Titre: Title:	Development of an Artificial Intelligence Model Able to Generate Designs for Software Interfaces
Auteur: Author:	Andrey Sobolevsky
Date:	2023
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Sobolevsky, A. (2023). Development of an Artificial Intelligence Model Able to Generate Designs for Software Interfaces [Mémoire de maîtrise, Polytechnique Montréal]. PolyPublie. https://publications.polymtl.ca/56715/

URL de PolyPublie: PolyPublie URL:	https://publications.polymtl.ca/56715/
Directeurs de recherche: Advisors:	Guillaume-Alexandre Bilodeau, & Jinghui Cheng
Programme: Program:	Génie informatique

POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

Development of an artificial intelligence model able to generate designs for software interfaces

ANDREY SOBOLEVSKY
Département de génie informatique et génie logiciel

Mémoire présenté en vue de l'obtention du diplôme de *Maîtrise ès sciences appliquées*
Génie informatique

Octobre 2023

POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

Ce mémoire intitulé :

Development of an artificial intelligence model able to generate designs for software interfaces

présenté par **Andrey SOBOLEVSKY**
en vue de l'obtention du diplôme de *Maîtrise ès sciences appliquées*
a été dûment accepté par le jury d'examen constitué de :

Amal ZOUAQ, présidente
Guillaume-Alexandre BILODEAU, membre et directeur de recherche
Jinghui CHENG, membre et codirecteur de recherche
Jin GUO, membre et codirectrice de recherche
Maxime LAMOTHE, membre

DEDICATION

To all those who are dear to me,
To my family who supported all my choices,
To my partner for her love

ACKNOWLEDGEMENTS

I would like to share my deepest gratitude to Pr. Guillaume-Alexandre Bilodeau, Pr. Jinghui Cheng and Pr. Jin Guo who guided me during my maitrise. The advises were always helpful, the encouragements always motivating, and the discussions always allowed to learn something new. With their help the progress of the the work was consistent.

I would like to also to thank my lab mates from LITIV who were always accessible for discussing technical problems to solve them. More importantly, the reading clubs each them allowed to share knowledge with everyone and discover new things.

RÉSUMÉ

La génération d'interfaces graphiques (GUI) est un domaine de recherche récent qui ne compte pas encore de nombreuses œuvres. Cependant, l'application de ce domaine peut avoir un impact positif dans la création de sites Web, d'applications et d'autres logiciels. Lors de la conception d'une interface utilisateur graphique (GUI), les concepteurs professionnels s'inspirent souvent du travail des autres pour définir le style lors de la phase de conception. Bien que cela soit une partie critique de la conception d'interface, la recherche d'exemples de conception inspirants et pertinents prend du temps. La plupart des concepteurs utilisent des mots-clés pour rechercher de l'inspiration, mais une entrée basée sur des contraintes pourrait être bénéfique pour les concepteurs afin d'obtenir de l'inspiration à partir d'exemples ayant la structure attendue. Dans notre recherche, nous nous intéressons à la génération de conceptions GUI nouvelles, créatives et intuitives basées sur les contraintes des concepteurs.

Notre contribution dans ce domaine commence par une méthode proposée, appelée GUILGET. GUILGET prend comme entrée un graphe de contraintes basé sur des graphiques appelé « graphique d'agencement GUI » (GUI-AG) et produit une disposition GUI réaliste en respectant la contrainte donnée. Notre méthode est basée sur des transformateurs et met en œuvre de nouvelles techniques inspirées du processus de conception GUI du monde réel. Ces techniques comprennent l'ajout d'informations hiérarchiques à l'entrée, l'inclusion des éléments enfants à l'intérieur de leur parent en utilisant une perte, la séparation des composants enfants partageant le même parent en utilisant une perte, et une grille avec une taille de cellule prédéfinie où tous les bords des composants sont ajustés au bord de grille le plus proche.

Dans la deuxième partie, nous avons étudié un pipeline complet appelé GILD en produisant des conceptions GUI à partir de GUI-AG. GILD est composé de deux étapes similaires aux étapes employées par les concepteurs GUI professionnels. Plus précisément, nous construisons d'abord les mises en page GUI à partir de contraintes sous forme de GUI-AG, après quoi nous générons la conception GUI. La première étape utilise GUILGET et la deuxième étape est construite avec le ControlNet, qui est une méthode pour affiner un modèle de diffusion latent pré-entraîné sur un grand ensemble de données.

Nous avons utilisé un ensemble de données disponible avec des captures d'écran annotées, l'ensemble de données CLAY. Lors de l'évaluation des mises en page GUI générées par GUILGET, nous avons découvert que la génération de mise en page à partir de GUI-AG capture mieux la sémantique et la logique des GUI-AG lorsque nous utilisons les transformateurs

plutôt que le réseau de convolution de graphes (GCN). De plus, la mise en œuvre de techniques inspirées du processus de conception GUI a montré de meilleures performances en ce qui concerne le respect des contraintes et des principes de conception GUI. Pour l'évaluation du style, certaines mesures quantitatives peuvent être utilisées, telles que le FID pour évaluer la qualité par rapport aux conceptions GUI de référence et le score de diversité qui mesure la diversité entre les conceptions GUI générées. Notre pipeline proposé présente les meilleurs résultats avec ces deux mesures. Nous avons ensuite mené une étude utilisateur avec cinq concepteurs UI/UX professionnels ayant des expériences différentes. Les participants à notre évaluation utilisateur ont décrit nos exemples de conception générée comme inspirants, visuellement attrayants et diversifiés. Cet outil pourrait faire partie de leur flux de travail de conception. En conclusion, nous avons apporté plusieurs contributions lors du développement d'un pipeline complet en deux étapes appelé GILD pour la génération de conceptions GUI à partir de GUI-AG. Ces contributions sont inspirées du processus de conception GUI.

ABSTRACT

GUI generation is a recent area of research that does not involve many works yet. However, the application of this area can have a positive impact in the creation of websites, applications, and other software. When designing a Graphical User Interface (GUI), often professional designers get inspired by others' works to define the style during the ideation phase. While this is a critical part of the interface design, searching inspiring and relevant design examples is time-consuming. Most designers use keywords for searching inspiration, but constraint-based input could be beneficial for designers to get inspiration from screens having the expected structure. In our research, we are interested in generating new, creative, and user-friendly GUI designs based on designer constraints.

Our contribution in this area starts with a proposed method, named GUILGET. GUILGET takes a graph-based constraint as input called GUI arrangement graph (GUI-AG) and produces a realistic GUI layout in respect to the given constraint. Our method is based on transformers and implements new techniques inspired from the real world GUI design process. These techniques comprise the addition of the hierarchical information in the input, the inclusion of the children inside their parent using a loss, the separation of children components that share the same parent using a loss function, and a grid with predefined cell size where all component borders are snapped to the closest grid edge.

In the second part, we investigated a complete pipeline called GILD by producing GUI designs from GUI-AGs. GILD is composed of two steps that are similar to the steps employed by professional GUI designers. More precisely, we first construct the GUI layouts from constraints in the form of GUI-AGs, after what we generate the GUI design. The first step uses the GUILGET and the second step is built with the ControlNet, which is a method for finetuning a latent diffusion model pretrained on a large dataset.

We used an available dataset with annotated screenshots, the CLAY dataset. During the evaluation of the generated GUI layouts by GUILGET, we discovered that the layout generation from GUI-AG better captures the semantic and logic of GUI-AGs when we use the transformers rather than Graph Convolutional Network (GCN). Moreover, implementing techniques inspired from GUI design process showed better performances in respecting constraints and GUI design principles. For evaluating the style, some quantitative metrics can be used such as FID for evaluating the quality in regard to the ground-truth GUI designs and the diversity score that measures the diversity among generated GUI designs. Our proposed pipeline shows the best results with the two metrics, which is in agreement with the results

from the user evaluation. We further conducted a user-study with five professional UI/UX designers with different background. Participants to our user evaluation described our generated design examples as inspirational, visually appealing and diversified. This tool could be part of their ideation workflow. In conclusion, we have made several contributions during the development of a complete two-step pipeline called GILD for GUI designs generation from GUI-AGs. Those contributions are inspired from GUI design process.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS AND ACRONYMS

GUI	Graphical User Interface
GUI-AG	Graphical User Interface Arrangement Graph
MLP	MultiLayer Perceptron
ML	Machine Learning
LT	LayoutTransformer
GAN	Generative Adversarial Network
LSTM	Long-Short Term Memory
CNN	Convolutional Neural Network
NLP	Natural Language Processing
FID	Fréchet Inception Distance
GCN	Graph Convolutional Network
CRN	Cascade Refinement Network
WSGC	Weighted Scene Graph Canonicalization
FC	Fully Connected
cLSTM	Convolutional Long-Short Term Memory
ROI	Region Of Interest
VAE	Variational Auto-Encoder
VT-CAtt	Visual-Textual Co-Attention
LPIPS	Learned Perceptual Image Patch Similarity
NDN	Neural Design Network
GMM	Gaussian Mixture Model
KL	Kullback-Leibler
MSE	Mean Square Error
CPI	Children-Parent Inclusion
CCS	Children-Children Separation
GUI-AGC	Graphical User Interface Arrangement Graph Correctness
IS	Inception Score

LIST OF APPENDICES

CHAPTER 1 INTRODUCTION

1.1 Context

Graphical User Interfaces (GUIs) are essential for enhancing user experiences. A GUI is composed of two types of components [11]:

- **Widgets** are small graphical control elements that serve as building blocks for constructing a user interface. Widgets are visual elements that enable users to interact with the software or application, such as buttons, menus, sliders, and text fields.
- A **spatial layout** that refers to a component responsible for the organization and arrangement of other components in the interface. The navigation bar is an example of spatial layout, several buttons are usually placed inside it.

GUIs are present in numerous applications and websites that we consume in big quantities every day. There are for instance some 255 billion downloads in 2022 on the mobile apps distribution platform *Google Play Store* [12], and this number is continuously growing every year. On a monthly basis, more than 75,000 mobile applications are released on the *Google Play Store* [13]. Nevertheless, after 30 days, there are less than 11.3% of those mobile apps that are still used [14]. This is due to high competition pressure and the quality of the app interface plays a role in the success. The illustrative example of Macintosh from Apple can be cited. Indeed, when this product from Apple was released, it had a more elaborate GUI than the one from Windows, which led to a higher appreciation of Macintosh and an increase of faithful clients by 20% [15].

Having established that GUI design is important for the success of software applications, it is however not an easy task. Designers face two major challenges when building GUIs:

1. Constraints and guidelines must be respected.
2. The design fixation results in GUIs that lack creativity.

Designing an effective GUI is a task that might take a lot of time and is difficult due to the large amount of guidelines and constraints to follow in order to have an app that is easy to use [16–20]. There are indeed many constraints and guidelines to follow such as respecting alignments, proportions, and defining an appealing color palette. Those guidelines allow to user to know where to look first and have an intuitive experience with the app. To stress

more on the fact that it is time-consuming to build a visually appealing interface, based on a survey study, it is observed that often in companies there are not enough GUI designers, so the company will transfer this task to the developers. The mentioned survey was conducted with 5,700 developers. Among them, 51% affirmed that they were involved in the GUI design task [21]. Another aspect that makes the GUI design process so complex is the fact that we must be up to date about the current trends and rules in this industry to follow the best current practices [16]. This is necessary to make the website intuitive for the user.

However, GUIs often suffer from a sense of tediousness as well as a lack of innovation. In recent years, the field of GUI design has become stagnant as there is a standardization in the interface designs that lead to a lack of excitement for the user [22]. The constant repetition of templates to which the designers are used to see is a limitation to the creativity and the exploration of novel ideas. In order to engage the designer in the interface exploration for GUI design, it is necessary to elaborate an innovative GUI design experience to make them curious to see the entire app or website. As Steve Krug [23] describes it in his book, innovative designs help to catch the curiosity of users who will explore further the website or the app. This observation may appear as a contradiction to the fact that the designers must follow the latest trends. However, the latest trends will help the designer in making an intuitive GUI for its exploration, but will not help in encouraging the creativity of the designer or in catching the curiosity of the user. Moreover, the combination of existing designs that were not combined previously is a creative performance that is only possible when the designer is aware of GUI design trends.

For all those reasons, a machine learning method to guide more GUI designers can help to reduce the time required to build an intuitive GUI design and improve creativity. This method should take as input the constraints from the GUI designer and the model should provide several examples of design respecting the constraints. The elaboration of a tool based on machine learning to help designer faces several challenges.

1.2 Challenges of ML-based Methods

In the following, we describe concrete challenges to be solved in the design of a ML-based method.

- There are a lot of possible types of input for the task of GUI design generation that might be considered. It is challenging to find an input that will allow at the same time freedom in expressing the constraints, and that can be interpreted by a ML method. If we consider an input with a high range of freedom, such as a textual input, it will

be harder for the method to simultaneously produce a visually appealing GUI design, and a design that corresponds to the description. The freedom makes the semantics of the input harder to capture. To make it easier for the method to understand what the designers desire, the usage of wireframes as input is a possibility. A wireframe is a spacial arrangement of components, more precisely, it is the tuple of position and the size of each component that is given as input. However, if the required input is a wireframe, it implies that the user of the method must be able to design wireframes and be an expert in the field of GUI design. Since this is not the case for most people, we might consider a directed graph as input. In the graph, we would have nodes as components that must be part of the interface, and directed edges indicating the relationship between two nodes. For instance, if we have a node "button", another node "text", and an edge from "button" to "text" that is labeled with "above", it can be read as "the button is above the text". This type of input allows to have a standard form of input, contrary to a textual input, which will help the method to capture better the semantic. Moreover, the expertise in the GUI design is not required with such constraints.

- The input size may vary a lot and be confusing for a ML-based method. For illustration, it is often the case that on a logging screen there are fewer components to be created than on a registration screen. It can become a problem since it is one of the goals of a ML method to have similar performances for different inputs. More specifically, for different complexity of inputs for this case. Figure 1.1 shows two different screenshots with its associated layout. Based on observation, it is clear that the first screen is smaller in terms of content than the second screen. Describing the more complex screen with an input is a more tedious task and may cause ambiguity in the ML-based method.

- The ML-based method must support creativity during GUI design. Since creativity is subjective, it is not intuitive how to formulate the objective of the algorithm. Concretely, the method might have difficulties to capture different aspects of human creativity such as colors combination, shadows size, and round corners. This leads to a lack of visual quality and an unsatisfying user experience. Moreover, the evaluation of a ML-based method that aims to generate new GUI designs is difficult. Indeed, there is a lack of quantitative metrics in the field of image generation mainly because it is hard to evaluate creativity.

- Two general aspects make a GUI design visually appealing. The first aspect is to produce a GUI design where the arrangement of GUI components respects rules imposed

Figure 1.1 Example of screenshots (left column) and its associated layout on the right. The screen on the top has a low amount of components inside while the screen on the bottom has a high amount of components.

in the field of GUI design to improve user interaction with the interface. Another important part is to fill all components with a style. The style of each component must give an interface with a high visual quality, and the components must fit with each other. This can be hard to perform all at once, hence an alternative option might be to consider a two-step ML-based method. The first step would be responsible for producing the GUI layout and the second one must add style to each component from the previously produced wireframe.

- A common problem for most ML-based methods is the lack of data. A generative method must be able to produce new designs from inputs that never appeared during the training. Without a large dataset, the effectiveness of the method will be reduced due to the lack of diversity in terms of design styles, user preferences, content and its

different possible arrangements. A limited dataset may result in generating GUI designs that are biased towards the patterns present in the training data, lacking innovation or generalization to novel input scenarios.

1.3 Research objectives

The main goal of this thesis is to design a method that generates GUI designs based on constraints given in the form of a directed graph. In fact, we aim to develop a machine learning (ML)-based method that will be as robust as possible toward the different challenges enumerated in the previous section. In summary, the research objectives are:

- Build directed graphs from ground-truth GUI layouts that capture all the positional constraints given by the designer.
- To develop a ML-based method for the new task of GUI design generation from graphs. At first, the ML-based method must produce a GUI layout from a directed graph given as input. Then at the second stage of the method, the GUI design with style should be generated from the GUI layout.
- To select and propose metrics to evaluate the proposed method and compare its performance another relevant method from the literature. At the same time, we will evaluate the performance of our method on the challenges stated in section 1.2

1.4 Thesis outline

In the following sections of the thesis, we will first describe the related works in chapter 2. After that, we start by giving an overview of the work that has been done and explain how it aligns with the research objectives we have set in chapter 3. In the two following sections that are chapter 4 and chapter 5, we describe our ML-based method that we developed. Then, we discuss the results in chapter 6 and finally, we conclude the work in chapter 7.

CHAPTER 2 BACKGROUND AND RELATED WORKS

In this section, we provide a comprehensive overview of existing research related to the work of GUI generation with AI. However, most of the works in the field are not solving the same task as we do, which is the GUI design generation from directed graphs. Moreover, the GUI generation task by itself is an emerging task and it does not include a lot of proposed works. So, we extend the literature review also on the image generation task. Indeed, image synthesis is one of the most active field of research with many great advances with for instance DALL-E 2, Midjourney and many others. It is relevant to learn about the existing research in the image synthesis field since the GUI generation is a close subject and many of its strategies can be also exploited. In addition to GUI generation and image generation, it is relevant to describe research about layout generation that can be used as a first step. Indeed, there are one-step methods that directly produce images from constraints, and two-steps methods that usually produce intermediary results in the form of a layout. Hence, it makes relevant to study the field of GUI layout prediction as well. And finally, one of our challenges will be to evaluate our method and it requires a deep understanding of the current evaluation metrics in the field of GUI and image generation.

2.1 GUI generation

2.1.1 GUIGAN

The field of GUI design is undergoing significant advancements, but the usage of AI for this field is still recent and there is not a lot of work yet. In 2021, GUIGAN [6] was the first proposed automated approach for the task of the GUI design generation. More precisely, the task in this method is to compose new GUI designs using a set of existing GUI elements that were extracted from a dataset. Even though the task is different for ours, it present many relevant strategies to deal with GUIs. Figure 2.1 is an illustration of the GUIGAN method and we will describe the different steps that are composing it:

Preprocessing: The RICO [24] dataset is the most common dataset in the field of GUI generation with 72219 UIs from 9772 mobile apps, spanning 27 GUI component categories. As explained in chapter 1, there are two types of components: widgets and spatial layouts. Those components are organized in a hierarchical structure that is exploited by GUIGAN. Indeed, the task is to compose new GUI designs by using existing subtrees of GUI elements. Subtrees are hierarchical representations with a parent component that contains children

Figure 2.1 Diagram of the method proposed by Zhao and al. [6]. The GUI dataset is preprocessed into a collection of subtrees that can be reused. A subtree is a portion of a GUI layout that has an hierarchical structure where components are grouped inside other components. Then a seqGAN [7] based model is used is used to generate new designs. ©2021 IEEE

components. The particularity is that all subtrees in an interface are considered, so there is not one tree which is considered but multiple different subtrees inside the tree. In fact, the goal is to produce the sequence $S_{1:T} = (s_1, ..., s_t, ..., s_T), s_t \in S$ of GUI component subtrees, where S is the subtree repository. Hence for the preprocessing, the subtrees of components from the GUI screenshots must be extracted. Another important point in the preprocessing is that some of the screens in the dataset are not relevant and must be removed for the training. In fact, the apps that are in the game category are different from regular apps in terms of structure and style so they may be removed. A game has typically a more uniform distribution of colors compared to regular apps where a little amount of colors is used in the entire interface. Some other apps where the whole screen is occupied by a large pop-up or an advertisement or a poster are low-quality GUIs and must be removed to not bias the training.

Model and Training: The model is mainly based on Generative Adversarial Networks [25] (GANs), which are made of a generative network and a discriminative network. The generator learns the features from the real-world data and produces new samples to fool the discriminator. The discriminator aims to distinguish the true sample from the fake one. The two networks are trained against each other and this training finishes once the generator is able to fool the discriminator most of the time.

More precisely the method in GUIGAN is based on seqGAN [7] that aims to produce a sequence of discrete tokens. Those tokens are encodings of design attributes into vectors. Based on random noise and a vector of tokens of real world GUIs as input, the generator aims to generate a new GUI design that matches the design attributes given as input. The generator in the architecture is a Long-Short Term Memory (LSTM) [26] and the discriminator is a Convolutional Neural Network (CNN) [27] with a highway. A highway [28] is a layer that allows the network to learn when to skip or propagate information through the layers.

It increases the convergence with deep CNN networks. The input will be the sub-images showing the subtrees that are converted to embeddings using a siamese network [29], which aims to determine if two images are cut from the same app. The entire GUI layout tree can be considered as a sentence and the subtrees are the words. This makes the problem very similar to a Natural Language Processing (NLP) problem.

Several aspects from the GUI design are taken into account through multiple losses. The first important aspect is to produce GUIs with a great visual quality. This is the role of the siamese network to achieve it with a cross-entropy loss over the prediction if the pair of images are coming from the same app. By doing so, the embedding vectors of screenshots will be close to vectors related to screenshots with similar appearance. This will help to combine subtrees of components that are visually fitting together. Indeed, they apply a technique named homogeneity (HOM) that evaluates the aesthetic compatibility of subtrees in the sequence. More precisely, it is measuring the proportion of clusters containing only members of a single app.

Another important aspect is to produce GUI designs with a realistic structure, which means the placement of components and their sizes are important in a GUI design to make it practical for the user. The minimum edit distance is used to quantify the structural similarity between two GUIs. The distance is measured between subtree structure vectors. Explicitly, one GUI is the generated one, and the other is a real world GUI. This distance must be minimized by the generator to learn better the structural combination.

Experiments: Since it was the first work published in the field of GUI generation, there was no other direct work to compare with. Hence, it was compared to similar works in image generation domain. For the quantitative results, two metrics were use: 1) the Fréchet Inception Distance (FID) [30] that measures the quality and diversity of the generated images and 2) the 1-Nearest Neighbor Accuracy [31, 32] to determine if samples have an identical distribution. The quantitative and qualitative results have shown that the GUIGAN performed better than other method. Moreover, removing either the structural loss or the appearance loss that was presented previously from GUIGAN reduces the performances. This proved that the appearance as well as the structure are important in a GUI design and moreover there are rules that are not present in real-world images that must be taken into account when designing a method for GUI generation.

To conclude on this work, there are many mechanisms that are useful to be considered, especially in the preprocessing and losses computation. However, the main architecture does not allow to choose the constraints that must be respected in the generated GUI design. Moreover, the produced screens are using existing components from the training data, which

does not allow to produce innovative interfaces.

2.1.2 GANspiration

Model and Training: Another work for the task of GUI generation is GANspiration [4], and as it is mentioned in itsname, it is mainly based on GANs like the previous method. The goal here is different. The method aims to produce variations of designs for a design given as input. To that end, the authors make use of styleGAN [33] to encode the input image to a latent code. However, by opposition to GUIGAN, it is not considering a tree structure in the encoding, but the screenshot directly. This obtained source latent code is then going through a new example synthesizer that is also made of a styleGAN. The entry of this component is composed of 1) the source latent code that was produced with the previous module, and 2) the target latent codes that are any vectors from the latent space. The styleGAN aims to merge the styles of the source image and of the target image by replacing the latent code of the source image. This component of GANSpiration produces 136 image variations for each pair of source and target inputs in order to cover all possible levels of granularity of the inference. In order to reduce the amount of produced images and to filter the most relevant ones, another component is used in the method which is called representative examples selection. This module is applying a clustering technique, DBSCAN [34], on the large amount of produced images from the previous step. It clusters the images through perceptual similarity that is calculated between every combination of pair of produced images. Withing each cluster, the image, considered by the discriminator from the styleGAN that is the most realistic GUI image, is used to represent the entire cluster and the rest of it is removed.

Experiments: Once again, due to the lack of dataset in the field of GUI, only the RICO [24] dataset is used to train and evaluate the method. For the experiments, a new notion appears that was not present in GUIGAN, which is the notion of screenshot complexity, or number of unique component types. In fact, the complexity was directly correlated to the performances of the method. This is a relevant study that must be done in every method related to GUI generation to have an understanding about how deep the model understands the GUI and at what granularity. In this work we made a study in the chapter 4 to observe the impact of the complexity on the performance of GUI layout generation method.

The method is limited by a fixed UI that must be provided to produce new ones. Indeed, components that are composing a UI cannot be removed or added. For this reason, we aim to propose a method that is based on constraints between components given as an entry. Also, since there is a style transfer that is applied, several artifacts may appear in the design.

Generating a GUI design from scratch would help to keep every component coherent with each other.

Even though our task does not match with what has already been proposed in those research works, we use in our method several techniques that are directly inspired from what is applied in those works to process the GUI dataset, to improve the method by adding task specific losses and to correctly evaluate the generated GUIs.

2.2 Image generation

Since GUI generation is a center of interest in the research only recently, there are not many works done yet that can help to understand the best practices in GUI generation and to guide our choices for the method. On the other hand, the image generation is a well-known field and has several interesting works that can be used as inspiration for our research.

Generating images from text is a hard task, therefore, the method called SG2IM [2] is the first one to make use of scene graphs as constraints for the task of image generation. It showed better performances than the state-of-the-art in the image generation from text at that time, which was the StackGAN [35]. Scene graphs provide a higher level of abstraction of an image by capturing the object and their positional relationships within the image. SG2IM is an approach that aims to produce visually and structurally coherent images from scene graphs.

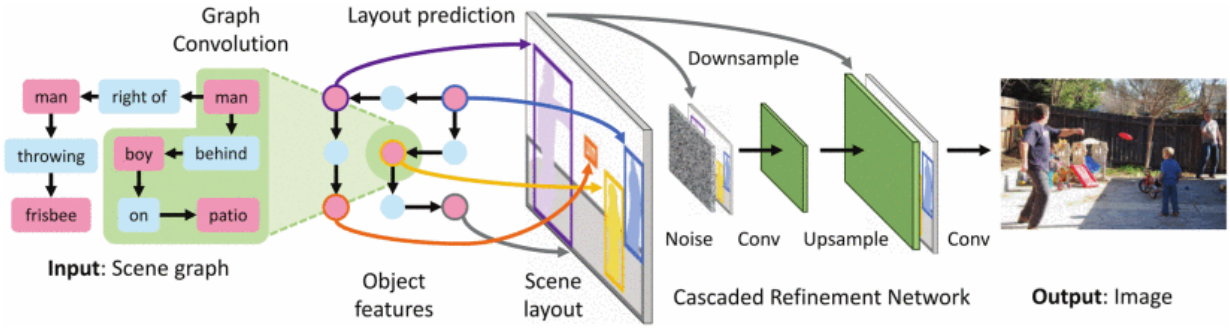


Figure 2.2 Diagram of the method proposed by Johnson and al. [2]. Stating from a scene graph representing the relationships between components as input, the object features are extracted using a graph convolutional network. These features are then processed by a layout prediction network to produce the layout of the scene. Finally, the image can be produced with the cascade refinement network. ©2018 IEEE

The SG2IM method is illustrated in Figure 2.2. It is a two-step method. It first generates a layout, and then the image. As we can observe, it is combining the effectiveness of the

Graph Convolutional Network (GCN) and the Convolutional Neural Network (CNN). More precisely, the GCN is encoding the scene graph for representing the structure of the image that must be produced. A scene graph, as illustrated in Figure 2.2, is a directed graph with edges $E \subseteq O \times R \times O$, where O is a set of objects and R is a set of relationships. At the first stage, each node and edge of the graph are converted from a categorical label to a dense vector (embedding). The GCN will process three objects o_1, o_2 and o_3 and two edges that share a common object (o_1, r_1, o_2) and (o_3, r_2, o_2) as shown in Figure 2.3. The embeddings of the objects are then processed in a object layout network to predict the mask and the bounding box for each object to build the scene layout. The final step of the method is to produce an image from the obtained scene layout. To achieve this, a Cascade Refinement Network (CRN) is used. It takes as input the scene layout and 2D random noise map and the network consists of a series of convolutional refinement module with spatial resolution doubling between modules.

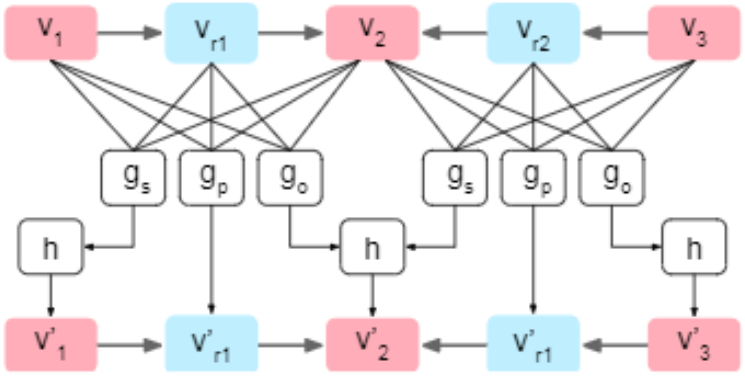


Figure 2.3 Illustration of one layer of the GCN used in SG2IM, where we have input vectors $v_i, v_r \in \mathbb{R}^{D_{in}}$ for all objects $o_i \in O$ and edges $(o_i, r, o_j) \in E$. The output vectors $v_i', v_r' \in \mathbb{R}^{D_{out}}$ are computed with functions g_s, g_p, g_o that are multilayer perceptrons (MLP). For computing v_r, the function directly updates the vector, while for the objects both functions g_s and g_o produce a set of candidate vectors. The symmetric function h is then applied to that pools a set of vectors by averaging them and outputs a single vector by feeding a MLP. ©2018 IEEE

It is important to describe how the GCN works here. Indeed, as illustrated in the Figure 2.2 with the green highlight over the scene graph, each node is encoded with the knowledge of its direct neighbours before and after. This implies that semantic equivalents may not be captured. To illustrate that if we have an object A that is above object B, it is also correct to say that object B is below object A. However, only one of the two relationships is kept in the scene graph. In consequence, multiple scene graphs can represent the same configuration. Those equivalences are hard to be captured by prior methods, especially for large scene graphs. Moreover, it captures well the local relationships, but some global

incoherences may appear. For all those reasons, Herzig et al. [36] proposed a similar method with numerous modifications to solve the cited issues. The main contribution was to integrate the canonicalization to the scene graph before it is encoded using a new Weighted Scene Graph Canonicalization (WSGC) method. In practice, a scene graph may have several possible logically-equivalent scene graphs. Hence the goal of this work was to propose an approach to canonicalize graphs to enforce invariance. This approach comes from observations that the relations in the scene graph are often dependent. Indeed, if we consider transitive relations such as "Left" with the two edges (o_1, Left, o_2) $\in E$ and (o_2, Left, o_3) $\in E$, we can directly conclude that (o_1, Left, o_3) $\in E$. Another case, is the case of converse relations. In fact, if (o_1, Left, o_2) $\in E$ we can say also that (o_2, Right, o_1) $\in E$. The WSGC method aims to produce the same image for all logically equivalent graphs to avoid being overly sensitive to the edge information. To do so the scene graphs are transformed to a canonical form. The most natural canonical form is the "relation-closure" which is the graph containing all relations implied by the initial scene graph. In practice, this is achieved by:

- The converse completion: We add the converse relations, e.g. if there is an edge with the "Left" relation, we add the equivalent "Right" edge in the other direction.
- The transitive completion: We calculate the transitive closure for each relation that is transitive. It is calculated with the Floyd-Warshall algorithm and is added to the graph.

However there is something that is still unknown: How to determine if a relation is transitive or converse? There is no certainty that a relation will always be a converse relation or a transitive relation. By consequence, weights or probabilities are assigned on the edges to reflect the uncertainty.

To produce the images from the layout obtained from the previous step, the model proposed a novel generator named AttSPADE. It is made of a series of residual blocks with upsampling layers. The main characteristic of this generator is that the condition on which the model is based is not only the class of the object in the layout but also its attributes. By attributes, we refer to some features of the object, such as its color. Those attributes are encoded via a multi-hot encoding, and then a Fully Connected (FC) layer is applied to obtain a vector $v \in \mathbb{R}^d$.

Experimentally, the observations made in this research were shown to be correct, the method outperformed SG2IM that was presented previously. Indeed, layouts produced by the method from Herzig and al. have better mean IOU and recall values in general. The generated images

were evaluated by humans, and 96.8% of these humans preferred the images produced by the method from Herzig and al. rather than by SG2IM. It shows that the task of scene graph preprocessing is important and can improve results significantly.

Another alternative is to generate images from layouts. It is common at the first stage of layout to image generation to produce a feature representation for each bounding box based on their class. Then this representation goes through a generator to produce an image. Depending on how this feature vector is processed, there are two separate groups of generators.

- Layout to image models with Encoder-Decoder Generators: These models convert the feature vectors obtained previously into realistic images. These models consist of an encoder component that processes the input, extracting relevant feature maps, and a decoder component that reconstructs the image based on those feature maps. As an example in this category we can cite the Layout2Im [37] method where the encoder network maps the layout into a latent space, which is then decoded by the generator network to produce the final image.
- Layout to image models with Decoder-only Generators: They directly produce images from given feature vectors as input. So the model is not extracting the most important information but rather uses the raw features directly. The work of this thesis for instance is a Decoder-only generator as we will see later as well as the method of He and al. [8].

As mentioned, the Layout2Im [37] falls into the category of Encoder-Decoder Generators. In this method, the object feature maps are constructed based on the latent code of objects, the word embedding representing the encoding of the class, and the layout. The feature map is then fed into the object encoder made of several convolution layers and object fuser sequentially, which is built with a convolutional Long-Short-Term Memory (cLSTM) network [38], generating a fused hidden feature map containing relevant information about all the objects in the layout. The image is reconstructed from the fused hidden feature map by a decoder. The model is trained adversarially with a discriminator evaluating the image as well as objects individually.

The context of an object given as input is important and this is even more accentuated in the work from He and al. [8]. In this work, the task is to produce visually realistic images from given layouts. The layout is made of bounding boxes with a class associated to each one. The method proposed by He and al. uses the Decoder-only generator as shown in

Figure 2.4. We can observe in the first part of the method that there is a context-aware feature generation. This part is the main contribution. Indeed, prior models processed each bounding box independently without considering the other objects in the layout. In consequence, the generated objects did not fit well together. The context-aware feature generation module is a way of overcoming this issue. In practice, it integrates contextual information into the feature representation of each bounding box by allowing each feature to cross-examine all other features via self-attention [39]. At the end of the process, the feature vector has additionally to it own information, the global context in the layout.

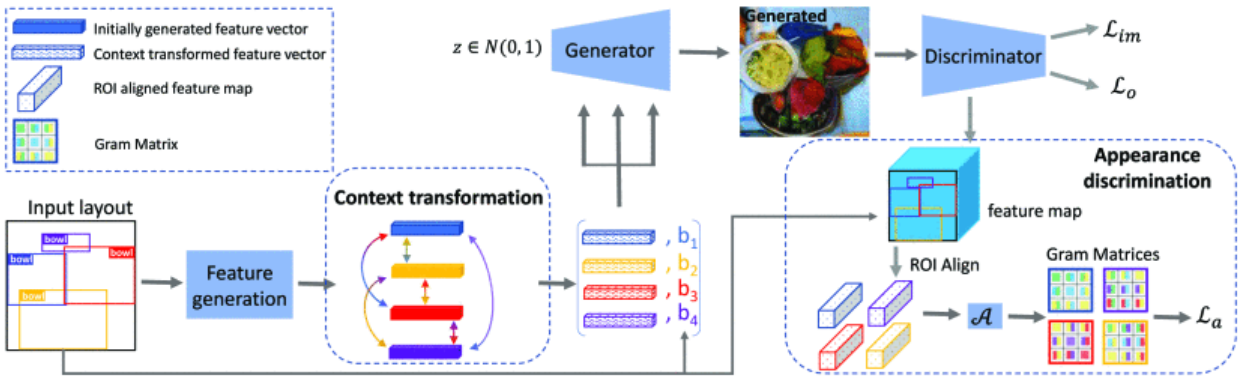


Figure 2.4 The architecture proposed by He and al. [8]. In this architecture, features are first extracted for the representation of each bounding box of the input layout. The context transformation is then applied to include the global context to each of the feature representation. Using as entry a random distribution z, coordinate of the bounding box b_i and the feature representation of the bounding box, the generator aims to produce an image. The generated image is compared to the real images by the discriminator with three losses: (1) image-level loss (2) object-level semantic loss (3) object-level Gram matrix loss. ©2021 IEEE

The generator takes as input, random noise and the new feature representation and the discriminator plays against the generator by trying to differentiate its produced images from real images. Previous methods usually have two losses coming with such discriminator: the image-level loss according to the entire image, and the object-level loss that is based on the Region Of Interest (ROI) pooled feature of each object in the image. We can notice that the two losses are insensitive to the location. So the authors proposed another module that is inserted in the method which is the appearance discrimination module. This module copes with the lack of location-sensitive appearance representation in the existing discriminator. The Gram matrix is introduced in this work as feature maps that are appearance representations. The loss of this module penalizes the spacial misalignment of each semantic feature between the generated and real images. Concretely, the object Gram matrix of feature maps is fed to the discriminator classification layer.

The experiments have shown a significant improvement in the quantitative and qualitative results. For the quantitative part, the metrics FID [30], inception score [40] and diversity score [41] were used. For each of them, the method from He and al. [8] outperformed all the previous ones. This is also the case with humans that rated the produced images. The images generated by this method had better rating than the others as well. We can conclude that the knowledge of the context when processing each object in a model can improve the results significantly.

The transformers have shown great performances in NLP for understanding the semantic of the input with the help of attention. Indeed, transformers have great abilities to capture complex and long-range dependencies in data, which make then also highly effective for image generation. There are several works that used them for this purpose. We can cite for instance the work from Zhao and al. [9], which aims to produce images from scene graphs. This is a one-step method as we will see in more details.

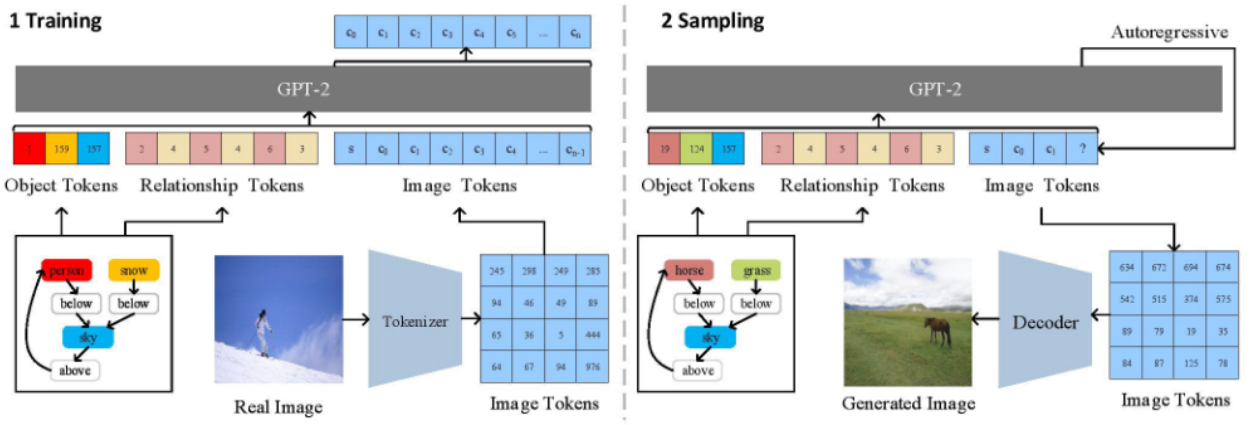


Figure 2.5 The method proposed by Zhao and al. [9]. For the training, the scene graph as well as the image are tokenized. The image tokenizer is based on VQGAN encoder [10]. The tokens are concatenated before being fed into the GPT-2, which goal is to predict image tokens from the scene graph in an autoregressive way. The image is generated from those tokens using the VQGAN decoder. ©2022 IEEE

A transformer requires a sequential input, and as we can observe in the Figure 2.5, this is done by tokenizing the input. Let us consider that the scene graph is defined by (O,S), where $O = o_1, ..., o_n$ is a set of objects and $S \subseteq O \times R \times O$ is a set of directed edges with $R = r_1, ..., r_n$ being the set of relations. (O,S) can be encoded as a linear structure with the concatenation of the object tokens and the relationship tokens. The process is illustrated in Figure 2.6.

Going back to Figure 2.5, a discrete autoencoder VQGAN [10] can be used as the tokenizer

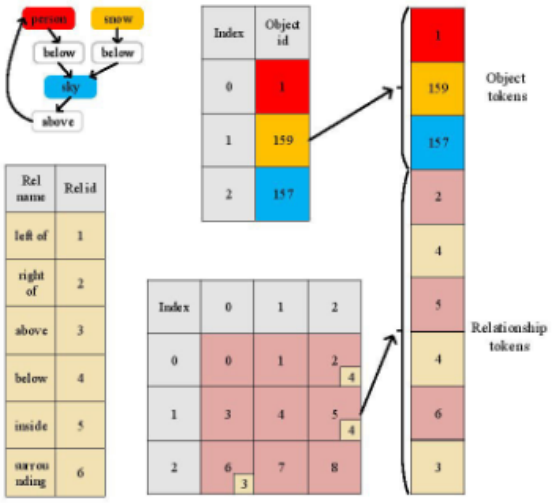


Figure 2.6 A schematic diagram of the proposed method to tokenize the scene graph by Zhao and al. [9] ©2022 IEEE. The object ids are placed at the beginning of scene graph tokens. The relationship tokens contain the edge and weight information then follow. Edge information is expressed by an adjacency matrix grid. Weight information is the relationship id.

of the image but also for decoding the image tokens. Those image tokens can be seen as high-dimensional representation of the image, and the transformer has only to predict those. The GPT-2 is used as transformer network for the autoregressive learning. The experiments showed better results for the method of Zhao and al. [9] when compared to the methods with GCNs presented earlier. It shows that transformers have a better understanding of the linear input that is given.

In recent years, diffusion models [42, 43] have gained considerable attention due to their unique approach to probabilistic modelling and generation. The strength of diffusion models is the ability to outperform the other models in the image generation due to it ability to capture fine-grained information more than other methods. Moreover, it has also the ability to produce diverse samples. Unlike traditional generative models that directly model the data distribution, diffusion models operate by iteratively refining a random noise source to approach the target distribution. This iterative refinement process allows diffusion models to capture intricate details and high-frequency patterns in the data, resulting in visually compelling and coherent samples. If we dive into details, during the training, the source image goes iteratively through noise adding steps and this is called the forward process and is formulated as

$$q(x_{1:T}|x_0) := \prod_{t=1}^{T} q(x_t|x_{t-1}) \tag{2.1}$$

with

$$q(x_t|x_{t-1}) := \mathcal{N}(x_t; \sqrt{1-\beta_t}x_{t-1}, \beta_t I), \tag{2.2}$$

where x_0 corresponds to the source image given for training. This is a Markov chain that gradually adds Gaussian noise to the data according to a variance schedule $\beta_1, ..., \beta_T$.

Conversely, there is the reverse process that aims to reconstruct iteratively the source image from the noise obtained from the forward process. It is defined as a Markov chain with learned Gaussian transitions starting at $p(x_T) := \mathcal{N}(x_T; 0, I)$. This can be formulated as following

$$p_\theta(x_{0:T}) := p(x_t) \prod_{t=1}^{T} p_\theta(x_{t-1}|x_t) \tag{2.3}$$

with

$$p_\theta(x_{t-1}|x_t) := \mathcal{N}(x_t; \mu_\theta(x_t, t), \Sigma_\theta(x_t, t)). \tag{2.4}$$

When sampling is performed, random noise is provided and the reverse process from the trained diffusion model will produce a new image. This is an unconditional generative method but some other work proposed to guide the diffusion model with a condition mechanism. This is the case with the recent ControlNets method [5]. The main contribution of this work is that it takes not only the random noise and a text prompt as entry but also a visual type of condition such as Canny edges, Hough lines, segmentation map, etc. This enables to guide better the model toward the results we would like to see, which is really hard to do sometimes with only a text prompt e.g. if we want to see a building from a specific perspective for instance, it is hard to describe textually. An advantage of this method is that it is fine tuning a text to image diffusion model to preserve the advantages of a large model trained on billions of images.

ControlNets introduces two main ideas. The first one is to lock the parameters of the diffusion model, it is called the "locked copy", and copy them in a duplicated network where the parameters are trainable, it is called the "trainable copy". So the original network is not modified while the copy is trained, this allows to have a more robust model while not reducing the performances of the large diffusion model. The second idea is the "zero convolution" layer, which is a 1x1 convolution layer with both weights and biases initialized with zeros. It allows to fine-tune the diffusion model without adding noise to the deep features since at the beginning the model will be equivalent to a diffusion model. If the weights and biases were initialized with random values, it would modify the diffusion model and produce poor results. The research presented only a positive qualitative evaluation and we can see that

the images produced by the model are of a very high quality.

In conclusion, the image generation is a very vast area where a lot of different techniques are explored, much more than in GUI generation currently. Several techniques from this field can be applicable to GUI generation since they have shown great performances. Moreover, some of those methods are one-step methods and other are two-steps. One-step methods offer simplicity and efficiency by directly mapping input data to the desired output images. For instance, the method from zhao et al. [9] is directly predicting the image from scene graph. In case of GUI design generation, the GUI design would be produced from scene graph. They are suitable when the task can be effectively solved using a single model. Two-step methods allow to decompose a task in multiple steps to make problems less complex. SG2IM [2] falls in this category as example by predicting the layout then the image. In a GUI generation process, one can see a GUI layout generated from constraints in the form of scene graph, then, based on the GUI layout, the GUI design is generated.

2.3 Layout generation

As discussed in the introduction of chapter 2, methods in two steps or one step are possible for GUI generation. In two-step methods, the first one is to produce a GUI layout from user constraints and the second one is to produce the GUI design from the GUI layout. For this reason, we also explore the layout generation task in the literature review. Several works from the previous section include the layout generation in their method such as SG2IM [2] and the work from Herzig et al. [36]. But there are many other works that are only focused on the layout generation and are relevant.

In the category of GANs, the layoutGAN method [44] was proposed. In this method, the generator takes as input a random object class and its bounding box; those random values are selected from Uniform and Gaussian distribution respectively. After the input is defined, it is fed into an encoder that embeds the input and feed the output to a stacked relation module. This module is a self-attention module inspired by Wang et al. [45], and is used to embed the relation of a graphic component with all the other elements in the design. Finally, a decoder decodes the embeddings back to class probabilities and bounding box. Since it is a GAN model, there is also discriminator that is applying a CNN on the 2D wireframe images that can be drawn with the knowledge of the class the geometric parameters. The discriminator predicts if the wireframe design is a real one or a fake.

Paired with GANs, Variational Auto-Encoders (VAE) are also well-known since they are also a type of model to perform generation from a latent space extracted from high dimensional

input. LayoutVAE [46] is a proposed method to generate layout from user constraints. The constraints required from the users are only the class of objects that they would like to see on the resulting layout. The method consists of two parts: 1) It starts at the first stage to predict the count for each class in the set of classes defined at the beginning. This is performed with the CountVAE, which is a conditional VAE designed to predict conditional count distribution for each class in an autoregressive manner. More simply, it generates the number of objects of each class to generate and put in the layout, 2) The second stage of the method is the BBoxVAE, which is very similar to CountVAE but rather than predicting count distribution, it is designed to predict the distribution of the coordinates of the bounding boxes autoregressively.

Even though GANs and VAE have shown great performances at that time, the ability of transformers to understand the semantic of the input with the help of the attention allows them to outperform such models. Therefore, Gupta et al. [47] proposed an autoregressive method using a transformer. The model takes as input layout elements and predicts the next layout element and it bounding box as output. For the training process, the teacher forcing is used, meaning that the ground-truth layout tokens obtained with an encoder are used as input to a multi-head decoder block. The first layer of this block is a masked self-attention layer, each block of the layer consider only the previous input in order to predict the following one. In terms of experiments, the transformer outperformed the LayoutGAN [44], LayoutVAE [46] but also the SG2IM [2]. This comes from the strength of the attention to understand the complex and long range previous elements that are in the scene.

The only problem with the previous method is that it does not allow to choose the constraints properly. Concretely, the methods does not take as input the information about what components should be part of the layout and their relative positions, only one component is given as input and the rest is chosen by the model. Hence, another transformer-based method, named LayoutTransformer [3], proposed to use scene graphs as input to produce layouts. As we previously saw, scene graphs are a simple and explicit way of defining constraints; but also they enable to modify the nodes and edges easily. LayoutTransformer [3] method is composed of three main components, the first one being the Object/Relation Predictor that encodes the scene graph to have a better semantic and logical understanding of the objects that are part of the graph. Then the output of the Object/Relation Predictor is used as input for the Layout Generator to produce bounding boxes in an autoregressive manner. Since those bounding boxes are produced sequentially, they may be not taking into account the entire layout. So there is a third component, the Layout Refiner, which is a Visual-Textual Co-Attention (VT-CAtt) module [3]. This module predicts for each bounding box the spatial shift that should be applied.

We were considering to build a two-step method in which the first step is to produce the GUI layout from the graph and then the design from the layout. This enable a more flexible workflow and steps that are dedicated to only one task, which is easier to evaluate in terms of performances. There are several works presented in this section that can be inspiring for designing the first step of our method.

2.4 Evaluation metrics

We are interested in metrics that are related to the GUI generation task. Since the method can have two steps, we should be able to evaluate the two different outputs. The first one corresponds to the evaluation of the GUI layouts, while the second one is the evaluation of the GUI design in terms of visual quality.

In the field of scene layout, several evaluation metrics have been proposed:

- **Alignment** [48]: The Alignment is required in most design creation so this is an important factor to consider when producing GUI designs. Concretely, we expect a component that is part of the layout to align with an edge or the center of another component. In the work from Lee et al. [48] only the vertical alignemnt (left-, center-, right-aligned) is considered. Mathematically, the quality of alignment is formulated as

$$\frac{1}{N_D}\sum_{d}\sum_{i}\min_{j,i\neq j}\{\min(l(c_i^d,c_j^d),m(c_i^d,c_j^d),r(c_i^d,c_j^d))\}, \tag{2.5}$$

  where N_D is the number of generated layouts, c_k^d is the k_{th} component of the d_{th} layout. Moreover, l, m and r are alignment functions where the distances between the left, center and right borders of components are measured, respectively.

- **Overlap** [44]: Often, components in a real document layout are not overlapping. To evaluate the quality of a document in terms of overlapping, Li et al. [44] proposed to use an overlap index that represents the percentage of the total overlapping area among any two bounding boxes inside the entire layout.

- **DocSim similarity metric** [49]: DocSim is a metric measuring how similar are two document layouts, which is very similar to the BLEU metric [50] for machine translation task that measures sentence similarity. In the DocSim metric, two documents D_1 and D_2 are compared as follows: to any pair of bounding boxes $B_1 \in D_1$ad $B_2 \in D_2$, we assign a *weighted edge* that indicates how similar are those bounding boxes in terms of size, position and class. It is formulated as

$$W(B_1.B_2) = \alpha(B_1, B_2)2^{-\Delta_c(B_1,B_2)-C_s\Delta_s(B_1,B_2)}, \tag{2.6}$$

where Δ_c is a location parameter, more precisely, it is the euclidean distance between the center of the two bounding boxes. The shape parameter is defined by $\Delta_s(B_1, B_2) = |w_1 - w_2| + |h_1 - h_2|$, where w_i and h_i are respectively the width and the height of the i_{th} bounding box.

As larger bounding boxes have a more significant impact on the appearance of the document, we would like to assign more weight to edge between larger boxes. So we can write that $\alpha(B_1, B_2) = \min(w_1h_1, w_2h_2)^C$, where we choose $C = \frac{1}{2}$ and $C_s = 2$.

To calculate the score, we have now to aggregate all the weighted edges. If we consider a bipartite graph where one part contains all boxes of D_1 while the other part has all the boxes from D_2. We find a maximum weight matching $M(D_1, D_2)$ in this bipartite graph using the well-known Hungarian method. We can then compute the score as

$$\textsc{DocSim}(D_1, D_2) = \frac{1}{|M(D_1, D_2)|} \sum W(B_1, B_2). \tag{2.7}$$

- **Wasserstein distance** [51]: Wasserstein distance is a efficient way of computing the diversity in the produced layouts. This distance is measured between the distribution of the real data and the generated one. In practice, this distance is computed for two marginal distributions: the class distribution and the bounding box distribution (x,y,w,h). A class distribution is extracted from the training data and another class distribution is extracted from the produced layouts for computing the wasserstein distance. Similarly, the bounding box distributions are used to measure the distance. The particularity with bounding boxes is the fact that each value from the tuple is considered independently as a distribution.

- **IoU** [52]: IoU stands for intersection over union and aims to measure how similar is a bounding box from a predicted layout compared to the same bounding box from the ground-truth data. If we consider that B_1 and B_2 are respectively the bounding box from a generated layout and the bounding box from the ground-truth layout, we can define IoU as

$$\text{IoU} = \frac{|B_1 \cap B_2|}{|B_1 \cup B_2|}. \tag{2.8}$$

- **Relation accuracy** [3]: this metric was proposed by Yang et al. [3], and it is specific

to the scene graph to layout task. For this metric, only relations with explicit spatial meaning (i.e. *left*, *right*, *above* and *below*) are considered. Then the relation accuracy is measured for each pair of objects by measuring the x, y distances between the boxes.

It is relatively easy to quantify the performance of a model generating layouts, but it is not the case with models generating images, or GUI designs in our case. Indeed, producing images is a creative work and in real-life it is mostly evaluated qualitatively. However, there are still a few evaluation metrics that are commonly used to evaluate an image:

- **Fréchet inception distance (FID)** [30]: This is a commonly used evaluation metric to assess the quality and diversity of generated images produced by generative models. It measures the similarity between the distribution of generated images and the distribution of real images. FID combines both the mean and covariance of the feature representations of the generated and real images to compute a distance score. The feature representation is obtained from pre-trained inception V3 model [53]. A lower FID score indicates better similarity between the generated and real images, implying higher quality and diversity of the generated samples. We can formulate it as

$$\text{FID} = \|\mu_{\text{real}} - \mu_{\text{fake}}\|^2 + \text{Tr}(\Sigma_{\text{real}} + \Sigma_{\text{fake}} - 2(\Sigma_{\text{real}} \cdot \Sigma_{\text{fake}})^{0.5}), \tag{2.9}$$

  where μ is the mean value of the feature representation, while Σ is the covariance. Tr refers to the trace of a matrix.

- **Learned Perceptual Image Patch Similarity (LPIPS)** [41]: This is a perceptual similarity metric often used in image generation works. It is used to quantify the perceptual distance between two images. The strength of this metric comes from an evaluation of the resemblance of images as perceived by humans. This is possible with the help of deep learning model trained on very large datasets. This network is used to compute distances between patches in images. The most reliable network to compute this distance is the VGG [54].

- **Diversity score** [3]: In the field of image generation, this is a metric that quantifies the variety or diversity of the generated images produced by a generative model. It measures how distinct and different the generated samples are from each other. LPIPS is commonly used to compute the distance between each pair of generated images and diversity score can be obtained by calculating the mean distance.

- **Inception score** [40]: The Inception Score is an evaluation metric used to assess the quality and diversity of generated images in the field of image generation. It measures

both the quality of individual generated images and the diversity of the generated image set. The Inception Score utilizes a pre-trained Inception Network, originally designed for image classification tasks. The basic idea is to calculate the entropy of the class probabilities predicted by the Inception Network for each generated image. High entropy indicates diverse and varied predictions, while low entropy suggests repetitive or biased outputs.

CHAPTER 3 OVERVIEW OF THE METHOD

GUI designers usually never perform the process of GUI design at once. They most often follow two steps to build a GUI design [19]:

1. The first step, as shown in Figure 3.1, is to sketch out a **wireframe** or the **GUI layout** as we will name it. A GUI layout is a visual arrangement of elements in a user interface. More precisely, it is the structure of the interface with the position and size of its components without providing their style.

2. The next step is the **visual design**, which is the process of adding style to each component of the GUI layout. The result of it will be the final GUI design product.

In this work, we followed these two main steps. The first step is researched and presented in an article in the next chapter. The second step is described in chapter 5, more precisely that section is a presentation of the entire two-step pipeline with a focus on the style part.

Our method is illustrated in Figure 3.1. It is composed of two different neural network models that will work together. There are several advantages to make this process in two steps instead of doing it in a single one, which is also possible as seen in the literature review:

- It allows the user to modify the layout before the final designs generation without being distracted by style elements. Indeed, the first step will produce GUI layouts based on user constraints but the user may want to validate and modify the position and/or the size of some of the components.

- We train two models independently which means if the results from one of the two step is satisfying, we do not need to train it again. With a 1-step method, improving one of the aspect of the design could lead to a decrease of performance of another aspect. It might be a limitation in the performance of the model.

- With two separate steps, we can evaluate the arrangement of GUI components with quantitative metrics to develop a better method. This is not the case if we have a 1-step method which gives directly the design.

The choice of how to represent user constraints is crucial. There are several options to represent constraints, such as using text or labels for instance, but in our method we instead propose to use what we call GUI Arrangement Graphs (GUI-AGs) as illustrated in

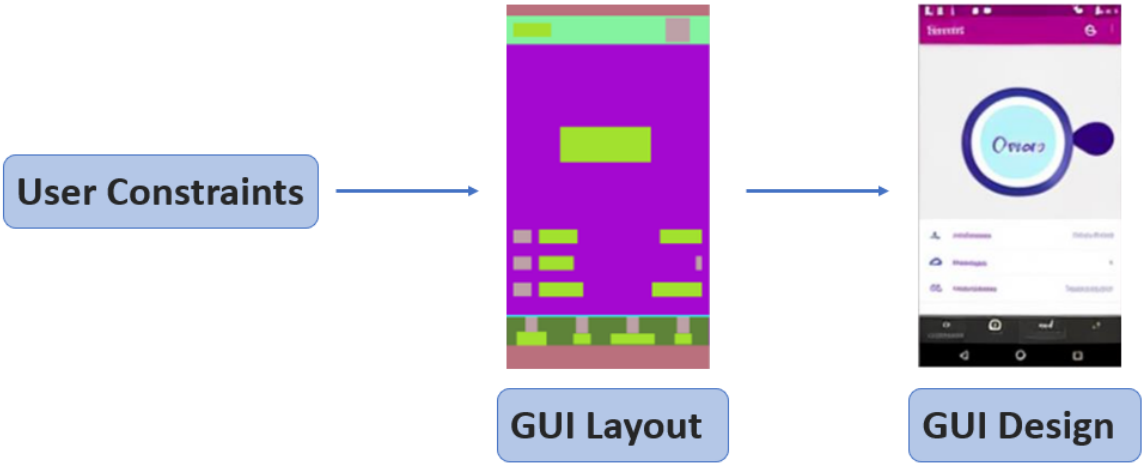
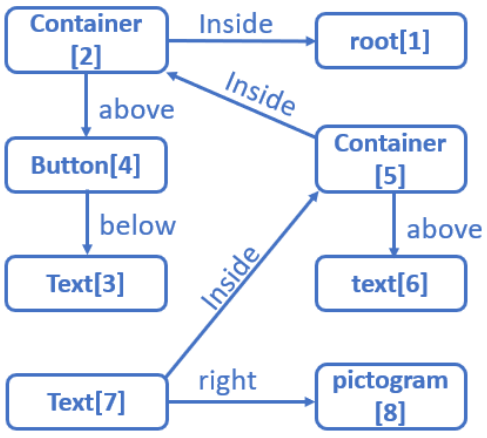
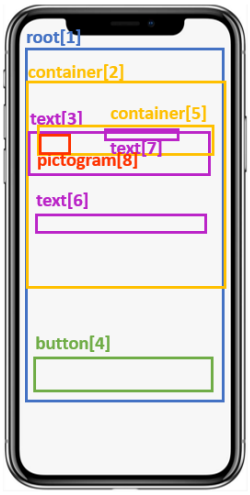


Figure 3.1 The overview of the two-steps pipeline of GUI design generation from constraints. First, we have to produce a GUI layout from user constraints then, the second step is to produce the GUI design screenshot from the generated GUI layout.

Figure 3.2. A GUI-AG is a directed graph where each node represent a component from the GUI and arrows are the relative position between two components and express the constraint between them. For instance, in the example of Figure 3.2a there is a constraint that we want *Container[2]* inside the *root[1]*. We can notice that GUI-AG are basically made only of triplets, which we will name relationship and represent *subject-predicate-object* (e.g. *container[2]-inside-root[1]*).



(a) GUI arrangement graph (GUI-AG)



(b) GUI layout

Figure 3.2 A GUI layout (b) corresponding to the GUI-AG (a) given by the user.

There are several advantages to the use of GUI-AG to represent the user constraints in our method compared to other type of constraints (e.g. textual):

- We can explicitly define the components that are part of the GUI and the relationships between them. Moreover, there is no restriction over the number of relationships that a component may have.
- A GUI-AG provides a hierarchical structure with the *inside* predicate for organizing GUI components. The parent-children relationship is important in GUI because it enables an efficient management of elements inside the GUI. Indeed, by grouping several GUI elements (i.e. children) inside another one (i.e. parent) we can then easily control all the collection of children by controlling only the parent.
- GUI-AGs allow flexibility. In fact, GUI elements and predicates that are used in the GUI-AG can be easily modified, rearranged or extended without impacting the rest of the constraints. It makes easier to adapt the GUI by simple modifications.

3.1 Challenges

Our generative 2-step method must tackle multiple challenges:

- GUI-AG complexity: The complexity of a GUI-AG is related to the number of nodes and relationships there are. Dealing with the complexity and ensuring that all constraints are respected as well as producing a visually appealing design is challenging. The other GUI generation works described in the chapter 2 do not have to deal with this challenge since they do not have structural constraints. The goal of those works is rather to provide diverse inspirational designs based on a design given as input. In contrast, in our work we want to instead use the design by constraints paradigm. In image generation this is also a major challenge especially in works based on GCN where a GCN layer considers only two relationships at a time which makes it harder to include global contextual information. Therefore, transformers appear to be a better choice to handle this complexity.
- Understanding of the semantic: It is important to capture the meaning of each GUI element in order to understand their purpose. Moreover, the understanding of the relationships may also be a challenging part. Indeed, this is due to the hierarchical structure of GUIs which is represented in GUI-AGs by the *children component-inside-parent component* relationships. As opposed to real-world images where the bounding

boxes can be placed anywhere as long as it is respecting the constraints, in a GUI layout, the children components are constrained to be placed inside the associated parent component. This reduces the possibilities where the component can be placed and makes it more difficult to capture the semantic of relationships. Hence, this is a challenge that does not appear in image generation works. From the literature, it seems that synthesizing GUI-AGs as sentences is the most promising way to capture the semantic [3].

- Respect of GUI layout design principles: The GUI differentiates from real-world images by the multiple rules that are involved. One of the most important one is the alignment. Indeed, each GUI element that is part of an interface is aligned with some other GUI element. Another rule is based on the overlaps between components. In facts, in a GUI the overlap between GUI components that share the same parent must be avoided in order not to hide the widgets. However, some overlaps are necessary, as it is the case for the children component that must be placed inside the parent component. In such case, the overlap of the surface of the children component with the surface of the parent component must be the highest. Following all those rules is challenging but it makes the GUI design visually appealing. In other GUI generation works, the GUI design is directly generated which makes it challenging to quantify the quality of placement and sizing of components. In image generation there are less principles to follow and more freedom is allowed without impacting the quality of the scene layout. Therefore, applying a two-step method is a better option, because explicit loss can be formulated over the generated GUI layouts.

3.2 Overview of the proposed method

In summary, we aim to develop a new method in the novel field of research that is GUI generation from constraints that will help professional UI/UX designers in their creation process. To the best of our knowledge, there is no other work that make use of GUI-AGs or any form of graph constraints to produce GUI designs. In our proposed solution that we named GILD, we take into account the above challenges to overcome them and produce the most realistic GUI designs. This solution is built with two separate steps. At first, a transformer-based method called GUILGET is implemented to produce GUI layouts from given GUI-AGs. GUI layouts are then fed into the ControlNet, a method that finetunes the latent diffusion model while using GUI layout as input. In the following we describe and experiment the first step of our method in chapter 4 and the entire pipeline in chapter 5. Then we present additional results as well as a discussion about all the results. And finally,

we conclude the work with a summary, limitations it has, and the possible future works.

CHAPTER 4 ARTICLE 1: GUILGET – GUI LAYOUT GENERATION WITH TRANSFORMER

published at Canadian AI Conference (CANAI) 2023 on June the 5th 2023 and accepted on April the 4th 2023

Andrey Sobolevsky[†,*], Guillaume-Alexandre Bilodeau[†], Jinghui Cheng[†], Jin L.C. Guo[‡]

[†] Polytechnique Montréal

[‡] McGill University

***Contribution:** Development of the method, experimentation, analyses of results, redaction, literature review

4.1 abstract

Sketching out Graphical User Interface (GUI) layout is part of the pipeline of designing a GUI and a crucial task for the success of a software application. Arranging all components inside a GUI layout manually is a time-consuming task. In order to assist designers, we developed a method named GUILGET to automatically generate GUI layouts from positional constraints represented as GUI arrangement graphs (GUI-AGs). The goal is to support the initial step of GUI design by producing realistic and diverse GUI layouts. The existing image layout generation techniques often cannot incorporate GUI design constraints. Thus, GUILGET needs to adapt existing techniques to generate GUI layouts that obey to constraints specific to GUI designs. GUILGET is based on transformers in order to capture the semantic in relationships between elements from GUI-AG. Moreover, the model learns constraints through the minimization of losses responsible for placing each component inside its parent layout, for not letting components overlap if they are inside the same parent, and for component alignment. Our experiments, which are conducted on the CLAY dataset, reveal that our model has the best understanding of relationships from GUI-AG and has the best performances in most of evaluation metrics. Therefore, our work contributes to improved GUI layout generation by proposing a novel method that effectively accounts for the constraints on GUI elements and paves the road for a more efficient GUI design pipeline.

4.2 keywords

Graphical User Interface, GUI arrangement graphs, deep learning, transformer, generative model, GUI layout

4.3 Introduction

The design of Graphical User Interface (GUI) is an important aspect that affects the success of many software applications. The first step for the GUI designers is often sketching out the interface layout with wireframes, based on design constraints such as users' needs and software requirements [55]. These GUI layouts define the visual arrangement of elements in a user interface, such as buttons, text fields, and containers. Creating these layouts and variations of them manually, however, can be time-consuming. In this paper, we explore an automated technique that can support designers creating GUI layouts. In our approach, we capture the design constraints that the designers have to consider through *GUI arrangement graphs* (GUI-AGs) and generate graphical user interface layouts from those constraints. GUI-AGs specify elements required in the UI design and the relationships among them, as illustrated in Figure 4.1. We use GUI-AGs because they can be used to describe the requirements with an explicit definition of components that are part of the screen and the definition of the visual relations between those components. These graph models offer the flexibility to automatically create layout variations by modifying relations in the graph.

There are several technical challenges for achieving the automatic generatation of GUI layouts from GUI-AGs. One challenge is to accurately capture the logical and semantic relationships between GUI elements, such as hierarchical structures and functional dependencies. Another challenge is to generate visually appealing and functional layouts that adhere to design principles and constraints [56]. In addition, the generation process should be efficient to be used in a design workflow. To address these challenges, we propose a transformer-based approach for generating GUI layouts from GUI-AGs. Transformer networks have recently achieved state-of-the-art results in a wide range of natural language processing and computer vision tasks. They are particularly well-suited for generating GUI layouts from GUI-AGs, as they can effectively capture the dependencies between elements in the GUI-AG and generate GUI layouts that reflect these dependencies. Our transformer-based model takes as input a series of tokens that express the GUI-AG relationships; it then outputs a realistic GUI layout. We demonstrate the effectiveness of our approach through a series of experiments on real-world datasets [1, 24] using metrics specific to this task. Results indicated that our approach produces the most relevant GUI layouts in regard to specified requirements.

Our contributions can be summarized with the following:

- We propose a new transformer-based method to generate GUI layouts from GUI-AGs that takes into account the GUI design constraints. Our experiment demonstrates that it better captures the intended GUI layout compared to previously proposed methods [2, 3];
- We introduce new loss functions and metrics for quantitative measurement of the quality of generated GUI layouts; Those metrics can be used for future work on similar tasks.
- To encourage reproduction or replication of our study, the source code of our experiment is publicly available with a pre-trained model at `https://github.com/dysoxor/GUILGET`.

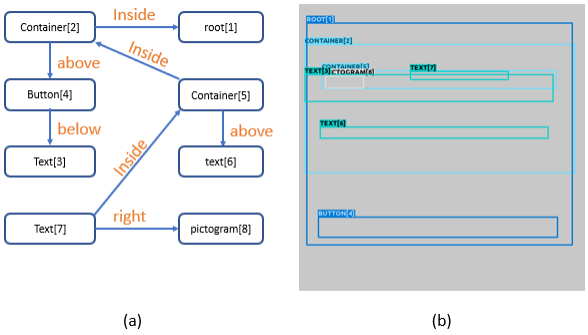


Figure 4.1 GUI layout generation goal is to generate a realistic GUI layout (b) from a given GUI-AG (a).

4.4 Method

Our work is based on the use of transformers [39] that have been shown to be state-of-the-art in multiple natural language processing tasks. A transformer uses the concept of self-attention, which allows to give different weights depending on the part of the input to make predictions. GUI-AGs can be then viewed as a natural language input to the transformer since it is a logical sequence of relationships.

Our proposed method is based on the LayoutTransformer (LT-net) [3], a transformer-based model that aims to generate diverse layouts from scene graphs of images. It consists of (1) an object/relation predictor $\mathcal{P}$ that encodes the input scene graph into contextual features f using transformer encoder layers, (2) a layout generator $\mathcal{G}$ that, from contextual features

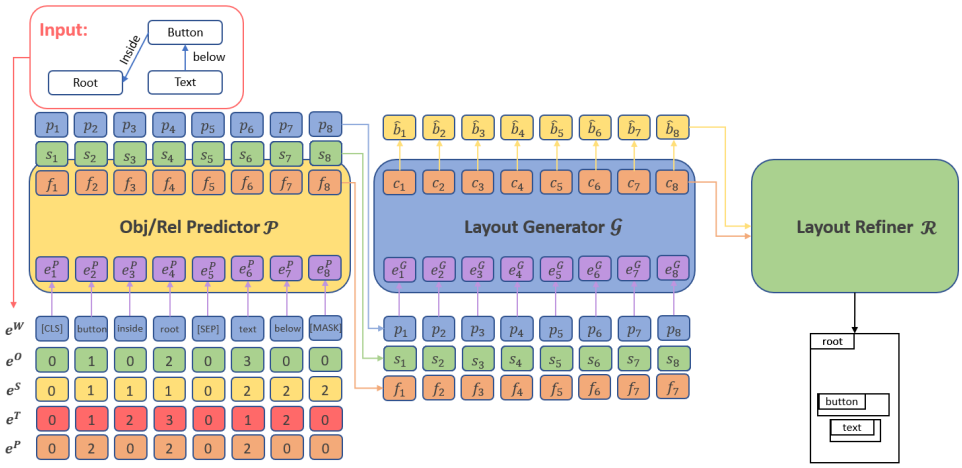


Figure 4.2 Architecture of our model with three main components. **Obj/Rel Predictor** $\mathcal{P}$ takes as input an embedding e^P which is a concatenation of several embeddings that describe different information about the given GUI-AG, and it produces contextual features f. **Layout Generator** $\mathcal{G}$ takes as input contextual features f, predicted sizes s and predicted positions p to translate it into a layout-aware representation c and bounding boxes b. **Layout Refiner** $\mathcal{R}$ uses co-attention module with predicted bounding boxes b and layout-aware representation c to improve the layout.

f, generates bounding boxes b with distributions matching a learned Gaussian distribution model and layout-aware representations c with transformer decoder layers, and (3) a layout refiner $\mathcal{R}$ made of a co-attention module. Our architecture is presented in Figure 4.2. Compared to LT-net, we introduced several improvements for GUI layout generation that will be presented in the following. It is relevant to mention that each of the module is trained from scratch without any use of pretrained model.

4.4.1 Building a GUI arrangement graph

GUIs are made of two types of components: (1) widgets (e.g. button, pictogram, text), which are leaf nodes in a GUI-AG representation and do not contain any other component, and (2) spatial layouts (e.g. container, list item, toolbar), which are intermediate nodes that allow to organize the structure of widgets [6]. This tree structure does not exist in images and it changes the way GUI-AGs should be modeled and processed compared to scene graphs for images. In contrast to our work, most of current works that are done in layout generation for GUI ignore the tree structure and only consider widgets to remove the complexity of organizing those widgets inside layouts.

GUI-AGs are directed graphs made of relationships that are triplets of *subject-predicate-object* [2]. A GUI does not require as much variety of relations and objects as images.

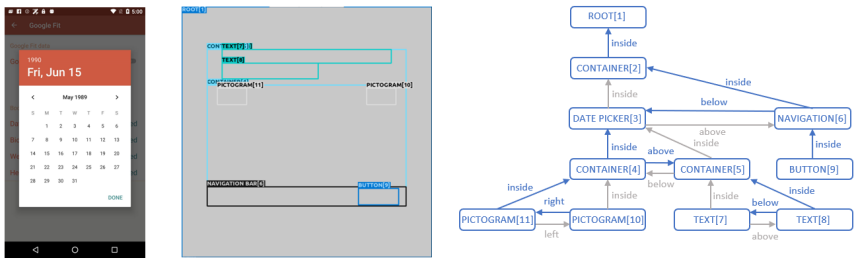


Figure 4.3 Heuristic process of modeling GUI-AG (c) based on given layout (b) and associated screenshot (a). All nodes represent components and all arrows represent possible relationships. We will keep only one *inside* relation among all children from a component (in blue) randomly, while the others are not used as input (in gray). The other relations are only possible between components that are inside the same parent and we keep only one relation between them.

However, it has complexity since GUI layouts have more rules and principles to be learned than in a layout from an image [56]. To train a neural network model, GUI-AGs from ground truth layouts are required. We define five types of possible predicate specifying the relationship among GUI components: *left, right, top, bottom* and *inside*. We build a GUI-AG from a layout by parsing a layout description. First, to reduce the size of the GUI-AG in order to have a smaller input in the transformer, we keep randomly only one *inside* relation within a group of components that are children to the same parent. This step is beneficial due to memory limitations when the input is too large. On the other hand, this might reduce the performances due to additional information the model has to infer. To illustrate this step, in Figure 4.3 we notice that *CONTAINER[2]* has two children, *DATE PICKER[3]* and *NAVIGATION[6]*, but only one *inside* relationship is kept as input to the model and the other one is removed. This processing method does not lose information since all components that do not have an *inside* relationship are considered at the same level and are implicitly inside the parent component. We also randomly choose a sequence of components inside the layout and determine relationship between each pair of components within the sequence. More precisely, as shown in Figure 4.3, *DATE PICKER[3]* and *NAVIGATION[6]* are inside the same parent; in this situation, we randomly choose the sequence in the set of children *[NAVIGATION[6], DATE PICKER[3]]* and then add the relation between each component of the sequence to form triplets (*[NAVIGATION[6], below, DATE PICKER[3]]* in this case). By doing so, we get a simplified input that captures most information from the graph, as we will see in subsection 4.4.2. It is to note however, that we may miss some relationships. For example, if there are three components (a,b,c) and we keep only two relationships (a-b, b-c), the relation between a-c can be uncertain in some cases. A GUI-AG is often larger

than 3 components and the number of possible relationships increases significant for each additional component. Finally, in GUI-AGs, the relationships are reversible; e.g., if in the layout from Figure 4.3 the *NAVIGATION[6]* is below *DATE PICKER[3]*, the reverse, *DATE PICKER[3]* is above *NAVIGATION[6]*, must also be true. Hence, we can keep only one relationship between them.

4.4.2 Object/Relation Predictor

To construct the input for our transformer model, we first convert the GUI-AG into a sequence of relationship triplets s_i. We refer to this sequence of relationships as $S = \{s_1, s_2, ...s_T\}$ where T is the number of relationships. Relationships are separated by a special token *SEP*, and a special token *CLS* is used at the beginning of the entire sequence S.

Instead of using directly the sequence S as input in the Object/Relation Predictor, we embed $s_{1:T}$ into $e^P_{1:T}$, which allows to take into consideration different features: word embedding $e^w_{1:T}$ that allows to identify the class of the object (e.g. "button" or "container") or the relation (e.g. "inside" or "right), object ID embedding $e^o_{1:T}$ to differentiate instances of the same object, relationship ID embedding $e^s_{1:T}$ to separate each relationship, type of word embedding $e^t_{1:T}$ to identify parts of a relationship (*subject = 1, predicate = 2, object = 3)* for example the part of the the relationship "button inside container" are "subject predicate object" and instead of writting the plain text, we rather associate an ID to each part of the relationship to differentiate them, and parent ID embedding $e^p_{1:T}$, which is a feature that allows for each of the component to know its parent. Those features are concatenated to form the input embedding for the object/relation predictor. It is given by

$$e^P_{1:T} = [e^w_{1:T} \bigoplus e^o_{1:T} \bigoplus e^s_{1:T} \bigoplus e^t_{1:T} \bigoplus e^p_{1:T}]. \tag{4.1}$$

The object/relation predictor learns to produce three different outputs: (1) the contextualized feature vectors $f_{1:T}$, (2) the size vectors $s_{1:T}$, and (3) the position vectors $p_{1:T}$. The contextualized feature vectors $f_{1:T}$ describe objects, relations and their context with features from the input. In order to capture conceptually diverse embedding and exploiting the co-occurrence among objects, predicates and parents, we follow the technique used in BERT [57] and randomly mask words from the input that must be predicted by the object/relation predictor. The size vectors $s_{1:T}$ are predictions of the bounding boxes size for each object made by the object/relation predictor. It is used as indicator later in the GUI layout generator to generate final bounding boxes size. The position vectors $p_{1:T}$ are predictions of bounding boxes position for each object.

Finally, in order to compute the objective function to train this part of the network, we need to predict $\hat{e}^P_{1:T}$ from the features $f_{1:T}$ using a single linear layer. Indeed, since there is no ground truth for $f_{1:T}$, we predict $\hat{e}^P_{1:T}$ from $f_{1:T}$ and match it with $e^P_{1:T}$ to minimize the reconstruction error. The objective function for training the module is composed of cross-entropy losses $\mathcal{L}_{predSem}$, given by

$$\mathcal{L}_{predSem} = CrossEntropy(e^P_t, \hat{e}^P_t), \tag{4.2}$$

for the matching word, object ID, type of word, parent ID which are all extracted from the input GUI-AG, and regression losses given by

$$\mathcal{L}_{predBox} = Regression(s_t, \hat{s}_t) + Regression(p_t, \hat{p}_t), \tag{4.3}$$

which are computed on predicted positions $\hat{p}_{1:T}$ and sizes $\hat{s}_{1:T}$ with their corresponding ground truth positions and sizes. The total loss of the predictor $\mathcal{L}_{pred}$ is a combination of the two losses and given by

$$\mathcal{L}_{pred} = \mathcal{L}_{predSem} + \mathcal{L}_{predBox}. \tag{4.4}$$

4.4.3 Layout generator

The goal of the layout generator module is to produce layout-aware representations $c_{1:T}$ and bounding boxes $\hat{b}_{1:T}$. This module is made of transformer decoder layers and interprets jointly and sequentially contextual features $f_{1:T}$, predicted bounding box sizes $s_{1:T}$ and predicted bounding box positions $p_{1:T}$ that are computed by the object/relation predictor module. The three given inputs are concatenated and expressed as $e^G_{1:T}$. After that it is translated into diverse bounding box output $\hat{b}_{1:T}$. A bounding box is described by its top-left corner position in a 2D Cartesian coordinate system and its size in terms of width and height, i.e., $\hat{b}_t = (x_t, y_t, w_t, h_t)$, and there is a bounding box produced for each subject, predicate and object. The bounding box of the predicate is the difference between the position of the object and the one of the subject, i.e., $\hat{b}_t = (x_{t+1} - x_{t-1}, y_{t+1} - y_{t-1})$.

To sequentially produce the bounding boxes $\hat{b}_t$, the features from the input e^G_t are also concatenated with the previously produced bounding box $\hat{b}_{t-1}$. This input is not directly translated into the bounding box but to a layout-aware representation c_t that is used to model a distribution in order to use *Gaussian Mixture Models (GMM)* [58]. We use this instead of directly predicting the bounding box from layout-aware representation in order to have a generative ability. Given a bounding box distribution, the bounding box $\hat{b}_t$ will be sampled from the posterior distribution $p_{\theta_t}(\hat{b}_t|c_t)$ knowing c_t. It can be described as follows:

$$p_{\theta_t}(\hat{b}_t|c_t) = \sum_{i=1}^{K} \pi_i \mathcal{N}(\hat{b}_t; \theta_{t,i}), \tag{4.5}$$

where i indicates the i-th distribution out of K multivariate normal distributions, $\theta_{t,i}$ are the parameters of each distribution defined by $(\mu^x_{t,i}, \mu^y_{t,i}, \sigma^x_{t,i}, \sigma^y_{t,i}, \rho^{xy}_{t,i})$ where μ, σ and ρ denote respectively the mean, standard deviation and the correlation coefficient, π_i is a magnitude factor, and $\mathcal{N}$ is the multivariate normal distribution.

To define the objective function of the generator $\mathcal{L}_{gen}$, we start by defining the box reconstruction loss $\mathcal{L}_{box}$, which maximizes the log-likelihood of the generated GMM to fit the training data where the ground-truth bounding boxes are denoted as $b_t = (x_t, y_t, w_t, h_t)$.

$$\mathcal{L}_{box} = -\frac{1}{K} \log(\sum_{i=1}^{K} \pi_i \mathcal{N}(b_t; \theta_{t,i})). \tag{4.6}$$

To avoid the over-fitting with this loss function, the GMM distributions are fitted to a multivariate normal distribution Q using a Kullback-Leibler (KL) divergence loss:

$$\mathcal{L}_{KL} = \sum_{i=1}^{K} D_{KL}(P_i||Q_i). \tag{4.7}$$

Finally, a relation consistency loss $\mathcal{L}_{rel}$ is also used since the two previous losses focuses only on the bounding boxes. It is given by:

$$\mathcal{L}_{rel} = \frac{1}{N} \sum (\Delta\hat{b}_t - \hat{b}^{rel}_t)^2, \tag{4.8}$$

where N denotes the number of relationships in S. It calculates the Mean Square Error (MSE) between the box disparity of the relation we get, i.e. $\hat{b}^{rel}_t$ which is the predicted bounding box for the predicate, and the corresponding box disparity we calculate from the object and the subject $\Delta\hat{b}_t = (x_{t+1} - x_{t-1}, y_{t+1} - y_{t-1})$. The layout generator is trained by minimizing the weighted sum of losses using

$$\mathcal{L}_{gen} = \lambda_{box}\mathcal{L}_{box} + \lambda_{KL}\mathcal{L}_{KL} + \lambda_{rel}\mathcal{L}_{rel}, \tag{4.9}$$

where λ_{box}, λ_{KL} and λ_{rel} are weighting factors for each corresponding loss.

4.4.4 Layout refiner

Since the bounding boxes are generated sequentially, they require refinement in the layout in order to consider the semantic $c_{1:T}$ and the bounding box $\hat{b}_{1:T}$. This is done in the layout refiner using the Visual-Textual Co-Attention (VT-CAtt) [3], which predicts the residual $\Delta\hat{b}_{1:T}$ for updating the bounding boxes. The objective function of this module $\mathcal{L}_{ref}$ is defined with multiple losses. The first one is a regression loss $\mathcal{L}_{reg}$ between predicted bounding boxes $b'_{1:T}$ by the layout refiner and the ground truth bounding boxes $b_{1:T}$. Another new and task-specific loss that we implemented is the overlap between children loss $\mathcal{L}_{CC}$, which aims to minimize the overlap of components that share the same parent in the GUI interface. This is specific to design principles in GUI since we do not want the components to overlap because it will hide some components on the final interface. This loss is given by

$$\mathcal{L}_{CC} = \frac{C_1 \cap C_2}{\min(C_1, C_2)}, \tag{4.10}$$

where C designate the area of a children. Another principle to follow is that a child must be inside its parent. To enforce this principle, we define the overlap between children and parent loss $\mathcal{L}_{CP}$ and we express it as

$$\mathcal{L}_{CP} = 1 - \frac{C \cap P}{C}, \tag{4.11}$$

where C is the area of the children and P is the area of its parent that is defined in the input GUI-AG. The objective function on which the layout refiner is trained is a weighted sum of those losses, that is

$$\mathcal{L}_{ref} = \lambda_{reg}\mathcal{L}_{reg} + \lambda_{CC}\mathcal{L}_{CC} + \lambda_{CP}\mathcal{L}_{CP}. \tag{4.12}$$

where λ_{reg}, λ_{CC} and λ_{CP} are weighting factors for each corresponding loss.

4.5 Experiments

In our experiment, we validated our proposed method by generating layouts from GUI-AGs. We aim to evaluate how close the generated layouts are to the ground truth layouts based on the graphs associated to them.

4.5.1 Dataset

We tested our method on the **CLAY dataset** [1], which is a UI design dataset. UI layouts in RICO dataset [24] are often noisy and have visual mismatches hence CLAY is a dataset that improves RICO by denoising UI layouts. It contains 59,555 human-annotated screen layouts, based on screenshots and layouts from RICO. A total of 24 component categories (e.g. Image, button, text) are available in layouts and 5 predicate categories (above, below, right, left, inside) are considered in GUI-AGs.

By observing data from the CLAY dataset, we decided to remove automatically several irrelevant GUIs and their layouts using a script. Firstly, we removed GUIs that contain two or less types of components that we know using the information of the class of each object present in the ground truth layout. Those screenshots are usually not representing an application GUI but rather GUIs with, for instance, full screen image and video screenshot that do not contain useful information for our task, as there is a lack of component and interactions (predicates) between components. Then, we also removed screens that contain only a navigation bar or popup for example, for the same reason. Also usually in the dataset there are several screenshots associated to the same application but some contain only the navigation bar and others have the navigation bar and also some content. So we removed the former to avoid overfitting. In practice, we achieved that by removing GUIs in which components cover less than 25% of the total area of the screen.

4.5.2 Evaluation metrics

To evaluate the generated layouts with respect to the ground truth, we used the following metrics.

CP Inclusion (CPI) is a metric that we introduce for capturing the overlap of children with its parent. The metric is computed as $1 - \mathcal{L}_{CP}$ (see Equation 4.11). Hence, the goal of this metric is to indicate if the generated GUI layouts tend to satisfy the UI design principle that states that children must be fully inside its parent as we see in Table 4.1 with ground truth data.

CC Separation (CCS) is another metric we introduce that aims to evaluate if the UI design principles tend to be satisfied. It is computed by $1 - \mathcal{L}_{CC}$ (see Equation 4.10), so the metric measure the ratio of components that does not overlap between each other and in the same time share a common parent.

Alignment [48] metric evaluates an important design principle that is components must be either in center alignment or in edge alignment (i.e. left-, right-, bottom- or top-aligned).

We computed alignment with the following:

$$1 - \frac{1}{N_C} \sum_{d} \sum_{i} \min_{j, i \neq j} \{\min(l(c_i^d, c_j^d), m(c_i^d, c_j^d), r(c_i^d, c_j^d), t(c_i^d, c_j^d), v(c_i^d, c_j^d), b(c_i^d, c_j^d))\}, \quad (4.13)$$

where N_C is the number of components, c_k^d is the k_{th} component of the d_{th} layout and l, m, r, t, v and b are alignment functions where the distance between the left, horizontal center, right, top, vertical center and bottom are measured, respectively. The metric as described in [48] captured only horizontal alignment while we also included the vertical alignment. This is useful for the considering some cases where the component is only aligning with other components vertically such as in a navigation bar where this is often the case.

W bbox [51] is the similarity between bounding box properties $(x_{left}, y_{top}, w, h)$ distribution of the generated GUI layouts and the ground truth GUI layouts. This is computed using Wasserstein distance and inverting it to be a similarity, between 0 and 1, by subtracting the maximum possible distance by the actual distance and normalizing the value. It is a way to measure if the generated bounding boxes are as diversified as in the ground truth data.

GUI-AG Correctness (GUI-AGC) computes the average number of correct relationships that appears in the generated GUI layout. In practice, we compare the input GUI-AG with the corresponding generated GUI layout and count the number of satisfied relationships divided by the total number of relationships. This metric measures how well the constraints given by the designer are respected. This metric does not capture the creativity and two totally different designs could respect the same GUI-AG with the same value of GUI-AGC but have completely different structures.

4.5.3 Quantitative results

Table 4.1 gives the results of our method compared to SG2IM [2] and to LayoutTransformer [3]. As we can observe, there are several metrics where our model gives the best results but not with all metrics. If we compare our model to SG2IM, we can see that in terms of *CCS* we do not get as good results. However, we notice that *CPI* and *GUI-AGC* are the worst for SG2IM. In particular, *GUI-AGC* shows that only 36.9% of the relations given as input to SG2IM are satisfied in the output, which means that the design constraints are not met with this method. Moreover, we can see with this model that the *CCS* is high while *CPI* is low, which means that SG2IM does not organize components inside the layout but uses the whole screen, as it is also shown in the results from Figure 4.5, which makes it easier to get a high

Table 4.1 Quantitative evaluation on CLAY dataset [1]. 3000 layouts are generated with each method and are compared using metrics presented in subsection 4.5.2. For SG2IM [2] as well as for LayoutTransformer (LT) [3], we only consider the parts of the architectures responsible to generate the layout.

	Metrics				
	CPI ↑	**CCS ↑**	**Alignment ↑**	**W bbox ↑**	**GUI-AGC ↑**
GT data	1.0	0.987	1.0	-	-
SG2IM [2]	0.191	**0.974**	0.997	0.81	0.369
LT [3]	0.392 ± 0.001	0.805 ± 0.002	0.998 ± 2E-5	**0.834** ± 1E-4	0.797 ± 0.001
GUILGET (ours)	**0.592** ±0.001	0.623 ± 0.002	**0.9983** ± 2E-5	0.811 ± 8E-5	**0.868** ±0.001

CCS since it does not learn the *inside* constraint. It is easy not to have overlap between child components in that case. In other words, it is not meaningful to have a high *CCS* if *CPI* and *GUI-AGC* are not also high. The LayoutTransformer model has more understanding of predicates but it is still worse than with our model. We can also see that there is a negative correlation between *CCS* and *CPI* – if the model learns to place components inside its parent, there are more possible overlaps between components inside a layout. We want both of these metrics to be similarly high to respect both of those GUI design constraints as we can observe it in the GT data, where all metrics related to GUI design constraint are close to 1. *W bbox* metric is similar for all of the models which is understandable since our model and LayoutTransformer model generate bounding boxes based on distribution of bounding boxes from the training GUI layouts. On the other hand, SG2IM is not a generative model and aims to predict bounding boxes which leads to learn the most common sizes and positions for different types of component.

Influence of UI category

In order to see if the screen category has an impact on the performance, we conducted an experiment where we compute each evaluation metric for each category separately. Figure 4.4 (a) summarizes the results. Overall, our model yields similar performances among different app categories. This is a conclusive result which shows that our model is not biased toward certain types of screen categories and is able to produce equally good GUI layout for any of the category.

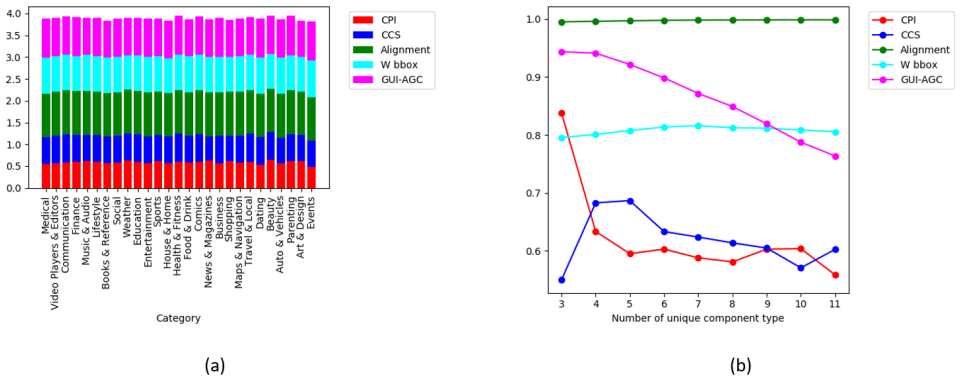


Figure 4.4 Quantitative evaluation on different screen categories (a) from CLAY dataset [1]. Evaluation metrics are applied on all 27 screen categories separately. (b) shows the influence of number of unique type components on evaluation metrics.

Influence of UI complexity

Similarly as with the previous experiment, we want to understand the influence of the UI complexity (indicated by the number of unique component types in the UI [4]) on the performance. Figure 4.4 (b) shows that with smaller number of unique component types most of the evaluation metrics are better; in other words, our model achieved better performances when the UI is less complex. Particularly, the *GUI-AGC* metric is inversely proportional to the number of unique component types in the UI. This shows clearly that the model is not consistent in capturing the semantic, it is less respecting the given constraints if it contains more complexity while for very low complexity of three the GUI-AGC is around 85%. For both *CCS* and *CPI* metrics, these results are expected due to the fact that with more diverse components that have various sizes and position standards, it becomes harder to organize all elements inside spatial layouts. There are however two metrics performing equally well over all number of unique component types, which are the *Alignment* and *W bbox* metrics. This shows that our model succeeds to always align components whether the complexity is low or high, and generated bounding boxes have almost the same similarity in distribution with the ground truth distribution.

4.5.4 Qualitative results

Figure 4.5 shows two examples produced by our model, LayoutTransformer, and SG2IM. Examples were chosen manually based on number of unique component types inside the GUI layout. We show results for low complexity (3-4 unique component types) and medium complexity (5-7 unique component types). We do not show GUI With large complexity (8

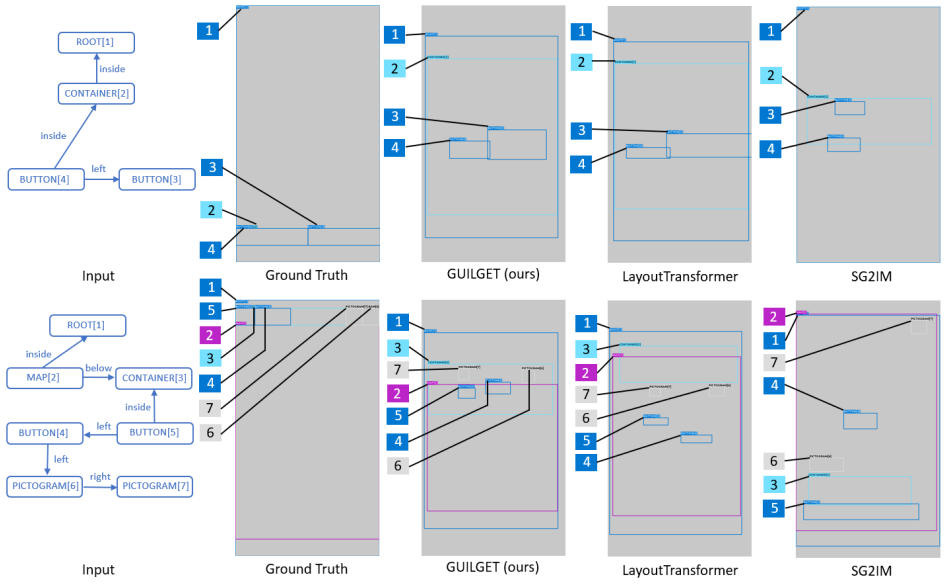


Figure 4.5 Qualitative comparison between our model, LayoutTransformer and SG2IM. The same input is given for the three models. The input from the first row has a low complexity with 3 unique component types while the second input has a larger complexity with 5 unique component types.

or more unique component types) as it is harder to analyze visually because of the larger number of components. The results of Figure 4.5 are aligned with the quantitative results from subsection 4.5.3. Indeed, we can see that SG2IM has a poor semantic understanding of relations for both cases in Figure 4.5; it also struggles to place components inside their parent as we can observe in the second row from Figure 4.5 where all components that are supposed to be inside the *CONTAINER[3]* are not and the container which is supposed to be below the *MAP[2]* is actually entirely inside it instead. We can note however that the sizes of bounding boxes and their alignments are realistic. The LayoutTransformer shows a better understanding of relations but is not able in the second case to place components inside its parent as exemplified by *BUTTON[4]* and *BUTTON[5]* that are outside the *CONTAINER[3]* in the second row from Figure 4.5. In contrast, our model respects all the given constraints and correctly placed buttons inside the container. Also, GUILGET generates plausible bounding boxes even though the generated layouts are not aligned in the way it is in the ground truth. However, information from GUI-AG is not complete enough to reproduce the same alignment. Adding global positioning constraints on components to the GUI-AG could be an interesting avenue to investigate.

4.6 Conclusion

This work proposes a transformer-based model that generates a GUI layout from a given GUI-AG. Our approach is the state-of-the-art in quantitative performance across several metrics and in visual quality. We saw that using attention provides a higher performance than using graph convolution network in capturing semantic of the GUI-AG. The new components from our model compared to LayoutTransformer bring also more understanding in GUI layout constraints. This work also introduces new loss functions and evaluation metrics specific to this task of GUI layout generation. Future work is to generate GUI from the layouts to complete the GUI design pipeline.

CHAPTER 5 GUI DESIGN SCREENSHOT GENERATION

5.1 Introduction

In the previous chapter, we presented a new method to solve the first step of the GUI design process by producing GUI layouts from constraints. Without concrete generated GUI design screenshot examples, this partially solves the problem of GUI design generation for inspiration and can not be well exploited by the professional UI/UX designers. With the purpose of assisting professional UI/UX designers, in this chapter we propose a complete pipeline called GILD that produces GUI designs from constraints in the form of a graph.

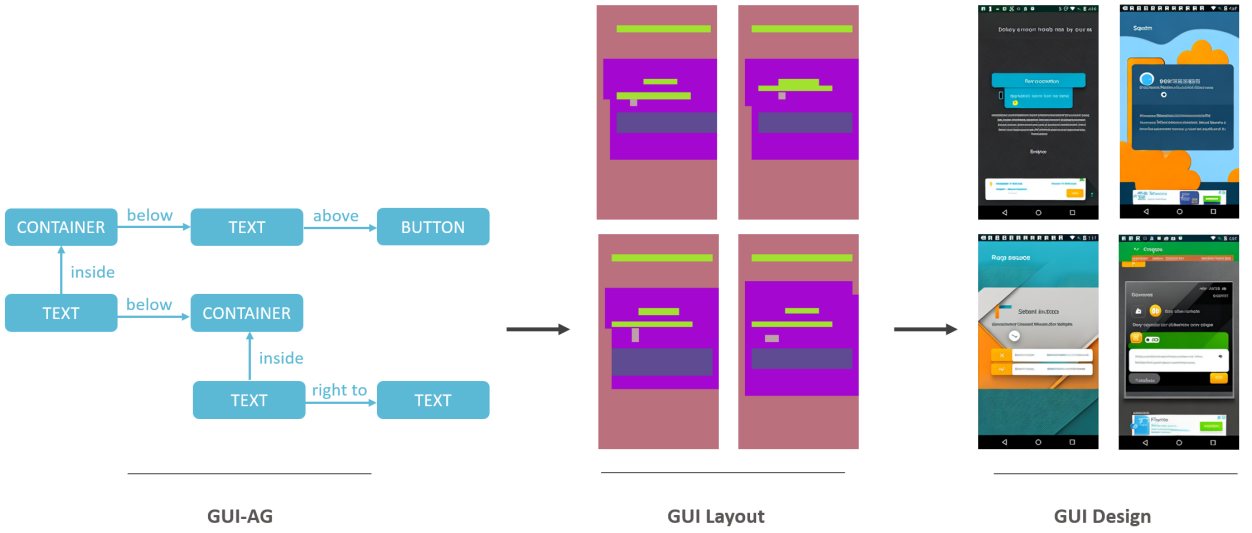


Figure 5.1 Overview of the proposed GILD method. From a graph expressing design constraints, several layouts obeying those constraints are first generated, and from the layouts, several screenshots are generated to showcase various possible styles.

We build upon GUILGET presented in chapter 4, with which we can produce GUI layouts from GUI-AGs. This transformer-based method is the initial step in GILD. The following step in the method is based on ControlNet [5] that uses latent diffusion models [43] to synthesize new images. We adapted ControlNet for GUI designs using a GUI layout as an image generation condition. The overview of those steps is illustrated in Figure 5.1, where we can draw multiple layouts with GUILGET and, for each of them, generate different screenshot ideas. The training is done separately for the two steps using the same large CLAY dataset [1], a dataset with human-annotated mobile app screen layouts.

The contribution in this chapter are:

- We propose a complete GUI generation pipeline in two steps. The first step aims to produce GUI layouts from GUI-AGs given as input. The second step generates GUI designs based on previously produced GUI layouts.
- We made improvements to the GUILGET method resulting in improved graph layouts.
- We conducted a user-study that aims to answer several research questions related to our method and also more general questions related to the input useful in a GUI inspiration tool.

5.2 Method

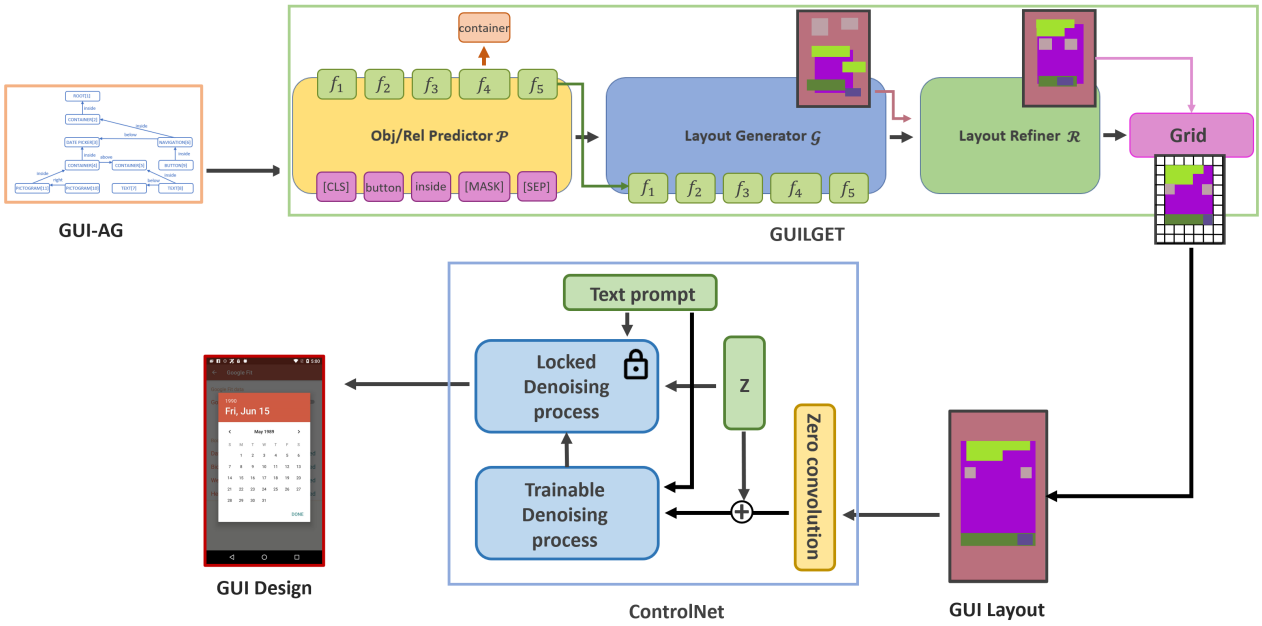


Figure 5.2 Neural network architecture of GILD. A GUI-AG is first processed by a modified version of the GUILGET method that generates a GUI layout based on the constraints expressed in the GUI-AG. Having the GUI layout, the ControlNet module will add style to each component in the layout to generate screenshots. GUILGET is able to produce multiple variations of GUI layout with the use of Gaussian Mixture Models (GMM) implemented in Layout Generator G. Then, ControlNet can generate multiple example designs by using different latent distribution z for each generation. Neural Processing modules are in blue while data vectors in green.

The detailed architecture of our proposed method is presented in Figure 5.2. Our method consists of two sub-tasks using generative deep learning methods. First, we aim to produce

realistic GUI layouts that respect not only the constraints given by the GUI designer but also the design principles to produce visually appealing and intuitive designs. This is performed using a transformer-based method which is a modified version of GUILGET presented in chapter 4. We improved it for better widget alignment. Second, with the produced GUI layouts as input, we use ControlNet [5] to generate screenshots that include concrete designs for each component in a layout. ControlNet has shown great performances in fine-tuning the large latent diffusion model [43] by converging to expected results in terms of layout and visual quality while conserving the power of the latent diffusion model trained on a large dataset. In the following, we describe each step in more details.

5.2.1 GUI layout generation

To generate layouts, we used and adapted GUILGET (chapter 4). It takes GUI-AGs as inputs and generates layouts (similar to wireframes) made of bounding boxes for each component as outputs. To obtain a more appropriate representation to be used for the next step, we fill each component with 24 different colors that correspond to the 24 classes of components that can compose a GUI. This gives a color-coded layout image that will guide the style generation process (see Figure 5.3 (a), and more details in the next section). One drawback of GUILGET is that the elements in the generated layouts are not always well-aligned (see Figure 5.3 (a)). In order to improve alignment, we defined a grid with 18×32 cells to cover the output layout, as illustrated in Figure 5.3 (b). This approach is inspired by the practices of many design systems and guidelines that make use of grid systems (e.g., [59]). Using this grid system, we snap the boxes predicted by GUILGET to the closest edge in the grid. The snapping allows to have more aligned components and helps to have the child components inside their parent components. In the experiments section, we show that this improves the performance compared to the original GUILGET.

5.2.2 GUI design generation

We used ControlNet [5] to generate styles from layouts. ControlNet leverages the latent diffusion model [43], which has shown superior performance in image generation. This generative model relies on a diffusion mechanism. This mechanism adds noise step by step during the training process to the training data until it has a random distribution. The neural network model will then aim to perform the reverse path by denoising the distribution step by step until it retrieves the original image. After the training of the model, the latent diffusion model requires only an initial noise distribution to generate a new image from it.

The latent diffusion models are most often pretrained with only a textual prompt input to

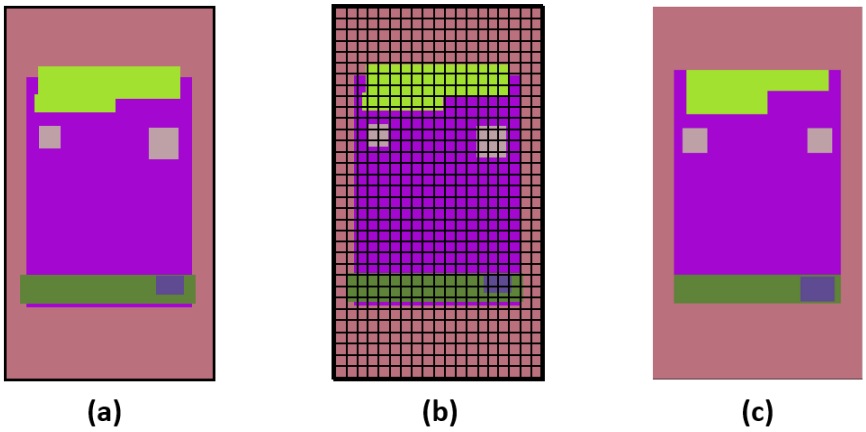


Figure 5.3 Illustration of the grid implementation used along with GUILGET. The elements of the GUI layout produced by GUILGET (a) are aligned with the grid edges (b) to obtain the final output (c).

guide the denoising process. ControlNet adds another type of input (i.e., visual prompts) to further guide the denoising process. ControlNet actually uses two denoising processes. In the *locked denoising process*, the parameters of the latent diffusion model are locked and cannot be changed. In the *trainable denoising process*, on the other hand, the model copies the same parameters that can be modified during training. Both processes work together to generate screenshots (see Figure 5.2). Such a mechanism allows a more robust model while retaining the performances of the large diffusion model. Additionally, the *zero convolution layer* is a 1×1 convolution layer with both weights and biases initialized with zeros. It supports fine-tuning the diffusion model without adding noise to the deep features.

In our pipeline, we used two copies of the latent diffusion model [43] built upon U-nets [60], and pretrained on a large dataset of images, ImageNet [61]. At each step of the inference, the diffusion model aims to denoise a random latent distribution z. In order to guide the denoising process, there are two other inputs to the model apart from z. First, we have a text prompt, which is encoded and mapped to each layer of the U-nets with a cross-attention mechanism. This text prompt comes with the pre-trained latent diffusion model and we chose to use a default text prompt (i.e., "High quality, detailed, and professional app interface") for each training data to give more importance to the visual input. We use the color-coded GUI layouts as the second input of the model (see Figure 5.2). The color code indicates to the model which pixel corresponds to what component. Like in ControlNet [5], this visual input is processed by the trainable denoising process as shown in more details in Figure 5.4, and with the help of *zero convolution layer*, the trainable denoising process is adjusting the

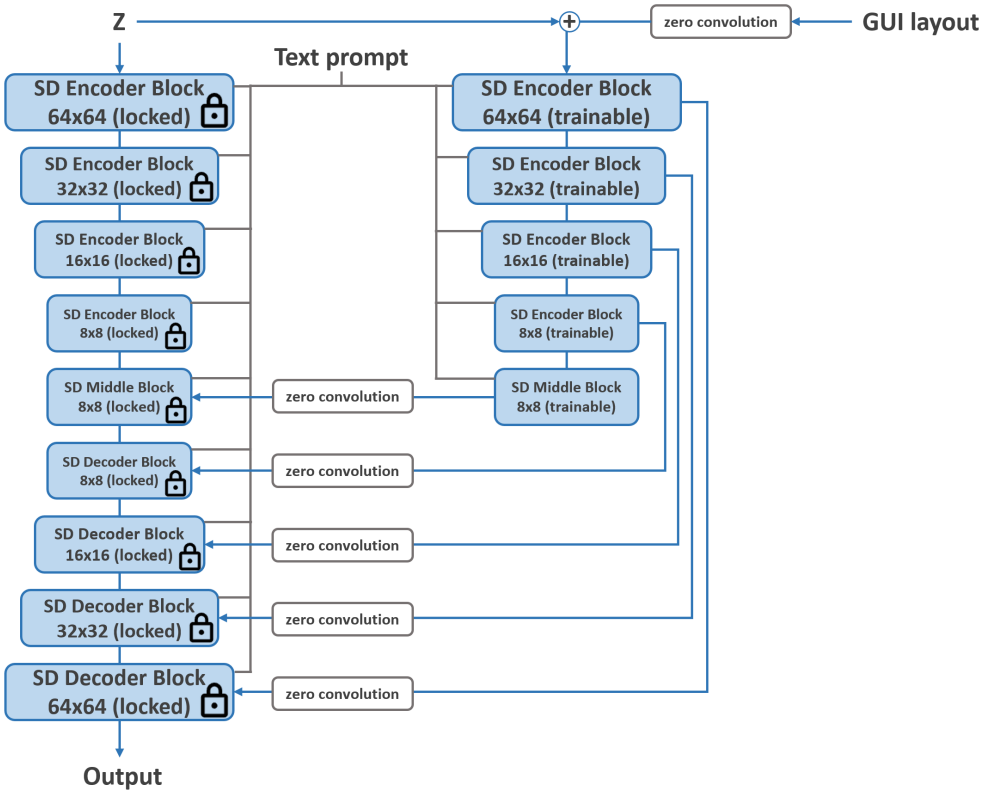


Figure 5.4 Detailed illustration of the *locked denoising process* with blocks on the left, the *trainable denoising process* with blocks on the right, and their interactions illustrated with zero convolution layers that concatenate vectors in decoder blocks of the locked copy.

outputs of layers of the locked denoising process to produce screenshots respecting the layout. More precisely, the outputs of the encoding blocks of the trainable copy are concatenated to the outputs of the decoder blocks of the locked copy after being fed to a zero convolution layer. We fine-tune the latent diffusion model on a mobile app screenshot dataset, CLAY dataset [1].

5.3 Experiments

In this section, we explain how we trained the neural network models and assess quantitatively their performances.

5.3.1 Dataset

To evaluate our proposed method, we use the CLAY dataset [1] with same preprocessing as in chapter 4. Additionally to that, a color-coded layout is generated for each screen in the

dataset for fine-tuning ControlNet.

5.3.2 Training

The training process involves two steps. First, we train GUILGET, implemented with PyTorch transformers, in a similar way as presented in chapter 4. The color coding and grid mechanism are applied after inference. Then, we used a pre-trained ControlNet model [5] and fine-tuned it using the color-coded layouts extracted from the CLAY dataset, as described above, that represent GUI layouts to generate the associated GUI design screenshots. We fine-tuned the model for 90,000 iterations, at which point we found that more training would lead to overfitting. The overfitting is a state where the method becomes worse in generalizing and is biased toward data it was trained on. In our case, it can be expressed through GUI's that have very similar colors for instance and the model is less likely to create new color palettes.

5.3.3 Evaluation of the Generated GUI layouts

In Table 5.1 we aim to evaluate the performances of the GUILGET method implemented along with the grid. For quantitative evaluation of the GUI layout generation methods, we consider the same metrics as in the GUILGET method we presented previously in chapter 4, which are all bounded between 0, being the worst value, and 1, being the best value.

Table 5.1 presents the results of our improved GUILGET method (GUILGET+grid) as compared to other methods. The grid significantly improves the performance of GUILGET in multiple metrics. The CPI particularly shows the greatest improvement and the value is much closer to ground-truth layouts (i.e., layouts directly extracted from the CLAY dataset). In GUILGET (chapter 4) work, it was shown that the increase in the *CPI* compared to other works leads to a decrease in the *CCS*. This is due to the higher complexity of arranging child components within their parent component without overlapping the child components. In contrast, GUILGET+grid increases the *CCS* as well as the *CPI*. The grid allows to resize children components. With that, borders of children components within a parent component are aligning on the same cell edge of the grid instead of having overlapped children components. We can notice that the *alignment* is high for all of the methods. However, our GUILGET+grid, in which components are snapped to grid cells, led to the best alignment. This shows that alignment itself is not sufficient to evaluate a layout. This good alignment should also lead to a good CPI and good CCS to correspond to a realistic GUI layout. Our GUILGET+grid also achieved a better *w bbox* compared to the original GUILGET, indicating that the positioning and sizing of components are more realistic with GUILGET+grid.

Table 5.1 Quantitative evaluation of GUI layouts on CLAY dataset [1]. 3000 layouts are generated with each method and are compared using similar metrics as presented in GUILGET chapter 4. For SG2IM [2] as well as for LayoutTransformer [3], we only consider the parts of the architectures responsible for generating the layout and the results are taken from the experiments part of GUILGET. GT data corresponds to ground-truth GUI layouts collection. Best results are in **boldface**.

	CPI ↑	**CCS ↑**	**Alignment ↑**	**W bbox ↑**	**GUI-AGC ↑**
GT data	1.0	0.987	1.0	-	-
SG2IM [2]	0.191	**0.974**	0.997	0.81	0.369
LT [3]	0.392 ± 1E-3	0.805 ± 2E-3	0.998 ± 2E-5	0.834 ± 1E-4	0.797 ± 1E-3
GUILGET	0.592 ± 1E-3	0.623 ± 2E-3	0.9983 ± 2E-5	0.811 ± 8E-5	**0.868** ± 1E-3
GUILGET+grid	**0.816** ± 4E-3	0.841 ± 1E-3	**0.9995** ± 2E-5	**0.869** ± 1E-8	0.651 ± 2E-5

Finally, we noted one drawback of GUILGET+grid. The relationships are less accurate, as shown by the *GUI-AGC* metric, compared to the previous work but still respect most of the constraints.

5.3.4 Evaluation of the Generated GUI designs

Evaluating the quality of generated images is not an easy task. One way to quantify the generated image quality is to quantify the similarity between the real data distribution and the generated sample distribution. If the distribution of generated images resemble the distribution of actual data, it means that the generated image are very likely to be of high quality and realistic. This can be computed with the Fréchet Inception Distance (FID) [30]. The FID relies on the Inception Score (IS), comparing the statistics of negative and positive samples using the Fréchet distance between two multivariate Gaussians:

$$\text{FID} = \|\mu_{\text{real}} - \mu_{\text{fake}}\|^2 + \text{Tr}(\Sigma_{\text{real}} + \Sigma_{\text{fake}} - 2(\Sigma_{\text{real}} \cdot \Sigma_{\text{fake}})^{0.5}), \tag{5.1}$$

where μ is the mean value of the feature representation, while Σ is the covariance. Tr refers to the trace of the matrix. The feature representation is a 2048-dimensional activation of the Inception-v3 [53] pretrained on ImageNet [61].

Although the model was pretrained on a dataset from a different domain of images than CLAY dataset, ImageNet is large and diverse. The learned features by this model can capture many aspects of visual content, making the model flexible for different image domain. A lower FID shows that the two feature distributions are closer between two collections of GUI designs.

This is the consequence of a higher quality and diversity in the generated images. In our experiments, we use the same number of real and generated images to compute the FID.

Another aspect, on which generated GUI designs must be evaluated, is the diversity of the generated collection of designs. Indeed, the FID assesses the quality of generated designs in regards to the real-world collection of app screenshots. However, it does not fully give an idea of the diversity in style among the generated designs. For this reason, we use the *Diversity Score* [3]. To compute this metric, we measure first the distance between each pair of the generated images using *Learned Perceptual Image Patch Similarity* (LPIPS) [41]. We used the pretrained LPIPS model that is built upon VGG [54]. Having all the distances, the *Diversity Score* is computed as the mean value across all those distances. Values of this metric typically range from 0 to 1, where a higher diversity score indicates a greater variation, while a lower score means a higher similarity among designs.

For evaluating the GUI design images, we compare our method with GANSpiration [4], which is at this time the only proposed work for pixel-based GUI design generation for inspiration purposes. Moreover, in order to evaluate the effects of the first step of the pipeline, we compared GUI designs generated from our complete pipeline GILD and GUI designs generated from real-world GUI layouts (GT Layout+ControlNet).

The FID score compares the distribution of features in generated GUIs with the distribution of features of ground-truth GUIs. Hence, a smaller FID means that the generated data is more realistic in regard the the ground-truth data. On the other hand, the diversity score evaluate the diversity of feature distributions inside a collection of generated GUIs. As shown in Table 5.2, our method GILD achieved the lowest *FID* score and the best *Diversity Score*. ControlNet played a considerable role in reducing the FID score drastically from GANSpiration. The difference in the FID values between GT Layout+ControlNet and GILD are similar, indicating that screenshots generated with GUILGET are of similar quality as screenshots generated from ground-truth GUI layouts.

In terms of visual quality, Figure 5.5[1] shows a clear difference between GANSpiration and methods that make use of ControlNet and layouts. This result is aligned with the FID scores presented previously. With GANSpiration we can often observe visual distortions and unrealistic GUI designs. However, the ControlNet-based variants, which incorporate clear layout information, present a variety of different designs where the components can be clearly distinguished. Interestingly, from our manual inspection of the generated images, we found that while the designs are diverse with the GT Layout+ControlNet method, the structure remains the same across all generated designs, but it is not the case for GANSpiration and

[1]More examples can be found in this repository: `https://github.com/dysoxor/design_examples`

Table 5.2 Quantitative evaluation of GUI designs on CLAY dataset [1] using the FID and diversity score as metrics. We compare GANSpiration [4], GT Layout+ControlNet that only considers the GUI layout to GUI design step, and GILD, which is our complete pipeline generating GUI designs from GUI-AGs. With GILD, we produce 4 different layouts and multiple screenshots for each of those. The same ground-truth GUI design collection was used for computing FID for each method. For computing metrics, 400 ground-truth screenshots and 400 generated screenshots were used for each method. We used the pretrained version of GANSpiration [4]. GUILGET and ControlNet [5] were trained by ourselves. The number of generated designs is the same for all methods. Best results are in **boldface**.

	FID ↓	**Diversity score ↑**
GANSpiration [4]	146.345	0.654
GT Layout+ControlNet	59.583	0.68
GILD (ours)	**52.872**	**0.71**

the GILD pipeline (see Figure 5.5). Particularly, GANSpiration and our pipeline also change the structure of the GUI design, in addition to modifying the visual representation. In fact, GILD can generate several layouts from the same GUI-AG resulting in higher diversity as shown in Figure 5.5. Despite the good visual quality, our method is not able to reproduce the type of screen we aim to reproduce. This is due to the fact that we do not communicate the information about the category of the app we want to produce.

5.4 User study

***Contribution**: Another student, Atefeh Shokrizadeh, has contributed to this section for the recruitment, writing of questions and conducting of the interviews. My personal contribution was to write some of the questions to be asked during the interview, prepare the results to be shown to participants, and transcribe the review of participants in the results of this section.*

In order to understand the capacity of our pipeline to support the GUI design ideation process, we conducted a user study with five professional UI/UX designers. The user study was approved by the ethics review board of all involved institutions. In the following, we first describe how the user study was conducted, and then we present the results of the user study.

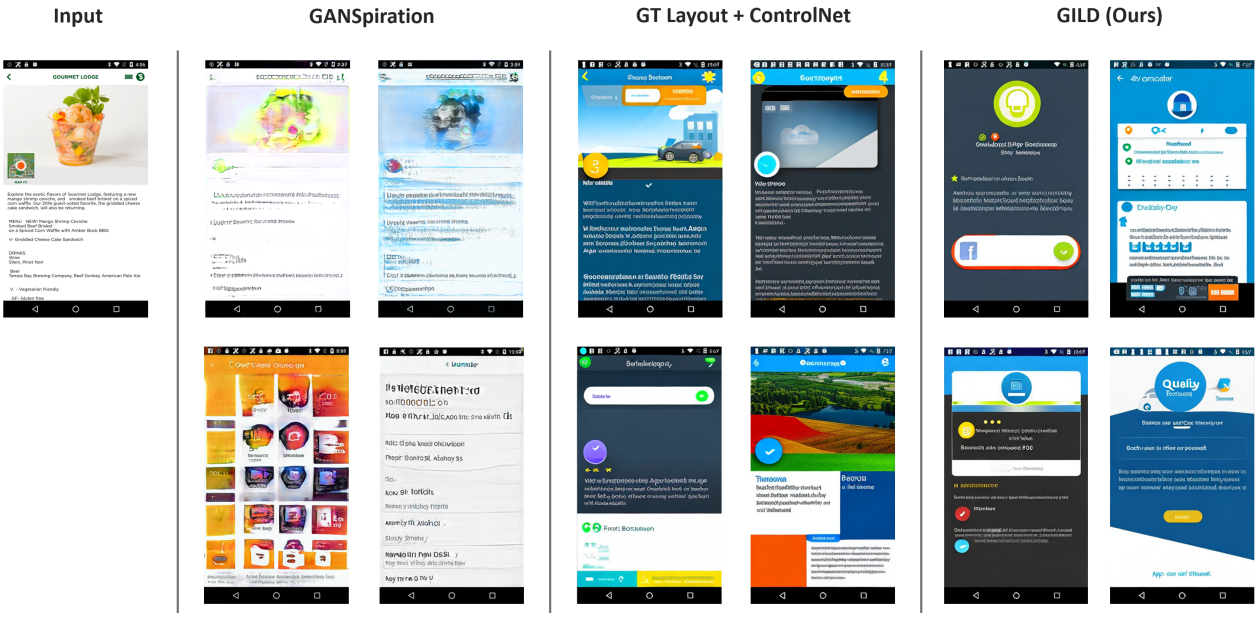


Figure 5.5 Qualitative comparison between GANSpiration, ControlNet, and our proposed pipeline. For GANSpiration, the image input is used to produce alternative designs. ControlNet uses the ground-truth layout associated with the input image to produce alternative designs. Finally, GILD takes the GUI-AG that can be built from the image to provide the same positional constraints. For GILD, 4 different layouts were produced from the same GUI-AG and we generated 1 design for each.

5.4.1 Methods

The user study was conducted with five participants who were all professional UI/UX designers. Each participant had between 1 and 8 years of professional experience in GUI design and had different levels of experience and backgrounds. Table 5.3 summarizes the level of experience of each participant. The diverse levels of experience of our participants allowed us to have an evaluation of our designs and method from different points of view and from designers used to different workflows in the ideation process.

The user studies were conducted on the Zoom platform and were recorded with the consent of the participants. Before entering into the evaluation of the designs, we first asked participants about their experiences in UI/UX design and their usual workflow. Then the participants were asked to reflect on two design ideation scenarios and examine the associated design examples. The entire scenario script and content presented to participants are described in Appendix A.

First, we introduced a scenario in which participants were asked to design a news app GUI based on a predefined layout, as shown in Figure 5.6(b). This news app must contain a

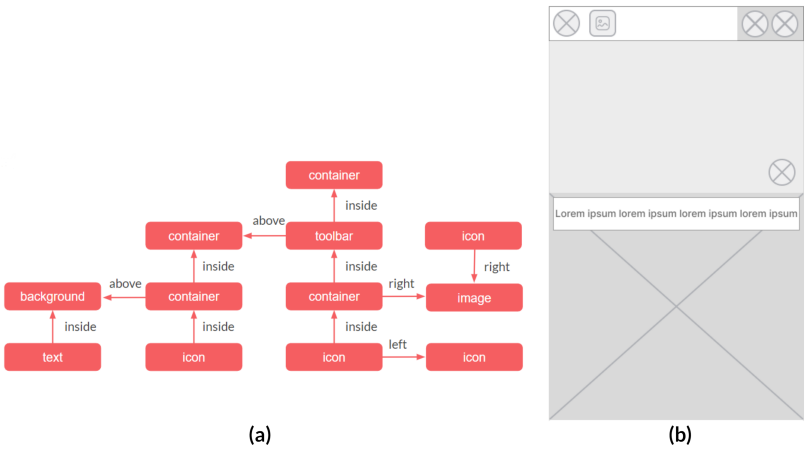


Figure 5.6 Inputs representing the constraints of a News app in the form of (a) GUI-AG and (b) GUI layout.

navigation bar on the top, a video or an image below with a caption attached to it, and a background component that contains a written description of the news. This layout was extracted from the ground truth dataset. A set of 27 screenshots were presented to the participants and can be found in the Appendix A. The screenshots were generated with the GT Layout+ControlNet method using the layout. Participants were asked to reflect on the scenario, describing what they liked and disliked about the collection and how this method can be incorporated into their ideation process.

Then, another scenario was presented where a GUI-AG, shown in Figure 5.6(a), describes the same news app structure. We first asked the participants about their perceptions of the use of a graph in the ideation workflow. Then, a set of 26 designs were presented that are generated with our GILD pipeline using four different GUI layouts obtained from the GUI-AG. Those screenshot examples are given in the Appendix A. Then, the same questions were asked as in the first scenario.

Finally, the participants are asked about the integration of this type of tool in their ideation process, reflecting on the ideal type of input. At the end of the study, participants have the opportunity to ask questions and make comments about anything related to the work. We transcribed the user studies verbatim and conducted a thematic analysis [62] to extract common themes in participants' answers.

5.4.2 Results

All of the participants mentioned that they used online tools for seeking inspiration (e.g., google.com, Pinterest.com, Behance.net, and dribble.com). Their input in those platforms

Table 5.3 Information about the participants in the user study.

ID	Years of experience	Projects contributed to
P1	8 years	20-30 projects
P2	5 years	15 projects
P3	2 years	6 projects
P4	2 years	3-5 projects
P5	1 year	3-4 projects

was usually a set of keywords that correspond to the page, style, or component they are looking to be inspired for. To illustrate this ideation process from online tools, P3 mentioned "*If I'm looking for some ideas for a dashboard design, I simply just look and search for the keyword... For example, if I'm designing a poster for an event, I just search for poster ideas with some additional keywords. And these additional keywords would be for example the style that I have in mind. For example futuristic poster design ideas or vintage poster design ideas or minimalist dashboard design ideas.*" Another participant, P2, had a different approach than using standard tools for getting ideas: "*Actually I'm paying attention to the needs of the project – it means why we are working, and what we want to design for this project. And after that, I gather information about the needs, I research about the competitor.*"

According to participants, the use of those tools presented some challenges. For example, P4 faced the following issue: "*Sometimes I'm looking for inspiration for some application or websites that I cannot find on dribble.com.*" P3 also commented on a lack of innovation in existing designs, elaborating: "*Because as designers, we see all kinds of designs every day, and personally I always try at least to come up with a new way to design.*" When designers do not use tools for inspiration, there were often difficulties in finding ideas. For instance, P5 describes the following problem: "*I have problems with finding proper colors. I tried to come up with an idea for the color palette, but at the end of my design, I usually find it's not interesting.*"

Evaluation of generated designs from a predefined layout

All participants agreed that designs directly generated with a predefined layout were not engaging and not effective for inspiration. Several justifications are given to this by different participants. The major problem that every participant except one noticed was the **overload of information in the designs**. For this, P4 said: "*There are too many elements on each page which makes them look cluttered.*" Three out of the five participants also mentioned

that the **generated components are not realistic**. For example, P1 described: "*It is like these are some random shapes in a page, maybe we cannot even call them 'components', because a component has a definition in each design system.*" Participants also considered the design examples **simple and repetitive**. For example, P5 stated: "*You can see that it's all about simple shapes like rectangles, circles, and no shadowing ... and simple typography.*"

Despite the generally negative view of those design examples, some positive points were identified in some particular screenshots. Specifically, some design examples may include **interesting elements** for inspiration, and in general, the generated GUIs respected the layout. For example, P1 stated: "*This screenshot is inspiring because it uses dividers correctly and I can get the idea of showing related news at the bottom of the page - using little images in each item of the list is very good.*" Additionally, although the structure of the generated design examples remains the same, respecting the input layout, the participants considered **the styles in some design examples** to have inspirational values. For example, P3 said: "*I think [the designs] are not similar ... I can see different types of styles here, different approaches.*"

Evaluation of generated designs from GUI-AGs using GILD

All participants found this set of designs **more visually appealing**. For example, P1 said: "*The designs are more neat than in the other set.*" P4 also described those designs as follows: "*They are more appealing and pleasurable to look at.*" Additionally, compared to the previous set, designs generated through the GILD pipeline were **more diverse**, leading to more inspiration power. For example, P5 considered these design examples as inspiring and useful: "*I saw, for example, you can see many different types of navigation bars, which is useful for an application. ... And also the colors are a lot better.*" Participants said that they are **more likely to use these designs for inspiration** than the previous ones. P3 stated this straightforwardly: "*I could get some ideas from these examples, at least I can see that. But from the previous one it had no chance.*"

Regarding points for improvement, four participants found it **challenging to identify designs related to the application domain**, i.e., a news app. For instance, P4 said: "*They are not the news pages we are looking for... Maybe some parts of some of them might help me.*" Because of the loose specifications of the layout with GUI-AG and the structural change performed by GUILGET, some participants **cannot track the components specified in the GUI-AG**. About this, P3 said: "*I can't find the place of the elements that I wanted for this design.*" Participants also voiced **concerns on the visual design of certain examples**. For example, P1 described: "*The contrast rate of the color palettes used in some of*

them, like image [X], violates the WCAG [i.e. Web Content Accessibility Guidelines]."

Evaluation of input types

We have presented two different types of input for inspiration under constraint, i.e., the GUI layout and the GUI-AG, each presenting advantages and disadvantages. In general, participants found **layouts as a more familiar and direct way** to express their ideas. P3 justified this as follows: "*I think layout would be so much easier to use as an input ... because I think it is both easier and more usable because I can put everything exactly the way I want.*" As said by P1, the GUI-AGs are not used by designers yet and could be difficult to master even though it could improve the workflow: "*Using the graph is not easy and needs training. I don't know where is the starting and ending point of the graph. If we can have both the graph and the wireframe it could be better.*" Additionally, compared to the current representation of GUI-AGs used in this study, some participants **preferred layouts because of their visual presentation**. For example, P4 said: "*I'm a visual person... So the texts [used in the GUI-AG] don't work that much for me.*"

The GUI-AG is, according to four participants, a representation that could be beneficial if integrated into the workflow. Participants considered the value of **GUI-AGs as an overview of the constraints**. According to P3, using the GUI-AG "*is like drawing a map of what you're going to do, and I think it gives you structure.*" P5 shared similar thoughts and added that **GUI-AGs can serve as a reminder of the constraints**: "*I have to refine the design. And every time I forgot some type of stuff and I forgot if I wanted to put the icon on the right side or on the left side, for example.*" Participants also discussed how **GUI-AGs can help accelerating design**. For example, P2 considered the GUI-AG: "*beneficial because in the design of an app, we need to create some repetitive elements in different pages, like the navigation bar. So by creating a graph and just modifying a few components we can have different pages.*" Participant P1 indicated: "*This could be useful if you are designing components. ... I can say this could be beneficial for creating a design system.*"

5.5 Discussion

Our study results have shown that GILD produced examples that are diverse, respect GUI layout constraints, and can support design inspiration. Insights from our results allowed us to provide several implications for tools that support GUI design under constraint, potentially using GILD. We discuss these implications in this section.

Presenting layout constraints in a way familiar to designers. In GILD, the layout

constraints are represented in the GUI-AG structure. Although the GUI-AG can be visualized in a graph, such a visualization is not intuitive for designers, as indicated by our participants during the user study. Participants were reluctant to directly use the GUI-AG structure, worrying about its learnability, even though they saw the benefit of generating examples from GUI-AG constraints. Thus, it would be beneficial to incorporate the GUI-AG structure in a type of artifact that is already familiar to designers. For example, constraints embedded in GUI-AG can be represented as an overlay on top of a visual GUI layout, represented as, e.g., a wireframe. This will make it easier for the designers to understand and create those constraints.

Empowering designers to edit the layout constraints. Although not directly explored in our study, our participants mentioned a desire to be able to edit the layout constraints embedded in the GUI-AG. After all, the same as the design process itself, ideation is an iterative process. Allowing designers to expand their ideas by adding, removing, or changing constraints during the ideation process would be not only practical but also beneficial for promoting design creativity. Related to the previous point, the editing process needs to be designed in a way that is familiar to designers. Ideally, designers would be able to perform the editing on a type of artifact that they frequently use, such as wireframes.

Supporting traceability between constraints and the examples. In GILD, the GUI-AG only loosely defined relationship constraints of GUI components, leaving a lot of flexibility for generating concrete design examples. While this approach boosts the diversity of the generated examples, some of our participants found it difficult to make sense of some examples in relation to the layout constraints. This indicated the usefulness of providing traceability between the constraints and each generated example, allowing the designers to understand why and how well the design follows the constraints. Technically, in GILD, this can be done through the intermediate layout generated with GUILGET. From the tool design perspective, however, the presentation and user interaction of the traceability links need to be explored.

Combining layout constraints with other types of design constraints. Our GILD technique only considers constraints related to GUI layouts. While we intentionally focused on this smaller scope to demonstrate the potential of design example generation under constraint, we acknowledge that designers face many other types of design constraints in practice. For example, in our user study, participants voiced the need to see more examples from the same application domain. Some participants also preferred a certain visual design, such as a particular color pallet or shadowing effect. Moreover, a few participants discussed the irrelevancy of certain generated examples when a design system [63] is used to uniform the design style in a company. Thus, these other types of design constraints need also to be

incorporated to create a practical design inspiration tool.

5.6 Conclusion

We proposed the first constraints-based pipeline for GUI design examples generation, named GILD, aimed to support design inspiration. More precisely, GILD first produces GUI layouts from GUI-AGs using a modified version of GUILGET, a transformer-based method integrating new techniques inspired by the design practice. From the layouts, GILD then generates GUI designs with the help of ControlNet, a method based on the Latent Diffusion Model pretrained on a large dataset of images and fine-tuned on a GUI design dataset. The quantitative evaluation studies showed that the generated designs are both realistic and diverse. A user study with professional UI/UX designers also confirmed these results from the user's perspective and indicated the potential of our approach in supporting design inspiration in practice. Our results further provided implications for the design of practical tools for addressing the intricate problem of design inspiration under constraints.

CHAPTER 6 GENERAL DISCUSSION

In this chapter, the discussion is developed about the results from chapter 4, chapter 5 and additional results are presented in order to make a complete discussion about the entire work.

6.1 GUI layout generation

There are not many works proposed in the domain of layout generation from a scene graph. This is mainly due to a high complexity of processing the graph and capturing all local and global contexts of each component. We have trained in similar conditions the available methods for comparison and gathered the results in Table 5.1. Methods with GCN were the first being explored, specifically SG2IM [2]. The results showed that even though there are some design principles respected in the generated GUI designs, such as the alignment and separation of children (CCS), the *inside* relationships are not respected at all and it leads to a wrong hierarchical structure in the output. This is shown by the low CPI and GUI-AGC. This is improved with existing transformer-based method, like the LayoutTransformer [3]. However it is possible to have better performances if the GUI design principles are taken into account more explicitly in a method. Being aware of GUI design principles, hence allowed to improve the results. To the best of our knowledge, we are proposing the state-of-the-art method in the field of GUI layout generation from GUI-AG with GUILGET.

Since the publication of GUILGET, several other works have been published proposing methods with a similar application. The method LayoutFormer++ [64] also employs a transformer to process a serialized constraint. They emphasis on the fact that serialization allows to be flexible on the constraint for layout generation. It allows to meet the user needs that are diverse, including the specification or not of element types, attributes, relations, or coordinates. An encoder-decoder framework with a transformer is used to perform a sequence-to-sequence task. The encoder processes the serialized constraints in a biderectional way like in BLT [65]. The decoder predicts the layout sequence autoregressively, where multiple decoding steps are composing the decoder and the model samples one token from the predicted distribution at each decoding step. A new mechanism is introduced called decoding space restriction that does not allow to sample a portion of the distribution in the case where this portion would lead to an unfollowed constraint. The method is not compared to GUILGET and it does not use the same metrics, therefore, it is difficult to compare the two methods. Nevertheless, LayoutFormer++ was evaluated on different tasks, including the layout generation from relationships like in GUILGET, with a large variety of other methods and showed good

resultsoverall. Another recent work presents LayoutDM [66], a method based on diffusion model. In image generation, the diffusion model has shown it power to produce realistic and diverse images using a denoising process made of convolutional neural networks. However, this is not suitable for layout denoising since layout is a non-sequential data structure consisting of varying length samples with discrete (classes) and continuous (coordinates) elements simultaneously, instead of pixels laid on a regular lattice. LayoutDM propose to use transformers instead of convolutional neural networks to address this issue and capture the high level relationship information between elements. Moreover, in order to include the conditions, a conditional layout denoiser is introduced and its specificity is that it omits the positional encoding since there is no sequence in the layout. These conditions are in the form of a set of elements and their attributes that must appear in the layout. Hence, the method is not inferring layouts from relationships like ours. The method was compared to methods employing GANs and VAEs and showed the best quantitative results overall and the qualitative evaluation allowed to conclude that diffusion models lead to good diversity and good realism in layout generation.

It is worth mentioning that some other metrics were considered. In fact, the alignment metric from [48] was initially proposed for measuring the horizontal alignment only. However, if we observe the real-world designs, it happens that the alignment is more important at the vertical than at the horizontal. Indeed, often the component from a horizontal navigation bar with all menu buttons are aligned between them and are placed on the same vertical line. This is the reason why the alignment metric was extended to a larger scope by considering both orientations for alignment. Also, the DocSim similarity metric [49] was not used due to it complex implementation and the similar information it aims to quantify as in Wasserstein similarity, which is the similarity of produced GUI layouts in comparison to ground-truth GUI layouts in terms of placement and size of components.

As we saw with the construction of GUI-AGs in chapter 4, all the possible relationships are not kept. We remove redundant ones, because of the limitation of computer resources. This led to experimentation with data augmentation since each GUI-AG has multiple other different GUI-AGs that are logically equivalents. For each screen from the training data collection, four GUI-AGs are produced with a random selection of which relationships are removed. Against expectations, this led to worse results overall as illustrated in Table 6.1. One hypothesis is the fact that the method is overfitting on some patterns that appears much more often with an augmentation. Hence, the method will see a greater proportion of navigation bar examples and will be less able to generalize and follow correctly the constraints.

We are building relationships in GUI-AGs in a heuristic way. Some prior experiments were

Table 6.1 Comparison of GUILGET methods trained in the same condition but one is trained on augmented dataset and the other without the augmentation.

	Metrics				
	CPI ↑	**CCS ↑**	**Alignment ↑**	**W bbox ↑**	**GUI-AGC ↑**
GUILGET with augmentation	0.356	**0.835**	**0.998**	0.809	0.698
GUILGET w/o augmentation	**0.592**	0.623	**0.998**	**0.811**	**0.868**

conducted to determine the best way to define relationships. First, the most trivial way was comparing the center position of each component. In fact, after checking if the component is inside another component, to determine if it is *right to*, *left to*, *above*, *below* another component, the centers of both component are compared in terms of distances. However, another way of determining relationships is also possible that is based on the boundary edges of components. In this scenario, the distances between boundary edges of components are considered to determine the relationships. This method appeared to be the most reliable of the two for defining relationships. Indeed, the various sizes of components can lead to an inconsistent definition of relationships in the case where the centers are used. On the contrary, if the relationship is based on boundary edges, the size of the components is not impacting the computation. Surprisingly, as shown in Table 6.2, the use of centers to determine the relationships gave better results at the end. Not having directly the coordinates of the borders might complexify the interpretation of the relationships based on edges since it requires to consider the size and coordinates simultaneously. The results with relationships based on centers give better results due to the fact that it is directly based on the (x, y) coordinate that the network predicts.

Table 6.2 Comparison of GUILGET methods trained in the same condition but one is trained on dataset where relationships are defined with edges positions and the other trained on dataset where relationships are defined with centers positions.

	Metrics				
	CPI ↑	**CCS ↑**	**Alignment ↑**	**W bbox ↑**	**GUI-AGC ↑**
Relationships based on edges	**0.709**	0.491	**0.998**	0.801	0.863
Relationships based on centers	0.592	**0.623**	**0.998**	**0.811**	**0.868**

6.2 GUI design generation

In the early exploration of a possible solution, the usage of GUI arrangement graphs (GUI-AGs) as input came naturally. This type of constraint allows to structure the desired app without the need to create the exact structure neither the style of the interface. From graphs, SG2IM [2] was one of the most well-known work in the field of image generation. In fact, it introduced the scene graph constraints. However, the major problem of this method is that generating image of sizes of more than 64×64 was technically not possible with our resources and by exploring the results obtained by the authors, the visual quality was not detailed enough. We abandoned this solution. We remarked that ControlNet was able to produce images with more fine details and the overall images were more realistic.

Two metrics are used to provide a quantitative value about the performances of the GILD method. We were concerned about the fact that for the FID and Diversity Score, models pretrained on ImageNet were used. Indeed, this is a dataset made of images without screenshots. To understand if these metrics are valid for evaluating GUI design, experiments were conducted. Starting with the diversity score, it can be easily shown that the metric is reliable or not with the following. By taking a set of very different GUI designs and a set of very similar designs, we can compute the metric for the two sets. We used screenshots from GANSpiration since we can easily observe differences in structure and in style. If we take for example the screenshots of Figure 6.1 and Figure 6.2, we can conclude that the metric shows a clear gap between similar or different images. The reason it is working even though the model was pretrained on a different dataset, is because the metric uses the hidden features from the model that are trained to capture the fine details, such as texture, colours, etc. Similarly for the FID, by building two sets that are different allows to assess the difference in quality between the two sets using the FID. Figure 6.1 and Figure 6.2 are showing two different FIDs and this shows that a set of screenshots that are visually similar to the reference set, have a higher quality and similar feature distribution in regard to the reference set. Since the metric provide the same interpretation as a human, we can conclude that the metric is reliable enough. Therefore, the use of a model pretrained on different dataset gives reasonable and useful results.

In quantitative evaluation, our method showed similar results as in the qualitative user study. The FID was the highest with our pipeline and indicates that we have the most relevant design examples in comparison to GANSpiration [4] and GT Layout + ControlNet. Moreover, the diversity is also the highest, which can be explained by the fact that each screenshot has it own style, and the structure is varying since GUILGET is a generative method that can produce different layouts.

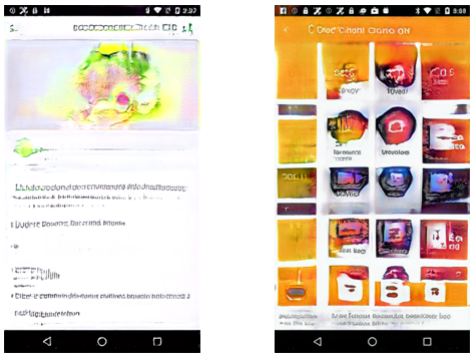


Figure 6.1 Two GUI designs selected from two different sets and produced by GANSpiration [4] having a very different structure and style. The diversity score of 0.712 and a FID of 213.492 are computed for the two images.

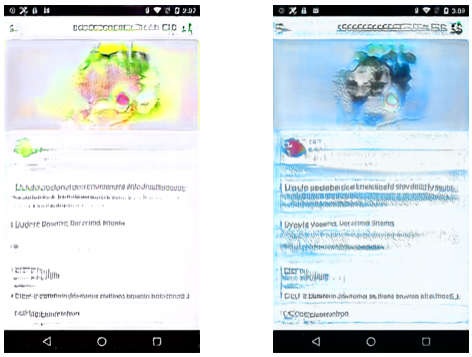
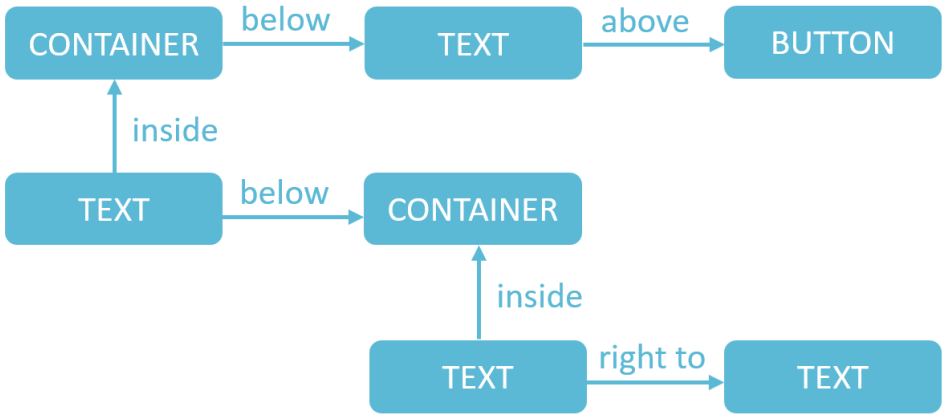


Figure 6.2 Two GUI designs selected from two different sets and produced by GANSpiration [4] having a very similar structure and style. The diversity score of 0.353 and a FID of 137.549 is computed for the two images.

6.3 One-step method vs two-step method

Different inputs combinations can be considered for the second step in our pipeline. More precisely, in the previous chapter, we used a default text prompt and the layouts produced by the first step as input, however it is also possible to use a text prompt describing the graph along with a layout, or only a text prompt describing the graph. The text prompt describing the graph would then be the GUI-AG put in a textual form. To do so, we applied the same process as for GUILGET, by aligning in a sequence all the triplets that are part of the GUI-AG. Moreover, we also add the ID of each component, next to the class of the component, in order to recognize different instances of the same component as illustrated in Figure 6.3. Using the GUI-AG text prompts alone is not requiring GUILGET in the pipeline, hence it would be a one-step method that produces GUI designs directly from GUI-AGs. Using the



CONTAINER 1 below **TEXT 2**, **TEXT 2** above **BUTTON 3**, **TEXT 5** below **CONTAINER 4**, **TEXT 5** inside **CONTAINER 1**, **TEXT 6** right **PICTOGRAM 7**, **TEXT 6** inside **CONTAINER 4**

Figure 6.3 Example of a GUI-AG being serialized to a GUI-AG text prompt. In addition to component classes and predicates, IDs are added in order to identify the instances of the same object in the GUI layout.

same metrics as in the previous section, we obtain the results in Table 6.3.

The quantitative results shows us a greater performance from the method with the usage of a GUI-AG text prompt combined with GUI layout as input. Indeed, the FID as well as the diversity score are better than with the other two types of input, even if the difference is small. This leads to the interpretation that providing more details about the expected GUI design as output is closer in terms of features distribution with the ground-truth data. Moreover. those features distribution are more diversified. These quantitative results may lead to the conclusion that using GUI-AG text prompt with GUI layout is the type of input that must be used. However, they do not evaluate the quality of the layout, only the overall appearance. It is difficult to evaluate the layout of screenshots, as one would need to extract all the IU elements. As a result, we also performed a qualitative evaluation of the visual quality. Figure 6.4 shows some results for a randomly selected input of a high complexity. A high complexity corresponds to a GUI composed of 8 or more different types of UI components. In the given input screenshot there are exactly 8 different types of UI components. The input design is used to extract the GUI-AG and/or GUI layout that are used as input for

Table 6.3 Quantitative evaluation of GUI designs on CLAY dataset [1] using the FID and diversity score as metrics. We compare our method using different inputs, more precisely the inputs are (1) GUI-AG text prompt alone (2) GUI-AG text prompt combined with GUI layout (3) Default text prompt with GUI layout. The default text prompt is the same as previously: "High quality, detailed, and professional app interface". We used 400 ground-truth screenshots and 400 generated ones for the evaluation.

	FID ↓	**Diversity score ↑**
GUI-AG text prompt	56.232	0.716
GUI-AG text prompt + Layout	**50.799**	**0.736**
Default text prompt + Layout (GILD)	52.872	0.710

generating the results appearing in next columns. GUI design screenshots generated from a GUI-AG text prompt lack of fidelity in regards to the layout of the ground-truth, even if they look as possible GUI designs. Moreover, there are components that are missing and others that are used even though they were not in the constraints. For example, there is an excessive amount of images in some of those designs, while there is only one in the constraint in the input, representing the logo of the app. On the other hand, there must be a container containing 9 distinct text fields, which is appearing only in the design on bottom left. If the GUI layout is used additionally, the layout fits with the layout of the ground-truth. However, we can notice that visual distortions are appearing in screenshots generated from *GUI-AG text prompt+layout*. This is not the case when the GUI layout is used with the default prompt as input. In this scenario, components are distinct and perfectly fitting the ground-truth structure. For these reasons, it is preferable to use a default text prompt with a GUI layout as input. We can conclude that the GUI-AG text prompt has a limitation of interpretation by the model.

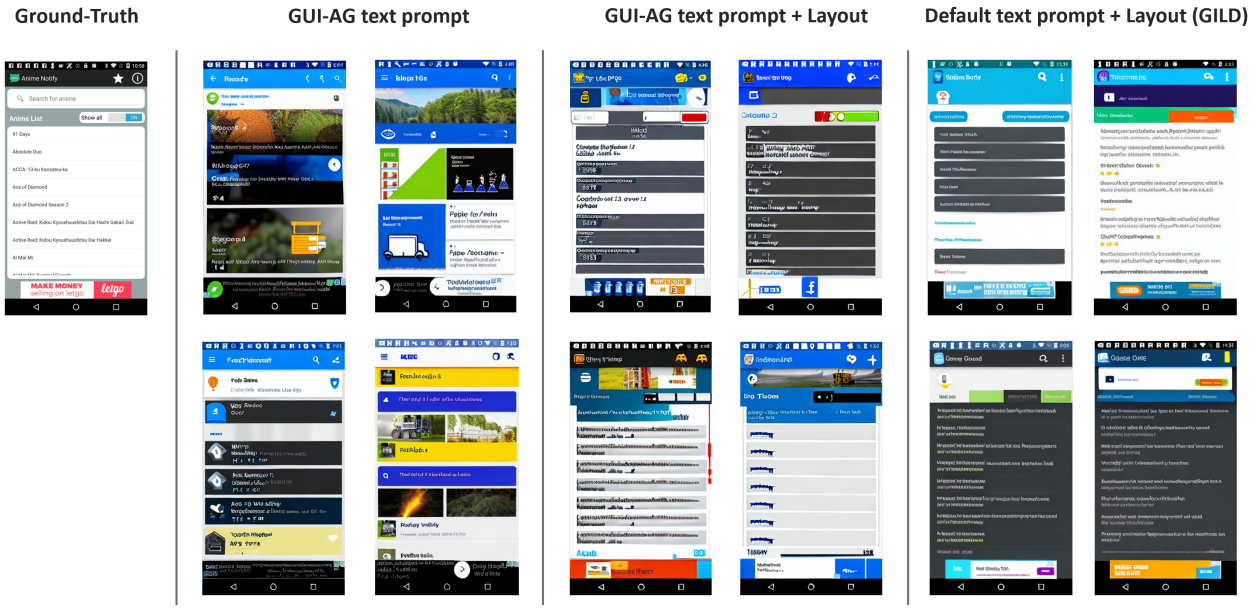


Figure 6.4 Qualitative comparison of our method using different inputs, more precisely the inputs are (1) GUI-AG text prompt (2) GUI-AG text prompt combined with GUI layout (3) Default text prompt and GUI layout (GILD). In the first column, the ground-truth screenshot from which the GUI-AG and GUI layout are extracted and passed as input for generating the other screenshots.

CHAPTER 7 CONCLUSION

In this section, we conclude on our research work and provide an insight about the possible future researches that are possible in this field based on knowledge we acquired.

7.1 Summary of Works

We proposed a new complete pipeline for the task of GUI generation from GUI-AGs, a task that we are the first to explore. This pipeline consists of two steps: (1) GUI layouts generation from GUI-AGs and (2) GUI design screenshots generation from GUI layouts.

If we enter into more details, we have shown that a transformer-based method is better in interpreting the semantic and logic in a GUI-AG. Indeed, methods employing GCN do not have a good understanding of relationships especially for *inside* relationships as shown in the results. Beside the introduction of GUI-AG for representing GUI layouts, we have introduced new techniques, used in our method named GUILGET, that are specific to the field of GUI design that allowed to improve performances compared to a classic transformer-based method. These techniques consist of (1) a new hierarchical structure to express parent-children relationships to let the model capture the parent of each UI component, (2) novel loss functions and metrics that allow to include better the children component inside its parent and separate children that share a common parent, and (3) a grid with fixed cell sizes inspired from techniques used by professional GUI designers. We have shown that the category of GUI does not have any impact on the performances of GUILGET while the number of different types of UI component has one. Indeed, the more unique type of component there are, the harder is the problem and our method performance decreases.

To complete the method, we proposed a method called GILD that uses the ControlNet method that takes as input a color-coded version of the GUI layout produced by GUILGET and a default text prompt that is "High quality, detailed, and professional app interface". The output of this step are GUI design screenshots corresponding to the GUI layout structure. The training of the two steps are done separately, so the second step is trained on generating GUI designs from ground-truth GUI layouts. The evaluation of the generated designs was done qualitatively by comparing design screenshots from different methods, but also quantitatively to measure the quality and diversity of generated design screenshots respectively. And finally, a user evaluation is part of the evaluation of the pipeline. This part is essential due to lack of reliable metrics to evaluate the quality of a GUI design in terms of creativity,

human interactions, and respect of GUI design principles. Our method for GUI design generation from constraints give the best results compared to prior art in terms of diversity and visual quality. As well, professional UX/UI designers all agreed to say that the generated designs by our pipeline are relevant and diverse in regard to the real-world designs. This tool is something they would integrate in their workflow.

7.2 Limitations

Several limitations can have an impact on the performances and evaluation of our proposed method and should be considered:

- Limited dataset availability: There is only one dataset with mobile interfaces so it makes it impossible to test the model on websites, computer software or tablet interfaces for instance.
- Consistency: Our method does not allow to produce GUI designs in a particular style. In consequence, it is a challenging task to produce several GUI designs for a specific application category. The method should be used only for inspiration rather than for the design of an entire application.
- Evaluation: Assessing the quality of the produced GUI design screenshots is a challenge. It is mostly done qualitatively by an evaluation from users. Moreover, evaluating the visual quality would be enough for an image but it is not for a GUI, indeed, the usability of the produced GUIs must be also evaluated.
- Computational requirements: A major problem we faced is the fact that graphs can have important growth just by adding a few components due to multiple relationships that can be linked to a single component. This implies that often the input is large and requires high computational capabilities. Moreover, it is well-known that training of high-quality generative models requires also a lot of computational resources to work with large images.

7.3 Future Research

Since this work is the first proposing to generate GUI designs from given constraints, it leaves many research opportunities. There are several potential improvements that can be explored:

1. One interesting idea would be to generate GUI designs based not only on the positional constraints but also the style. Hence, considering to add the category of the interface

in addition to the GUI-AG is a possible improvement that can be brought. For the second step it is straightforward. The use of a text prompt different from the default propose one can be done. For the first step it is less trivial. Possibly, the best choice is to reserve a portion of the input for the screen category and add a new value in type of word embedding $e^t_{1:T}$ value 4 corresponding to the *screen category*.

2. The first step has room for improvements, especially due to lower performances for high complexity screens. To overcome that, one could try to employ knowledge from GUI design to improve performances of a GCN that processes a GUI-AG to generate a GUI layout. Also, since transformers have shown to be the most effective to solve NLP problems, it might be also interesting to exploit transformers and implement new techniques. For instance, one could think of skip connections, indeed, there are three modules responsible for different tasks. Skip connections could help to allow the information travel across each module. Also, as we saw, the implementation of technique based on knowledge of GUI design principles and process allowed to improve the results. I would be recommended to explore more principles from this area and implement them in the method to improve the results.

3. Using a text prompt for guiding a GUI design generation task is a more complex process compared to using GUI-AGs as input. The reason comes from the freedom the text prompt allows in terms of formulation. Exploring the field of text prompt to GUI design generation is a possible research direction as perhaps there is a better way to encode constraints than making a textual version of the GUI-AG.

REFERENCES

[1] G. Li *et al.*, "Learning to denoise raw mobile ui layouts for improving datasets at scale," in *CHI Conference on Human Factors in Computing Systems*, 2022, pp. 1–13.

[2] J. Johnson, A. Gupta, and L. Fei-Fei, "Image generation from scene graphs," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 1219–1228.

[3] C.-F. Yang *et al.*, "Layouttransformer: Scene layout generation with conceptual and spatial diversity," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2021, pp. 3732–3741.

[4] M. A. Mozaffari *et al.*, "Ganspiration: Balancing targeted and serendipitous inspiration in user interface design with style-based generative adversarial network," in *CHI Conference on Human Factors in Computing Systems*, 2022, pp. 1–15.

[5] L. Zhang and M. Agrawala, "Adding conditional control to text-to-image diffusion models," *arXiv preprint arXiv:2302.05543*, 2023.

[6] T. Zhao *et al.*, "Guigan: Learning to generate gui designs using generative adversarial networks," in *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 2021, pp. 748–760.

[7] L. Yu *et al.*, "Seqgan: Sequence generative adversarial nets with policy gradient," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 31, no. 1, 2017.

[8] S. He *et al.*, "Context-aware layout to image generation with enhanced object appearance," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2021, pp. 15 049–15 058.

[9] X. Zhao *et al.*, "High-quality image generation from scene graphs with transformer," in *2022 IEEE International Conference on Multimedia and Expo (ICME)*. IEEE, 2022, pp. 1–6.

[10] P. Esser, R. Rombach, and B. Ommer, "Taming transformers for high-resolution image synthesis," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2021, pp. 12 873–12 883.

[11] "Understanding layout," https://m2.material.io/design/layout/understanding-layout.html#principles, accessed: September 20, 2023.

[12] "Number of mobile app downloads worldwide from 2016 to 2022," https://www.statista.com/statistics/271644/worldwide-free-and-paid-mobile-app-store-downloads/, accessed: August 5, 2023.

[13] "Average number of new android app releases via google play per month from march 2019 to june 2023," https://www.statista.com/statistics/1020956/android-app-releases-worldwide/, accessed: August 5, 2023.

[14] "Retention rate on day 30 of mobile app installs worldwide in 3rd quarter 2022," https://www.statista.com/statistics/259329/ios-and-android-app-user-retention-rate/, accessed: August 5, 2023.

[15] T. Winograd, "From programming environments to environments for designing," *Communications of the ACM*, vol. 38, no. 6, pp. 65–74, 1995.

[16] L. Bruton, "How to become a better ui designer: 9 expert tips," https://www.uxdesigninstitute.com/blog/become-a-better-ui-designer, accessed: August 5, 2023.

[17] "What is a gui (graphical user interface)? definition, elements and benefits," https://www.indeed.com/career-advice/career-development/gui-meaning, accessed: August 5, 2023.

[18] E. Wong, "User interface design guidelines: 10 rules of thumb," https://www.interaction-design.org/literature/article/user-interface-design-guidelines-10-rules-of-thumb#:~:text=Nielsen%20and%20Molich%27s%2010%20User%20Interface%20Design%20Guidelines,8%20Aesthetic%20and%20minimalist%20design.%20...%20%C3%89l%C3%A9ments%20suppl%C3%A9mentaires, accessed: August 5, 2023.

[19] "6 stages of ui design and what's involved," https://designerup.co/blog/6-stages-of-ui-design-and-whats-involved/, accessed: August 5, 2023.

[20] J. Hong, "Matters of design," *Communications of the ACM*, vol. 54, no. 2, pp. 10–11, 2011.

[21] "It's not easy to develop mobile apps," https://www.cmswire.com/mobile-enterprise/its-not-easy-to-develop-mobile-apps/, accessed: August 5, 2023.

[22] D. A. Norman, "The design of everyday things." 1988.

[23] S. Krug *et al.*, "Don't make me think, revisited: A common sense approach to web usability," *A Common Sense Approach to Web and Mobile Usability*, 2014.

[24] B. Deka *et al.*, "Rico: A mobile app dataset for building data-driven design applications," in *Proceedings of the 30th annual ACM symposium on user interface software and technology*, 2017, pp. 845–854.

[25] I. Goodfellow *et al.*, "Generative adversarial nets," *Advances in neural information processing systems*, vol. 27, 2014.

[26] A. Graves and A. Graves, "Long short-term memory," *Supervised sequence labelling with recurrent neural networks*, pp. 37–45, 2012.

[27] K. O'Shea and R. Nash, "An introduction to convolutional neural networks," *arXiv preprint arXiv:1511.08458*, 2015.

[28] R. K. Srivastava, K. Greff, and J. Schmidhuber, "Highway networks," *arXiv preprint arXiv:1505.00387*, 2015.

[29] J. Bromley *et al.*, "Signature verification using a" siamese" time delay neural network," *Advances in neural information processing systems*, vol. 6, 1993.

[30] M. Heusel *et al.*, "Gans trained by a two time-scale update rule converge to a local nash equilibrium," *Advances in neural information processing systems*, vol. 30, 2017.

[31] G. Yang *et al.*, "Pointflow: 3d point cloud generation with continuous normalizing flows," in *Proceedings of the IEEE/CVF international conference on computer vision*, 2019, pp. 4541–4550.

[32] Q. Xu *et al.*, "An empirical study on evaluation metrics of generative adversarial networks," *arXiv preprint arXiv:1806.07755*, 2018.

[33] T. Karras, S. Laine, and T. Aila, "A style-based generator architecture for generative adversarial networks," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 4401–4410.

[34] M. Ester *et al.*, "A density-based algorithm for discovering clusters in large spatial databases with noise," in *kdd*, vol. 96, no. 34, 1996, pp. 226–231.

[35] H. Zhang *et al.*, "Stackgan: Text to photo-realistic image synthesis with stacked generative adversarial networks," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 5907–5915.

[36] R. Herzig *et al.*, "Learning canonical representations for scene graph to image generation," in *European Conference on Computer Vision*. Springer, 2020, pp. 210–227.

[37] B. Zhao *et al.*, "Image generation from layout," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 8584–8593.

[38] X. Shi *et al.*, "Convolutional lstm network: A machine learning approach for precipitation nowcasting," *Advances in neural information processing systems*, vol. 28, 2015.

[39] A. Vaswani *et al.*, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[40] T. Salimans *et al.*, "Improved techniques for training gans," *Advances in neural information processing systems*, vol. 29, 2016.

[41] R. Zhang *et al.*, "The unreasonable effectiveness of deep features as a perceptual metric," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 586–595.

[42] J. Ho, A. Jain, and P. Abbeel, "Denoising diffusion probabilistic models," *Advances in neural information processing systems*, vol. 33, pp. 6840–6851, 2020.

[43] R. Rombach *et al.*, "High-resolution image synthesis with latent diffusion models," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2022, pp. 10 684–10 695.

[44] J. Li *et al.*, "Layoutgan: Generating graphic layouts with wireframe discriminators," *arXiv preprint arXiv:1901.06767*, 2019.

[45] X. Wang *et al.*, "Non-local neural networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 7794–7803.

[46] A. A. Jyothi *et al.*, "Layoutvae: Stochastic scene layout generation from a label set," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 9895–9904.

[47] K. Gupta *et al.*, "Layouttransformer: Layout generation and completion with self-attention," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 1004–1014.

[48] H.-Y. Lee *et al.*, "Neural design network: Graphic layout generation with constraints," in *European Conference on Computer Vision*. Springer, 2020, pp. 491–506.

[49] A. G. Patil *et al.*, "Read: Recursive autoencoders for document layout generation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, 2020, pp. 544–545.

[50] K. Papineni *et al.*, "Bleu: a method for automatic evaluation of machine translation," in *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, 2002, pp. 311–318.

[51] D. M. Arroyo, J. Postels, and F. Tombari, "Variational transformer networks for layout generation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 13 642–13 652.

[52] D. Zhou *et al.*, "Iou loss for 2d/3d object detection," in *2019 international conference on 3D vision (3DV)*. IEEE, 2019, pp. 85–94.

[53] C. Szegedy *et al.*, "Rethinking the inception architecture for computer vision," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 2818–2826.

[54] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[55] R. Hartson and P. S. Pyla, *The UX book: Agile UX design for a quality user experience*. Morgan Kaufmann, 2018.

[56] J. Tidwell, *Designing interfaces: Patterns for effective interaction design*. " O'Reilly Media, Inc.", 2010.

[57] J. Devlin *et al.*, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[58] D. A. Reynolds *et al.*, "Gaussian mixture models." *Encyclopedia of biometrics*, vol. 741, no. 659-663, 2009.

[59] "Responsive layout grid," https://m2.material.io/design/layout/responsive-layout-grid.html#columns-gutters-and-margins, accessed: August 5, 2023.

[60] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015: 18th International Conference, Munich, Germany, October 5-9, 2015, Proceedings, Part III 18*. Springer, 2015, pp. 234–241.

[61] J. Deng *et al.*, "ImageNet: A Large-Scale Hierarchical Image Database," in *CVPR09*, 2009.

[62] M. Vaismoradi, H. Turunen, and T. Bondas, "Content analysis and thematic analysis: Implications for conducting a qualitative descriptive study," *Nursing & Health Sciences*, vol. 15, no. 3, pp. 398–405, 2013.

[63] Y. Lamine and J. Cheng, "Understanding and supporting the design systems practice," *Empirical Software Engineering*, vol. 27, no. 6, p. 146, 2022. [Online]. Available: https://doi.org/10.1007/s10664-022-10181-y

[64] Z. Jiang *et al.*, "Layoutformer++: Conditional graphic layout generation via constraint serialization and decoding space restriction," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023, pp. 18 403–18 412.

[65] X. Kong *et al.*, "Blt: bidirectional layout transformer for controllable layout generation," in *Computer Vision–ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part XVII*. Springer, 2022, pp. 474–490.

[66] N. Inoue *et al.*, "Layoutdm: Discrete diffusion model for controllable layout generation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023, pp. 10 167–10 176.

APPENDIX A USER-STUDY SCENARIOS

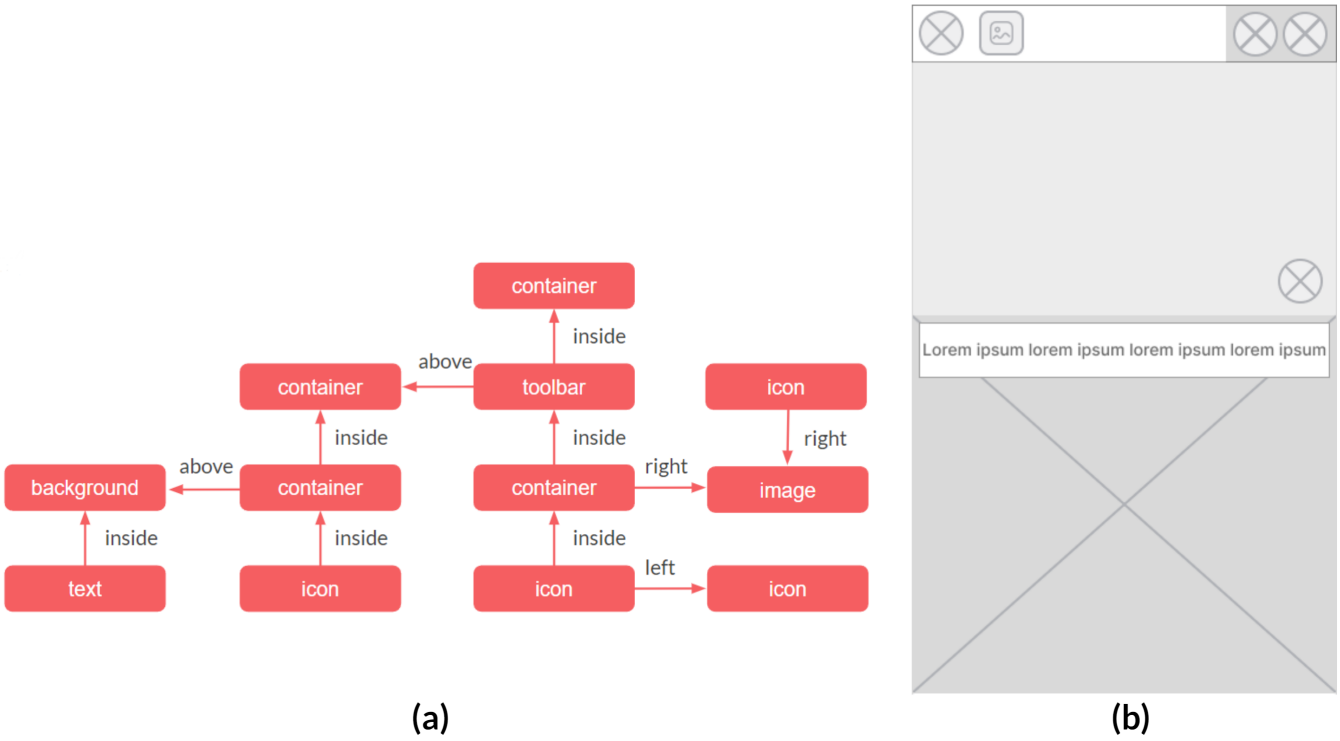


Figure A.1 Inputs representing the constraints of a News app in the form of (a) GUI-AG and (b) GUI layout.

In the following we present the user study script prepared to conduct the interviews and ask relevant questions about our results and our method.

Demographic Questions

These questions will be asked before the study.

- How long have you been working as a user experience designer?
- How many design projects have you contributed to?
- How do you usually approach design ideation?
  - What is your general design ideation process?
  - What are the main challenges you encountered related to design ideation?

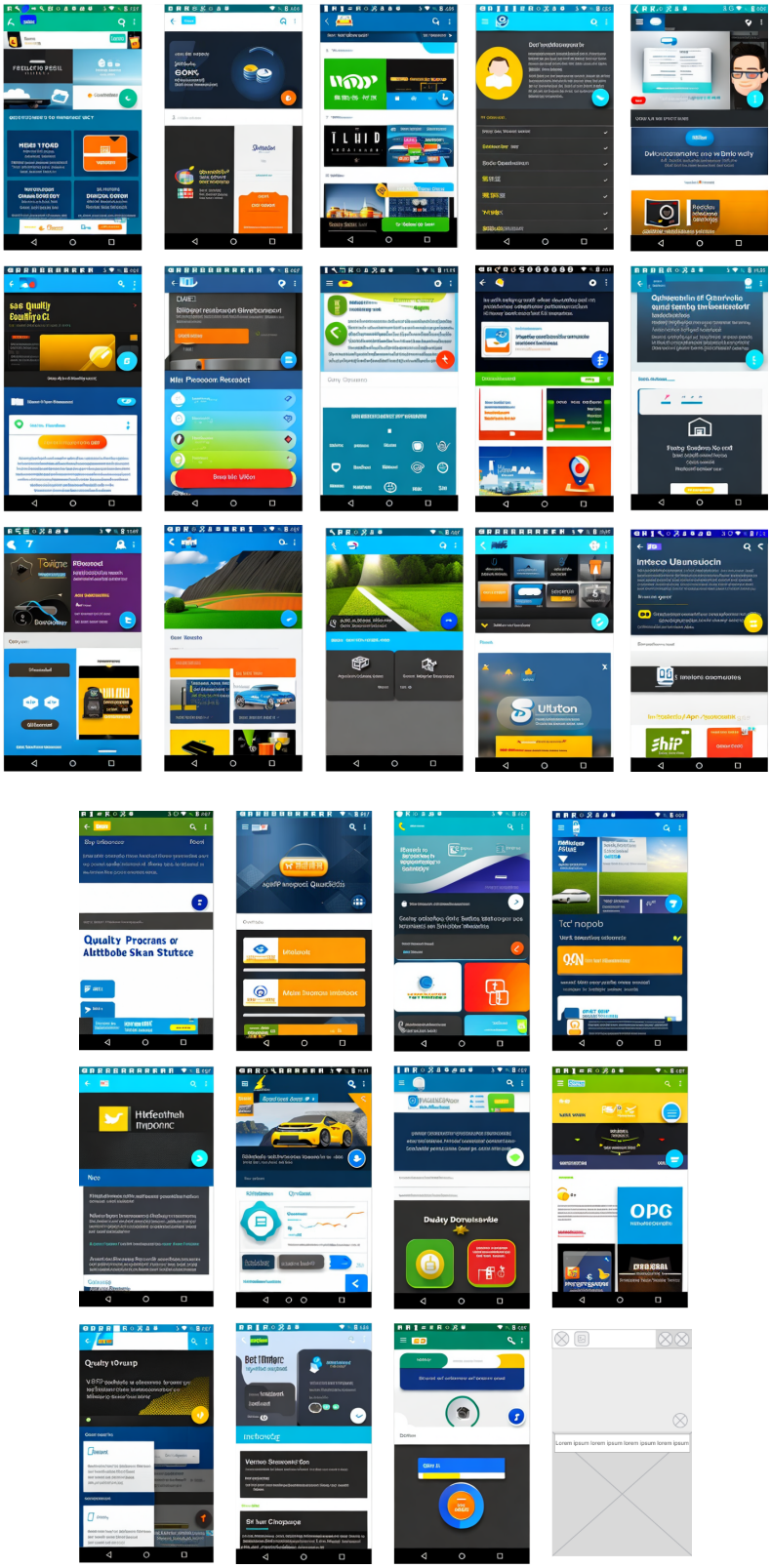


Figure A.2 Designs generated using ControlNet with Figure A.1 (b) as input GUI layout.

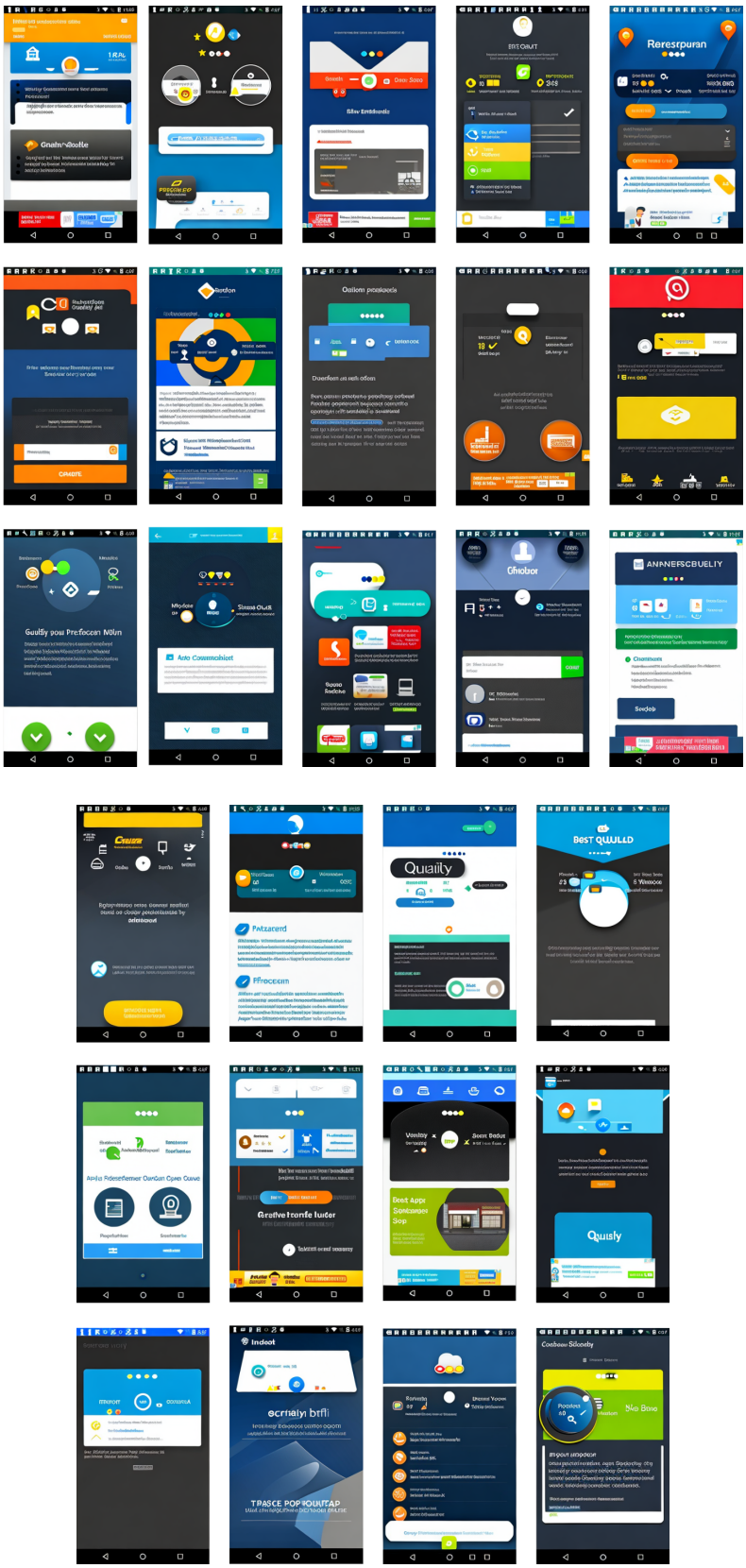


Figure A.3 Designs generated using our complete pipeline with Figure A.1 (a) as input GUI-AG.

- How frequently do you search for examples before beginning your design process for inspiration?
  - How have you performed the search? What is usually your input?

Study Script

We sincerely appreciate your participation in this research study. As mentioned in the consent form you've reviewed, our focus is to understand the perspectives of UX designers regarding design examples generated using a machine learning algorithm.

Before we proceed, please remember that this study will be recorded. We are committed to maintaining the confidentiality of your identity and the information shared. You have the right to withdraw from the study at any point, and if you choose to do so, all data associated with your participation will be deleted.

Additionally, it's important to emphasize that there is no definitive right or wrong way to answer any of the questions. Our intention is not to assess the correctness of your answers; rather, we aim to gain insight from your feedback. Do you have any questions before we continue? Please feel free to ask.

[*explain the app and present the wireframe Figure A.1(b):*]

We would like to build a news app with a toolbar on the top containing the logo and several clickable icons, just below we would have a container that will show either a video or an image to illustrate the news, below that we would like to have a text as caption for the video/image and then a background which is describing the news in the written form.

Now, I'll display the corresponding design examples Figure A.2 that were generated based on the wireframe you just saw. [*Ask follow-up questions when interesting answers emerge*]:

- Imagine you are designing a news application, which examples would you pick to start ideating?
  - Why would you pick those examples?
- What are the two things you like about the design examples?
- What are the two things you dislike about the design examples?
- How diverse are the design examples?
- How effective do you think these design examples can inspire a UI designer?

Now, I will present you with a graph Figure A.1(a), which is a visual representation showing the different elements of a UI page and how they relate in terms of position in the UI. The goal of this graph is to represent the requirements and constraints that the UI must meet.

- Do you think this type of graph can be useful in your UI/UX design workflow? If so, how? If not, why?

Now, I'll display the corresponding design examples Figure A.3 that were generated based on the graph. [*Ask follow-up questions when interesting answers emerge*]:

- Imagine you are designing a news application, which examples would you pick to start ideating?
  - Why would you pick those examples?
- What are the two things you like about the design examples?
- What are the two things you dislike about the design examples?
- How diverse are the design examples?
- How effective do you think these design examples can inspire a UI designer?

[*Ask the following question after all designs are reviewed.*]

- In general, what do you think about the generated design examples?
- Supposed that someone has developed a design tool that uses this technique, how would this tool be integrated in your design workflow?
- What do you think about the graph now? Do you think you could use it to obtain design examples?
- What about the wireframe? Do you think you could use it to obtain design examples?
- What kind of input would you prefer to have? The graph or the wireframe? Why?
- Are there other kinds of input you would rather use? If so, what kinds and why?
- Do you have any other comments or questions?

Thank you so much for your participation!