CoGen:

Creation of Reusable UI Components in Figma via Textual Commands

Specifications document

Table of Contents

1. Introduction to the Project

UI design involves the whole process of designing for relevant projects based on the requirements; from the smallest element to the final design screens. Due to the importance of UI design, authors have come up with many systems to reduce the time and burden when building UI designs. Currently authors and organisations use ML and AI to automate the process as much as possible.

The following shows an overview of the most related projects and tools available that align closely with the idea of this project.

Projects based on research papers	**Tools based on online sites or plugins**
GUIGAN- UI designs generated as images (Zhao et al., 2021): This project focuses on the creation of images of UI designs as a means of inspiration to designers and developers.	**Uizard:** Creation of wireframes and UI designs from descriptions. Link: https://uizard.io/solutions/
Creating UI mockups from descriptions (Huang et al., 2021): Creation of LoFi UI mockups from natural language descriptions using transformer based deep learning models.	**Figma plugin- Anima:** Generation of HTML, React and Vue code from designs. Link: https://www.animaapp.com/figma
Akin- UI wireframes generated for UI design patterns (Gajjar et al., 2021): The project focuses on generating wireframes by allowing a designer to choose a design pattern.	**Galileo AI:** Creation of UI designs by using generative AI and analysing prompts. Link: https://www.usegalileo.ai/explore

2. Importance of the Project

This project idea was formulated around the Atomic design concept. This is a concept that encompasses 5 stages;

- Atoms
- Molecules
- Organisms
- Templates
- Pages

Atoms are the building blocks of a UI design, that represent the initial design of the simple component or UI elements like buttons, input fields and so on. Molecules are a combination of atoms like navigation bars. The end result of any UI design would be a design project that consists of Pages.

2.1 Importance of focusing on Atoms

The projects and tools that were listed above fully focused on ‘Templates’ and ‘Pages. This disrupts the flow of this concept as each stage is built on the result of the previous stage. Therefore there is a lack of focus on the initial stage of this concept. This is disadvantageous in terms of complex and large UI design projects, which causes discrepancies in the flow while scaling up. For instance, without a strong foundation of atomic components like buttons, labels, and input fields, the higher-level structures such as forms and navigation bars can become inconsistent and harder to manage. As another example, if atoms are not well-defined, changes in design or functionality require extensive rework across all related templates and pages.
Thus this project was built with the idea of building the Atoms of UI design.

2.2 Contribution to the body of knowledge

- Limited number of components: CoGen only works with six components (’Button’, ’Input field’, ’Icon button’, ’Menu list’, ’List items’, ’Label’), and four styles (`Basic`, `Trendy`, `Playful`, `Professional`). This is mainly due to resource limitations and the lack of publicly available Figma datasets.
- Creation of component sets with variants: Figma allows for the creation of component sets with multiple variants of a component within. However, at present, CoGen is able to create only a single component which includes one variant with a single prompt. This again is mostly due to time and resource constraints relating to the training of models. In future, CoGen could be expanded to include more variants within a component set.
- Lack of a vast dataset: The T5 model requires more input with nested components to understand the relationship between the node type (frame, text, icon) and the properties of each node. This will result in better component generation within Figma as well.
- Lack of complex models: The model’s ability to understand diverse prompts is very limited. Access to paid models like GPT-4 would make this process faster and efficient as they are more refined in terms of understanding prompts and generating content. Moreover, this would aid in creating nested components as well.
- Lack of complex integrations: The ability to integrate with sites such as Google Fonts to include more font options, or connecting to an icon library set, will be a huge added advantage.
-

3. Use of JSONs and Figma

3.1 Why use Figma

CoGen was built with the intention of being used on Figma. Figma is a UI design tool used by designers. This tool facilitates the development of custom plugins. There are two reasons as to why this project was created to optimise and use in Figma.

3.1.1 Reusability and editing

Unlike the tools and the projects from existing systems, this project focuses on the UI elements or components. As there is a need for reusability and further editing there's no reason to create these components on a website or web app. Thus the most feasible approach would be to connect the output to a design tool.
As a result Figma was selected as it is the most used software for UI design.

3.1.2 Dataset

Currently the available dataset that is used for most projects in most literatures is the Rico dataset. This dataset wasn't chosen as it was mainly focused on mobile screens, lacked the needed component details such as buttons and also focused mainly on the positioning of UI design elements . This was because the dataset also consists not only of JSON but also of images. These JSON consist of data such as the ancestors and resource IDs which are also relevant to the whole UI design rather than individual components (Manandhar, Jin and Collomosse, 2021).
As a result a dataset was created using the free and open source templates available in the Figma community.
These project file links consist of all the components that were used:
https://www.figma.com/design/vyKyzo7v71QHiJPbi3vlmB/Components---1?t=uKVS3OCFfq73JLdQ-1

https://www.figma.com/design/fcnUCGmVtluR6Q4qLJA3Qo/Components---2?t=uKVS3OCFfq73JLdQ-1

3.2 Why use JSON?

When extracting details from the dataset in Figma, the Figma API returns them in JSON form. Therefore it is essential to manipulate JSON throughout the project, so that the model gives an output JSON that can be post processed to be used on Figma using methods defined in the project files.

3.3 JSON extraction

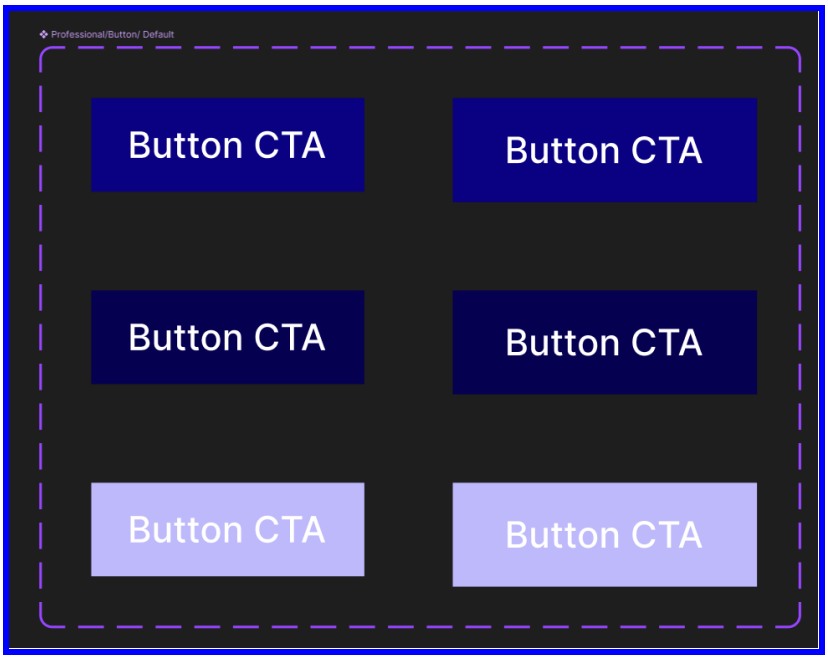
The JSON extracted consist of the following details;

- Component types - Button, label, icon button, input field, list item, menu item
- Component styles - Professional, Trendy, playful, basic
- Component subtype- Dark, Light, Default
- Variant properties like the colour, stroke details (stroke weight, stroke colours), font detail (characters, font family, font weight, and font size), padding, width , height, x and y values, variant details with the key value pairs of the variants.

Figma deals with ‘Component_sets’ and ‘Variants’. A component set would include one or more variants. For example when a button component exists, all its states (default, hover, disabled) will all be shown as individual variants inside the component set.

Therefore the extracted data will include the variant details. This is because the dataset is from Figma and it is important to focus on the individual variants first before upscaling to component sets.

The following is an example of a component set;



4. Implementation of the problem

The project implementation was carried out in 2 major stages.

1- Generation of text descriptions (Prompt generation)

2- Generation of JSON (Figma component JSON generation)

4.1 Generation of text descriptions (Prompt generation)

This stage mainly aided the completion of data needed for the next stage. After collecting the JSON data from the Figma dataset, synthetic data generation was done using a rule based approach in order to generate the descriptions. This was because the datasets' major flaw was the lack of descriptions or prompts relevant to the data. As a result, descriptions were generated and paired with their corresponding JSON.

These descriptions can be found in https://github.com/KI-5/CoGen/blob/main/synthetic_descriptions.jsonl.

The following image is a snippet of synthetic descriptions for a button component JSON:

```
Create a Button with a style of Professional and State of Default, Size of Small.
Generate a Professional Button with border radius default border radius and State of Default, Size of
Small.
Generate a Professional Button with State of Default, Size of Small.
Create a Professional Button with a border radius of default border radius.
Generate a Button with State of Default, Size of Small.
Generate a Professional Button in custom color color.
Create a Button styled Professional with custom color background and State of Default, Size of Small.
Produce a Button with State of Default, Size of Small and border radius default border radius.
Construct a Professional Button having a border radius of default border radius and State of Default, Size
of Small.
```

Following this step, the authors use an approach to generate descriptions for any input JSON, based on this dataset. Here a **LSTM based Seq2Seq model, GRU based Seq2Seq model and a fine tuned T5 model were used**.

The LSTM Seq2Seq baseline model is a standard custom implementation followed by common PyTorch tutorials and literature on sequence-to-sequence architectures. It consists of a two-layer LSTM encoder-decoder with embedding layers and fully connected output projection, trained with cross-entropy loss and Adam optimizer.

The GRU Seq2Seq model is a custom implementation consisting of an embedding layer followed by a three-layer GRU encoder and decoder. The decoder outputs are projected to token logits via a linear layer. Training uses weighted cross-entropy loss with class weights calculated based on component type frequencies to address class imbalance.

We fine-tuned a pretrained T5-small model to generate UI component descriptions from flattened JSON representations. The model was trained with the Hugging Face Trainer API, using 256 token max length inputs and outputs, gradient accumulation, and mixed precision. The input JSON strings were tokenized with T5Tokenizer, and the model was optimized using AdamW. Training ran for 12 epochs with evaluation and checkpointing each epoch.

The results showed that the T5 model produced the best prompts. The descriptions were then taken and paired with the related JSON.

4.2 Generation of JSON (Figma component JSON generation)

Here a variation of T5-based models were used: a **simple T5 model and a complex T5 model with BERT embeddings and cross attention**.
Another important factor to note is that two variations of the JSON data were used to train both the models: **a flat-structured simple JSON and a multi-layered nested JSON structure.** This was purely for research purposes to identify the competency of each model to work with nested and simple hierarchies.
The links for both these datasets can be found in https://github.com/KI-5/CoGen/blob/main/simple_components_dataset.jsonl and https://github.com/KI-5/CoGen/blob/main/nested_dataset_snippet.jsonl respectively.
The major challenge in this stage was the generation of the whole JSON based on the input description string. The most important aspect was that the keys of the JSON had to be present. For their values, the most important factors that had to be derived from the description were the 'component_name' and the 'style' of the component.
Based on the results the simple JSON gave the best results, while the simple T5 model gave the best flat-structured simple JSON outputs.

5. Results

The subsets relevant for both components were created by random sampling without replacement from the full dataset. We did not apply stratification but made sure that the samples reflected the overall data distribution.

5.1 Generation of prompts

To evaluate our T5 model, we used five random subsets of our dataset (100, 200, 300, 400, and 500 samples). For each subset, we calculated accuracy, precision, recall, and F1-score independently. The final reported scores are averages across all five subsets.

The progressive subsets were used to evaluate how model performance scales with data size, to look into learning behavior and output. Due to computational constraints, full dataset evaluation was limited.

The evaluation dataset was drawn from our main dataset, for consistency.

Model	Accuracy %	F1%	Recall%	Precision%
LSTM based	56.8	52.58	56.8	55.18
GRU based	96.2	96.08	96.2	96.52
T5	98.0	98.2	98.0	98.44

To assess this the main aspects considered were the ‘component_name’, ‘subtype’, ‘style’ and other attributes like ‘effects’,’stroke-weight’ and ‘border_radius’.
The following were the main aspects considered when generating the descriptions;

- ‘component_name’: Identifying the name of the component from the JSON correctly.
- ‘subtype’: This is needed to identify the additional features of the components (Dark theme, light theme)
- ‘style’: This is essential as this reflects the style of the component (Professional, playful, trendy).

Since this step focuses on the creation of descriptions that completely align with an input JSON, it is important that the accuracy in terms of the component name, is high. Therefore, the T5 model is considered as the best option.
The second priority goes to the BLEU scores and ROUGE scores.

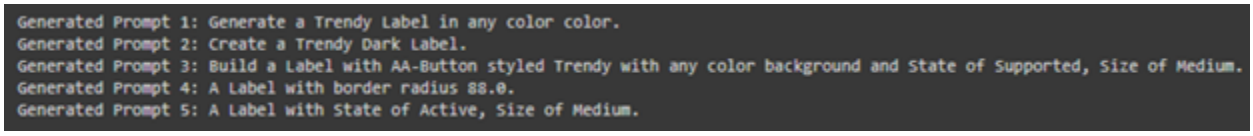
Model	**BLEU score**	**ROUGE-1**	**ROUGE-2**	**ROUGE-L**
LSTM based	0.2515	0.8758	0.7657	0.8752
GRU based	0.2402	0.9738	0.9581	0.9734
T5	0.2668	0.5486	0.3967	0.5293

Based on these it can be observed that T5 has a score of 0.2668, indicating that the model is learning to generate diverse descriptions that are different from the dataset while also ensuring that the component name (and other elements) is (or are) extracted accurately in the descriptions.

The following is a JSON structure from the dataset:

```
sample_json = {
    "variant_properties": {
      "component_name": "Label",
      "style": "Trendy",
      "subtype": "Dark",
      "variant_details": {
        "State": ["Enabled", "Focused"],
        "Size": ["Medium"]
      },
      "borderRadius": "8.0",
      "strokeWeight": "2.0",
      "effects": [
        {"type": "SHADOW", "color": "rgba(0,0,0,0.2)"}
      ]
  }
}
```

The relevant prompts generated are shown in the following image:



As seen from the results, T5 was more successful in terms of generating descriptions with relevance to the JSON, in terms of understanding the aspects and attributes mentioned. As a result T5 was chosen to generate the descriptions. The following is a snippet of the dataset consisting of the JSON and prompt pairs. It consists of the prompts generated from the T5 model along with their relevant JSON:

```
        "json": "{\"variant_properties\": {\"color\": \"rgba(235, 235, 245, 1.0)\", \"strokes\": [],
\"strokeWeight\": 0.0, \"text\": \"\\udbc0\\ude84\", \"textColor\": \"rgba(235, 235, 245, 1.0)\",
\"borderRadius\": 500.0, \"fontFamily\": \"SF Pro\", \"fontWeight\": 400, \"fontSize\": 15.0, \"effects\":
[], \"padding\": 0, \"width\": 34.0, \"height\": 34.0, \"x\": -3918.0, \"y\": 998.0, \"hasIcon\": false,
\"style\": \"Trendy\", \"component_name\": \"Button\", \"subtype\": \"With Icons\", \"variant_details\":
{\"Mode\": [\"Dark\"], \"Size\": [\"Medium\"], \"Style\": [\"Filled\"], \"State\": [\"Disabled\"], \"On
material\": [\"True\"], \"Label Type\": [\"Symbol\"]}}}",
        "description": "Generate a Button with Mode of Dark, Size of Medium, Style of Filled, State of
Disabled, On material of True, Label Type of Symbol."
    },
    {
        "json": "{\"variant_properties\": {\"color\": \"rgba(235, 235, 245, 1.0)\", \"strokes\": [],
\"strokeWeight\": 0.0, \"text\": \"\\udbc0\\ude84\", \"textColor\": \"rgba(235, 235, 245, 1.0)\",
\"borderRadius\": 500.0, \"fontFamily\": \"SF Pro\", \"fontWeight\": 400, \"fontSize\": 15.0, \"effects\":
[], \"padding\": 0, \"width\": 34.0, \"height\": 34.0, \"x\": -3918.0, \"y\": 998.0, \"hasIcon\": false,
\"style\": \"Trendy\", \"component_name\": \"Button\", \"subtype\": \"With Icons\", \"variant_details\":
{\"Mode\": [\"Dark\"], \"Size\": [\"Medium\"], \"Style\": [\"Filled\"], \"State\": [\"Disabled\"], \"On
material\": [\"True\"], \"Label Type\": [\"Symbol\"]}}}",
        "description": "Generate a Trendy Button in any color color."
    },
    {
        "json": "{\"variant_properties\": {\"color\": \"rgba(235, 235, 245, 1.0)\", \"strokes\": [],
\"strokeWeight\": 0.0, \"text\": \"\\udbc0\\ude84\", \"textColor\": \"rgba(235, 235, 245, 1.0)\",
\"borderRadius\": 500.0, \"fontFamily\": \"SF Pro\", \"fontWeight\": 400, \"fontSize\": 15.0, \"effects\":
[], \"padding\": 0, \"width\": 34.0, \"height\": 34.0, \"x\": -3918.0, \"y\": 998.0, \"hasIcon\": false,
\"style\": \"Trendy\", \"component_name\": \"Button\", \"subtype\": \"With Icons\", \"variant_details\":
{\"Mode\": [\"Dark\"], \"Size\": [\"Medium\"], \"Style\": [\"Filled\"], \"State\": [\"Disabled\"], \"On
material\": [\"True\"], \"Label Type\": [\"Symbol\"]}}}",
        "description": "Generate a Button in any color color."
```


The link to this can be found in https://github.com/KI-5/CoGen/blob/main/t5_generated_pairs_data_snippet.jsonl.

5.2 Generation of JSON

The JSON generation was evaluated using the outputs from two different architectures of a fine tuned T5. The first one was a simple T5 model and the second one was a complex T5 model with cross attention and BERT. The evaluation dataset was drawn from our main dataset, for consistency.
The metrics were calculated for 100 samples as follows;

Simple JSON			
Model	**BLEU score**	**ROUGE-1**	**ROUGE-L**
Simple T5	0.6071	0.6659	0.6433

T5 with cross attention and BERT	0.5574	0.6142	0.5940
Nested JSON			
Model	**BLEU score**	**ROUGE-1**	**ROUGE-L**
Simple T5	0.3769	0.4544	0.2414
T5 with cross attention and BERT	0.3771	0.4511	0.2503

Based on this the Simple T5 model with simple JSON works best. While for the nested JSON the complex T5 model works better.
The following table shows the accuracy, f1 score, precision and recall related to the component type identification;

Simple JSON				
Model	**Accuracy %**	**F1%**	**Recall%**	**Precision%**
Simple T5	86.2	89.6	86.2	94.2
T5 with cross attention and BERT	97.4	97.6	97.4	98.2
Nested JSON				
Model	**Accuracy %**	**Precision%**	**Recall%**	**F1%**
Simple T5	100	100	100	100

T5 with cross attention and BERT	98	99	98	98.5

Further evidence was needed to test which model to choose as T5 model with cross attention and BERT generated better scores. The success rate for the prompts were tested using 5 prompts surpassing the different component types, styles and other properties. This was assessed based on the following criteria;

- **In terms of the features checked.**
  For example the size of a button being small in size/ drop shadow effects/ border radius effects, stroke weights etc.
- **Component type identification**
  For example if the description says ‘Create a button’ then the component_name should be ‘button’
- **Style identification**
  For example if the description says ‘Create a Professional {component_name}’ then the style should be equal to ‘Professional’
- **JSON keys reflect the JSON keys in the dataset**
  All the keys present in the dataset should be reflected in the output regardless of syntax errors in the JSON.

Based on the above information and the evidence of test cases given below, it is evident that manipulating the simple JSON is a better idea for the Figma integration. Another reason was also because the model failed in identifying inner details in the nested JSON despite having the JSON structure down correctly.

Based on this the model chosen was the simple T5 fine-tuned model.

5.2.1 For simple JSON

Each of the factors above are given a weightage of 25% each for each prompt and based on that the success rate is calculated. The following table shows overview of the results;

Simple JSON with T5 fine-tuned model			
Prompt	**Pass**	**Fail**	**Success rate**
Button	5	0	100%
Label	5	0	100%
Input fields	3	2	60%
menu	3.75	1.25	75%
List-item	5	5	100%
Icon button	3.75	1.25	75%
T5 with Cross mechanism and BERT embeddings			
Prompt	**Pass**	**Fail**	**Success rate**
Button	3	2	60%
Label	4.5	0.5	90%
Input fields	2	3	40%
menu	1.75	3.25	35%

List-item	3.25	1.75	65%
Icon button	4.75	0.25	95%

The following are the test cases relevant to the components;

5.2.1.1 Simple JSON with T5 fine-tuned model

Buttons

Prompt	Values checked	Success of feature	Component type	Style	JSON keys	Difference
"Generate a Professional button with a State of hover."	State of hover	PASS	PASS	PASS	PASS	N/A
"Generate a Professional button with a size of small, state of default."	Size of small, state of default	PASS	PASS	PASS	PASS	N/A
"Generate a Basic button with	Drop shadow effect	PASS	PASS	PASS	PASS	N/A

DROP_SHADOW effects."						
"Generate a Professional Button with a border radius of 10.0"	Border radius 10	PASS	PASS	PASS	PASS	N/A
"Generate a Trendy Button"	Trendy	PASS	PASS	PASS	PASS	N/A

Labels

Prompt	**Values checked**	**Success of feature**	**Component type**	**Style**	**JSON keys**	**Difference**
"Generate a Professional label with a State of hover."	State of hover	PASS	PASS	PASS	PASS	N/A
"Generate a casual label with a size of small"	Size of small, state of default and casual for the	PASS	PASS	PASS	PASS	Style was not labelled as ‘Casual’,

						instead as ‘Label’
"Generate a label with DROP_SHADOW effects."	Drop shadow effects.	PASS	PASS	PASS	PASS	N/A
"Generate a default label with a border radius of 10.0"	Default label and border radius	PASS	PASS	PASS	PASS	Labeled the default style as basic
"Generate a Professional label"	Professional and label	PASS	PASS	PASS	PASS	N/A

Input fields

Prompt	**Values checked**	**Success of feature**	**Component type**	**Style**	**JSON keys**	**Difference**
"Generate a Basic input-field with a state of hover."	State of hover	PASS	PASS	PASS	PASS	N/A

"Generate a input-field."	-	FAIL	FAIL	FAIL	FAIL	Didn’t generate the JSON
"Generate a Professional input-field with a state of small."	State of small	PASS	PASS	PASS	PASS	N/A
"Generate a default input field with a border radius of 10.0"	Border radius value	FAIL	FAIL	FAIL	FAIL	Didn’t generate the JSON
"Generate a playful input field with drop_shadow effects"	10	PASS	PASS	PASS	PASS	N/A

Menu

Prompt	**Values checked**	**Success of feature**	**Component type**	**Style**	**JSON keys**	**Difference**

"Generate a Basic menu with a state of hover."	State of hover	PASS	PASS	PASS	PASS	N/A
"Generate a menu."	-	PASS	PASS	PASS	PASS	N/A
"Generate a Professional menu item with a small state."	Small state	FAIL	PASS	PASS	PASS	Returned large state
“Generate a default menu item with a border radius of 10.0"	Border radius	FAIL	FAIL	FAIL	FAIL	No proper json
"Generate a playful menu list with drop_shadow effects"	Drop shadow	PASS	PASS	PASS	PASS	N/A

List item

Prompt	Values checked	Success of feature	Component type	Style	JSON keys	Difference
"Generate a Basic list item with a state of hover."	State of hover	PASS	PASS	PASS	PASS	N/A
"Generate a list item."	-	PASS	PASS	PASS	PASS	N/A
"Generate a playful listitem with a state of hover."	hoverstate	PASS	PASS	PASS	PASS	N/A
"Generate a default list with a border radius of 10.0"	Border radius	PASS	PASS	PASS	PASS	N/A
"Generate a trendy menu list with	Drop shadow	PASS	PASS	PASS	PASS	N/A

drop_shadow effects"						

Icon button

Prompt	Values checked	Success of feature	Component type	Style	JSON keys	Difference
"Generate a Basic icon button with a state of hover."	State of hover	PASS	PASS	PASS	PASS	Returned Icon
"Generate a iconbutton."	-	FAIL	FAIL	FAIL	FAIL	Wrong JSON
"Generate a playful iconbutton with a state of hover."	hoverstate	PASS	PASS	PASS	PASS	N/A
"Generate a default icon button with a border	Border radius	PASS	PASS	PASS	PASS	Returned Icon

radius of 10.0"						
"Generate a trendy icon button with a stroke weight of 2"	Stroke weight values	FAIL	PASS	PASS	PASS	Returned value as 1.0

5.2.1.2 Simple JSON with T5 with Cross mechanism and BERT embeddings.

Buttons

Prompt	**Values checked**	**Success of feature**	**Component type**	**Style**	**JSON keys**	**Difference**
"Generate a Professional button with a State of hover."	State of hover	PASS	PASS	PASS	PASS	N/A
"Generate a Professional button with a size of small, state of default."	Size of small, state of default	PASS	PASS	PASS	PASS	N/A

"Generate a Basic button with DROP_SHADOW effects."	Drop shadow effect	PASS	PASS	PASS	PASS	N/A
"Generate a Professional Button with a border radius of 10.0"	Border radius 10	FAIL	FAIL	FAIL	FAIL	Wrong JSON
"Generate a Trendy Button"	Trendy	FAIL	FAIL	FAIL	FAIL	Wrong JSON

Labels

Prompt	**Values checked**	**Success of feature**	**Component type**	**Style**	**JSON keys**	**Difference**
"Generate a Professional label with a State of hover."	State of hover	PASS	PASS	PASS	PASS	N/A

"Generate a casual label with a size of small"	Size of small, state of default and casual for the	FAIL	PASS	FAIL	PASS	Style is labeled as ‘Trendy’ and State small is not included.
"Generate a label with DROP_SHADOW effects."	Drop shadow effects.	PASS	PASS	PASS	PASS	N/A
"Generate a default label with a border radius of 10.0"	Default label and border radius	PASS	PASS	PASS	PASS	Labeled the default style as basic
"Generate a Professional label"	Professional and label	PASS	PASS	PASS	PASS	N/A

Input fields

Prompt	**Values checked**	**Success of feature**	**Component type**	**Style**	**JSON keys**	**Difference**

"Generate a Basic input-field with a state of hover."	State of hover	PASS	PASS	PASS	PASS	N/A
"Generate a input-field."	-	FAIL	FAIL	FAIL	FAIL	Didn’t generate the JSON
"Generate a Professional input-field with a state of small."	State of small	PASS	PASS	PASS	PASS	N/A
"Generate a default input field with a border radius of 10.0"	Border radius	FAIL	FAIL	FAIL	FAIL	Didn’t generate the JSON
"Generate a playful input field with drop_shadow effects"	Drop shadow	FAIL	FAIL	FAIL	FAIL	Didn’t generate the JSON

Menu

Prompt	Values checked	Success of feature	Component type	Style	JSON keys	Difference
"Generate a Basic menu item with a state of hover."	State of hover	PASS	PASS	PASS	PASS	N/A
"Generate a menu."	-	FAIL	FAIL	FAIL	FAIL	Didn’t generate the whole JSON
"Generate a Professional menu item with a small state."	Small state	FAIL	PASS	PASS	PASS	Returned large state
“Generate a default menu item with a border radius of 10.0"	Border radius	FAIL	FAIL	FAIL	FAIL	No proper json

"Generate a playful menu list with drop_shadow effects"	Drop shadow	FAIL	FAIL	FAIL	FAIL	No proper json

List item

Prompt	Values checked	Success of feature	Component type	Style	JSON keys	Difference
"Generate a Basic list item with a state of hover."	State of hover	FAIL	PASS	PASS	PASS	Shows "Show supporting text": ["Hover"]
"Generate a list item."	-	PASS	PASS	PASS	PASS	N/A
"Generate a playful listitem with a state of hover."	hoverstate	FAIL	PASS	PASS	PASS	Shows "Show supporting text": ["Hover"]

"Generate a default list with a border radius of 10.0"	Border radius	PASS	FAIL	PASS	PASS	Style is professional
"Generate a trendy menu list with drop_shadow effects"	Drop shadow	FAIL	FAIL	FAIL	FAIL	Wrong JSON created

Icon button

Prompt	Values checked	Success of feature	Component type	Style	JSON keys	Difference
"Generate a Basic icon button with a state of hover."	State of hover	PASS	PASS	PASS	PASS	Returned Icon
"Generate a iconbutton."	-	PASS	PASS	PASS	PASS	N/A

"Generate a playful iconbutton with a state of hover."	hoverstate	PASS	PASS	PASS	PASS	N/A
"Generate a default icon button with a border radius of 10.0"	Border radius	PASS	PASS	PASS	PASS	Returned Icon
"Generate a trendy icon button with a stroke weight of 2"	Stroke weight values	FAIL	PASS	PASS	PASS	Returned value as 1.0

5.2.2 For nested JSON

The same weightage was added as done above. The following table shows overview of the results;

Simple T5 fine-tuned model			
Prompt	**Pass**	**Fail**	**Success rate**
Button	3.5	1.5	70%

Label	3	2	60%
Input fields	3.5	1.5	70%
menu	3.25	1.75	65%
List-item	2.5	2.5	50%
Icon button	3	2	60%
T5 with Cross mechanism and BERT embeddings			
Prompt	**Pass**	**Fail**	**Success rate**
Button	4.5	0.5	90%
Label	4	1	80%
Input fields	2.75	2.25	55%
menu	3.75	1.25	75%
List-item	3	2	60%
Icon button	3	2	60%

The following are the test cases relevant to the components;

5.2.2.1 Nested JSON with Simple T5 .

Buttons

Prompt	Values checked	Success of feature	Component type	Style	JSON keys	Difference
"Generate a Professional button with a State of hover."	State of hover	PASS	PASS	PASS	FAIL	The frame was not created initially.
"Generate a Professional button with a size of small, state of default."	Size of small, state of default	PASS	PASS	PASS	PASS	N/A
"Generate a Basic button with DROP_SHADOW effects."	Drop shadow effect	FAIL	PASS	PASS	PASS	No effects
"Generate a Professional Button with a border radius of 10.0"	Border radius 10	FAIL	PASS	PASS	FAIL	Wrong and Frame doesn’t exist and therefore

						the border radius doesn’t either.
"Generate a Trendy Button"	Trendy	FAIL	PASS	PASS	FAIL	Wrong JSON. The frame doesn’t exist.

Labels

Prompt	**Values checked**	**Success of feature**	**Component type**	**Style**	**JSON keys**	**Difference**
"Generate a Professional label with a State of hover."	State of hover	PASS	PASS	PASS	FAIL	No frame layer.
"Generate a casual label with a size of small"	Size of small, state of default	PASS	PASS	FAIL	PASS	N/A

	and casual for the					
"Generate a label with DROP_SHADOW effects."	Drop shadow effects.	FAIL	PASS	FAIL	FAIL	No effects, frame layer and wrong style was created.
"Generate a default label with a border radius of 10.0"	Default label and border radius	FAIL	FAIL	PASS	PASS	Border radius is 0, the style is basic and not default.
"Generate a Professional label"	Professional and label	PASS	PASS	PASS	FAIL	Frame is not created and JSON is wrong.

Input fields

Prompt	Values checked	Success of feature	Component type	Style	JSON keys	Difference
"Generate a Basic input-field with a state of hover."	State of hover	PASS	PASS	PASS	PASS	N/A
"Generate a input-field."	-	PASS	PASS	FAIL	PASS	Wrong style as it says ‘professional’
"Generate a Professional input-field with a state of small."	State of small	PASS	PASS	PASS	PASS	N/A
"Generate a default input field with a border radius of 10.0"	Border radius	FAIL	PASS	FAIL	FAIL	Border radius is 0. Wrong style instead takes subtype as Default.

"Generate a playful input field with drop_shadow effects"	Effects	FAIL	PASS	PASS	FAIL	No effects was created.

Menu

Prompt	Values checked	Success of feature	Component type	Style	JSON keys	Difference
"Generate a Basic menu item with a state of hover."	State of hover	PASS	PASS	PASS	PASS	N/A
"Generate a menu."	-	PASS	PASS	FAIL	PASS	Style is Professional
"Generate a Professional menu item with a small state."	Small state	FAIL	PASS	PASS	PASS	State is Default

“Generate a default menu item with a border radius of 10.0"	Border radius	FAIL	PASS	FAIL	FAIL	Wrong style, returns Professional, border radius is 0.
"Generate a playful menu list with drop_shadow effects"	Drop shadow	FAIL	PASS	PASS	FAIL	Icon and text nodes given together.

List item

Prompt	Values checked	Success of feature	Component type	Style	JSON keys	Difference
"Generate a Basic list item with a state of hover."	State of hover	FAIL	PASS	PASS	FAIL	No frame node initially and wrong JSON structure.
"Generate a list item."	-	PASS	PASS	FAIL	FAIL	Style is profession

						al, wrong JSON
"Generate a playful listitem with a state of hover."	hoverstate	FAIL	PASS	PASS	FAIL	No hover state and wrong JSON.
"Generate a default list with a border radius of 10.0"	Border radius	FAIL	PASS	FAIL	PASS	Style is professional, wrong JSON structure, border radius 0.
"Generate a trendy menu list with drop_shadow effects"	Drop shadow	FAIL	PASS	PASS	FAIL	Wrong JSON created and no effects

Icon button

Prompt	Values checked	Success of feature	Component type	Style	JSON keys	Difference

"Generate a Basic icon button with a state of hover."	State of hover	FAIL	PASS	PASS	FAIL	No hover state, text properties are there.
"Generate a iconbutton."	-	PASS	PASS	PASS	FAIL	Text properties are there.
"Generate a playful iconbutton with a state of hover."	hoverstate	PASS	PASS	PASS	FAIL	Text properties are there.
"Generate a default icon button with a border radius of 10.0"	Border radius	PASS	PASS	FAIL	FAIL	Text properties are there, basic style.
"Generate a trendy icon button with a stroke weight of 2"	Stroke weight values	FAIL	PASS	PASS	FAIL	Text properties are there, no stroke details.

5.2.2.2 Nested JSON with T5 with Cross attention and BERT embeddings.

Buttons

Prompt	Values checked	Success of feature	Component type	Style	JSON keys	Difference
"Generate a Professional button with a State of hover."	State of hover	PASS	PASS	PASS	PASS	N/A
"Generate a Professional button with a size of small, state of default."	Size of small, state of default	PASS	PASS	PASS	PASS	N/A
"Generate a Basic button with DROP_SHADOW effects."	Drop shadow effect	FAIL	PASS	PASS	PASS	No effects created.
"Generate a Professional Button with a border radius of 10.0"	Border radius 10	FAIL	PASS	PASS	PASS	No border radius details.
"Generate a Trendy Button"	Trendy	PASS	PASS	PASS	PASS	N/A

Labels

Prompt	Values checked	Success of feature	Component type	Style	JSON keys	Difference
"Generate a Professional label with a State of hover."	State of hover	PASS	PASS	PASS	PASS	N/A
"Generate a casual label with a size of small"	Size of small, state of default and casual for the	FAIL	PASS	PASS	PASS	Style is labeled as ‘Trendy’
"Generate a label with DROP_SHADOW effects."	Drop shadow effects.	PASS	PASS	FAIL	PASS	Trendy is the style.
"Generate a default label	Default label and	FAIL	PASS	FAIL	PASS	Labeled the default

with a border radius of 10.0"	border radius					style as basic, border radius is 0.
"Generate a Professional label"	Professional and label	PASS	PASS	PASS	PASS	N/A

Input fields

Prompt	Values checked	Success of feature	Component type	Style	JSON keys	Difference
"Generate a Basic input-field with a state of hover."	State of hover	PASS	PASS	PASS	PASS	N/A
"Generate a input-field."	-	PASS	PASS	FAIL	FAIL	Professional input field., creating icons inside the input field.

"Generate a Professional input-field with a state of small."	State of small	PASS	PASS	PASS	FAIL	Icon inside Input field.
"Generate a default input field with a border radius of 10.0"	Border radius	FAIL	PASS	FAIL	FAIL	Style is basic., icon inside inputfield, no border radius.
"Generate a playful input field with drop_shadow effects"	Effects	FAIL	PASS	PASS	FAIL	No effects, icon inside input field

Menu

Prompt	**Values checked**	**Success of feature**	**Component type**	**Style**	**JSON keys**	**Difference**
"Generate a Basic menu item with a	State of hover	PASS	PASS	PASS	PASS	N/A

state of hover."						
"Generate a menu."	-	PASS	PASS	FAIL	PASS	Style is professional.
"Generate a Professional menu item with a small state."	Small state	FAIL	PASS	PASS	PASS	Returned size small,
“Generate a default menu item with a border radius of 10.0"	Border radius	FAIL	PASS	FAIL	PASS	Professional style,no border radius
"Generate a playful menu list with drop_shadow effects"	Drop shadow	FAIL	PASS	PASS	PASS	No effects

List item

Prompt	Values checked	Success of feature	Component type	Style	JSON keys	Difference
"Generate a Basic list item with a state of hover."	State of hover	FAIL	PASS	PASS	FAIL	No hover state, children are not inside the JSON.
"Generate a list item."	-	PASS	PASS	FAIL	FAIL	Professional is the style, children are not inside the JSON.
"Generate a playful listitem with a state of hover."	hoverstate	FAIL	PASS	PASS	FAIL	No hover state and children are not inside the JSON.
"Generate a default list with a border	Border radius	PASS	FAIL	PASS	PASS	Style is professional

radius of 10.0"						
"Generate a trendy list item with drop_shadow effects"	Drop shadow	FAIL	PASS	PASS	PASS	Effects are not there.

Icon button

Prompt	Values checked	Success of feature	Component type	Style	JSON keys	Difference
"Generate a Basic icon button with a state of hover."	State of hover	PASS	PASS	PASS	FAIL	Returns text details.
"Generate a iconbutton."	-	PASS	PASS	PASS	FAIL	Returns text details. Returns basic style.

"Generate a playful iconbutton with a state of hover."	hoverstate	PASS	PASS	PASS	FAIL	Returns text details.
"Generate a default icon button with a border radius of 10.0"	Border radius	FAIL	PASS	FAIL	FAIL	Returns text details., returns default style, no border radius.
"Generate a trendy icon button with a stroke weight of 2"	Stroke weight values	FAIL	PASS	PASS	FAIL	Returns text details, no stroke details.

The relevant prompts and JSON outputs for these test cases can be found in https://github.com/KI-5/CoGen/blob/main/Models/Testing%20t5%20json%20component.ipynb.

6. Testing outputs

6.1 From the model :

The model does not generate correct outputs every single time.

Additionally the model works extremely well with certain components when compared to other components.
Please note that as the JSON dealt with does not reflect all the properties for a figma JSON, the plugin is not very complex. The JSONs generated were simplified as most models these days struggle with JSON generation. As a result, having simplified JSON was a requirement in this project.

Based on the fine tuned T5 model:

The following is a simple example of a button for the prompt "Generate a Professional button with a state of hover."
As the JSON is too long the following is a snippet of the JSON which specifies the 'component_name' and the 'style' correctly. The state is also specified correctly.


```
'style": "Professional", "component_name": "Button", "subtype": "Light", "variant_details": "State": ["hover"]
```


Based on the fine tuned T5 model with BERT and cross mechanism:

For a different component like a 'Label' with a prompt saying "Generate a label with DROP_SHADOW effects.", the model is able to generate the effects, and the component_name and then it creates default values for the 'style' and 'subtype' keys.
The following image shows the 'style', 'component_name' and the 'subtype';


```
style": "Trendy", "component_name": "label", "subtype": "Light",
```


The following image shows the 2 layers of drop shadow layers;

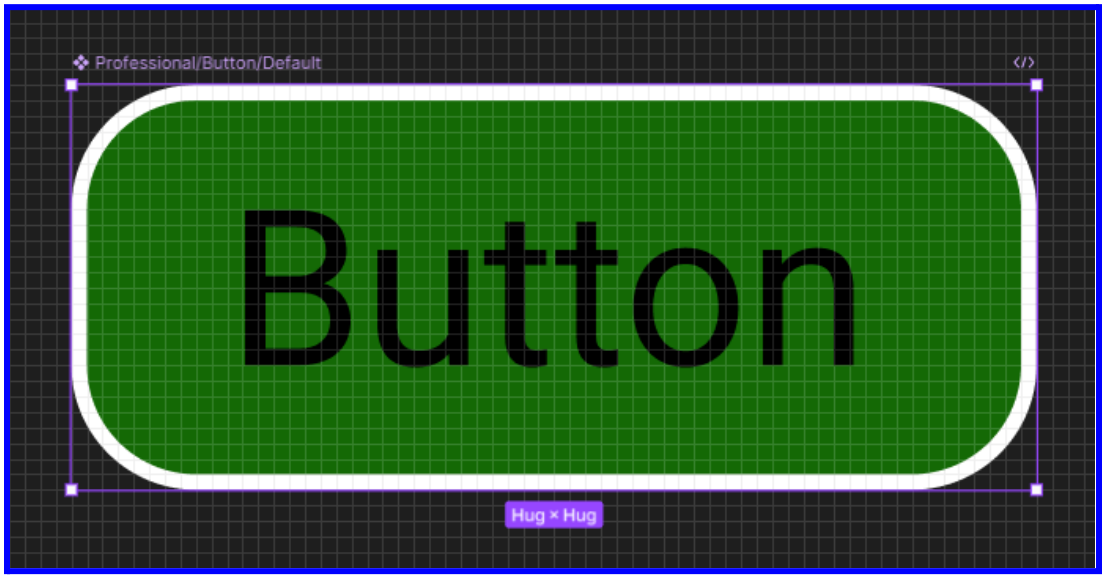

```
"effects": ["type": "DROP_SHADOW", "color": "rgba(0, 0, 0, 0.30000001192092896)", "type": "DROP_SHADOW", "color": "rgba(0, 0, 0, 0.15000000596046448)"
```

6.2 In Figma

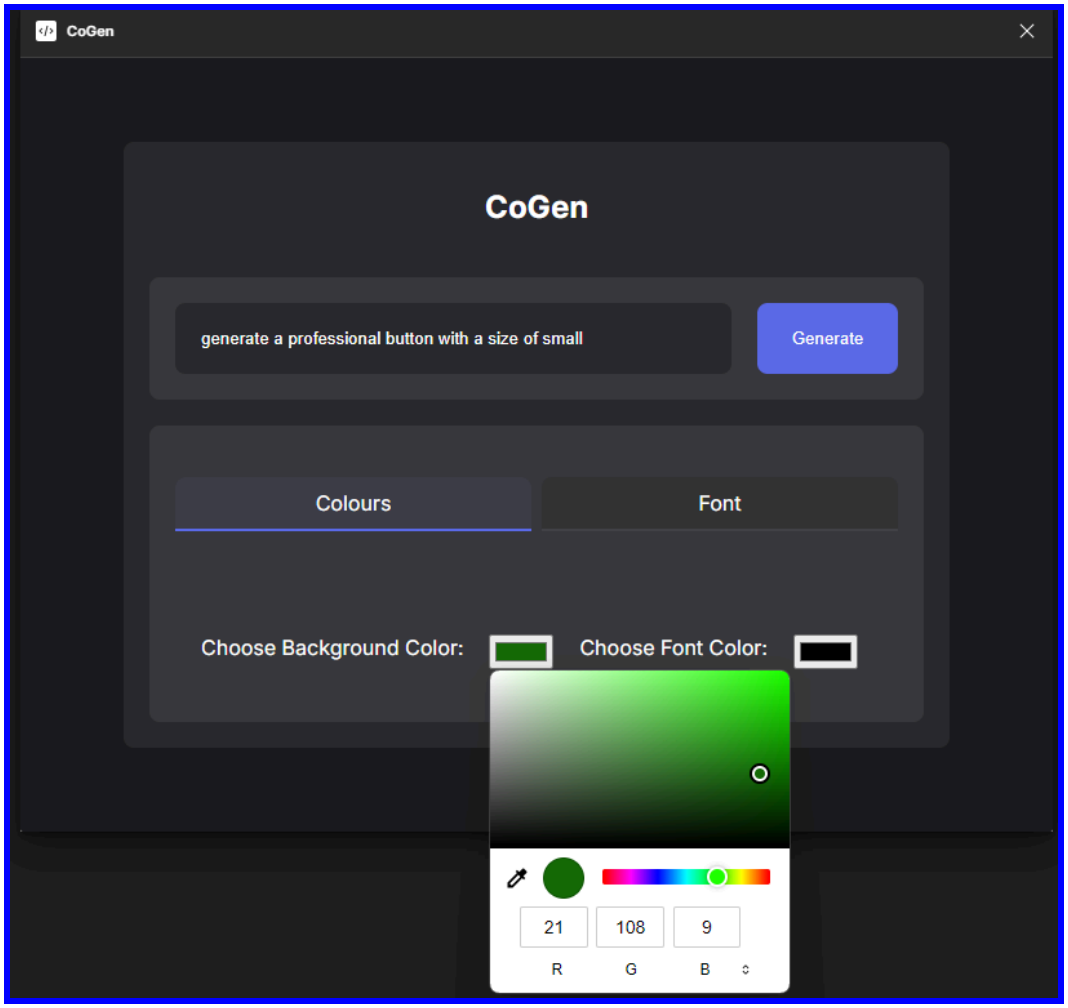
01 Button creation

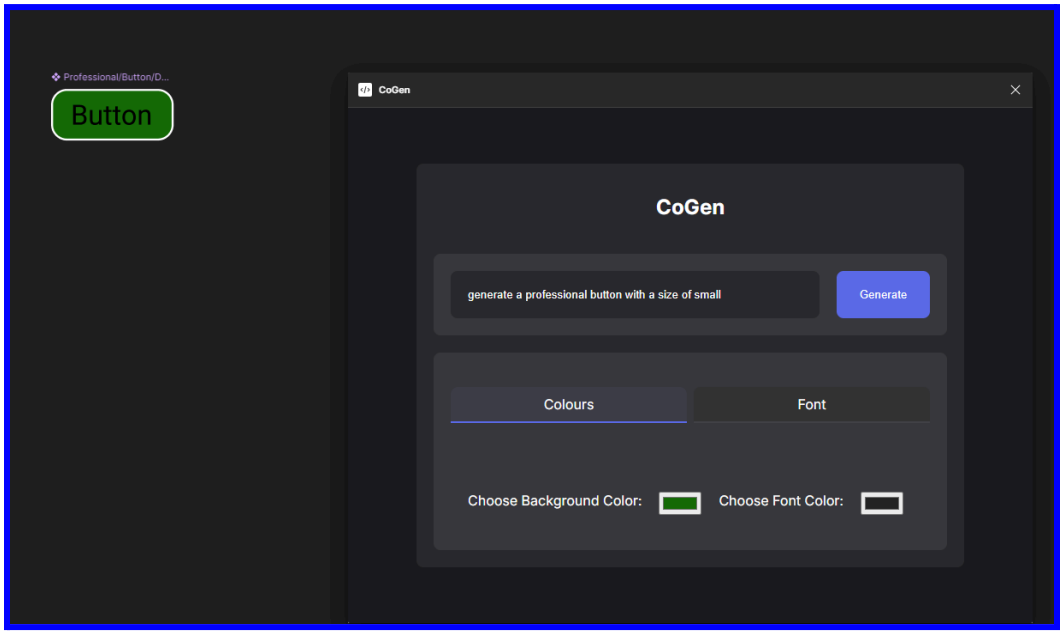
The Prompt will be “generate a professional button with a size of small”. The user is able to specify the colour and font properties.

The following image shows the button created;



The following images show the plugin UI with the tabs for colours and fonts.


CoGen
CoGen
generate a professional button with a size of small
Generate
Colours
Font
Choose Background Color:
Choose Font Color:
21
108
9
R
G
B


Professional/Button/D...
Button
CoGen
CoGen
generate a professional button with a size of small
Generate
Colours
Font
Choose Background Color:
Choose Font Color:

02 List item creation

The Prompt will be “Generate a list item.”The user is able to specify the colour and font properties.

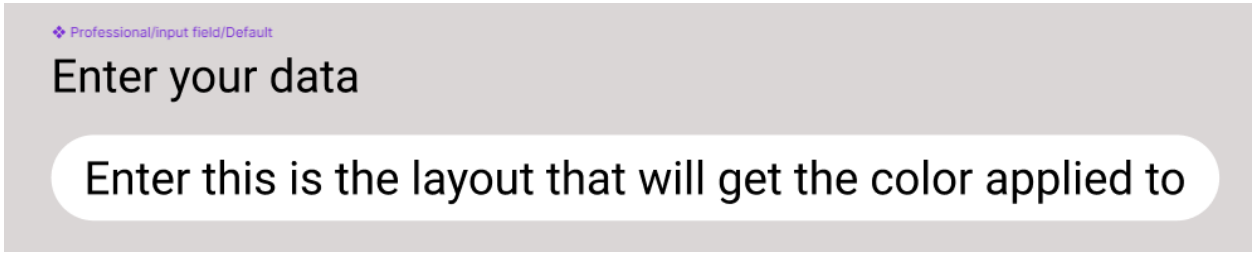
The following image shows the list item created;

This component includes the default space for the inclusion of an icon. Due to constraints in relevance to icon libraries, this could not be incorporated.

03 Input field creation

The Prompt will be “Generate a input-field.” The user is able to specify the colour and font properties.

The following image shows the input field created;



03 Icon button creation

The Prompt will be “Generate a icon-button.” The user is able to specify the colour and font properties.

The following image shows the input field created;

This component includes the default space for the inclusion of an icon. Due to constraints in relevance to icon libraries, this could not be incorporated.

6.3 Postprocessing

First, before creating the components the most important part is the post processing part where there are checks against syntax, and characters in the JSON. Following this there was a method to map them to Figma JSON where a more nested structure was used as the output.

The link relevant can be found in https://github.com/KI-5/CoGen/blob/main/Front%20end/src/code.js.

7. Future enhancements

- Limited number of components: CoGen only works with six components (’Button’, ’Input field’, ’Icon button’, ’Menu list’, ’List items’, ’Label’), and four styles (`Basic`, `Trendy`, `Playful`, `Professional`). This is mainly due to resource limitations and the lack of publicly available Figma datasets.
- Creation of component sets with variants: Figma allows for the creation of component sets with multiple variants of a component within. However, at present, CoGen is able to create only a single component which includes one variant with a single prompt. This again is mostly due to time and

resource constraints relating to the training of models. In future, CoGen could be expanded to include more variants within a component set.

- Lack of a vast dataset: The T5 model requires more input with nested components to understand the relationship between the node type (frame, text, icon) and the properties of each node. This will result in better component generation within Figma as well.
- Lack of complex models: The model's ability to understand diverse prompts is very limited. Access to paid models like GPT-4 would make this process faster and efficient as they are more refined in terms of understanding prompts and generating content. Moreover, this would aid in creating nested components as well.
- Lack of complex integrations: The ability to integrate with sites such as Google Fonts to include more font options, or connecting to an icon library set, will be a huge added advantage.

8. Conclusion

The project presents a novel approach to UI component generation, addressing a gap in current research. Unlike existing systems that primarily focus on the overall visual design or image-based representations, this work emphasizes the creation of UI components based on input prompts.
Additionally, prompt generation from component JSON adds significant value to the UI design field, as most datasets typically consist only of design details, rather than the descriptive information needed to guide component creation.

Code and project available at:
https://github.com/KI-5/CoGen

References

Gajjar, N. et al. (2021). Akin: Generating UI Wireframes From UI Design Patterns Using Deep Learning. *26th International Conference on Intelligent User Interfaces*. 14 April 2021. College Station TX USA: ACM, 40–42. Available from https://doi.org/10.1145/3397482.3450727 [Accessed 24 March 2024].

Huang, F. et al. (2021). Creating User Interface Mock-ups from High-Level Text Descriptions with Deep-Learning Models. Available from http://arxiv.org/abs/2110.07775 [Accessed 24 March 2024].

Manandhar, D., Jin, H. and Collomosse, J. (2021). Magic Layouts: Structural Prior for Component Detection in User Interface Designs. *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2021. Nashville, TN, USA: IEEE, 15804–15813. Available from https://doi.org/10.1109/CVPR46437.2021.01555 [Accessed 24 March 2024].

Zhao, T. et al. (2021). GUIGAN: Learning to Generate GUI Designs Using Generative Adversarial Networks. *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. May 2021. Madrid, ES: IEEE, 748–760. Available from https://doi.org/10.1109/ICSE43902.2021.00074 [Accessed 24 March 2024].