TRIBHUVAN UNIVERSITY

INSTITUTE OF ENGINEERING

THAPATHALI CAMPUS

Major Project Mid-term Report

On

User Interface Code Generation from Hand-drawn Sketch

Submitted By:

Manoj Paudel	(THA077BCT025)
Prince Poudel	(THA077BCT036)
Ronish Shrestha	(THA077BCT040)
Sonish Poudel	(THA077BCT042)

Submitted To:

Department of Electronics and Computer Engineering

Thapathali Campus

Kathmandu, Nepal

August, 2024

TRIBHUVAN UNIVERSITY

INSTITUTE OF ENGINEERING

THAPATHALI CAMPUS

Major Project Mid-Term Report

On

User Interface Code Generation from Hand-drawn Sketch

Submitted By:

Manoj Paudel	(THA077BCT025)
Prince Poudel	(THA077BCT036)
Ronish Shrestha	(THA077BCT040)
Sonish Poudel	(THA077BCT042)

Submitted To:

Department of Electronics and Computer Engineering

Thapathali Campus

Kathmandu, Nepal

In partial fulfillment for the award of the Bachelor's Degree in Computer Engineering.

Under the Supervision of

Er. Devendra Kathayat

August, 2024

ACKNOWLEDGEMENT

We would like to express our sincere gratitude towards the Institute of Engineering, Tribhuvan University for the inclusion of major project in the course of Bachelors in Computer Engineering. We are also thankful towards our Department of Electronics and Computer Engineering for the proper orientation and guidance during the project "**User Interface Code Generation from Hand-drawn Sketch**".

We would like to acknowledge the authors of various research papers and developers of various programming libraries and frameworks that we have reference for developing our project.

We would like to express our gratitude towards our Project Supervisor **Er.Devendra Kathayat** for continuous suggestions throughout.

Finally, we would like to thank all the people who are directly or indirectly related during our study and preparation of this project.

Manoj Paudel	(THA077BCT025)
Prince Poudel	(THA077BCT036)
Ronish Shrestha	(THA077BCT040)
Sonish Poudel	(THA077BCT042)

ABSTRACT

This report presents "User Interface Code Generation from hand-drawn sketch" to develop a machine learning-based system that can generate the code in HTML/CSS from the given sketch of the layout. This project covers the topics of Image processing, Transformer model, and compiler to translate the domain specific language to the specific language on the basis of the platform. This system will be very useful to the company that want to present the prototype of the user interface to the user. Our project reduces the development time required to go from design to deployment. Skilled man power in a company can focus on more complex and value adding aspect of the project. Developers can easily customize and extend the generated code as per their need. We can also add a dynamicity in the project by adding JavaScript code. The system will support multiple frontend frameworks providing flexibility and compatibility with various development environment. The project is expected to develop a user-friendly system that can generate code from the layout which can run on any platform.

Keywords: Artificial Intelligence, Domain Specific Language (DSL), Image Processing, Image synthesis, Self-Attention, Transformer Decoder, Vision Transformer

TABLE OF CONTENTS

ACKNOWLEDGEMENT .... **i**

ABSTRACT .... **ii**

List of Figures .... **vi**

List of Tables .... **vii**

List of Abbreviations .... **viii**

1 INTRODUCTION **1**

1.1 Background 1

1.2 Motivation 1

1.3 Problem Definition 2

1.4 Objectives 2

1.5 Scope and Applications 3

2 LITERATURE REVIEW **4**

2.1 Different methods used for this kind of task 4

2.2 Vision Transformer for Computer vision 6

3 REQUIREMENT ANALYSIS **7**

3.1 Software Requirement 7

3.1.1 Python 7

3.1.2 Numpy 7

3.1.3 Tensorflow 7

3.1.4 Keras 7

3.2 Hardware Requirement 8

3.2.1 Camera Device 8

3.2.2 Cloud Computing Platform 8

4 SYSTEM ARCHITECTURE AND METHODOLOGY **9**

4.1 Dataset 9

4.1.1 Dataset Generator 9
4.2 System Block Diagram 10
4.3 Working Principle 10
4.3.1 Image Processing 11
4.3.2 Convolutional Tokenizer 12
4.3.3 Transformer Model 12
4.3.4 Domain Specific Language (DSL) 17
4.3.5 Customization 18
4.3.6 Compiler 19
4.4 Activation Function 19
4.4.1 Softmax Function 19
4.4.2 ReLU Function 20
4.5 Loss Function and Optimizer 21
4.5.1 Categorical cross entropy 21
4.5.2 Adam Optimizer 21
4.6 Performance Metrics 22
4.6.1 BLEU 22
4.6.2 ROUGE 22
4.7 Flowchart 23
4.7.1 During Training 23
4.7.2 During Testing 24
5 IMPLEMENTATION DETAILS 25
5.1 Dataset Generator Implementation 25
5.1.1 Steps taken in Dataset Generation 25
5.1.2 Rules for creating DSL 26
5.2 Model Implementation 26
5.2.1 Input Processing: 26

5.2.2 Convolutional Tokenizer: 26
5.2.3 Positional Encoding: 28
5.2.4 Encoder: 28
5.2.5 Decoder: 29
5.2.6 Attention Mechanisms: 30
5.2.7 Masking: 30
5.2.8 Output Layer: 30
5.3 Model Complexity: 30
5.4 Vocabulary of Model 30
5.5 Training details: 31
5.6 Testing details: 31
6 RESULTS AND ANALYSIS 32
6.1 Model Training 32
6.1.1 Loss vs Epoch Graph 32
6.1.2 Evaluation 33
7 REMAINING TASK 36
8 APPENDICES 37
APPENDIX A: Project Schedule 37
APPENDIX B: DSL Generation Rules 37
Appendix C: Sample Generated Datasets 39
REFERENCES 44

LIST OF FIGURES

Figure 4-1: Block Diagram of Data Synthesis….……….……..…….….… 9

Figure 4-2: Sample of some sketched elements….……….……...………….. 10

Figure 4-3: Block Diagram of Working of system….…………..………..………..10

Figure 4-4: Block Diagram of Transformer model..………….………….………..13

Figure 4-5: Multilayer Perceptron….………….……………… ………………....16

Figure 4-6: Sample input image……….………….………..………………….. 17

Figure 4-7: Graphical Representation of ReLU function …….....………..……… 21

Figure 4-8: Flowchart for training….………….………..……………………….. 23

Figure 4-9: Flowchart for testing….……….……….…..……… …………………24

Figure 6-1: Loss vs Epoch graph for Training data……..…….. …………………32

Figure 6-2: Loss vs Epoch graph for Validation data……..…….. ….……………32

Figure 6-3: BLEU-10 scores in sorted order………..……..…….. ….……………33

Figure 6-3: ROUGE-L F1-scores in sorted order………………... ….……………35

LIST OF TABLES

Table 6-1: ROUGE Score for test data……………………..…………..........................34
Table 8-1: Gantt Chart……………………...……………….………..........................37

LIST OF ABBREVIATIONS

AI	Artificial Intelligence
API	Application Programming Interface
BLEU	Bilingual Evaluation Understudy
CNN	Convolutional Neural Network
CSS	Cascading Style Sheets
CV	Computer Vision
DSL	Domain Specific Language
DSLR	Digital Single-Lens Reflex
GUI	Graphical User Interface
HTML	Hyper Text Markup Language
JS	JavaScript
LDP	Line Drawing Prediction
LSTM	Long Short Term Memory Networks
MLP	Multi-level Perceptron
MP	Mega Pixel
NLP	Natural Language Processing
RELU	Rectified Linear Unit
UI	User Interface
ViT	Vision Transformer

1 INTRODUCTION

1.1 Background

In realm of software development process traditionally user interface development has been relied on manual interpretation and implementation of design specifications. Although, manual coding of UI elements such as buttons, input fields, and layouts consumes not only valuable time but also introduces multiple potential errors and variations between the intended design and the implemented interface. Upon discovering the fact this project aims to address these challenges by proposing a system Sketch-to-code, capable of transforming sketches into executable front-end code. It addresses the challenges that have been affecting designers and developers for years: that even though designers go to great lengths to create digital solutions that are perfectly measured and consistently crafted, design systems do not fluently translate to development systems.

In recent years, technology and artificial intelligence have significantly changed the process of converting graphical user interface (GUI) sketches into functional frontend code. Researchers are exploring machine learning and advanced image processing to develop automated systems capable of understanding and generating code directly from design sketches. These systems benefit developers by speeding up UI development, ensuring consistency between design and implementation, and allowing for more focus on refining user experience and functionality. Furthermore AI-driven solutions continue to advance, they promise to revolutionize software development practices, making them more efficient and accessible for developers across different industries.

1.2 Motivation

This project aims to solve a common problem in software development: the lengthy process of turning design ideas into working software. Developers often face challenges when the final product doesn't match the original design, which leads to making many adjustments and can cause delays in finishing projects on time. In order to solve this problem, we will be providing simple prototype front end code which will be easier for developer. By automating the first steps of creating user interfaces (UI), we want to make this process faster and smoother. This means developers can spend more time improving how the software works for users, rather than getting stuck on the technical details of building buttons and menus.

In essence, our goal is to use technology to simplify software development, making it easier for developers to create software that matches the original vision without wasting time on repetitive tasks. Moreover, not to create variance from the original intended design of user.

1.3 Problem Definition

The problem is to develop a system that can automate the process of turning sketches into functional front-end code. The system should be able to craft interactive User Interface that is more like designer intended.

Some of the key challenges are listed below:

- **Ambiguity in Requirements**: Sketches may not always clearly define all requirements, leading to ambiguity. This can result in misunderstandings between stakeholders, designers, and developers.
- **Technical Feasibility**: Sometimes, what looks good on paper (or screen) may not be technically feasible within the constraints of the project, platform, or technology stack.
- **Integration Complexity**: Projects often require integrating various components, APIs, or third-party services, which can introduce complexity beyond the initial sketch.
- **User Experience (UX) Considerations**: Implementing a sketch into code involves not just visual fidelity but also ensuring a smooth and intuitive user experience, which may require iterative improvements.

1.4 Objectives

- To construct a model able to generate quick GUI prototype from sketch into HTML code.
- To make interactive user interface for customizing and stylizing the generated code.

1.5 Scope and Applications

Computer science projects typically start with a concept or idea, often visualized through sketches or diagrams that outline the project's goals, user interfaces, and system architecture. These sketches serve as a blueprint for translating abstract concepts into tangible software solutions. The scope of converting these sketches into code lies in transforming visual representations into functional code that aligns with specific requirements and user needs.

The application of this sketch-to-code transition simplifies the development process by providing a clear roadmap for programmers to follow. It ensures that developers understand the project's scope, functionalities, and design aesthetics from the outset. By adhering to the sketches, developers can maintain consistency in design elements, streamline development iterations, and effectively manage project timelines and resources. Moreover, the sketches serve as a communication tool between stakeholders, designers, and developers, fostering collaboration and alignment throughout the project lifecycle.

In practical terms, the transition from sketches to code involves selecting appropriate programming languages, frameworks, and tools that best fit the project's requirements. Developers interpret the visual concepts into code syntax, ensuring that each component functions as intended and integrates seamlessly with other project modules. This process demands attention to detail, problem-solving skills, and a keen eye for user experience to deliver software that not only meets technical specifications but also provides a user-friendly interface.

Furthermore, the sketches-to-code approach facilitates iterative development and agile methodologies, allowing for continuous feedback and improvement based on user testing and stakeholder input. It empowers developers to make informed decisions, address challenges promptly, and adapt to evolving project requirements. Ultimately, by translating project sketches into code effectively, computer science projects can achieve their objectives efficiently, delivering robust software solutions that meet both technical standards and user expectations.

2 LITERATURE REVIEW

2.1 Different methods used for this kind of task

For the survey, databases like IEEE Xplore , Google Scholar, and Articles were searched manually, by using various keywords like “sketch to code”, “pics to code”, ”Image to functional code” etc. Research papers of various paper were shown on the search from the mentioned repositories like Journals, Conferences and Articles. Along with Research papers, certain existing systems were also reviewed. The research paper were studied and then this literature review explores the key methodologies, tools and technologies that have been developed and the progress made in this domain.

In 2017, Pix2code is an system described in the paper "Generating Code from a Graphical User Interface Screenshot.Pix2code is a technology that converts graphical user interface (GUI) designs, like screenshots, into executable code. Developed using deep learning techniques, it analyzes the visual layout of buttons, menus, and other elements to generate corresponding code in various programming languages. This innovation bridges the gap between design and development by automating the translation of visual concepts into functional software components. By interpreting the structure and properties of each GUI element, Pix2code ensures accuracy in code generation, speeding up the prototyping and development process significantly. This approach not only enhances efficiency but also facilitates collaboration between designers and developers, making it easier to translate design ideas into tangible applications seamlessly. [1]

In 2019, the paper titled “Eve: A sketch-based Software Prototyping Workbench” presents a novel approach to transforming hand-drawn sketches into software prototypes. Prototyping involves the evolution of an idea into various stages of design until it reaches a certain level of maturity. Conventional approaches to software prototyping typically involve Manual Coding, Graphic Design Tools & Hybrid Approaches. However they did it through a process wise approach which includes User Interface Sketch Recognition, Machine Learning, Prototype Generation, and User Interaction and at the end Feedback Loop. They present a promising approach to automating the translation of hand-drawn sketches into interactive software prototypes by streamline the prototyping process. However the system faces challenges related to recognition accuracy, complexity handling, scalability, usability and integration. [2]

In 2020, the paper titled "Automatic Code Generation from Low Fidelity Graphical User Interface Sketches Using Deep Learning" explores an advanced method for converting hand-drawn, low-fidelity GUI sketches into functional code using deep learning techniques. The process involves cleaning the sketches, recognizing different UI components using neural networks, and then generating the corresponding code with models designed for handling sequences, like LSTMs. Specifically, it uses Convolutional Neural Networks (CNNs) for image recognition and Long Short-Term Memory (LSTM) networks for sequence prediction, enhancing the system's ability to accurately interpret and generate code from sketches. The combination of CNNs for detailed image analysis and LSTMs for managing sequential data allows the system to better handle complex and densely packed UI layouts. This is an improvement over simpler models that struggled with intricate designs. This approach aims to make UI design faster and more efficient, but it faces challenges such as accurately interpreting diverse sketch styles, managing complex layouts, and ensuring the generated code integrates well with existing development workflows. [3]

Moreover, the paper "Automatic Code Generation from Sketches of Mobile Applications in End-User Development Using Deep Learning" presents a system that allows users to sketch mobile app interfaces, which are then automatically converted into functional code using deep learning techniques. This approach aims to democratize app development by enabling users without programming skills to create applications. The system utilizes Convolutional Neural Networks (CNNs) for accurate recognition of UI elements and Long Short-Term Memory (LSTM) networks for generating corresponding code sequences. Despite advancements, challenges such as ensuring accuracy across diverse sketch styles and integrating seamlessly into existing development workflows remain. Overall, the research marks a significant step towards empowering end-users in mobile app creation through automated code generation from conceptual sketches. [4]

Similarly, in 2024 a paper was published in IEEE, STC (Sketch To Code)-An Enhance Html & CSS Autocode Generator from Handwritten Text and Image Using Deep

Learning. The authors unfolds in four distinct phases: Pre-processing, Segmentation, Feature Extraction employs Line Drawing Prediction (LDP) for detailed interpretation and Classification Categorizes features into HTML and CSS elements. Early approaches relied heavily on rule-based systems and heuristics. These methods involved predefined templates and rules to convert sketches into code. However, these systems were inflexible and struggled with complex or unconventional sketches. [5]

2.2 Vision Transformer for Computer vision

Dosovitskiy et al. proposed the vision transformer, which can take image as an input and process it to classify the image at their paper “An Image is worth 16 x 16 words: Transformers for Image Recognition at scale”. Transformer have gained popularity in Natural Language Processing (NLP) and to extend this concept in computer vision, they designed this transformer model. [6]

3 REQUIREMENT ANALYSIS

3.1 Software Requirement

3.1.1 Python

Python is a high level, versatile and user-friendly general purpose, dynamic programming language where multiple programming paradigms, including object-oriented, imperative and functional programming are supported. It is an open source programming language, which let us work quickly and integrate system easily.

3.1.2 Numpy

Numpy stands for Numerical Python. Numpy is a package to perform numeric programming extension for python. It is the core library to compute scientific calculation in python. Numpy provide an array object which is up to 50x faster than traditional python lists.

3.1.3 Tensorflow

Tensorflow is basically defined as a open source software library, developed by google for various numerical computation using data flow. Tensorflow is used for building and training deep learning models, facilating the creation of computational graphs and efficient execution on various hardware platforms.

3.1.4 Keras

Keras is an open source library that offers a python interface for artificial neural network (ANN) and acts as interface to TensorFlow library. Keras acts as host tools to simplify programming language in deep neural network area while working with texts and images data. Keras is implemented in various neural-network building block area such as layers, optimizers and activation functions

3.2 Hardware Requirement

3.2.1 Camera Device

To collect the image of manually sketch design of paper any mobile phone with camera of minimum of 12 MP is required. More than 12 MP camera phone or may be DSLR camera will be additionally fruitful for us to capture high resolution images.

3.2.2 Cloud Computing Platform

For the training purpose of our model, it is essential for us with a dedicated GPU and CPU with sufficient capacity of RAM. For training purpose, we will be using colab or kaggle GPU platform, which both provide us a free hosted jupyter notebook environment in the cloud.

4 SYSTEM ARCHITECTURE AND METHODOLOGY

4.1 Dataset

Our approaches require a dataset which contains a wireframe sketch and associated DSL code. Sourcing a quality dataset is often a challenge in many machine learning projects. We were not aware of any dataset which contained wireframes sketches and associated DSL code, and therefore created our own.

4.1.1 Dataset Generator

The dataset generator is a crucial component of our project, designed to create a large and diverse set of training samples for our model. It operates through a multi-step process that begins with the random generation of Domain Specific Language (DSL) code, which describes the structure and elements of a web page layout. This DSL code is then compiled into HTML and CSS, which is subsequently rendered to create a visual representation of the layout. Then, it is analyzed to identify the positions and outlines of various elements. In the final step, hand-drawn sketches are placed of the elements at the appropriate position, then we get a sketch corresponding to the original DSL code. This automated process allows us to rapidly produce a large number of sketch-DSL pairs, each unique and representative of real-world web designs. The generator used the predefined rules and constraints which ensures that the produced layouts are diverse and realistic. This approach helps in addressing the challenges of requiring lot of dataset while training the model.

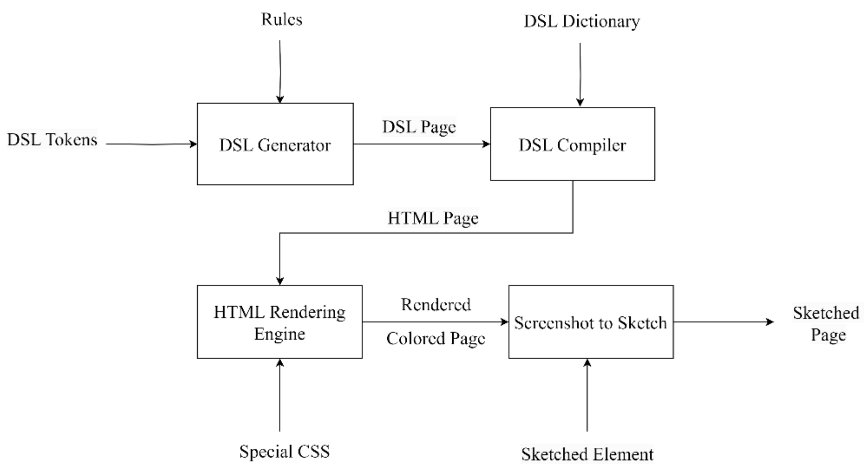


Figure 4-1: Block Diagram of Data Synthesis

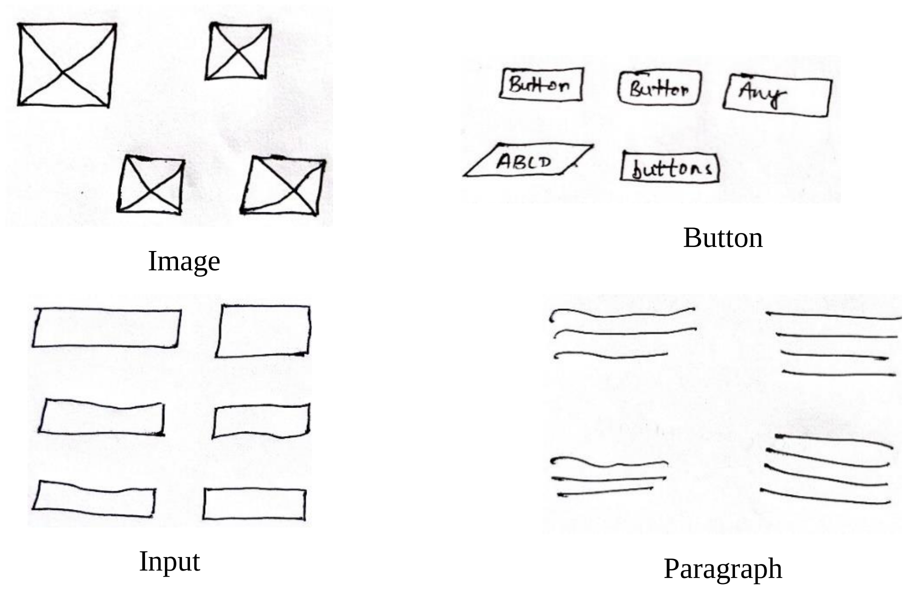


Figure 4-2: Sample of some sketched elements

4.2 System Block Diagram

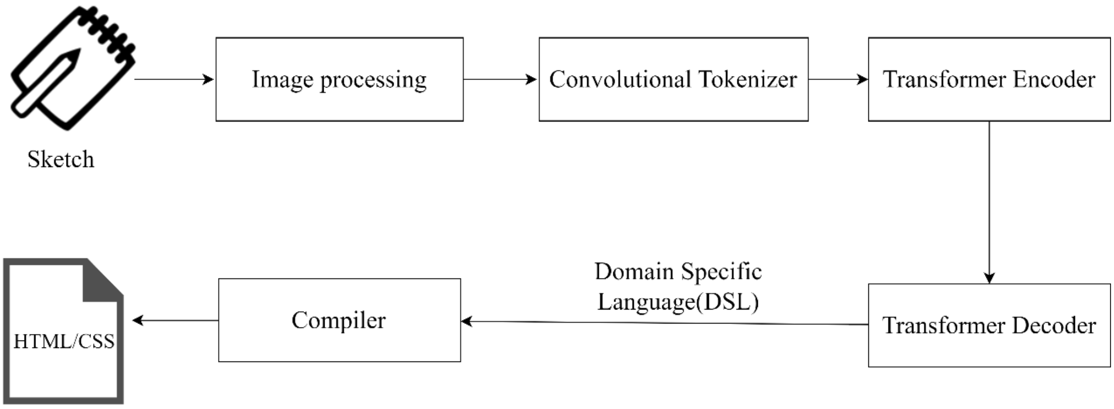


Figure 4-3: Block Diagram of Working of system

4.3 Working Principle

The input data for this project is the sketch of the UI. The sketch is transformed to the HTML code. It involves following steps:

1. DSL code extraction using Transformer model
2. Compiling DSL code to HTML code

4.3.1 Image Processing

Preprocessing is required to translate an image from a camera into an image which can be fed into the model. Due to the positioning of the camera or lighting conditions the raw image must be cleaned up before it can be processed.

The main challenges are:

i. The image may not fill the entire frame, as such the background must be removed.
ii. The paper may be skewed or rotated.
iii. The image may contain noise or alterations due to lighting.

For solving above challenges:

i. We will detect the paper in the image and crop it. As our requirements state that the medium must be white, to detect the paper we converted the image to HSV and used threshold filtering to remove all colors except white. This process reduces the background noise considerably. As the paper contains the sketch in a dark marker, these pixels will be filtered by the threshold filtering. As such, to fill the gaps left by the sketch we will apply a large median blur. We will apply canny edge detection and dilated the edge map to close small gaps between the edges. Finally, we will apply contour detection and find the largest contour with approximately four sides (as we assume the medium is four sided e.g. paper or a white board).
ii. We will use perspective warping to unwrap the four corners of the contour found and map them to an equivalent rectangle for correcting the orientation of the page.
iii. We will apply a median blur to the unwrapped cropped image to reduce noise. We then ran Canny edge detection and dilated the edge map to close small gaps between lines. The result of the post processing is an unskewed binary edge map of the sketch. This is fed into our model for processing.

4.3.1.1 Canny Edge Detection Algorithm

The process of canny edge detection algorithm can be broken down to five different steps: [7]

1. Apply Gaussian filter to smooth the image in order to remove the noise
2. Find the intensity gradients of the image
3. Apply gradient magnitude thresholding or lower bound cut-off suppression to get rid of spurious response to edge detection
4. Apply double threshold to determine potential edges
5. Track edge by hysteresis: Finalize the detection of edges by suppressing all the other edges that are weak and not connected to strong edges.

4.3.2 Convolutional Tokenizer

A convolutional tokenizer is a neural network component that processes image inputs using convolutional layers to create a sequence of tokens. A convolutional tokenizer works on visual data. It applies a series of convolutional filters across the image, capturing local patterns and features at different scales. These convolutional operations are typically followed by pooling layers, which reduce spatial dimensions while retaining important information. The result is a set of feature maps that represent high-level abstractions of the input image. These feature maps are then flattened or reshaped into a sequence of vectors, where each vector can be considered a "token" representing a specific part or feature of the image. This process bridges the gap between image data and sequence models, allowing techniques from natural language processing to be applied to visual tasks.

4.3.3 Transformer Model

Transformer model is used to generate structural description of the image in the form of DSL code. Transformer is a deep learning architecture that consists of encoder-decoder structure. The transformer processes input sequence of tokens to sequence of continuous representation. Given the output representation of the encoder, then the decoder generate output sequence of symbols. The transformer model consists of stacked self-attention and fully connected layer for both encoder and decoder.

Our project works similar to the image captioning. In the Image captioning model, input is the image and the output is the text describing the image. Whereas in our model, it will give a structural description of the interface as output in Domain Specific language.

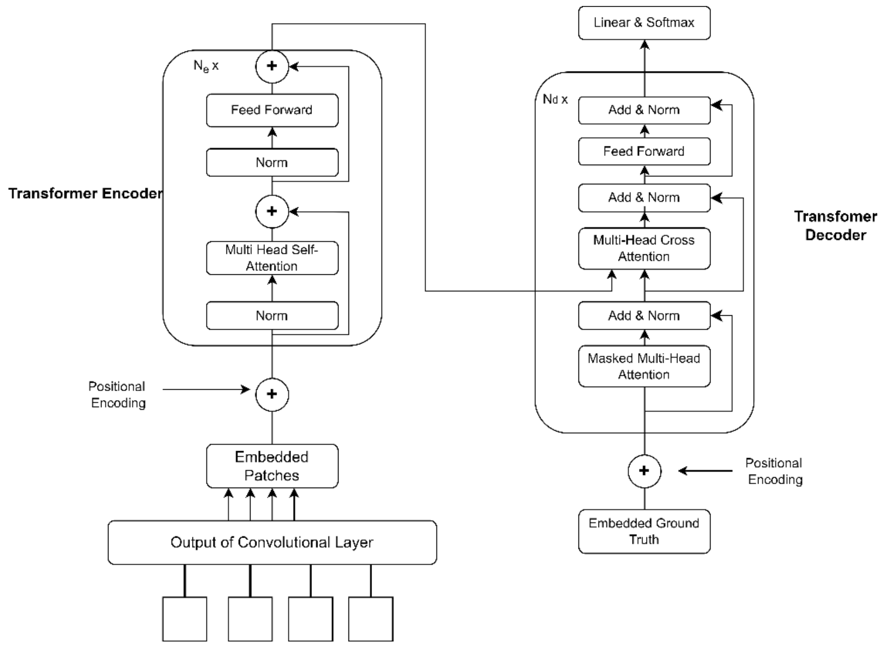


Figure 4-4: Block Diagram of Transformer model [8]

4.3.3.1 Transformer Encoder

The transformer encoder is a key component of the transformer architecture which is originally designed for NLP tasks. Now, it is widely used in various domains, including computer vision. It consists of multiple identical layers, each containing two main sub-layers: a multi-head self-attention mechanism and a position-wise fully connected feed-forward network. The self-attention mechanism allows the model to weigh the importance of different parts of the input sequence when processing each element, enabling it to capture complex dependencies regardless of their distance in the sequence. The feed-forward network processes each position independently, adding non-linearity and increasing the model's capacity to learn complex functions. Layer normalization and residual connections are employed around each sub-layer to facilitate training of deep networks.

4.3.3.2 Transformer Decoder

The transformer-based decoder consists of N_d stacked identical transformer block similar to the encoder. Each transformer decoder block is composed of a masked multi-head self-attention sublayer followed by a multi-head cross attention sublayer and a positional feed-forward sublayer sequentially. The decoder in our model takes in the encoded image embeddings and the embedded ground truth caption sequences. In addition, we add positional embedding to it for word embedding features, which is added to make use of the order of the ground truth caption sequences. The positional encodings have the same dimension as the sequence embeddings to be summed. The decoder block utilizes the last decoder block's output feature to predict the next word via a linear layer whose output dimension equals the vocabulary size.

4.3.3.3 Attention

Attention can be describe as a similarity between the query and key. They take the form:

$$\text{Attention} = \text{similarity}\ (q, k) \tag{4.1}$$

Where q represents a query and k represents a key. It's like accessing a database, where we query the database looking for the information we want. To find the similarity between queries and keys, dot product is normally used. It provide the value between 0 and 1. If they are different, obtained value is 0 otherwise it is 1. The output is computed as a weighted sum of the values, where the weight assigned to each value is computed by a compatibility function of the query with the corresponding key.

4.3.3.4 Self-Attention

In order to identify dependencies and relationships within input sequences, self-attention is a method employed in machine learning, especially in natural language processing and computer vision applications. By taking care of itself, the model is able to recognize and assess the relative value of various input sequence components. In self-attention, vector q, k, v which are actually neural networks (typically linear) have same input (q(x), k(x), v(x)), then they are self- attending.

4.3.3.5 Scaled Dot-Product Self-Attention

Here, the dimensionality of queries and keys are denoted by d_k and dimensionality of values is denoted by d_v. The scaled dot-product attention then receives these queries, keys, and input values and computes the dot-product of the queries with the keys. The attention scores are then obtained by scaling the result by the square root of d_k. After that, a collection of attention weights is obtained by feeding them into a softmax function. Lastly, a weighted multiplication operation is performed on the data using the attention weights to scale them. The complete procedure can be mathematically stated as follows, where Q, K, and V, represent the keys, values, and queries, correspondingly:

$$\text{Attention}(Q,K,V) = \text{softmax}\left(\frac{Q.K^T}{\sqrt{d_k}}\right)V \qquad (4.2)$$

4.3.3.6 Multi-Headed Self-Attention

Instead of performing a single attention function with keys, values and queries, it is more beneficial to linearly project the queries, keys and values *h* times with different, learned linear projections to d_k, d_k, and d_v dimensions, respectively. By performing the attention function in parallel on each of these projected versions of queries, keys and values, d_v –dimensional values are obtained as output. The ability of attending to input from several representation subspaces at different points is provided by multi-head attention to the model.

$$\text{MultiHead}(Q,K,V) = \text{Concat}(\text{head}_1,\ldots\ldots\text{head}_h)W^O \qquad (4.3)$$

$$\text{Where head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

4.3.3.7 Multi-Layer Perceptron

An MLP is a type of feed forward artificial neural network with multiple layers, including an input layer, one or more hidden layers, and an output layer. It is an Artificial Neural Network in which all nodes are interconnected with nodes of different layers. An input layer, an output layer, and one or more hidden layers with several neurons stacked on top of each other make up a multilayer perceptron. Additionally, neurons in a Multilayer Perceptron can employ any arbitrary activation function, unlike neurons in a Perceptron, which must have an activation function that enforces a threshold, such as ReLU or sigmoid.

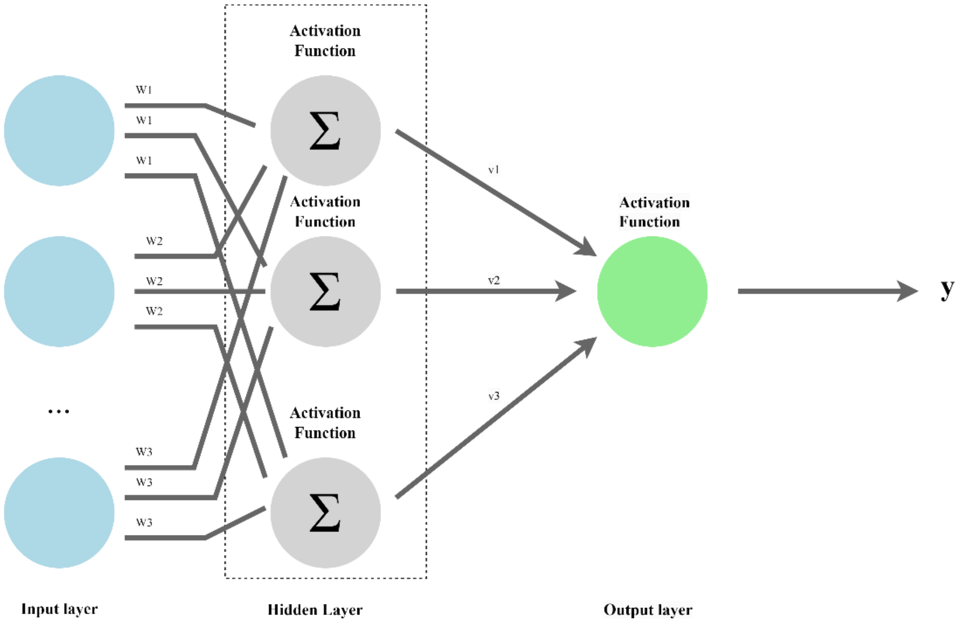


Figure 4-5: Multilayer Perceptron

4.3.3.8 Layer Normalization

In deep learning, layer normalization (LN) is a technique that helps to stabilize the training process and boost neural network performance. LN separately normalizes each layer's activations for every feature. This means that the activations are scaled and shifted to have a standard normal distribution (mean of 0 and variance of 1) after the mean and variance of the activations are determined independently for each layer.

4.3.3.9 Position Embedding

Position embedding is used to add spatial information to the image data. Since the model is actually uninformed of the token's spatial relationship, additional information expressing this relationship can be useful. Usually, this involves assigning tokens weights derived from two high-frequency sine waves, or using a learned embedding. This allows the model to understand that these tokens have a positional relationship. The standard Transformer uses sine and cosine functions of different frequencies:

$$PE(pos, 2i) = \frac{\sin\left(pos * 1000^{2i}\right)}{dmodel} \tag{4.4}$$

$$PE(pos, 2i + 1) = \frac{\cos\left(pos * 10000^{2i}\right)}{dmodel} \tag{4.5}$$

Where *pos* denotes the position and *i* denotes the dimension.

4.3.4 Domain Specific Language (DSL)

It is a specialized language designed to address specific aspect or needs of a particular language. Unlike general-purpose programming languages such as python, Java, or C++ which are intended for a wide range of applications, DSLs are optimized for a particular set of tasks within a specific domain.

We will be designing the simple lightweight Domain specific language to describe the Graphical User Interface. Elements in DSL will be categorized into different hierarchical structures on the basis of the position of the elements. Additionally to reducing the size of the search space, the DSL simplicity also reduces the size of the vocabulary (i.e. the total number of tokens supported by the DSL). To overcome this problem, compact convolution transformers is introduced. This assists to avoid overfitting and performs better than CNN for small datasets. This model is flexible to small model size and less parameter with achieving competitive results.

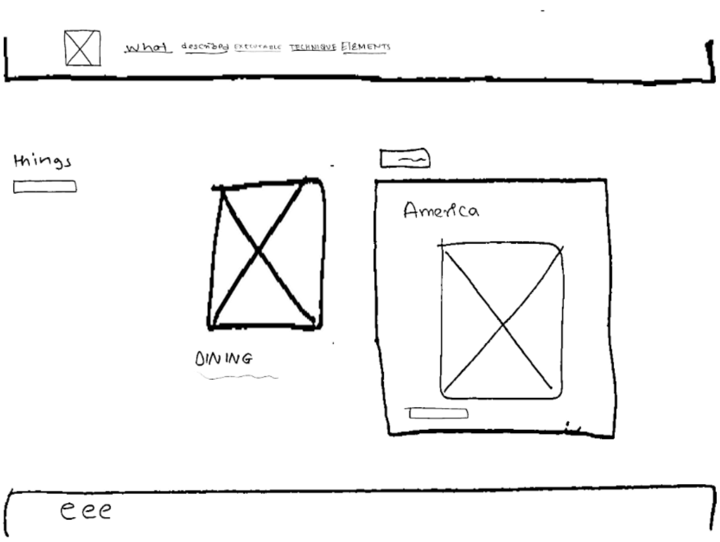


Figure 4-6: Sample input image

The sample DSL for above image:

```
header{
	flex{
		logodiv{
			image
		}
		nav{
			navlink
			navlink
			navlink
			navlink
			navlink
		}
	}
}
container{
	row{
		div-3{
			text-c
			input
		}
		div-3{
			image
			text
			paragraph
		}
		div-6{
			button
			card{
				text-c
				image
				input
			}
		}
	}
}
footer{
	text
}
```

4.3.5 Customization

Only transforming the sketch to HTML code is not enough. The sketch lacks colors, and styling for various elements. It is merely a blue print for where each component are located. The component must be styled accordingly for good look. This project will allow for some customizing by theme selection, color palate, font-selection, style selection etc. The user can select appropriate style and designs as required.

The text content will be randomly generated text which the user can change. For image, there is placeholder in which user can add link or choose from computer.

4.3.6 Compiler

Compiler is a computer program that translates computer code written in one programming language into another language. In our context, Compiler will translate the Domain Specific language that we designed into a HTML/CSS code. This compiler will read a JSON mapping file to define the conversion from DSL tokens to HTML tags. Then, it will systematically process the input, and build a hierarchical structure that mirrors the nested elements of an HTML document.

It helps us to develop a flexible architecture that accommodates various output formats beyond HTML, enabling cross-platform compatibility. Compiler can translate the Domain Specific Language into suitable format for web, Android, and iOS platforms. The compiler's ability to target multiple platforms from a single DSL source ensures that application maintain consistent functionality and appearance across different environments

4.4 Activation Function

Activation functions are mathematical equations that determine the output of a neural network model. It can also be defined as transfer function. No matter how many hidden layers are attached, every layer behaves the same way since the composite of two linear functions is also a linear function. A non-linear activation function is required to learn the difference between desired output and generated output. Thus the following activation functions have been decided to be used.

4.4.1 Softmax Function

The Softmax function is sometimes called the soft argmax function, or multi-class logistic regression since it is a generalization of logistic regression that can be used for 11 multi-class classification. The softmax function is ideally used in the output layer of the classifier where the probabilities are required to define the input images' class. A softmax function calculates the probabilities of each class which the input belongs. The softmax units in the output layer always be equal to number of classes. The probability distribution is different for different classes and the summation value of all probability distribution is 1. The softmax function 4.4 provides the probability values for each classes and class with highest probability value is consider as correct prediction.

$$\text{Softmax } \sigma(\vec{z})_{\text{I}} = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_i}} \qquad (4.6)$$

Where

$\vec{z}$ = input vector to the softmax function

z_{i} = elements of the input vector

e^{z_i} = standard exponential function applied to each element of the input vector

K = number of classes in the multi-class classifier

The Softmax function is a function that turns a vector of K real values into a vector of K real values that sum to 1. The input values can be positive, negative, zero, or greater than one, but the values are converted to values between 0 and 1 by softmax. Thus now they are interpreted as probability. Small or negative inputs are converted to a small probability value.

4.4.2 ReLU Function

ReLU is the most commonly used activation function in neural networks, whose mathematical equation is:

$$Relu\ f(x) = max\ (0,x) \qquad (4.7)$$

So,if the input is negative, the output of ReLU is 0 and for positive values, it is x.

$$Relu\ Derivative\ f'(x) = \begin{cases} 1\ for\ x > 0 \\ 0\ for\ x \leq 0 \end{cases} \qquad (4.8)$$

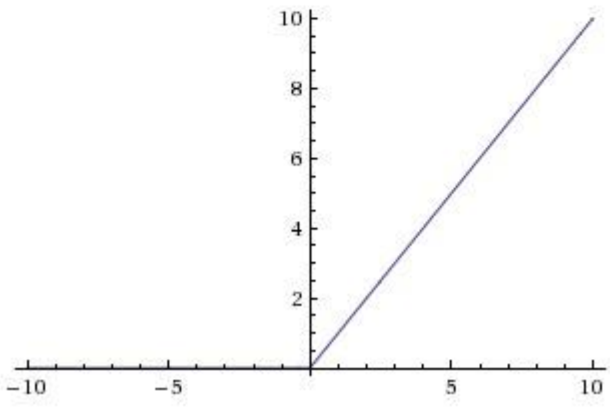


Figure 4-7: Graphical Representation of ReLU function

4.5 Loss Function and Optimizer

4.5.1 Categorical cross entropy

Categorical cross-entropy is a commonly used loss function in machine learning, particularly in classification tasks where the model predicts the probability distribution over multiple classes for each input sample. The loss is calculated by the following formula:

$$\text{Loss} = -\sum_{i=1}^{output\ size} y_i . \log \hat{y}_i \qquad (4.9)$$

Where

$\hat{y}_i$ = i^{th} scalar value in the model output

y_i = Corresponding target value

output size = the number of scalar values in the model output.

4.5.2 Adam Optimizer

It is a popular optimization algorithm used in training neural network. It stands for Adaptive Moment Estimation. It maintains two moving average vectors: the first moment m (the mean) and the second moment v (the uncentered variance) of the gradients. These moving average are used to adaptively adjust the learning rates for each parameters during training. Due to its simplicity of use, computational efficiency, low memory requirements, and invariance to diagonal rescaling of the gradients, this approach is well suited for scenarios involving high volumes of data and parameters.

4.6 Performance Metrics

4.6.1 BLEU

BLEU (bilingual evaluation understudy) is an algorithm for evaluating the quality of text which has been machine-translated from one natural language to another. This is a common metric used in machine translation tasks, which seeks to measure how closely a machine-generated text resembles what a human would have generated, given the same input. It is based on n-gram based precision. Four sub metrics are denotes as BLEUn, for n = 1, 2, 3, 4. For a candidate sentence a and a set of reference sentences b, the BLEU score is calculated as:

$$BLEU_n(a,b) = \sum_{w_n \epsilon a} \frac{\min\left(count_a(w_n).max_{j=1,\dots\dots,|b|} count_{b_j}(w_n)\right)}{\sum_{w_n \epsilon a} count_a(w_n)} \tag{4.10}$$

where w_n denotes n-gram, $count_a(w_n)$ denotes count of n-gram w_n in sentence

4.6.2 ROUGE

ROUGE (Recall-Oriented Understudy for Gisting Evaluation) is an evaluation metric specially designed for automatic summarization that can be used for machine translation. It measures the overlap of n-grams between the generated sequence and reference sequence. The ROUGE score is calculated as:

$$Recall = \frac{Overlapping\ number\ of\ n - grams}{Number\ of\ n - grams\ in\ the\ reference} \tag{4.11}$$

$$Precision = \frac{Overlapping\ number\ of\ n - grams}{Number\ of\ n - grams\ in\ the\ candidate} \tag{4.12}$$

$$Recall = \frac{2 * Precision * Recall}{Precision + Recall} \tag{4.13}$$

4.7 Flowchart

4.7.1 During Training

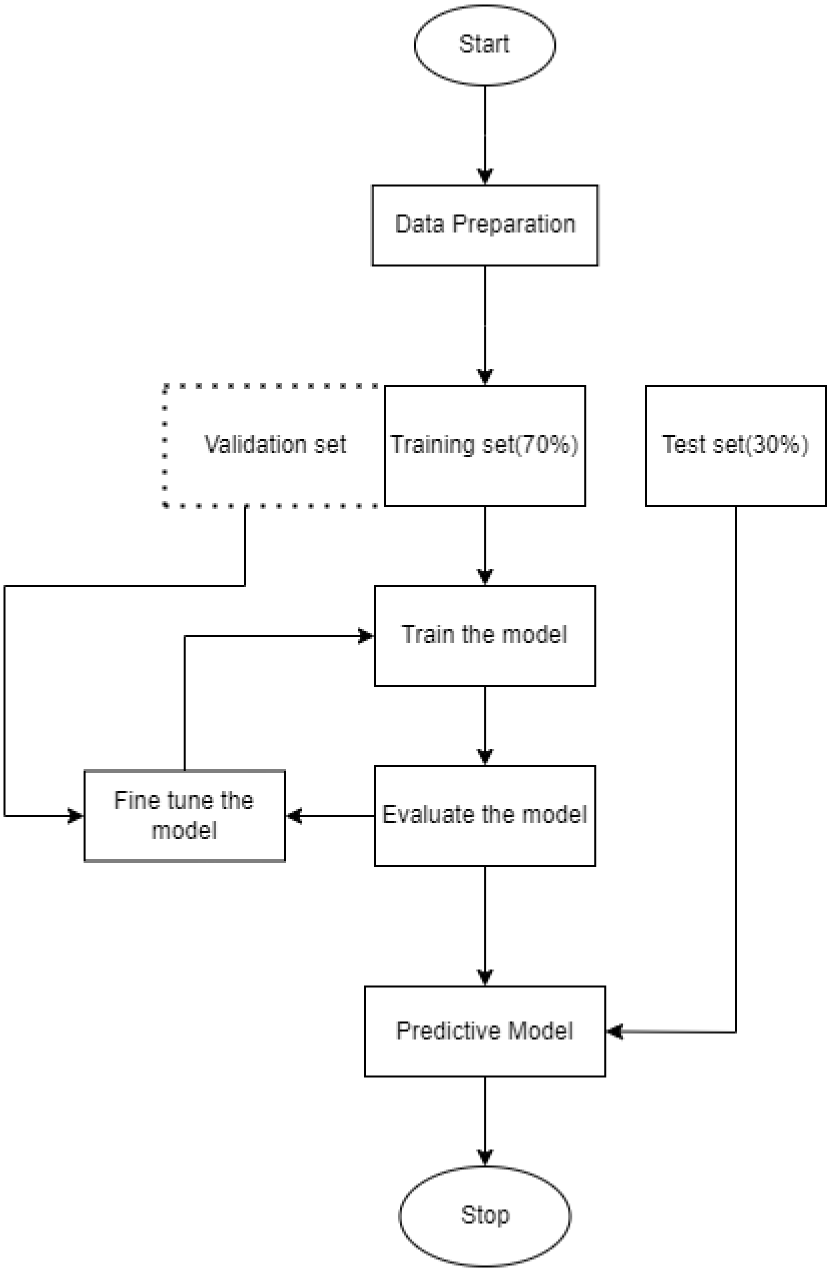


Figure 4-8: Flowchart for training

4.7.2 During Testing

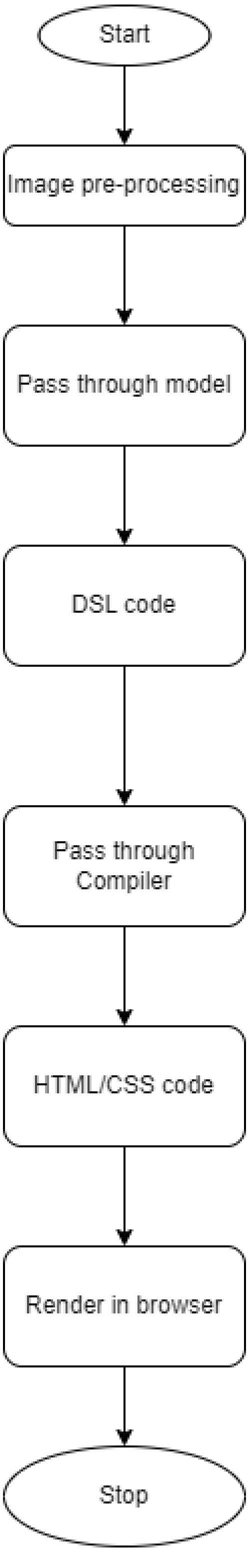


Figure 4-9: Flowchart for testing

5 IMPLEMENTATION DETAILS

5.1 Dataset Generator Implementation

Initially, we collected the multiple number of hand-drawn sketches for the each element under consideration. These sketches is combined to generate any number of data as required.

5.1.1 Steps taken in Dataset Generation

5.1.1.1 Random DSL generator

First we have to generate a DSL code which is created by using the custom made rule which is described below. Different DSL code is generated by mixing the different possible combination of the element.

5.1.1.2 Compiling the DSL Code

After generating the DSL Code, we compiled or mapped the produced DSL code to proper HTML code.

5.1.1.3 Rendering the produced HTML Code

We then rendered the HTML code in chrome browser and also use the special CSS file during rendering. Different elements are rendered with specified background color. The javascript file is also used to dynamically set random height, width and position for each element. Then, screenshot is taken of the rendered page.

5.1.1.4 Finding the outline of the different element

For detecting each element, we filter the background color of each element and by using contour detection obtain the dimensions and position of the specific element.

5.1.1.5 Placement of hand-drawn sketch

The aspect ratio of the hand drawn sketches is compared with the aspect ratio of the bounding box for each element. The sketches are sorted according to the best match aspect ratio. The random sketch among top 8 sketches is placed in the location.

5.1.2 Rules for creating DSL

We have defined the structure and rules for generating random website layouts in a Domain Specific Language (DSL). We established a hierarchical structure of webpage elements, where each key represents a component and its value lists possible child components. For instance, the 'root' element can contain 'header', 'container', and 'footer', while a 'row' can have various sizes of div elements. We also defined different ways to divide a row into columns, based on a 12-column grid system which common in responsive web design. There is also constraints and behaviors for different elements, such as the minimum and maximum number of occurrences, whether children should be generated in a specific order, and how to apply the division combinations for rows. This structure allows for the generation of diverse yet constrained layouts and also ensuring essential elements are present, allowing for variable numbers of containers and rows, providing flexibility in column divisions, and setting limits on the number of elements within components.

5.2 Model Implementation

We have designed a transformer-based architecture to convert the sketch images into the DSL code. It consists of an encoder that processes input images (sketches) and a decoder which generates the corresponding code. Our model combines convolutional neural networks (CNNs) for image processing with transformer layers for sequence modeling, which makes it well-suited for translating visual information in images into textual output, which is DSL code.

5.2.1 Input Processing:

The model takes two inputs: an image input with shape (None, 850, 600, 1), representing grayscale sketches, and a text input with shape (None, 120), representing the DSL code associated with the input sketch image.

5.2.2 Convolutional Tokenizer:

The first major component in our model is the Convolutional Tokenizer. This custom layer processes the input image through a series of convolutional and pooling operations, converting the 2D image into a sequence of feature vectors. Output shape of this component is (None, 513, 128) which means it transformed the image into 513 tokens, each with 128 dimensions.

Convolutional Layers:

The tokenizer uses a series of convolutional layers (Conv2D) with increasing numbers of filters (32, 64, 64, 128, 128). Each convolutional layer is followed by zero-padding and max-pooling operations. This structure helps in reducing the spatial dimensions of the image while increasing the depth, which is number of channels. Max-pooling layers are used after each convolution to reduce the spatial dimensions. This helps in capturing hierarchical features and reducing computational complexity. Dropout layers are inserted twice in the network after the second and fourth convolutional blocks to prevent overfitting.

The layers are as follows:

1. First convolutional layer: It contains 32 filters of kernel size of 7x7 with stride of 1 with no padding and have ReLU activation function. Then, ZeroPadding2D adds a padding of 1 around the feature map to preserve spatial dimensions. Then, the max-pooling layer of 3x3 pooling window and stride of 2 is added.

2. Second convolutional layer: It contains 64 filters of kernel size of 5x5 with stride of 1 with no padding and have ReLU activation function. Then, ZeroPadding2D adds a padding of 1 around the feature map. Then, the max-pooling layer of 3x3 pooling window and stride of 2 is added. And, dropout is applied with a rate of 0.1 to prevent overfitting.

3. Third convolutional layer: It contains 64 filters of kernel size of 3x3 with stride of 1 with no padding and have ReLU activation function. Then, ZeroPadding2D adds a padding of 1 around the feature map. Then, the max-pooling layer of 3x3 pooling window and stride of 2 is added.

4. Fourth convolutional layer: It contains 128 filters of kernel size of 3x3 with stride of 1 with no padding and have ReLU activation function. Then, ZeroPadding2D adds a padding of 1 around the feature map. Then, the max-pooling layer of 3x3 pooling window and stride of 2 is added. And, dropout is applied with a rate of 0.1.

5. Fifth convolutional layer: It contains 128 filters of kernel size of 3x3 with stride of 1 with no padding and have ReLU activation function. Then, ZeroPadding2D adds a padding of 1 around the feature map. Then, the max-pooling layer of 3x3 pooling window and stride of 2 is added.

Then the output from the convolutional tokenizer is flatten to shape of (513,128).

5.2.3 Positional Encoding:

After the convolutional tokenization, positional encoding is added to the token embeddings. We have used the sinusoidal Positional Encoding. This is crucial for the transformer architecture to understand the spatial relationships between different parts of the image. Then we combine the tokenized images features with positional information.

5.2.4 Encoder:

The encoder consists of three transformer blocks. Each block includes layer normalization, multi-head attention (self-attention), residual connection, feed-forward neural network, and another residual connection. The encoder processes the image tokens (513x128) to create a rich representation for the decoder. The output of each encoder block maintains the 513x128 dimension.

5.2.4.1 Layer normalization:

Each layer normalization has 256 parameters and normalizes the inputs across the 128 feature dimensions to improve training stability.

5.2.4.2 Multi-head attention (self-attention):

Each multi-head attention layer has 329,728 parameters. We have use 5 number of heads. The input and output dimensions are both 513x128.

5.2.4.3 Residual connection:

Residual connections add the input (513x128) to the output of the attention layer.

5.2.4.4 Feed-forward neural network:

It consist of two dense layers. The first dense layer of size 513x128 which consist of 16,512 parameters. Then, dropout layer is added. After that, second dense layer of same size with same number of parameters are added. Then, another dropout is added.

5.2.5 Decoder:

The decoder is more complex, consisting of four transformer blocks. It takes the embedding of the input sequence (120x128) and processes it while attending to the encoder's output (513x128).

Each block in the decoder includes:

5.2.5.1 Self-attention layer:

This layer has 329,728 numbers of parameters with input and output dimension of 120x128. This layer mask the future token ensuring that the prediction for a given token is done only using previous tokens.

5.2.5.2 Cross-attention layer:

It has 329,728 numbers of parameters and attends to the encoder output (513x128) while processing the 120x128 decoder sequence.

5.2.5.3 Feed-forward network:

It consist of two dense layer of size 120x128 consisting 16,512 numbers of parameters and each dense layer is followed by the dropout layer.

5.2.5.4 Layer normalization and residual connections:

Each layer normalization has 256 parameters, normalizing across the 128 feature dimensions. Residual connections add the input (120x128) to the output of each sub-layer.

5.2.6 Attention Mechanisms:

The model uses several multi-head attention layers. In the encoder, these are self-attention layers, allowing each token to attend to all other tokens in the image representation. In the decoder, there are both self-attention layers for processing the output sequence and cross-attention layers for attending to the encoder's output.

5.2.7 Masking:

The model implements causal masking in the decoder's self-attention layers. It ensures that each position in the output sequence can only attend to previous positions which is crucial for autoregressive generation.

5.2.8 Output Layer:

The final layer is a dense layer with 31 units, corresponding to the vocabulary size of the output tokens (minus one, accounting for a start token). This layer produces the logits for each token in the vocabulary at each position in the output sequence.

5.3 Model Complexity:

The model has a total of 3,488,447 trainable parameters, which is relatively compact for a task of this complexity. This model have a balance between model capacity and computational efficiency.

5.4 Vocabulary of Model

The DSL used in this project consists of total 32 different symbols and words. <Start>, <End> and <Pad> are special symbols. The textual representation is converted to vector form. For each words or symbols, unique number from 0 to 31 is assigned. <Pad> symbol is assigned 0 which is added to make input uniform. <Start> symbol is assigned 31. These vector information is passed as the input to model. The embedding layer converts one dimensional vector to vector of size corresponding to projection dimensions.

5.5 Training details:

The model is trained on paired sketch-code data using supervised learning. It employs teacher forcing by feeding ground truth tokens to the decoder during training. The objective is to minimize cross-entropy loss between predicted and actual tokens. End-to-end training optimizes both convolutional and transformer layers. Data augmentation is used to improve training stability and model generalization.

5.6 Testing details:

During testing, the model generates code iteratively, one token at a time. It processes the input sketch and predicts tokens sequentially, using previously generated tokens as input for subsequent predictions. Performance is evaluated using metrics like BLEU and ROUGE score. Human evaluation is also necessary to assess the generated code's quality and input image quality.

6 RESULTS AND ANALYSIS

6.1 Model Training

The model is trained for 8 epochs taking batch size of 32. The training set consists of 8951 samples and testing set consists of 2238 samples. All these samples are generated using random dataset generator.

6.1.1 Loss vs Epoch Graph

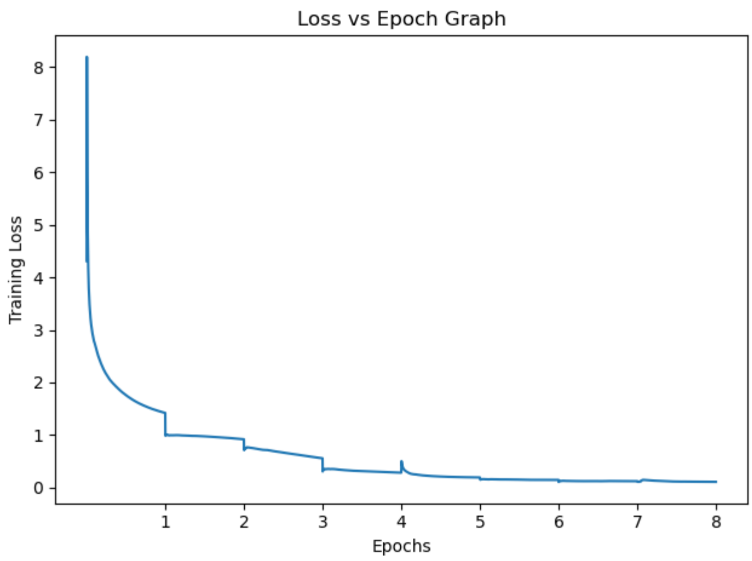


Figure 6-1: Loss vs Epoch graph for training time

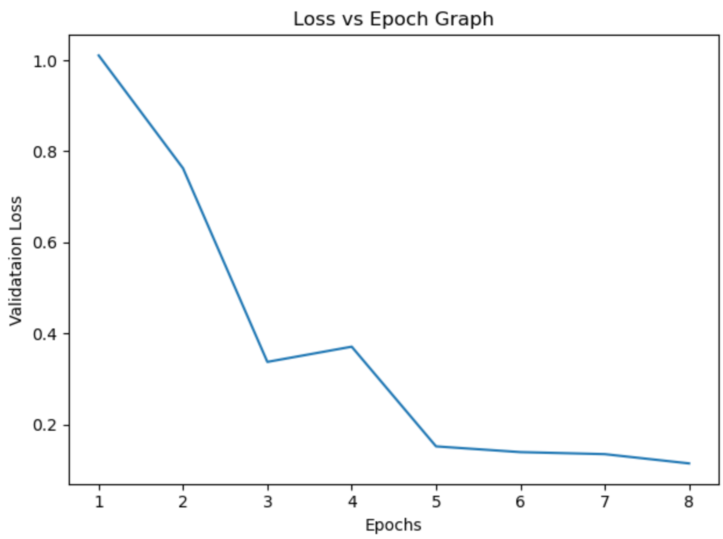


Figure 6-2: Loss vs Epoch graph for validation time

Initially, the loss for the model is very high. In first epoch, the loss gradually decreased and reached around 1.5. In the following epochs, the loss decreased steadily and reached to 0.1 in the final epoch. The loss in validation set also remain comparable to that of training set throughout the training which confirms model didn't over fit for the training data.

6.1.2 Evaluation

After training for 8 epochs, the model is evaluated on the test data. The BLEU and ROUGE evaluation metrics are used to evaluate the model.

6.1.2.1 BLEU Score

The average BLEU score for n-grams 1, 2, 5 and 10 is as follows:

- 1-grams BLEU Score: 0.9544114683715649
- 2-grams BLEU Score: 0.9378854888760932
- 5-grams BLEU Score: 0.8867411989388778
- 10-grams BLEU Score: 0.7982317463855622

The above observation shows that the model performed well for the given testing data. The average BLEU-1 score of 0.95 indicate that the generated DSL closely match the reference DSL. The BLEU score decreased for increasing n grams but the score is still good which indicates model is good predicting short sequences. The BLEU-10 score is also good (0.79). So, it also predicted longer sequence.

The individual BLEU-10 score in sorted order is plotted in the graph. It can be seen that the score is higher than 0.6 for most of the samples. There are plenty of samples with bleu scores above 0.9. So, the model performed well for the testing data.

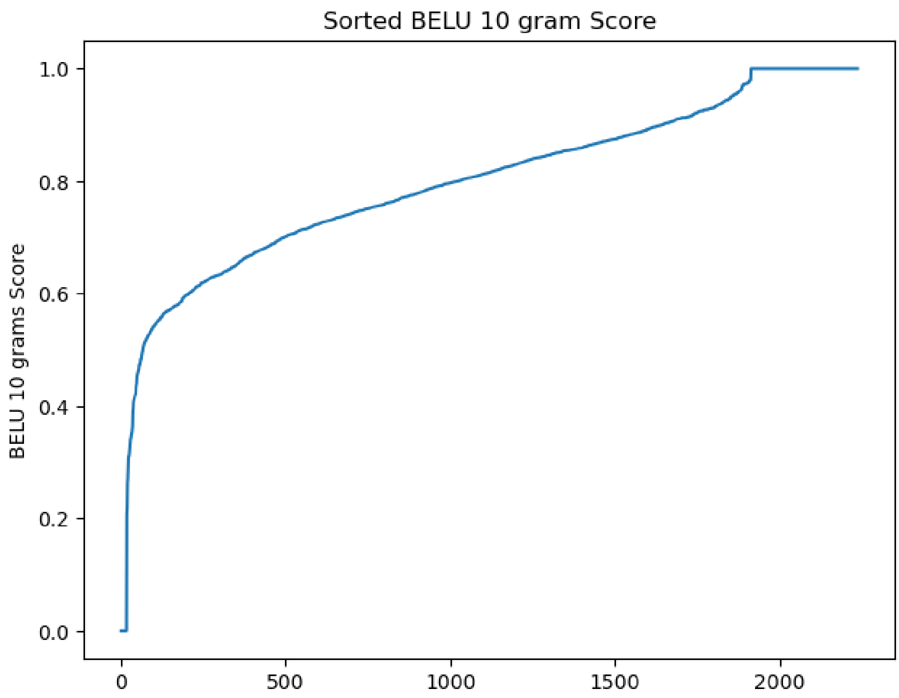


Figure 6-3: BLEU-10 Scores in sorted order

6.1.2.2 ROUGE Score

The average ROUGE Scores are tabulated below:

Table 6-1: ROUGE Scores for test data

Metric	Precision	Recall	F1-Score
ROUGE-1	0.9481	0.9846	0.9652
ROUGE-5	0.7806	0.8112	0.7948
ROUGE-L	0.9428	0.9791	0.9598

From above table, ROUGE-1 has high F1-score of 0.96. This indicates the predicted tokens matches 96% with the reference tokens. The ROUGE-5 F1-score is 0.79. It has decreased drastically but it is still a good score. The high recall of 0.97 for ROUGE-L indicates that the generated DSL captures most of the longest common subsequences from the reference. This indicates the generator maintains correct sequence and structure of DSL. The graph below shows that the F1-score for almost all samples is above 0.9 which is good performance.

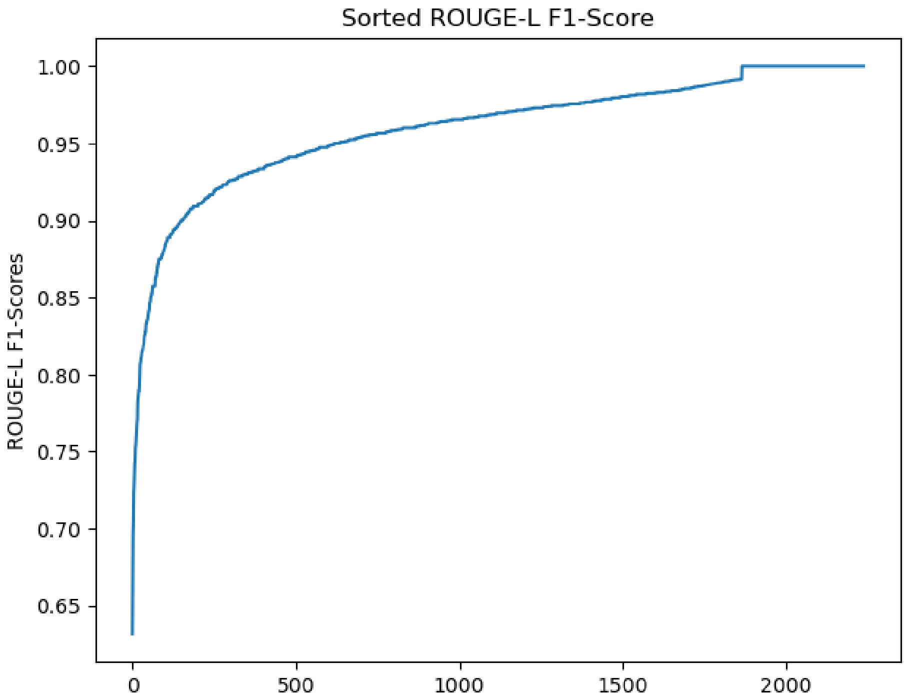


Figure 6-4: ROUGE-L F1-Scores in sorted order

7 REMAINING TASK

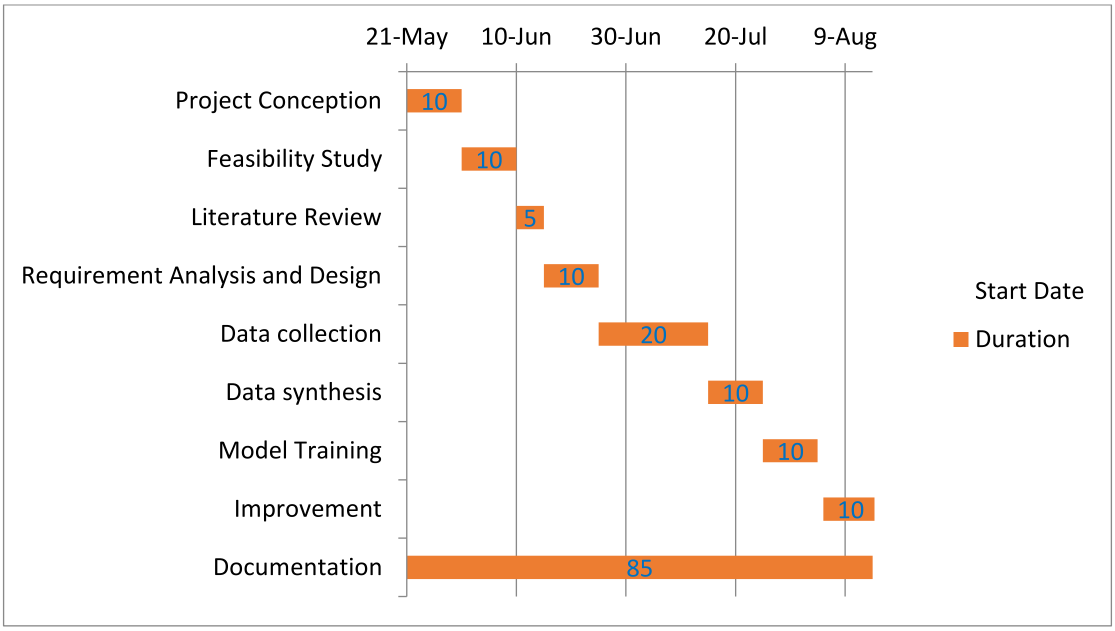
The remaining tasks are:

- Optimizing Dataset generation process to match real world data.
- Collect more Hand-drawn data to generalize better to real-world input.
- Fine-tuning the model for better performance.
- Improving the architecture of model.
- Improve the preprocessing steps for hand-drawn sketches.
- Creation of intuitive User interface for customization and styling of generated code.

8 APPENDICES

APPENDIX A: Project Schedule

Table 8-1: Gantt Chart



APPENDIX B: DSL Generation Rules

1. Dictionary for Child Elements

```
graph={
    'root':['header','container','footer'],
    'header':['flex'],
    'nav':['navlink'],
    'logodiv':['image','text'],
    'container':['row'],
    'row':['div-3','div-6','div-9','div-12'],
    'div-3':['text','paragraph','image','card','input','button',],
    'div-6':['text','paragraph','image','card','carousel','input','button','flex'],
    'div-9':['text','paragraph','image','card','carousel','input','table','button','flex'],
    'div-12':['text','paragraph','image','card','carousel','input','table','button','flex'],
    'flex':['text','button'],
    'card':['text','paragraph','image','input','button','flex'],
    'footer':['text']
}
```

2. Rules for Child Elements

```
rules = {
    'root': {'inOrder': True},
    'logodiv': {'inOrder': True},
    'header':{'min':1,'max':1},
    'container': {'min': 1, 'max':3},
    'row': {'combinations':True,'0': divCombinations,'1':divCombinations2,'proba':0.9},
    'div-3': {'min': 2, 'max': 4},
    'div-6': {'min': 2, 'max': 4},
    'div-9': {'min': 2, 'max': 4},
    'div-12': {'min': 2, 'max':4},
    'card': {'min': 2, 'max': 3},
    'footer': {'min': 1, 'max': 1},
    'nav':{'min': 1, 'max': 5},
    'flex':{'min':2,'max':4},
}
```

3. DSL to HTML mappings

```
{
  "opening-tag": "{",
  "closing-tag": "}",
  "body": "\r\n {} <style>\nmax-width: 900px;\nmax-height: 300px
!important;\n</style>\n",
  "root": "\r\n<div class=\"root\">{}</div>\r\n",
  "header": "\r\n<header class=\"header\">{}</header>\r\n",
  "nav": "\r\n<nav class=\"nav\">{}</nav>\r\n",
  "navlink": "\r\n<a href=\"#\" class=\"navlink\">[]</a>\r\n",
  "logodiv": "\r\n<div class=\"logodiv\">{}</div>\r\n",
  "container": "\r\n<div class=\"container\">{}</div>\r\n",
  "row": "\r\n<div class=\"row\">{}</div>\r\n",
  "div-3": "\r\n<div class=\"div-3\">{}</div>\r\n",
  "div-6": "\r\n<div class=\"div-6\">{}</div>\r\n",
  "div-9": "\r\n<div class=\"div-9\">{}</div>\r\n",
  "div-12": "\r\n<div class=\"div-12\">{}</div>\r\n",
  "flex": "\r\n<div class=\"flex\">{}</div>\r\n",
  "flex-sb": "\r\n<div class=\"flex-sb\">{}</div>\r\n",
  "flex-c": "\r\n<div class=\"flex-c\">{}</div>\r\n",
  "flex-r": "\r\n<div class=\"flex-r\">{}</div>\r\n",
  "text": "\r\n<div class=\"text\">{}</div>\r\n",
  "text-c": "\r\n<div class=\"text-c\">{}</div>\r\n",
  "text-r": "\r\n<div class=\"text-r\">{}</div>\r\n",
  "paragraph": "\r\n<p class=\"paragraph\">{}</p>\r\n",
  "image": "\r\n<img src=\"placeholder.jpg\"  class=\"image\">\r\n",
  "card": "\r\n<div class=\"card\">{}</div>\r\n",
  "input": "\r\n<input type=\"text\" class=\"input\" placeholder=\"Enter text\">\r\n",
  "button": "\r\n<button class=\"button\">{}</button>\r\n",
  "button-c": "\r\n<button class=\"button-c\">{}</button>\r\n",
  "button-r": "\r\n<button class=\"button-r\">{}</button>\r\n",
```

```
  "footer": "\r\n<footer class=\"footer\">{}</div>\r\n</footer>\r\n",
    "table":"\r\n<table class=\"table\">{}</table>\r\n",
  "carousel":"\r\n<div class=\"carousel\">{}</div>\r\n"
}
```

4. Color Mapping for each elements

```
{
    'paragraph':np.array([193,188,192]),
    'text':np.array([255,255,255]),
    'button':np.array([19,247,47]),
    'navlink':np.array([255,0,0]),
    'carousel':np.array([255,165,0]),
    'table':np.array([165,42,42]),
    'input':np.array([255,159,252]),
    'image':np.array([0,0,255]),
    'header':np.array([0,255,255]),
    'footer':np.array([154,128,235]),
    'card':np.array([0,128,128]),
 }
```

Appendix C: Sample Generated Datasets

Sample 1:

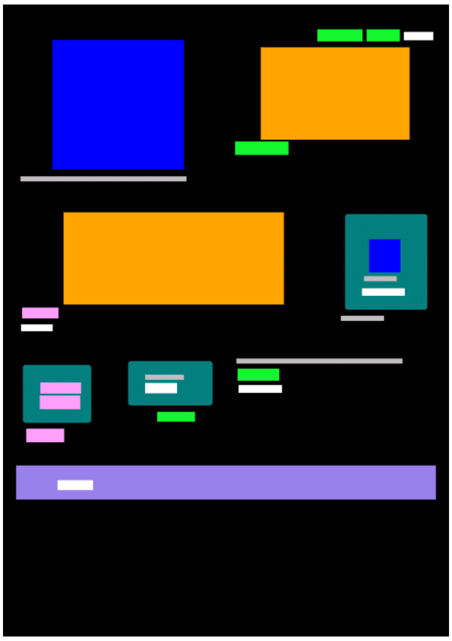
DSL Code:

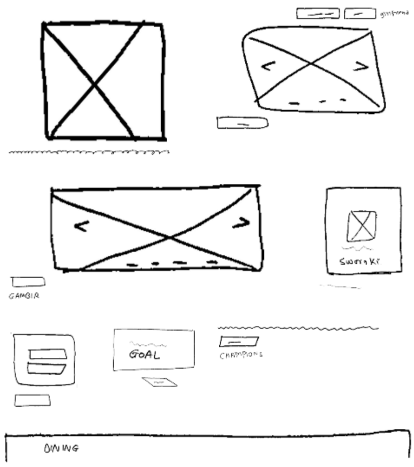
```
container{
        row{
                div-6{
                        image
                        paragraph
                }
                div-6{
                        flex-r{
                                button
                                button
                                text
                        }
                        carousel
                        button
```

```
        }
}
row{
        div-9{
                carousel
                input
                text
        }
        div-3{
                card{
                        image
                        paragraph
                        text-c
                }
                paragraph
        }
}
row{
        div-3{
                card{
                        input
                        input
                }
                input
        }
        div-3{
                card{
                        paragraph
                        text-c
                }
                button-c
        }
        div-6{
                paragraph
```

```
                        button
                        text
                }
        }
}
footer{
        text-c
}
```

Rendered HTML: *Sketch:*





Sample 2:

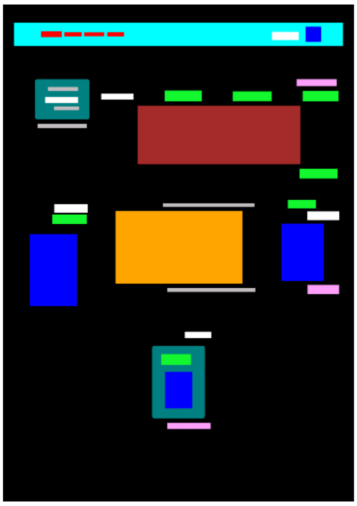
DSL Code:

```
header{
        flex-sb{
                logodiv{
                        image
                        text
                }
                nav{
                        navlink
                        navlink
                        navlink
                        navlink
                }
        }
```

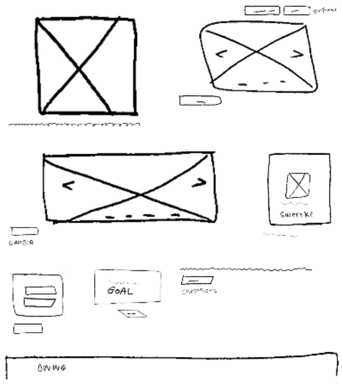
```
}
container{
    row{
        div-9{
            input
            flex-sb{
                button
                button
                button
                text
            }
            table
            button
        }
        div-3{
            card{
                paragraph
                text-c
                paragraph
            }
            paragraph
        }
    }
    row{
        div-3{
            button-c
            text
            image
            input
        }
        div-6{
            paragraph
            carousel
            paragraph
        }
        div-3{
            text
            button
            image
        }
    }
    row{
        div-3{
            text-c
            card{
                button
                image
            }
            input
        }
```

```
        }
}
```

Rendered HTML:



Sketch:



REFERENCES

[1] Tony Beltramelli,"Pix2code: Generating Code from a Graphical User Interface Screenshot," 2017. [Online]. Available: https://www.researchgate.net/publication/325920827_pix2code_Generating_Code_from_a_Graphical_User_Interface_Screenshot. [Accessed: June 2024].

[2] Sarah Suleri, Vinoth Pandian,Svetlana Shiskovets,Matthias Jarke,"Eve: A sketch-based Software Prototyping Workbench," 2019. Available: https://www.researchgate.net/publication/332777261_Eve_A_Sketch-based_Software_Prototyping_Workbench. [Accessed: June 2024].

[3] Biniam Behailu Adefris, Ayalew Belay Habtie, Yesuneh Getachew Taye, "Automatic Code Generation from Low Fidelity Graphical User Interface Sketches Using Deep Learning," 2020. [Online]. Available: https://ieeexplore.ieee.org/document/9971204. [Accessed: June 2024].

[4] Daniel Baulé, Christiane Gresse von Wangenheim, Aldo von Wangenheim, Jean C. R. Hauck and Edson C. Vargas Júnior, "Automatic Code Generation from Sketches of Mobile Applications in End-User Development Using Deep Learning," [Online]. Available: https://www.researchgate.net/publication/349963791_Automatic_code_generation_from_sketches_of_mobile_applications_in_enduser_development_using_Deep_Learning . [Accessed: June 2024].

[5] Jia Li, Yongmin Li, Ge Li, Zhi Jin, Yiyang Hao, and Xing Hu, "STC (Sketch To Code) - An Enhanced HTML & CSS Autocode Generator from Handwritten Text and Image Using Deep Learning," IEEE, 2024. [Online]. Available: https://ieeexplore.ieee.org/document/10537336. [Accessed: June 2024].

[6] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale," [Online]. Available: https://arxiv.org/abs/2010.11929. [Accessed: June 2024].

[7] "Canny edge detector," Wikipedia, The Free Encyclopedia. [Online]. Available: https://en.wikipedia.org/wiki/Canny_edge_detector. [Accessed: 17-Jun-2024].

[8] Q. Xin, Y. Zhang and B. Tan, "Image Captioning with Vision/Text Transformers," [Online]. Available: https://qi-xin.github.io/image%20caption%20generation.pdf. [Accessed: June 2024].