UI Generator: from Natural Language to Web Pages with Large Code Generation Model

Ziqing Huang
Institute for Interdisciplinary Information Sciences
Tsinghua University
hzq19@mails.tsinghua.edu.cn

Abstract

Large language models have been widely used in a wide range of NLP tasks. Based on the impressive ability of large code generation models, we proposed a generic method for designing web pages by generating front-end programs. With collected data and our original framework, we fine-tuned CodeGeeX, a multilingual code generative model, to generate HTML code from natural languages with control.

1 Introduction

Code generation is an important research area in Natural Language Processing. Compared to other NLP tasks, it is more challenging considering the strictness of the answers and the difficulty in evaluation. There have been a lot of successful trials in recent years, such as Codex (Chen et al., 2021), AlphaCode (Li et al., 2022), PaLM-Coder (Chowdhery et al., 2022), CodeGen (Nijkamp et al., 2022), and CodeT (Chen et al., 2022). They mainly use large language models trained on a large amount of data, in which we believe that the capacity is not explored.

In this work, we proposed a generic method for designing web pages by generating front-end programs. Eventually, we want to enable users to generate their customized pages by providing arbitrary information such as the content, desired styles, or existing pages as templates. Our long-term goal is to release the front-end engineers from the simple and repeated work, by automatically generating the website with different types of interfaces such as buttons and links. Figure 1 shows an overview of our work. Our code will be public in the future.

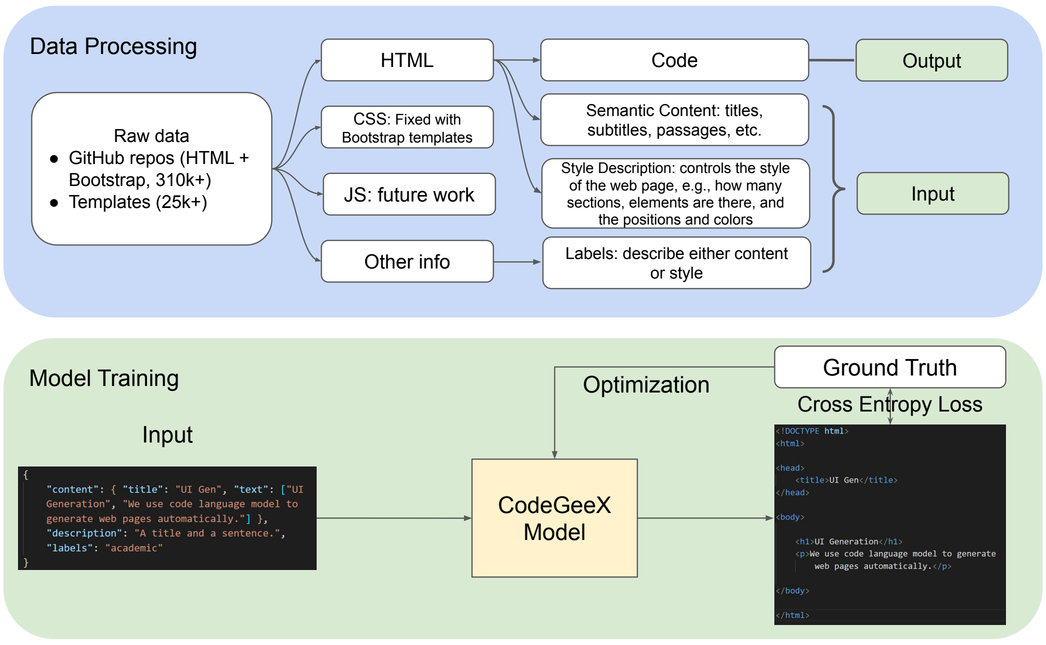
2 Related Works

2.1 Language Models

Language Model (LM) is a probability distribution over sequences of words. Generally, LMs can determine the probability of a given sequence in a sentence. LMs have wide applications in Natural Language Processing as generative models, such as question answering and machine translation.

In recent years, Pre-trained Language Models (PLMs) have become the mainstream in NLP research. One of the most representative series includes GPT (Radford et al., 2018), GPT-2 (Radford et al., 2019), and GPT-3 (Brown et al., 2020). With a large number of parameters, GPT-3 managed to deal with a variety of NLP tasks without further fine-tuning. Based on GPT, Codex (Chen et al., 2021) showed that the models can generate high-quality code that solves a large number of problems by fine-tuning on abundant publicly available code. Recently, OpenAI released their latest product ChatGPT[1] that interacts in a conversational way and performs surprisingly well on different kinds of tasks.

[1]https://openai.com/blog/chatgpt/


Data Processing
Raw data
• GitHub repos (HTML + Bootstrap, 310k+)
• Templates (25k+)
HTML
CSS: Fixed with Bootstrap templates
JS: future work
Other info
Code
Semantic Content: titles, subtitles, passages, etc.
Style Description: controls the style of the web page, e.g., how many sections, elements are there, and the positions and colors
Labels: describe either content or style
Output
Input
Model Training
Input
CodeGeeX Model
Optimization
Ground Truth
Cross Entropy Loss

```
{
    "content": { "title": "UI Gen", "text": ["UI
    Generation", "We use code language model to
    generate web pages automatically."] },
    "description": "A title and a sentence.",
    "labels": "academic"
}
```

```
<!DOCTYPE html>
<html>

<head>
    <title>UI Gen</title>
</head>

<body>

    <h1>UI Generation</h1>
    <p>We use code language model to generate
        web pages automatically.</p>

</body>

</html>
```

Figure 1: The overview of data processing and model training

2.2 Web Page Generation

On the one hand, there are numerous products and research on the generation of web pages. We divided them into two following categories.

The first type of work takes sketches as inputs, using vision models and pre-defined components to implement the design. As an instance, Microsoft AI Lab developed Sketch2Code[2] that transforms pictures to HTML code. CodeFun[3] is a platform that supports code generation from sketch, psd and svg format. Imgcook[4] shares a similar function with CodeFun. This line of thread focus on the accuracy of recognition rather than creative generation. The generated components are generally limited, and it takes the user's effort to provide a sketch of the design.

The other way is to use language models to generate HTML code directly. CodeGeeX, GitHub Copilot[5] and some other coding aids are already capable of generating HTML code. However, it is observed that it would be difficult to control the generation by the prompt in the zero-shot setting, and the generated web pages are usually of low quality in terms of both content and design. Whether and how we can generate ready-to-use web pages from natural language is still an open question.

3 Preliminaries

3.1 Bootstrap

Bootstrap[6] is a widely used open-source front-end framework that contains HTML, CSS and JavaScript-based design templates for a variety of different interface components, such as buttons and navigation bars. As of December 2022, it has received more than 161k stars in GitHub[7]. Its latest release is Bootstap 6.

In the current stage, rather than generating

[2]https://github.com/microsoft/ailab/tree/master/Sketch2Code
[3]https://code.fun/
[4]https://www.imgcook.com/
[5]https://github.com/features/copilot
[6]https://getbootstrap.com/
[7]https://github.com/twbs/bootstrap

web pages from scratch, we take advantage of the Bootstrap framework, generating only the HTML files. For simplicity, we extract the corresponding contents from the CSS files about the styles and put them into the HTML file as well.

3.2 CodeGeeX

CodeGeeX[8] is a large-scale multilingual code generative model with 13 billion parameters, pre-trained on a large code corpus of more than 20 programming languages. It contains several unique features, including multilingual code generation and cross-lingual code translation. It also supports a variety of real-world functionalities, such as code completion and explanation, as a VS Code extension.

4 Data Collection and Processing

4.1 Data Source

4.1.1 Web Page Templates

There are many websites that provide numerous templates for users to build their websites conveniently. We referred to about 3.4k templates from https://www.free-css.com/, 872 templates from https://www.templatemonster.com/cn/free-html-website-templates/, and more than 23k from https://sc.chinaz.com/moban/index.html. We want the model to generate well-designed pages rather than simple collections of plain text by learning from the source code of these templates.

4.1.2 GitHub Repositories

We collect more than 300k open-source repositories marked Bootstrap and HTML from GitHub. We also include some extra information, such as the tags and numbers of stars. They are filtered by a set of rules such as min and max length before usage.

[8] https://github.com/THUDM/CodeGeeX

4.1.3 Real-World Web Pages

In addition to the data source above, we also considered web pages in real life, especially those popular ones, since they could be more diverse and contain more useful information in the content. We leave this part as future work, however, considering the following problems:

1. Some pages are encrypted and the source code is inaccessible, or there could be a copyright issue;
2. Some pages require more complicated environments to display;
3. Some pages have extremely long source code which is hard to process.

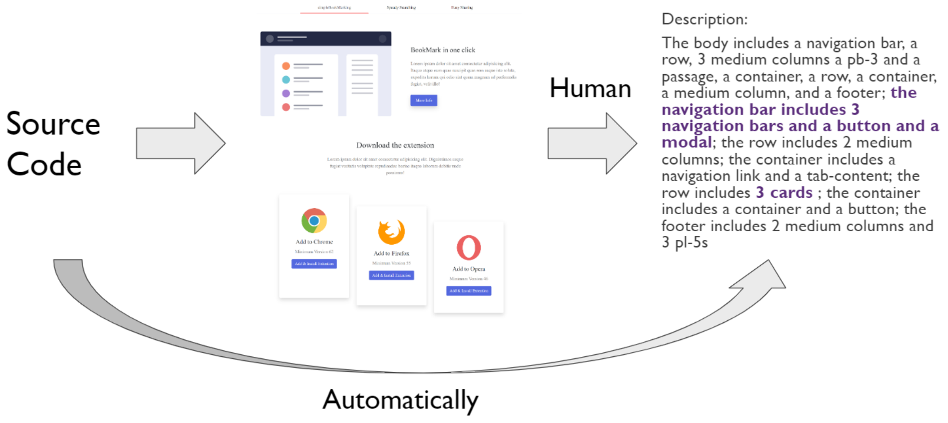
4.2 Data Cleaning

To equip the model with the ability to generate code with correct grammar and natural language, based on different types of input, we reorganize our data samples into the following parts.

Code. In the current stage, we consider only HTML files. We replace some unmeaningful contents such as placeholders and URLs with specific special tokens to improve efficiency and robustness.

Content. With an HTML parser, we extract all text on the pages, including titles, passages, text on the buttons, etc. To improve the quality of the content, we apply the following filtering rules.

1. Remove placeholders. Most of the web page templates and parts of the GitHub repositories contain few meaningful long paragraphs. They typically use some fixed passages, such as *Lorem ipsum*



The body includes a navigation bar, a row, 3 medium columns a pb-3 and a passage, a container, a row, a container, a medium column, and a footer; **the navigation bar includes 3 navigation bars and a button and a modal**; the row includes 2 medium columns; the container includes a navigation link and a tab-content; the row includes **3 cards** ; the container includes a container and a button; the footer includes 2 medium columns and 3 pl-5s

Figure 2: Two different ways to get the description of style.

(Wikipedia contributors, 2022), in place of the content. We remove such placeholders to prevent the model from learning disinformation from them.

2. Extract key information. We found that a large part of the text on a web page might contain little information, for example, "click" and "more info". We remove most of the text and keep only a small amount of important content, such as the title, subtitle, and extracted key words, as we associate the content with corresponding tags.

3. Keep long passages. For the longer passages other than placeholders that appear on the web page, we save them as part of the input.

Style Description. We introduce a rule-based script to describe the format of the web page by parsing it. An example is shown in fig. 3, where the generated description is poor in readability. It is generally not in the way that people will describe their goal, but there is still some key information captured, such as "3 navigation bars" and "3 cards".

We have also tried to use some language models to get the description from the code, but the performance is relatively poor. The model can generate a natural and smooth sentence, but it might not capture the web page's overall features at all.

We are now hiring annotators to write more natural and high-quality descriptions for future usage.

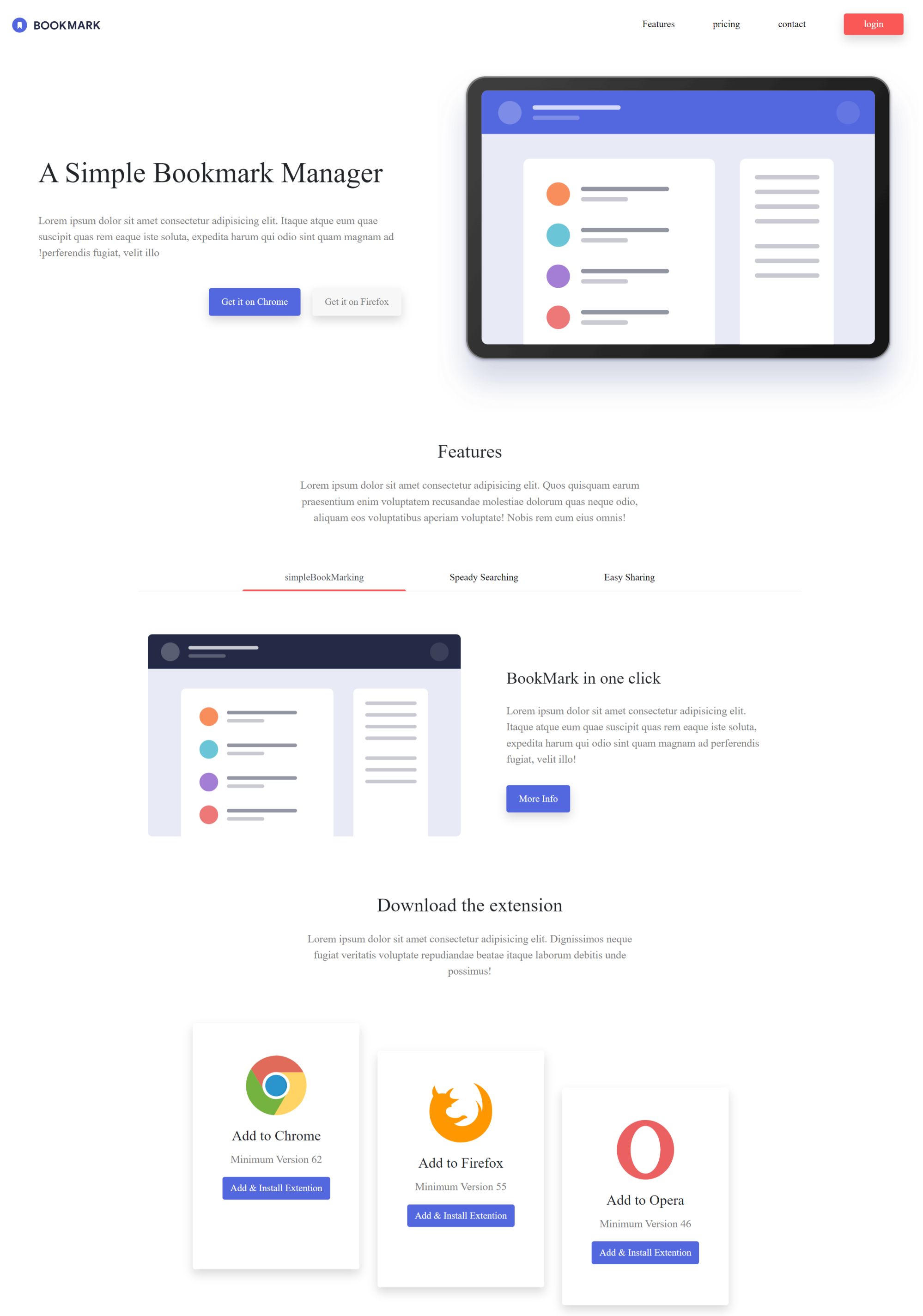
Tags. We include the tags that describe the style of the source, such as "food", "business", or "academic".

Metadata. We also have some additional information, including the number of stars obtained on GitHub. Such data can be used to assign a weight for the web page in the training.

Our long-term goal is to equip the model with the ability to generate a web page from different types and levels of inputs. For example, one can provide the contents to be shown on the page, or describe what the page should be like, in either a high-level or detailed way.

5 Method

Generally speaking, we started from CodeGeeX and fine-tuned it with our carefully designed data to get a model specified for generating HTML files.


BOOKMARK

Features pricing contact login

A Simple Bookmark Manager

Lorem ipsum dolor sit amet consectetur adipisicing elit. Itaque atque eum quae suscipit quas rem eaque iste soluta, expedita harum qui odio sint quam magnam ad !perferendis fugiat, velit illo

Get it on Chrome Get it on Firefox

Features

Lorem ipsum dolor sit amet consectetur adipisicing elit. Quos quisquam earum praesentium enim voluptatem recusandae molestiae dolorum quas neque odio, aliquam eos voluptatibus aperiam voluptate! Nobis rem eum eius omnis!

simpleBookMarking Speady Searching Easy Sharing

BookMark in one click

Lorem ipsum dolor sit amet consectetur adipisicing elit. Itaque atque eum quae suscipit quas rem eaque iste soluta, expedita harum qui odio sint quam magnam ad perferendis fugiat, velit illo!

More Info

Download the extension

Lorem ipsum dolor sit amet consectetur adipisicing elit. Dignissimos neque fugiat veritatis voluptate repudiandae beatae itaque laborum debitis unde possimus!

Add to Chrome
Minimum Version 62
Add & Install Extention

Add to Firefox
Minimum Version 55
Add & Install Extention

Add to Opera
Minimum Version 46
Add & Install Extention

Figure 3: Part of a web page example. The generated description is "The body includes a navigation bar, a row, 3 medium columns a pb-3 and a passage, a container, a row, a container, a medium column, and a footer; **the navigation bar includes 3 navigation bars and a button and a modal**; the row includes 2 medium columns ; the container includes a navigation link and a tab-content; the row includes **3 cards** ; the container includes a container and a button; the footer includes 2 medium columns and 3 pl-5s".

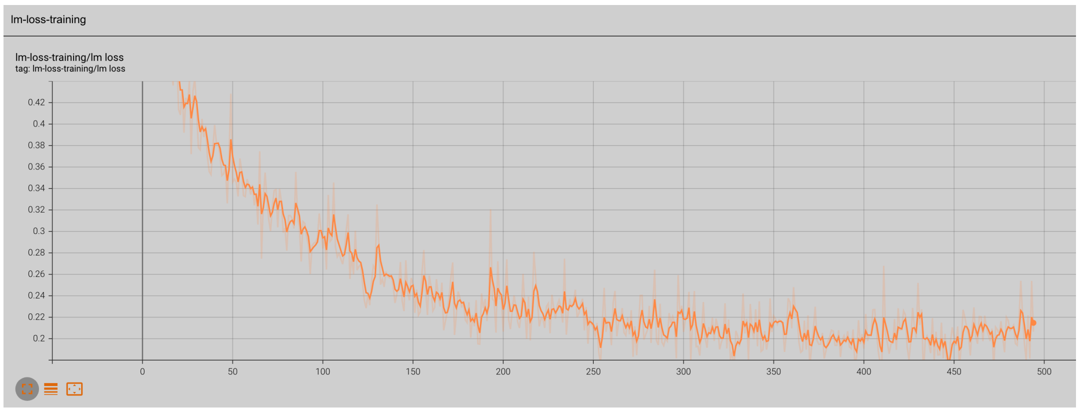


Figure 4: Training Loss

5.1 Tokenization

Using the tokenizer of CodeGeeX, there are about 10,601 tokens in each HTML file on average in the data we mentioned in section 4.1, which is too long to train. In this case, we improved the tokenization efficiency by adding extra tokens specified for HTML. For instance, by adding `<div>` and `</div>` we can "merge" three or four tokens into one. Also, we considered the line break and indent of the next line as one token to emphasize the role of the indent. By adding about 1.7k extra tokens, we manage to reduce the average number of tokens to 8,445, saving 20.3% of the tokens.

5.2 Hierarchical Generation

Though the number of tokens can be reduced by the specified tokenizer, the normal max sequence length that we can afford to train a large language model, typically 2,048 or 4,096, is still unable to cover real-world usage of HTML files. Additionally, the nested structure of HTML code limits the usage of simple sequential generation with truncation, since it might lose information in the former parts, and cause grammatical errors. In this case, we introduced **hierarchical generation**, in which the generation aligns with the action of expanding tags in HTML files. Figure 5 shows an example. Basically, each tag longer than the threshold would be generated by an independent model pass that takes the code frame with the part to generate being masked by a special token (`<|extratoken|>` in the figure) as input.

The architecture allows multiple masked parts in the framework as well as nested ones, enabling the generation of infinitely long code in theory.

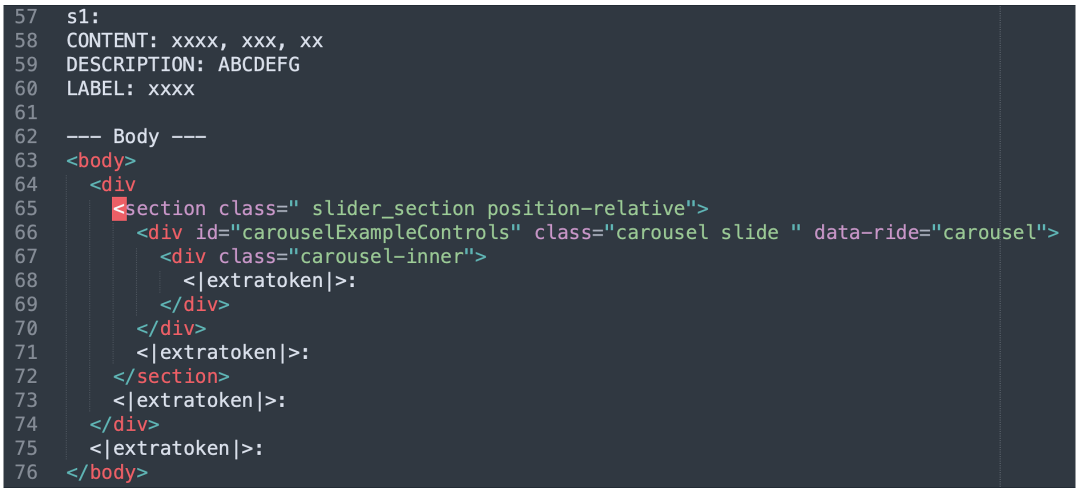
6 Results

We have trained a baseline of our model and managed to generate runnable HTML code with it. Figure 4 shows the change of loss in training progress. We will improve the quality of generation in both content and design in the future.

7 Conclusion and Future Work

In this work, we widely collected data from web pages from different resources and added additional descriptions. With such data and our original framework, we fine-tuned CodeGeeX, a multilingual code generative model, to generate HTML code with control. In the future, we are planning to further improve our work in the following aspects:

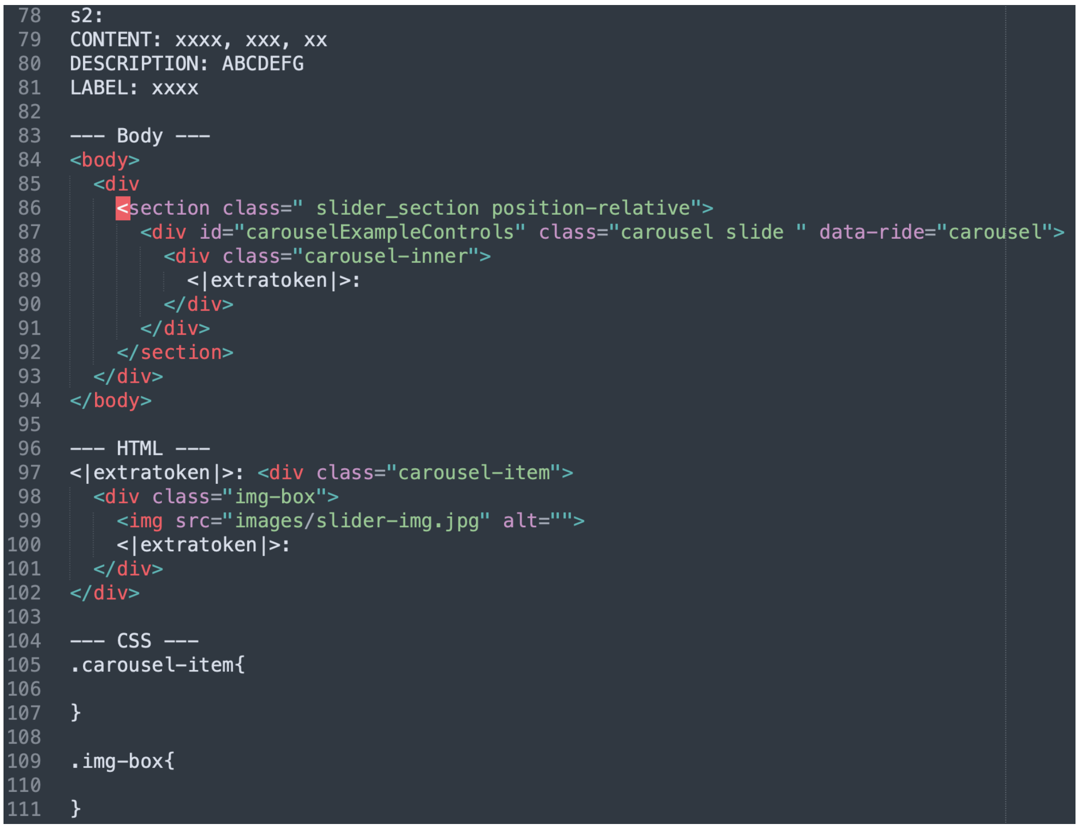
1. Bringing in human annotations of web



```
s1:
CONTENT: xxxx, xxx, xx
DESCRIPTION: ABCDEFG
LABEL: xxxx

--- Body ---
<body>
  <div
    <section class=" slider_section position-relative">
      <div id="carouselExampleControls" class="carousel slide " data-ride="carousel">
        <div class="carousel-inner">
          <|extratoken|>:
        </div>
      </div>
      <|extratoken|>:
    </section>
    <|extratoken|>:
  </div>
  <|extratoken|>:
</body>
```

(a) Input



```
s2:
CONTENT: xxxx, xxx, xx
DESCRIPTION: ABCDEFG
LABEL: xxxx

--- Body ---
<body>
  <div
    <section class=" slider_section position-relative">
      <div id="carouselExampleControls" class="carousel slide " data-ride="carousel">
        <div class="carousel-inner">
          <|extratoken|>:
        </div>
      </div>
    </section>
  </div>
</body>

--- HTML ---
<|extratoken|>: <div class="carousel-item">
  <div class="img-box">
    <img src="images/slider-img.jpg" alt="">
    <|extratoken|>:
  </div>
</div>

--- CSS ---
.carousel-item{

}

.img-box{

}
```

(b) Output

Figure 5: An example of the input and output of the hierarchical generation.

page description to improve the control of generation;

2. Adding real-world websites as training samples to improve the integrity and coherence of content;

3. Implementing the automatic matching of HTML and CSS files and enabling our generated files to work without redundant environments;

4. Expanding the range of generation to CSS and more types of files involved.

8 Acknowledgement

This project is done in cooperation with Qinkai Zheng, Ruijie Cheng, and Shengrong Qin under the direction of Zhilin Yang. The work of others would be marked exclusively.

References

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language Models are Few-Shot Learners.

Bei Chen, Fengji Zhang, Anh Nguyen, Daoguang Zan, Zeqi Lin, Jian-Guang Lou, and Weizhu Chen. 2022. CodeT: Code Generation with Generated Tests.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. 2021. Evaluating Large Language Models Trained on Code.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, Parker Schuh, Kensen Shi, Sasha Tsvyashchenko, Joshua Maynez, Abhishek Rao, Parker Barnes, Yi Tay, Noam Shazeer, Vinodkumar Prabhakaran, Emily Reif, Nan Du, Ben Hutchinson, Reiner Pope, James Bradbury, Jacob Austin, Michael Isard, Guy Gur-Ari, Pengcheng Yin, Toju Duke, Anselm Levskaya, Sanjay Ghemawat, Sunipa Dev, Henryk Michalewski, Xavier Garcia, Vedant Misra, Kevin Robinson, Liam Fedus, Denny Zhou, Daphne Ippolito, David Luan, Hyeontaek Lim, Barret Zoph, Alexander Spiridonov, Ryan Sepassi, David Dohan, Shivani Agrawal, Mark Omernick, Andrew M. Dai, Thanumalayan Sankaranarayana Pillai, Marie Pellat, Aitor Lewkowycz, Erica Moreira, Rewon Child, Oleksandr Polozov, Katherine Lee, Zongwei Zhou, Xuezhi Wang, Brennan Saeta, Mark Diaz, Orhan Firat, Michele Catasta, Jason Wei, Kathy Meier-Hellstern, Douglas Eck, Jeff Dean, Slav Petrov, and Noah Fiedel. 2022. PaLM: Scaling Language Modeling with Pathways.

Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, Thomas Hubert, Peter Choy, Cyprien de Masson d'Autume, Igor Babuschkin, Xinyun Chen, Po-Sen Huang, Johannes Welbl, Sven Gowal, Alexey Cherepanov, James Molloy, Daniel J. Mankowitz, Esme Sutherland Robson, Pushmeet Kohli, Nando de Freitas, Koray Kavukcuoglu, and Oriol Vinyals. 2022. Competition-Level Code Generation with AlphaCode.

Erik Nijkamp, Bo Pang, Hiroaki Hayashi, Lifu Tu, Huan Wang, Yingbo Zhou, Silvio Savarese, and Caiming Xiong. 2022. A Conversational Paradigm for Program Synthesis.

Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. 2018. Improving language understanding by generative pre-training.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019. Language models are unsupervised multitask learners.

Wikipedia contributors. 2022. Lorem ipsum — Wikipedia, the free encyclopedia. `https://en.wikipedia.org/w/index.php?title=Lorem_ipsum&oldid=1114552653`. [Online; accessed 9-November-2022].