



Graduation Report

Software Engineering Graduation Internship 2023-2024

Document details:

| | |
|-------------------------------|---|
| Project Name | "AI for Web Development: Creating an HTML Bootstrap Code Generator from both Images and Code Description Text" |
| Version | 02 |
| Owner | Name: Mooza Ali Salim Aamir Al Harrasi Email: 471987@student.saxion.nl moozaali98@gmail.com |
| University supervisors | Name: Manasse Siekmans E-mail: m.siekmans@saxion.nl Name: Etto Salomons E-mail: e.l.salomons@saxion.nl |
| Company supervisor | Name: George Petrov E-mail: george@wappler.io |
| Document Number | 02 |

Summary

This graduation report investigates the creation of an artificial intelligence-based tool for producing HTML Bootstrap code from images and codes description text provided by the user. The goal of this project is to automate the conversion of images and code description text into accurate HTML Bootstrap code. The report includes a detailed summary of the project's objectives, research subjects, user requirements, product design, decision-making process, and testing validation outcomes.

The report serves as a development roadmap for the AI tool, giving a thorough evaluation of the decision-making process and underlying logic. It demonstrates the student's ability to solve problems and create viable solutions by using AI approaches and software development practices.

In conclusion, this graduation report marks an important milestone in the student's academic path. It covers the complete project, from start to finish, demonstrating the effective creation of an AI-based solution for creating HTML Bootstrap code. By incorporating AI and machine learning approaches into software development, the student has proven not just technical skills but also a thorough awareness of the difficulties and possibilities that AI presents in the industry.

Table of Contents

| | |
|---|----|
| 1. Introduction | 7 |
| 2. Company information | 8 |
| 2.1 General information | 8 |
| 2.2 Vision | 8 |
| 2.3 Mission | 8 |
| 2.4 History | 8 |
| 3. Project description | 9 |
| 4. Stakeholders | 10 |
| 5. Research questions | 11 |
| 5.1 Main question | 11 |
| 5.2 Sub questions | 11 |
| 6. Process | 13 |
| 7. Requirements | 14 |
| 8. Product Design | 16 |
| 8.1 Use Case | 16 |
| 8.2 System architecture | 18 |
| 8.2.1 Image to Bootstrap HTML code : | 18 |
| 8.2.2 Text To Bootstrap HTML Code: | 22 |
| 8.3 Frontend and Backend Classification | 24 |
| 8.3.1 Image To Bootstrap HTML Code: | 24 |
| 8.3.2 Text To Bootstrap HTML Code | 25 |
| 8.4 User Interface Design | 26 |
| 8.4.1 Wireframes | 26 |
| 8.4.2 Product flow | 28 |
| 9. Design choices | 29 |
| 10. Implementation | 30 |
| 10.1 Abstract | 30 |
| 10.2 Introduction | 31 |
| 10.3 Techniques | 32 |
| 10.3.1 Computer vision techniques | 33 |
| 10.3.2 Machine learning techniques | 33 |
| 10.3.3 Fine-tuning Techniques | 35 |

Research Report

Software Engineering Graduation

| | | |
|-------------|-------------------------------------|-----------|
| 10.4 | Methodology | 35 |
| 10.4.1 | Language and frameworks..... | 35 |
| 10.4.2 | Dataset and Features | 36 |
| 10.5 | Experiment | 39 |
| 10.6 | Result | 41 |
| 10.7 | Discussion | 46 |
| 10.8 | Implementation Details | 48 |
| 11. | Project Conclusion..... | 59 |
| 12. | Reflection..... | 60 |
| 13. | References | 63 |
| 14. | Appendix..... | 64 |

List of Figures

| | |
|---|----|
| Figure 1 Use case for AI-powered Image OR Text to html code convertor | 16 |
| Figure 2 Sequence diagram: Flow and Interactions in an image to HTML Code Generation System | 19 |
| Figure 3 Sequence diagram: Flow and Interactions in an HTML Code Generation System | 21 |
| Figure 4 High level architecture: Using a Fine-Tuned GPT-3 Model to Generate Bootstrap HTML Code from Code | 22 |
| Figure 5 Sequence diagram: Interactive Code Generation with User Feedback and Re-Fine-Tuning in Wappler platform using the fine-tuned GPT-3 model | 23 |
| Figure 6 CNN Encoder and RNN Decoder | 34 |
| Figure 7 One Image from Sketch2Code Dataset | 37 |
| Figure 8 One Image from Pix2Code Dataset | 37 |
| Figure 9 Input and Output of GPT-3 Finetuned model | 38 |
| Figure 10 The Flow of Creating Text to HTML Code System | 40 |
| Figure 11 Model 1: Training and validation metrics of all the steps for model 1 | 42 |
| Figure 12 Model 2: Training and Validation mean metrics of all the steps for model 2 | 43 |
| Figure 13 Model 3: Training and validation mean metrics for generate complete html page | 44 |
| Figure 14 Sequence Diagram for Flask's Server-side | 49 |
| Figure 15 HTTP Requests and Responses Between User Browser and Flask Server | 51 |
| Figure 16 HTTP Request and Responses from Backend and Frontend | 51 |
| Figure 17 Sequence Diagram: Generating and Modifying HTML Code with OpenAI API | 54 |
| Figure 18 Displays the effort required for different tasks in relation to time for Text to Code tool | 62 |
| Figure 19 Example 1: Input Image (Sketch Design) | 64 |
| Figure 20 Example 2: Output Image (Generated Bootstrap HTML Code) | 64 |
| Figure 21 Example 2: Input Image (Sketch Design) | 65 |
| Figure 22 Example 2: Output Image (Generated Bootstrap HTML Code) | 65 |
| Figure 23 User-Interface for image to code tool | 66 |
| Figure 24 Text-to-Code System Output in OpenAI Playground | 67 |
| Figure 25 Chat between User and Wappler AI for Text-to-Code System | 68 |
| Figure 26 User-Interface for Image-To-Code Tool | 69 |

List of Tables

| | |
|---|----|
| Table 1 User-Friendly Interface for Efficient Image or Text to Bootstrap HTML Conversion | 14 |
| Table 2 Machine Learning-Based Conversion for Accurate and Efficient Image or Text to HTML Transformation | 14 |
| Table 3 Customizable Bootstrap HTML Output | 15 |
| Table 4 Fast and Efficient Conversion of Images or Text to HTML Code | 15 |
| Table 5 Design-to-HTML Conversion Tool for Web Designers | 17 |
| Table 6 Customizable HTML Code for Developers | 17 |
| Table 7 Streamlined Idea-to-HTML Conversion Tool for Business Owners | 18 |

List of Abbreviations and Terms

A list of abbreviations is provided below to help with understanding and navigating throughout this report. These abbreviations appear often throughout the paper and cover a wide range of topics and terminology relevant to the graduation project:

AI: Artificial intelligence

API: Application programming interface

CNN: Convolutional neural network

DNN: Deep neural network

DSL: Domain-specific language

GPT: Generative Pre-trained Transformer

GRU: Gated recurrent unit

GUI: Graphical user interface

LSTM: Long Short-Term Memory

NLP: Natural language processing

RNN: Recurrent neural network

WBA: A web-based application

1. Introduction

The use of artificial intelligence (AI) in software development has created a variety of new opportunities and made it possible to automate difficult and time-consuming operations that were previously impossible. One such work is creating an HTML Bootstrap code generator from code description text or images provided by the user.

I have set out to investigate this topic as a fourth-year student working on a graduation project in Wappler by developing a machine learning and AI tool. The project's objective is to create an AI-based tool that can efficiently and precisely translate screenshots from websites, images of sketch layouts or code description text provided by the user into HTML Bootstrap code.

Key features of the project are fully covered throughout the report. This comprises a brief company history, a definition of the parties involved, and a context for the project's development. The investigation's research themes are explained, demonstrating the student's grasp of the conceptual foundations and previous work on the subject.

The in-depth research on user needs for the AI tool is a significant part of the report. The report guarantees that the tool is customized to suit the wants and expectations of the target users by defining the relevant features and functions. This user-centred approach displays the student's commitment to developing a solution that tackles real-world problems.

In addition, the report discusses the product design process, including the concerns and design concepts used during development. Usability, scalability, and compatibility with the HTML Bootstrap framework are all aspects to consider. The study emphasizes the project's thinking and attention to detail by extensively reviewing the design process. This graduation report will serve as a roadmap for the AI tool's development, providing a complete examination of the decision-making process and underlying logic.

Validation and testing are critical to the project's success. The report demonstrates the tool's usefulness and accuracy in creating HTML Bootstrap code from code description text or images provided by the user, based on the results of rigorous testing and validation methods. These data are examined and interpreted, revealing information about the tool's performance and highlighting possible areas for improvement.

Overall, this report will be a key component of the graduation project, giving a thorough explanation of how the AI-based solution for turning screenshots from websites, sketches of layouts or user code descriptions into HTML Bootstrap code was created.

2. Company information

This section of the report contains basic information about the company, a brief history, a vision and mission statement, and facts about the Wappler company according to the Wappler Documentation(n.d).

2.1 General information

Dynamic Zones has already taken the lead by developing Wappler. Wappler is a single tool to allow designers and developers to build advanced data-driven solutions as websites and mobile apps – all visually but still empowering, with best practices and the best open frameworks available.

Wappler is a division of **Dynamic Zones** International BV with offices in the Netherlands (Enschede) and Bulgaria (Sofia). Wappler is a low-code, visual productivity tool allowing users to create rich data-driven, highly interactive websites and mobile apps. It offers a great visual design and a standard set of frameworks allowing for full freedom and rich productivity. Wappler supports many frameworks for front-end and back-end solutions as well as visual design tools and workflow editors. Its visual, low-code productivity tools for professionals, allows them to focus on high-level design thinking and code practices, making them hugely productive and still using the best practices and latest frameworks.

2.2 Vision

Wappler thinks that the creative process can be improved by bridging the gap between humans and machines. Wappler creates visual coding tools for experts so they can use their imagination to create complex, highly interactive web apps and games. Wappler is continuously on the lookout for individuals with a love for visual automation, future human interaction design, coding, and artificial intelligence.

2.3 Mission

“UNLEASH CREATIVITY”

Wappler is a visual tool for creating dynamic, information-driven, and highly interactive custom websites, content management systems, and mobile applications. Due to its integrated, biggest reactive front-end and backend frameworks, visual two-way data binding, and Bootstrap builder, there is no need to code. You only need to concentrate on your creativity and productivity.

2.4 History

The development of Wappler took place in the shadows of Adobe for 20 years, adding fantastic components and plugins to their products. Since its initial launch in 2018 - Wappler has drawn thousands of users from all around the world. Behind Wappler, there is a huge community of users around the globe using the product and delivering feedback on new ideas, which Wappler's development team promptly integrates into the platform.

3. Project description

The objective of this project is to build a machine learning-based AI tool that can translate screenshots from websites, images of sketch layouts or code description text into HTML code. The tool will include a call to a machine-learning model.

Users will be able to quickly convert their sketch layout images or code description text to Bootstrap HTML code using the tool, which is intended to be user-friendly, effective, and precise. This eliminates the need for manual coding. Web developers, designers, and other professionals that need to immediately and simply translate their designs into usable HTML code will find the tool to be helpful.

The design and building of the machine learning model, testing and evaluating the machine learning model, and the integration of the two elements are the main key milestones in the development of the tool. The machine learning algorithm will be able to accurately translate the images or words to HTML code after being trained on a dataset comprising screenshots of websites or code description texts. Users will be able to import their sketch layout images or write code description text and generate the HTML code with Bootstrap with just a few clicks.

A variety of design decisions will need to be made throughout the development process, such as those regarding the user interface, the machine learning techniques to be employed, and the features and functionalities to be included in the tool. A report that details these design decisions will give a thorough overview of the development process and the justification for the choices taken.

Ultimately, this project will make a significant addition to the field of web development by enabling experts to rapidly and simply translate their concepts into useful HTML Bootstrap code using AI and machine learning models.

4. Stakeholders

University Supervisor(Manasse Siekmans): The university supervisor will offer assistance in problem-solving and ensure that academic standards are upheld. Also, the university supervisor will stay in communication with me and with the company supervisor to share challenges and achievements. Also, the university supervisor will offer comments and feedback on the deliverables (like the graduation internship report) that must be submitted to the university during the graduation internship time and will offer me advice and support in achieving the project's objectives.

Company Supervisor(George Petrov): The company supervisor will give me the necessary training and regular performance and expectation feedback to help me reach the goals and learning objectives that are relevant to the professional goals of the intern's academic courses. The corporate supervisor will also help me establish contact with various software team members, also supervise and approve the weekly hours spent toward the internship's mandatory number of hours.

Product owner(George Petrov): Greater oversight of the products' functionality as well as appearance will be addressed by the product owner. User stories are accepted by the product owner for a particular sprint, and he is responsible for making sure they meet their requirements. Also, it is the product owner's responsibility and capacity to inform me of its advancement toward the sprint objective.

5. Research questions

This section of the report contains the main research question that can be answered by answering the three following sub-questions.

5.1 Main question

The goal of this project is to create an artificial intelligence-based tool that can translate hand-drawn layout images or code description text, into Bootstrap HTML code. This tool will make web development more effective and productive for users. As a result, the following main research question has been developed:

Main Question: How can AI be used to translate website layout sketches or code description text into HTML Bootstrap code while easily integrating with Wappler to give developers a user-friendly interface?

5.2 Sub questions

The technical details of creating the AI-based tool are the focus of the following sub-question. To transform sketches of webpage layout or code description text into Bootstrap HTML code, it is necessary to investigate several machine learning and AI techniques, such as image recognition, natural language processing, and deep learning.

Question 1: What artificial intelligence (AI) and machine learning (ML) approaches are available for automatically identifying and categorizing layout design characteristics in images or understanding human language in code description text to produce related HTML code in Bootstrap?

The first research question focuses on the creation of an AI-powered tool capable of analyzing images as well as code description text to determine layout design elements. It seeks to investigate various AI, NLP, and ML strategies for consistently recognizing layout features in images and understanding natural language code descriptions.

The question will almost definitely require a study into computer vision and image processing techniques for images, as well as natural language processing for code description texts. The study may also examine various machine learning methods, such as convolutional neural networks, decision trees, and support vector machines, and assess their performance in effectively identifying and categorizing layout design aspects in images or code description texts.

Overall, the goal is to produce a strong and efficient AI-based solution that can streamline the web development process and increase web developer productivity.

Research Report

Software Engineering Graduation

Question 2: How can the Python-based image-to-code or text-to-code tool be combined with Wappler to produce a user-friendly interface for generating HTML code from sketches of website layouts or code description texts?

The practical implementation of the Python-based image-to-code or text-to-code tool, specifically how it may be integrated into the online development platform Wappler, is the topic of this research question. The question intends to investigate the creation of a user-friendly interface that allows users to submit website layout sketches or code description texts and then obtain HTML code with Bootstrap created by the Python-based tool.

The question is expected to require web development and user interface design research, as well as practical programming abilities for integrating the Python-based application with Wappler. The study might also look into different ways to present the generated HTML code, such as a live preview or a downloadable file.

Ultimately, the goal is to establish a seamless interface between the Python-based image or texts to code tool and the Wappler platform, allowing web developers to produce HTML code from images of website layouts or code description texts in an understandable and fast way.

Question 3: What are the potential limitations and difficulties of using the AI-based tool in real-world scenarios, and how can its accuracy and efficacy be evaluated?

This research question is concerned with the practical considerations that may influence the development of the AI-based tool for producing Bootstrap HTML code from images depicting website layouts or code description texts. It aims to detect possible limitations and challenges that may develop in real-world circumstances, such as image quality changes, various coding standards, or compatibility concerns with other software tools.

The research may include testing the AI-based tool in various locations and use scenarios, as well as evaluating its correctness and efficacy in producing high-quality HTML code. It may also be necessary to create performance measurements and testing procedures to evaluate the tool's performance over time.

Finally, the goal is to determine the most effective methods of using the AI-based tool in practical contexts while ensuring its accuracy and reliability. Developers can construct more effective solutions that satisfy the needs of web development professionals in a range of scenarios by knowing the tool's potential limitations and boundaries.

The overall goal of the graduation assignment's main and supporting questions is to investigate the technical, practical, and evaluative aspects of developing an AI-based tool that transforms images of website layouts or converts natural human code description words into HTML code with Bootstrap and integrating that tool with Wappler to be displayed in a user-friendly interface.

6. Process

The Scrum technique was used for project management in this research study. Scrum is an Agile approach for managing and completing large projects. Working in iterations or sprints, each of which includes a set of tasks that must be done within a specific amount of time, often two weeks, is the methodology.

The Scrum framework was used to plan the project, and it was broken into sprints with particular tasks and deliverables. The sprints were managed using the Scrum board, which had tasks labelled as to-do, in progress, and completed. Each sprint had its sprint backlog, which was a list of activities that needed to be performed during the sprint.

Regular communication channels, such as email updates, progress reports, and presentations, can be used to keep stakeholders informed of progress. It is critical to identify key stakeholders early on and understand their expectations and project requirements.

Furthermore, regular check-ins with stakeholders can provide an opportunity to gather feedback, discuss any concerns or issues, and ensure that the project is on track to meet its objectives. The final report should also give stakeholders a full review of the project's progress, outcomes, and impact.

7. Requirements

A graduation report that highlights the essential features and capabilities of a software product or system must include user requirements. User requirements are used to direct the development process by defining what stakeholders or end users want from the product or system. To guarantee that the finished product satisfies the needs and expectations of the users, it is important to accurately collect and record user requirements.

During numerous meetings with Wappler's developers, those user requirements were created using a structured process that put the user's wants and expectations first.

1. **User-Friendly Interface:** The tool should have an intuitive and simple-to-use interface that enables users to quickly and easily import their sketches of website layouts or write code description texts and then generate HTML code in under 2 minutes.

| User-Friendly Interface | |
|-------------------------|--|
| Specific | The tool should be user-friendly and intuitive. |
| Measurable | In less than two minutes, users should be able to upload their images of website layouts or write code layout description text and transform them into HTML code with Bootstrap. |
| Achievable | The interface should be created in a way that makes it possible to finish the task in the given amount of time. |
| Relevant | For a seamless user experience, the interface must be user-friendly. |
| Time-bound | It should take less than two minutes to finish the conversion procedure. |

Table 1 User-Friendly Interface for Efficient Image or Text to Bootstrap HTML Conversion

2. **Accurate Conversion:** Using machine learning algorithms, the tool should be able to convert images of website layouts or code description texts into HTML code with Bootstrap without losing any important components.

| Accurate Conversion | |
|---------------------|---|
| Specific | For the tool to transform screenshots and images of website layouts or code description texts into HTML code with Bootstrap, machine learning algorithms should be used. |
| Measurable | The important elements should be maintained during the conversion process without any loss. |
| Achievable | The tool's machine learning algorithms should be able to precisely recognize and maintain design features for images or understand natural language for code description texts throughout the conversion process. |
| Relevant | To guarantee that the final Bootstrap HTML code closely matches the original design, accurate conversion is essential. |
| Time-bound | The conversion procedure should be carried out quickly and correctly. |

Table 2 Machine Learning-Based Conversion for Accurate and Efficient Image or Text to HTML Transformation

- 3. Modify HTML output:** The tool should give users the option to modify the HTML code and ask the tool to improve some parts of the design that it generates.

| Customized HTML Output | |
|------------------------|--|
| Specific | Users should be able to make modifications and requests for improvements to certain aspects of the design of the HTML code that the tool generates. |
| Measurable | The tool's UI should allow users to make changes in the code and submit requests for improvements. |
| Achievable | Modifying the HTML output is technically viable and can be accomplished through the use of a user interface that allows users to modify the code and submit ideas for improvements. |
| Relevant | The ability to change HTML output and request improvements is essential to provide users with a more customized and adjustable experience, which corresponds to the project's goal of improving web development productivity and efficiency. |
| Time-bound | This feature should be implemented within the project's timetable and milestones, with stakeholders receiving regular updates and progress reports. |

Table 3 Customizable Bootstrap HTML Output

- 4. Speed and Efficiency:** The tool should be rapid and effective, enabling users to quickly translate their designs of website layouts or code description texts into Bootstrap HTML code without needing to invest a lot of time in manual coding.

| Speed and Efficiency | |
|----------------------|--|
| Specific | The tool should be created to efficiently transform designs or descriptions into HTML code with Bootstrap. |
| Measurable | In comparison to manual coding, users should be able to finish the conversion process much faster. |
| Achievable | To provide a quick and efficient conversion procedure, the tool should use excellent algorithms and coding techniques. |
| Relevant | To give users a simplified and effective experience, the tool's speed and efficiency are essential. |
| Time-bound | The conversion procedure should be finished on time to meet customers' demands for effectiveness. |

Table 4 Fast and Efficient Conversion of Images or Text to HTML Code

These specifications will ensure that the tool satisfies the expectations of its intended users and offers a helpful solution for web developers and designers who wish to avoid wasting time and effort while transforming their designs of website layouts or code description texts into usable Bootstrap HTML code.

8. Product Design

Product design is the process of thinking of and developing a new product that satisfies the requirements of its target market. The product design in this project for an AI tool to convert images representing website layouts or code description texts into HTML code with Bootstrap would include use cases, system architecture, and user interface design that allows users to quickly and easily turn their website layout images or code description texts into Bootstrap HTML code.

8.1 Use Case

Web designers and developers that need to swiftly translate static image designs of website layouts or code description texts into HTML code could be one possible use case for the AI tool. Designers and developers could improve their workflows and save time by using an AI-powered image-to-code or text-to-code converter. Here are three use cases are presented below:

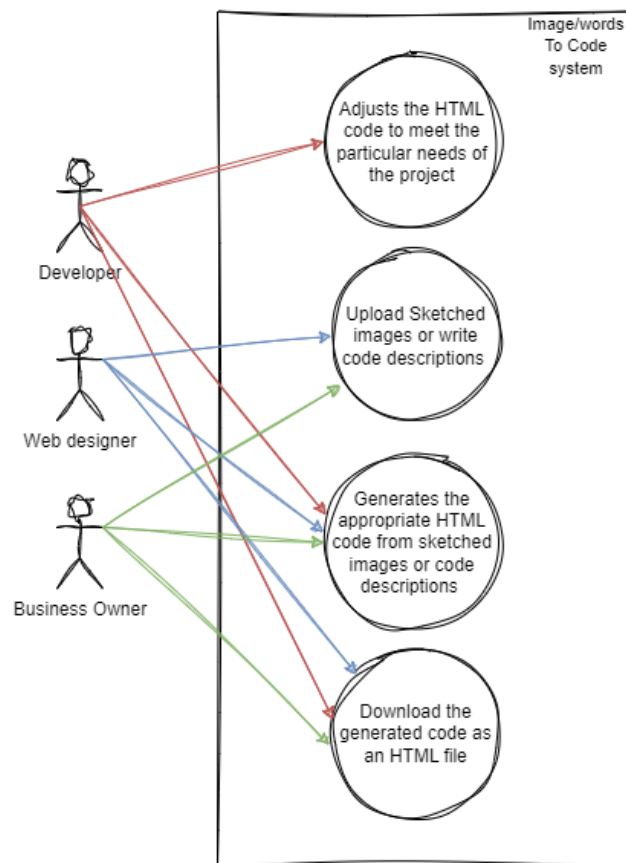


Figure 1 Use case for AI-powered Image OR Text to html code convertor

Research Report

Software Engineering Graduation

- As a web designer, I would like to generate HTML code from my design mockups rapidly and easily, without having to hand-code everything.

| | |
|-------------------|--|
| Goal | To give web designers a tool that quickly and easily converts their design mockups into Bootstrap HTML code. |
| Actors | Web designers are the main parties who must produce HTML code from their design mockups for website layouts. |
| Regular Steps | <ol style="list-style-type: none">1. The web designer enters the tool and uploads their design mockup.2. The tool creates the appropriate HTML code after analyzing the design.3. If necessary, the web designer can evaluate and edit the HTML code produced by the tool.4. The completed HTML code is downloaded by the web designer. |
| Alternative Steps | <ol style="list-style-type: none">1. The web designer can change the design mockup and try again if the tool is unable to produce the necessary HTML code.2. The web designer can ask for help from the tool's customer support if they run into any problems during the process. |
| Postconditions | <ol style="list-style-type: none">1. The final HTML code that exactly reflects the web designer's design mockup is delivered to them. |

Table 5 Design-to-HTML Conversion Tool for Web Designers

- As a developer, I want to modify the HTML code generated to meet the particular requirements of my project.

| | |
|-------------------|---|
| Goal | To give developers the flexibility to adjust the HTML code the tool produces to satisfy the unique needs of their project. |
| Actors | Developer: the main party wishing to adjust the HTML code that has been generated. |
| Regular Steps | <ol style="list-style-type: none">1. The tool's created HTML code is delivered to the developer.2. The developer looks over the HTML code that was generated to find the places that need to be changed.3. The developer adjusts the HTML code to meet the particular needs of their project.4. To confirm that the updated HTML code performs as anticipated, the developer tests it. |
| Alternative Steps | <ol style="list-style-type: none">1. The developer can ask for help from the tool's customer support if they run into any problems while adjusting. |
| Postconditions | <ol style="list-style-type: none">1. Modified HTML code that matches the exact specifications of the project is sent to the developer.2. The developer has verified that the changed HTML code is functional.3. By enabling developers to modify the resulting HTML code, the tool gives them freedom and personalization. |

Table 6 Customizable HTML Code for Developers

Research Report

Software Engineering Graduation

- As a business owner, I want to cut down on the time and work needed to turn my ideas and code description texts into usable HTML code with Bootstrap.

| | |
|-------------------|--|
| Goal | To give business owners a tool that makes it easier for them to convert their concepts and descriptions into practical HTML code. |
| Actors | The company owner is the main player who wishes to speed up the idea description-to-HTML conversion process. |
| Regular Steps | <ol style="list-style-type: none">1. The tool receives ideas or concepts from the business owner by describing the HTML code.2. The program generates the appropriate HTML code after analyzing the code descriptions.3. The business owner examines and accepts the HTML code that was prepared.4. The business owner gets the completed HTML code. |
| Alternative Steps | <ol style="list-style-type: none">1. The tool can be updated or given new ideas or concepts by the business owner if they are not happy with the HTML code that was generated.2. The company owner can ask for help from the tool's customer support if they run into any problems while using it. |
| Postconditions | <ol style="list-style-type: none">1. The application saves the business owner a lot of time and work by automating the conversion process.2. The business owner obtains the finished HTML code that precisely reflects their ideas or concepts.3. Without having to spend time and effort manually translating their concepts into HTML code, the business owner can concentrate on their primary business operations. |

Table 7 Streamlined Idea-to-HTML Conversion Tool for Business Owners

8.2 System architecture

The system architecture for both the Image-To-Code and Text-To-Code systems involves the design and arrangement of components that allow the translation of visual or textual inputs into appropriate code representations will be given below.

8.2.1 Image to Bootstrap HTML code :

The sequence diagram below shows the process flow and interactions inside Image To Code, an HTML code-generating system. It illustrates the stages involved, beginning with the User contributing layout images and ending with the produced HTML code with Bootstrap. Image Data Collection, Model Setup, Training, Evaluation, and Deployment are all critical components in converting images into HTML representations.

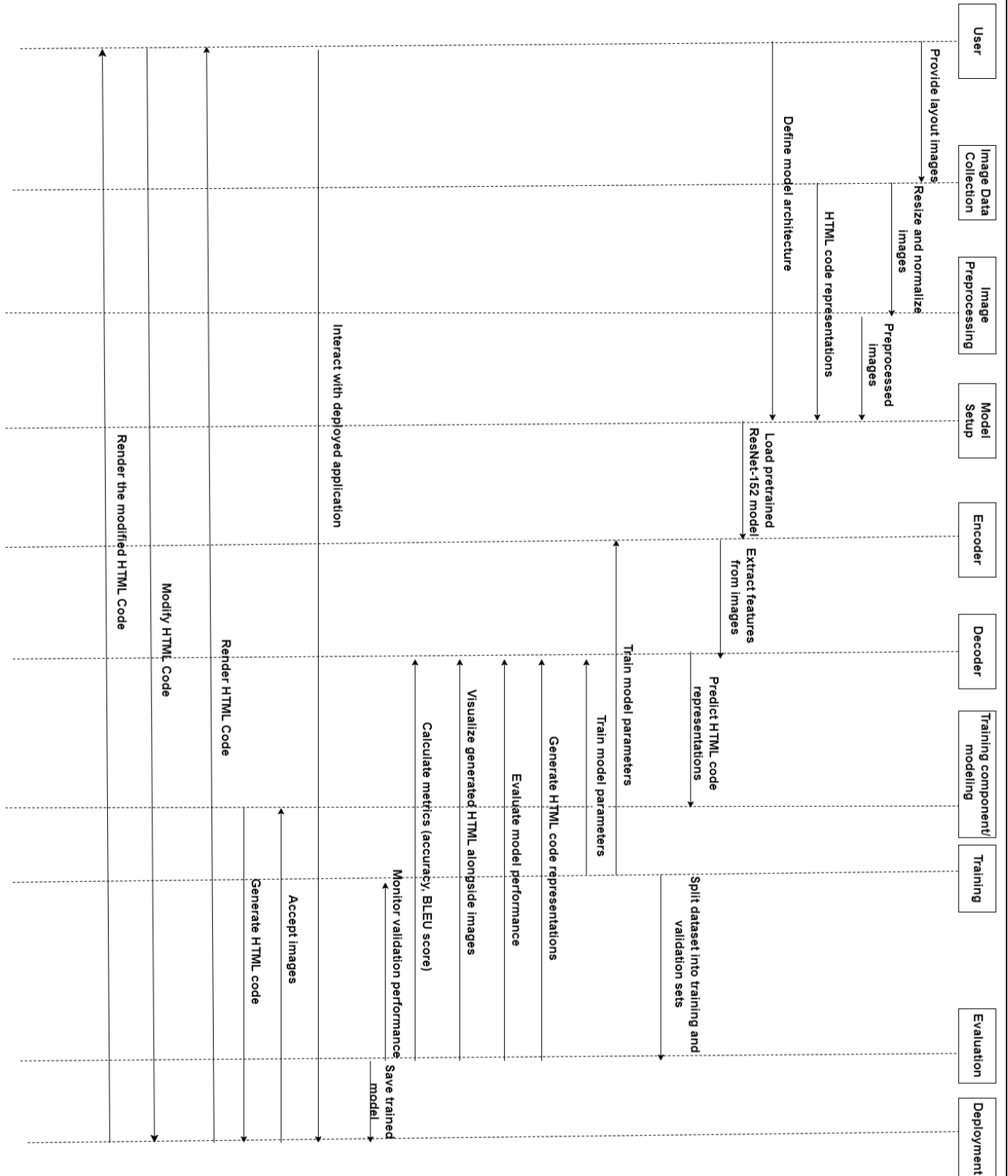


Figure 2 Sequence diagram: Flow and Interactions in an image to HTML Code Generation System

Research Report

Software Engineering Graduation

1. The sequence diagram begins with the User, who delivers sketches of website layouts to the Image Data Collection component as input.
2. The Image Data Collection component takes the given images and preprocesses them by scaling and normalizing them.
3. The preprocessed images, together with the HTML code representations given by the User, are then delivered to the Model Setup component.
4. The model architecture is defined in the Model Setup component. The Encoder loads a previously trained ResNet-152 model, and the Decoder is configured.
5. The Encoder uses the ResNet-152 model to extract features from the preprocessed images.
6. The Encoder extracts features, which are then sent to the Decoder, which outputs projected HTML code representations.
7. After receiving the anticipated HTML code representations, the Training component divides the dataset into training and validation sets.
8. The training set is used to train the model parameters of the Encoder and Decoder.
9. The validation set is used to evaluate the training model. The Decoder provides HTML code representations for examination and calculates performance measures like accuracy and BLEU score.
10. The Evaluation component displays the produced HTML code with the images for qualitative evaluation.
11. The Evaluation component additionally tracks validation performance to analyze the model's progress.
12. If necessary, the model is fine-tuned, in which hyperparameters are modified or regularization techniques are used.
13. Finally, the trained model is stored and integrated into a deployed application via the Deployment component.
14. The user can interact with the deployed application by providing images of website layouts as input.
15. The Deployment component takes the images and sends them to the Decoder, which creates the HTML code with Bootstrap.
16. The Deployment component takes the images and feeds them to the Decoder, which creates the relevant Bootstrap HTML code.
17. The produced HTML code is then returned to the user as the result of the Image to Code system.
18. The user can modify the produced HTML code and send a save request.
19. The new modified HTML code is sent back to the user so the user can download the HTML code.

The above sequence diagram illustrates the flow of interactions and phases in the Image to Code tasks, showing interactions between various components and the user throughout the process.

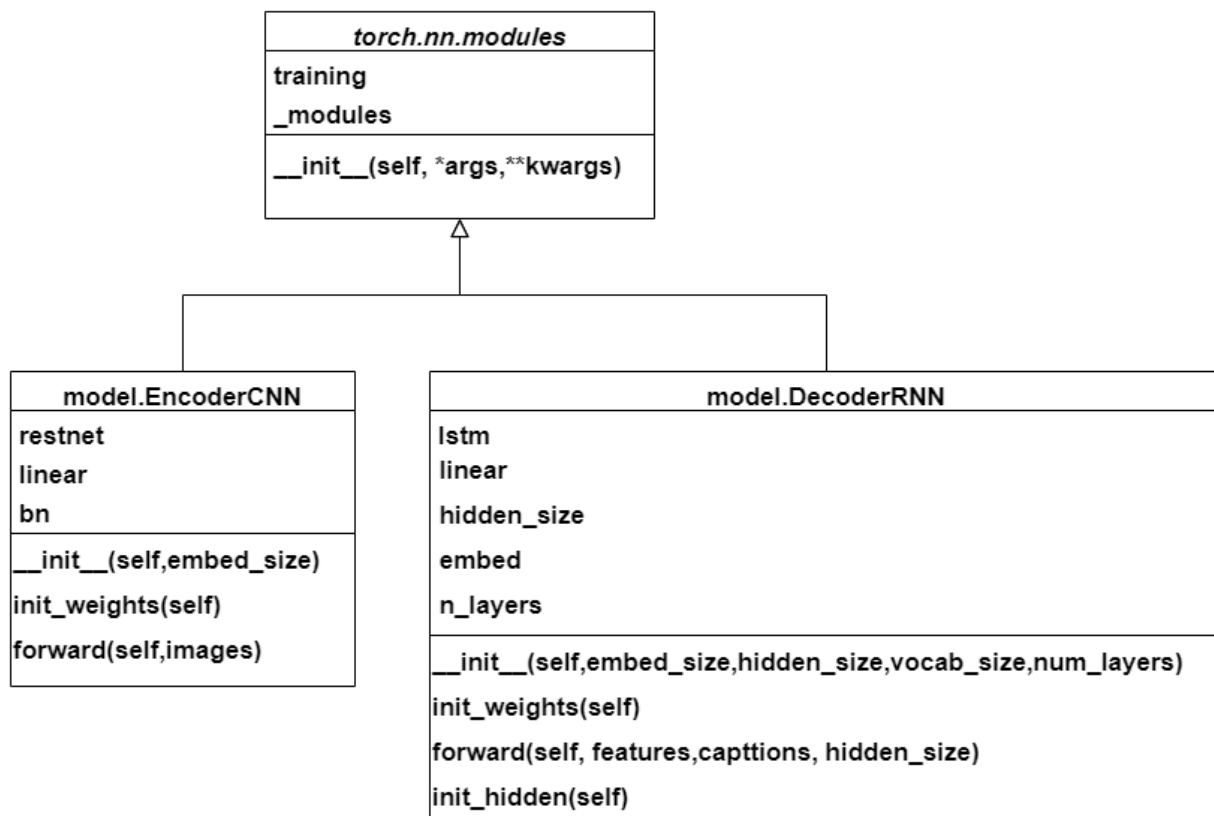


Figure 3 Sequence diagram: Flow and Interactions in an HTML Code Generation System

The encoder-CNN and decoder-RNN are components of a machine learning system that translates images to code that performs specific tasks in the overall architecture. The PyTorch library is commonly used to implement these components.

The encoder-CNN is in charge of extracting useful information from the input image. It starts with an image and runs it through a sequence of convolutional and pooling layers. Convolutional layers apply filters to the input image, collecting diverse visual patterns at different sizes. Pooling layers reduces the spatial dimensions of feature maps while keeping critical information. The encoder-CNN output is a concise representation of the image's features.

The decoder-RNN (Recurrent Neural Network) generates the code or sequence of tokens that represent the output based on the encoded image attributes. It accepts as input the encoded features and generates the code token by token. Because they can create sequential output based on past outputs and hidden states, recurrent neural networks (RNNs) are often used for this purpose.

We can build an end-to-end image-to-code system by merging the encoder-CNN and decoder-RNN. The encoder-CNN retrieves relevant features from the image, and the decoder-RNN constructs the matching code based on those features. Using proper loss functions and optimization techniques, the training procedure often entails reducing the difference between the produced code and the ground truth code.

8.2.2 Text To Bootstrap HTML Code:

The following high-level architecture shows gathering a dataset of code descriptions and accompanying bootstrap HTML code to train the model. Following data collection, the collected dataset requires preprocessing, which includes cleaning and tokenization. The fine-tuning stage uses the dataset to update the pre-trained GPT-3 model, allowing it to recognize patterns and correlations between code descriptions and HTML code (OpenAI API,n.d.). Once fine-tuned, the model can produce bootstrap HTML code based on input text or code descriptions, making it a handy tool for automating code-generating tasks.

1. **Data Collection:**

A dataset including pairs of code descriptions and related bootstrap HTML code must be collected. The code descriptions act as prompts for input, whilst the HTML code reflects the intended outcome. To guarantee that the model learns the patterns and structure of bootstrap HTML, collect a broad and representative group of samples.

2. **Data Preprocessing:**

After the model has been trained, the acquired data must be preprocessed. This will involve cleaning the data to remove any extraneous or distracting information. In addition, the input text and HTML code will be tokenized, which means they will be broken down into smaller units (tokens) such as words or sub-words. The data will be correctly prepared for training purposes as a result.

3. **Fine-tune GPT-3 Model:**

Using the preprocessed data, the GPT-3 model will be fine-tuned. Fine-tuning will comprise exposing the pre-trained GPT-3 model to the specific dataset to adjust it to the unique job. The GPT-3 from OpenAI allows for fine-tuning, allowing for training on a customized dataset. The model will be able to understand the deep patterns and interactions between code descriptions and bootstrap HTML code as a result of this fine-tuning process.

4. **Code Generation:**

Once fine-tuned, the GPT-3 model will be capable of creating bootstrap HTML code based on input text or code description. The model will be provided with the code description or prompt as input to produce the code. The model will then create the relevant HTML code as output.

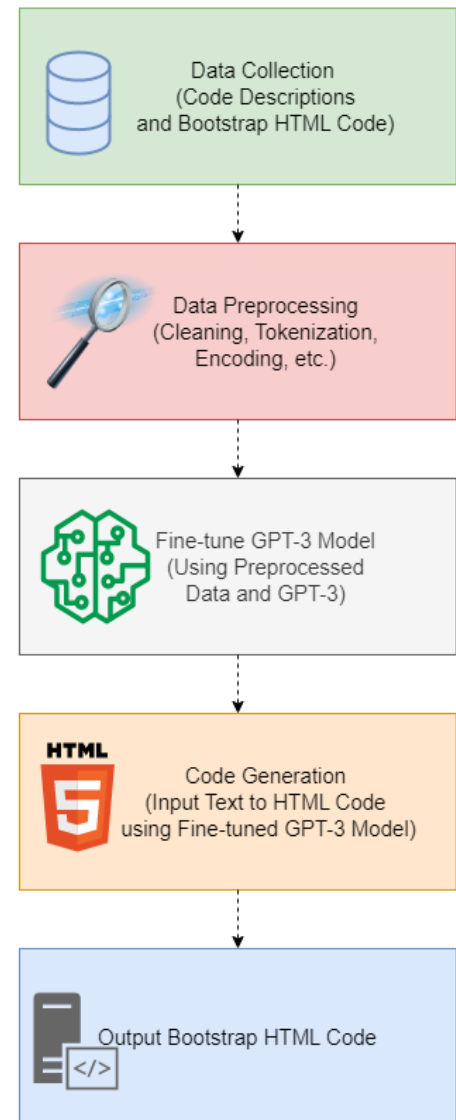
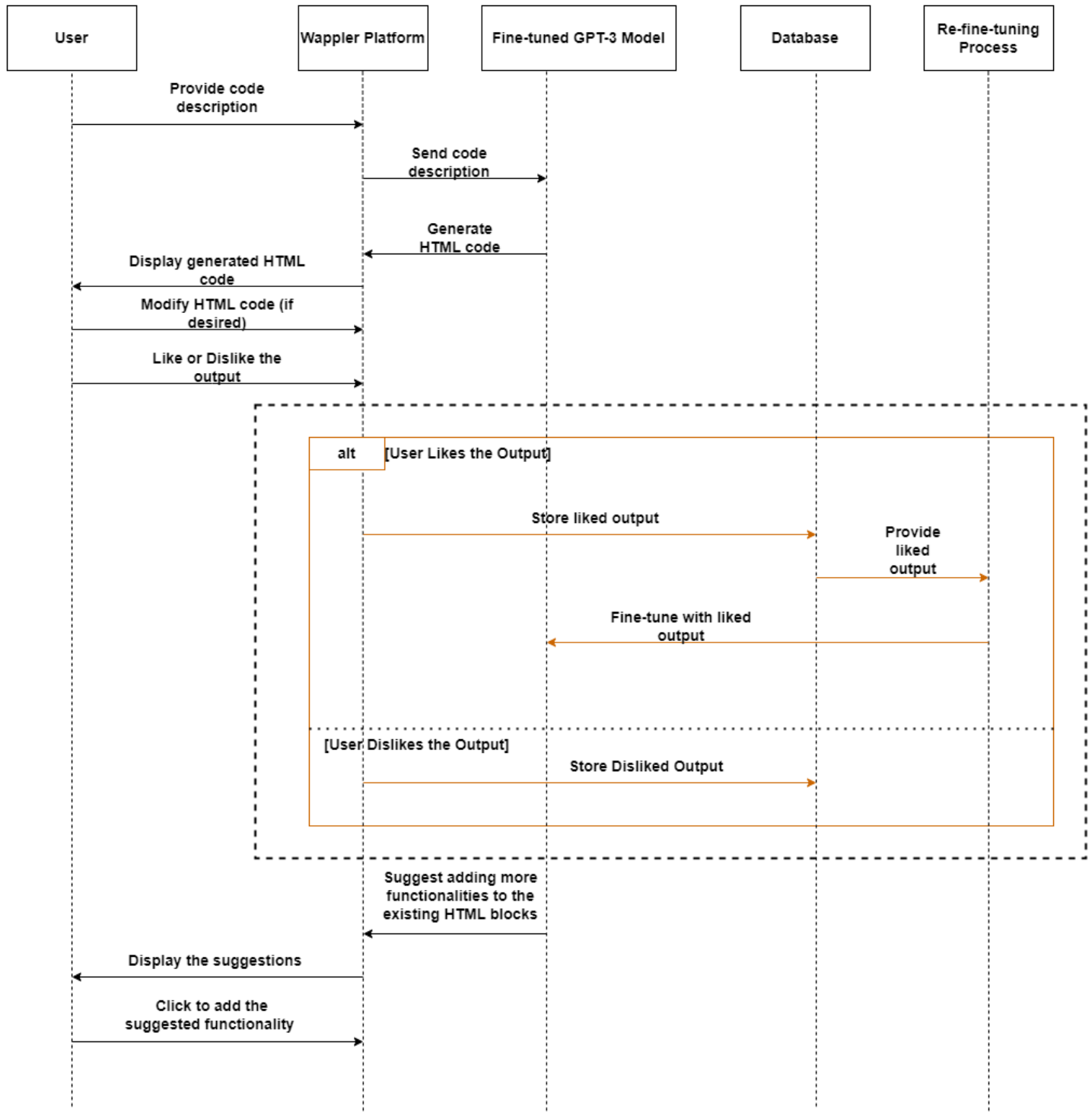


Figure 4 High level architecture: Using a Fine-Tuned GPT-3 Model to Generate Bootstrap HTML Code from Code

Figure 5 Sequence diagram: Interactive Code Generation with User Feedback and Re-Fine-Tuning in Wappler platform using the fine-tuned GPT-3 model



Research Report

Software Engineering Graduation

The flow of interactions between the user, the Wappler platform, and the fine-tuned GPT-3 model is shown in the above sequence diagram. The user enters a code description text, which is forwarded to the Wappler platform. Based on the description, the platform then employs the fine-tuned GPT-3 model to create HTML code with Bootstrap. The resulting HTML code is shown to the user, who may make changes as needed.

Future and optional implementations are shown in the dashed box in the above diagram: When the user can indicate their happiness by liking or disliking the output. If a user loves the created code, the Wappler platform records this information. If the user does not like the output, the Wappler platform records it as well.

The Wappler platform keeps the liked output in a database when the user displays an interest in it. The liked output that has been saved can be used as data for the re-fine-tuning procedure. The re-fine-tuning procedure combines the previously saved liked output into the already fine-tuned GPT-3 model to boost its performance even further.

In addition, the fine-tuned model in the Wappler platform can make suggestions to the user to add more functionalities to the existing Bootstrap HTML code blocks, if the user likes the suggestions, he can add it with only one click.

The data flow from the Wappler platform to the database, and then to the re-fine-tuning procedure, is now shown in the sequence diagram. This guarantees that the user's preferred results contribute to the model's continual development.

8.3 Frontend and Backend Classification

The terms "frontend" and "backend" relate to the separation of responsibilities and components between the user-facing interface and server-side operations. The categorization for both the image-to-code and text-to-code systems is shown below.

8.3.1 Image To Bootstrap HTML Code:

Frontend: The frontend refers to the application's client side, which is in charge of displaying the user interface and communicating with the user. It will run in the user's browser and interact with the backend via HTTP requests and responses. The following parts will make up the frontend component:

1. **User Browser:** This is the interface through which the user will interact. The web page will be shown, and the user will be able to submit an image for processing.
2. **User Browser Response:** After the backend creates the processed text and HTML code, it sends it to the user's browser as an HTTP response. After receiving the response, the browser will show the created HTML code, resulting in a visual representation of the processed content.

The major responsibility of the frontend will be to provide a user-friendly interface for uploading the image, obtaining the processed results, and presenting the output in the browser.

Research Report

Software Engineering Graduation

Backend: The backend refers to the application's server side. It will be in charge of the logic, data processing, and communication with other components and external services. The backend will be created in this scenario using the Wappler platform, which is a powerful web development platform. The backend component will consist of the following elements:

1. **Wappler Server:** It will accept HTTP requests from the user's browser and handle the associated logic. It will be in charge of forwarding requests to the proper services and providing HTTP replies.
2. **Image Resizing API:** This component will receive the picture from the Wappler server, resize it, and prepare it for further processing. It will carry out the necessary procedures to change the image as needed.
3. **ML Model (Encoder/Decoder):** This component will receive the scaled image and use a machine learning model to handle it. Based on the image content, it will produce the corresponding text. For image processing and text creation, the ML model will include complicated algorithms and neural networks.
4. **HTML Compilation Function:** After the ML model generates the processed text, this function will take it as input and transform it into HTML code. It might entail producing dynamic HTML content based on the processed text.

The backend will largely be concerned with processing the user's input (images), conducting complex tasks (image resizing and ML processing), and producing suitable output (processed text and HTML code).

To summarize, the backend will handle processing, logic, and connectivity with other services, whilst the frontend will focus on providing a user interface and rendering output in the user's browser. They will collaborate to develop a fully functional online application that will allow users to upload images, process them, and display the results.

8.3.2 Text To Bootstrap HTML Code

Frontend: The frontend is the component of the program or website that is visible to users. It involves the interface via which users will interact with the GPT-3 model to create HTML code. HTML, CSS, and JavaScript may be used to create the frontend. The following is an overview of the frontend components:

1. **User Interface (UI):** Create an intuitive and user-friendly UI that allows users to enter code descriptions or text inputs and submit them for HTML code production.
2. **Input form:** Create an input form where users may submit the code description text to be converted into HTML code.
3. **Submit Button:** Provide a button for users to click to send the input text to the backend for processing.
4. **Output display:** Set up a UI area where the produced HTML code will be shown after it is received from the backend.

Backend: The server-side logic and architecture that handles user requests, interacts with the GPT-3 model and returns the resulting HTML code is referred to as the backend. In this situation, the backend will handle communication with the GPT-3 model as well as provide the functionality required to interact with the Wappler platform.

The following is an overview of the frontend components:

1. **Server Infrastructure:** Set up a server environment where the backend code will be hosted. This is possible with technologies like Node.js, and Python.
2. **API Integration:** Use the OpenAI GPT-3 API to transmit the user's input text and retrieve the HTML code produced. API requests and parsing of the GPT-3 API response will be handled.
3. **Data Processing:** Once the frontend receives the user's input, preprocess the data to ensure it is in a format acceptable for input to the GPT-3 model. Tokenization, deleting extraneous characters, and other changes may be required.
4. **Interaction with the GPT-3 Model:** Use the fine-tuned GPT-3 model to produce HTML code from the specified input text. Using the OpenAI API, send the preprocessed input to the GPT-3 model and obtain the resulting HTML code as the response.

8.4 User Interface Design

The tool would need to be simple to use, intuitive, and have an interface that made it easy for users to upload their images or write their code descriptions. For the output to meet a variety of web design requirements, it would need to be adaptable. Here are a few crucial design elements:

8.4.1 Wireframes

Before any actual coding is done, stakeholders can gain a clear understanding of how the product will appear and function by including wireframes in the functional report. Early on in the development phase, this can assist in identifying future issues or challenges, saving time and resources later on. The project's wireframe components are listed below:

1. **Dashboard:** The primary dashboard will offer quick access to all of the tool's essential capabilities, such as data import and code generation.
2. **Import images or write code descriptions:** With a straightforward interface, users will be able to import their design prototypes. The tool will automatically preprocess the images and produce an output HTML code preview **OR** users will be able to describe the code and then the tool understands the human language and generate HTML code.
3. **Possibilities for modification:** The modification options will be displayed in a straightforward and user-friendly interface, with options for adjusting the layout schemes, and more. Real-time previews of the user's modifications will be available.
4. **Output Code Generation:** If the user is happy with their changes, they can easily produce the HTML code for the final output with a single click. The code will be neat, organized, and simple to include in their project.

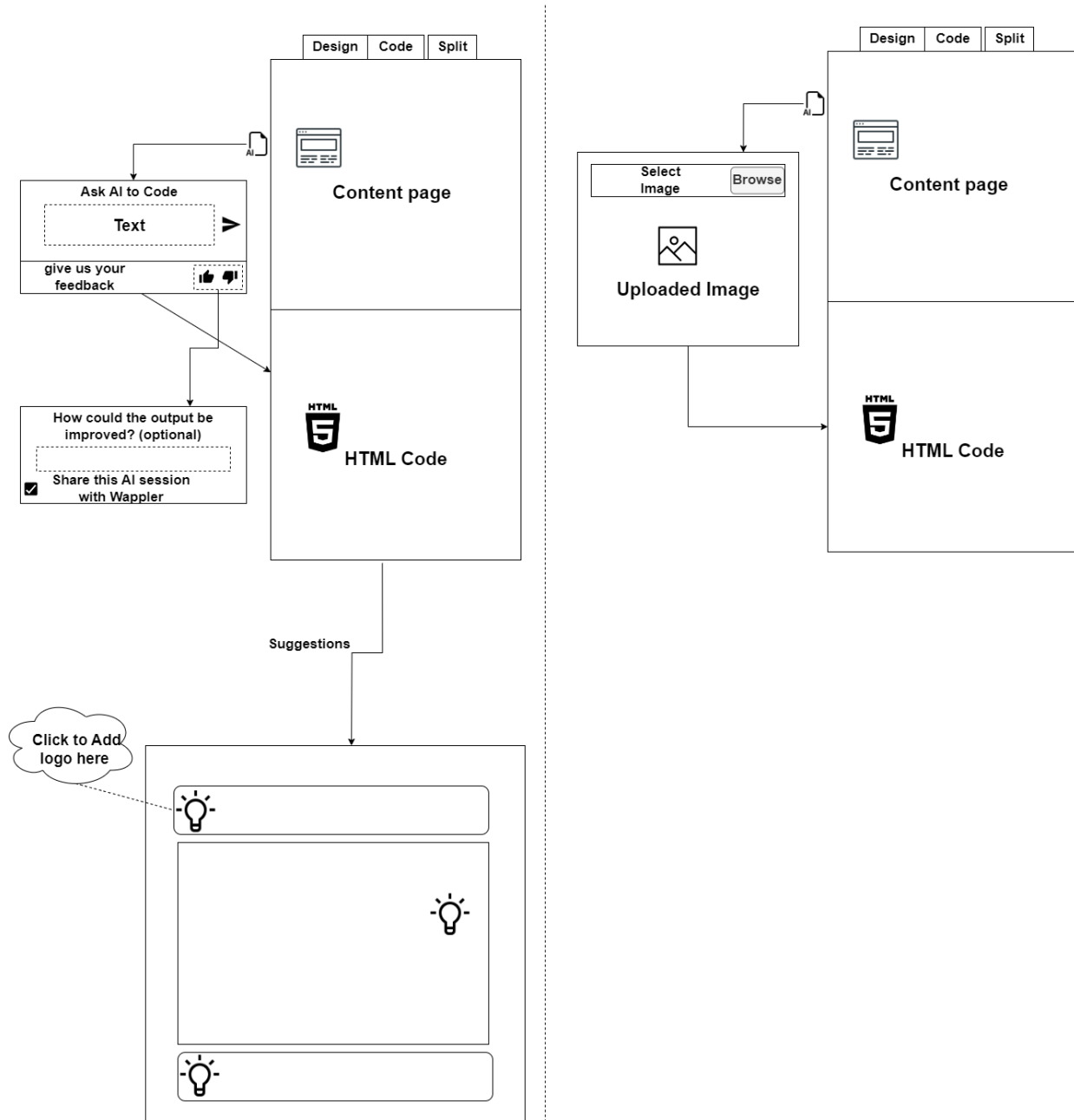
The tool's overall design aims to be effective, precise, and user-friendly, giving web designers and developers a strong tool for transforming their design mockups or code description text into useful HTML code.

Here is a wireframe for how the user interface will look in the frontend for either implementing image-to-code or text-to-code:

Research Report

Software Engineering Graduation

Figure 6 Wireframe design for converting Image to Code OR Text to Code



8.4.2 Product flow

The series of actions a user performs when interacting with a digital product is referred to as the "product flow." Here are the processes customers must take to convert sketches or translate code description texts into Bootstrap HTML code, covering everything from the beginning onboarding procedure to the last steps completed before leaving the product.

1. The user selects the "Image To Code" button and selects the sketch for the website layout or selects the "Text To Code" button and writes the code description text.
2. The user needs to upload the image in the tool or submit their code descriptions.
3. The tool preprocesses the images or code description texts and generates a preview of the output HTML code.
4. The user can modify the output HTML code in the code editor and immediately see the results of their changes in the design area.
5. When the user is happy with the changes, they may produce the HTML code for the final output.
6. The user can use the generated code for their projects.

The product flow is overall made to be easy and intuitive, enabling customers to import their data or write code description texts, adjust the output code, and produce the finished HTML code quickly and easily. Users of various skill levels can use the product effectively and efficiently by maintaining a clear and user-friendly interface.

9. Design choices

To start, a variety of potential users, such as web designers, developers, team leaders, and business owners, were taken into consideration while choosing the use cases. In doing so, it is made sure that a variety of people can use and access the product.

To ensure precise and effective conversion of design mockups or code description texts into HTML code, I decided to integrate machine learning techniques with the HTML framework for the system architecture. I can increase the tool's accuracy and make sure it works with a variety of design elements or a variety of code description texts and phrases by training the machine learning algorithms on a big dataset of design mockups for website layouts or HTML code description texts.

The flexible and user-friendly modification options enable customers to modify the output HTML code to suit the particular requirements of their project. Users may simply adjust their output code without having to have advanced coding knowledge due to a straightforward interface that makes changing code structures easy.

Lastly, the user interface was created to be intuitive and effective, with a user-friendly interface for adding design mockups or writing code description texts and a straightforward dashboard for accessing all of the tool's essential capabilities. I can guarantee that all users, regardless of ability level, can use the tool successfully and efficiently by keeping the interface straightforward and user-friendly.

Overall, the design decisions were taken to produce a tool that is precise, effective, and user-friendly, giving web designers and developers a potent means of turning their design mockups or ideas into usable HTML code with Bootstrap.

10. Implementation

Python is a popular programming language that may be used for a variety of tasks, including image processing and natural language processing (NLP). In this implementation, Python was used to create a machine learning program that can convert images to code and code description texts to code.

Image-to-code conversion is an interesting and difficult task in computer vision research. This includes converting an image into a structured HTML code format that can then be analyzed. Image-to-code conversion methods include deep learning-based approaches, rule-based approaches, and hybrid approaches. A hybrid strategy that blends deep learning and rule-based methods is used in this implementation. The deep learning model extracts features from the image, and the rule-based technique converts the features to a structured Bootstrap HTML code format.

The second stage of implementation entails converting text to HTML code. Text is processed and translated into a structured format that may be used for additional analysis, which is a common activity in NLP research. Fine-tuned GPT-3 with a specific dataset in this version to translate the code description text into HTML code was explained.

Overall, this implementation demonstrates Python's potential for image processing and natural language processing tasks.

10.1 Abstract

Recent improvements in machine learning have created new opportunities for automating web development processes. The purpose of this study is to investigate two different approaches for applying machine learning techniques for web development.

The first focus of the research is on automatically converting the web graphical interface mockups into Bootstrap HTML code. To convert screenshots of websites into code, this project developed a PyTorch implementation of an image-captioning model. The system feeds images into a ResNet-152-based Convolutional neural network (CNN) encoder model, which generates features for a decoder RNN model using word embeddings and either a gated recurrent unit (GRU) or long short-term memory (LSTM). Then, the trained model needs to be incorporated into the Wappler platform to create an online tool that essentially enables users to transform a hand-drawn image or a screenshot from a website into HTML code. The suggested method is trained using the Pix2Code dataset, and the models have been trained over a large dataset of web page Graphical User Interface(GUI) designs and their corresponding Domain-specific language (DSL) code in numerous batches. The project resulted in a peak Bilingual Evaluation Understudy (BLEU) score plateauing around 92 % after a few 32 epochs.

Furthermore, the study investigates the fine-tuning of GPT-3. GPT-3 has appeared as a game changer in the field of natural language processing, with the potential to generate human-like text. While OpenAI pre-trains the model, it can be fine-tuned on a smaller dataset for a specific application. The goal of the research is to collect information about Bootstrap HTML code and the code description texts that go with it. Using this data to fine-tune GPT-3, it will construct a highly flexible HTML Bootstrap code generator that understands code descriptions and phrases.

GPT-3 fine-tuning on Bootstrap HTML coding represents an innovative technique for creating highly responsive and user-friendly websites. HTML Bootstrap code generator that understands the code description texts and generates code accordingly by fine-tuning the model was created.

Research Report

Software Engineering Graduation

This reduces the time and effort required to create responsive web pages, allowing developers to focus on other elements of web development. The findings show the value of fine-tuning GPT-3 for web development and set the path for future advances in this field.

Keywords: Machine Learning, HTML, Deep Learning, Sketch-to-Code, Artificial Intelligence, Convolutional neural network, Domain-specific language, Graphical User Interface, Recurrent neural network

10.2 Introduction

Computer vision is a branch of artificial intelligence (AI) that enables computers and systems to extract useful information from digital photos, videos, and other visual inputs and to execute actions or make recommendations based on that information (IBM, n.d). The subject of this paper will be image recognition as it has always been a popular research topic in computer vision. According to Burns and Brush (2021, March 29), in the context of machine vision, image recognition refers to a computer program's capacity to recognize items, locations, persons, writing, and actions in digital photographs. To recognize images, computers can use machine vision technology in conjunction with a camera and artificial intelligence (AI) software.

Every website has a fundamental framework that can be seen in a wireframe, which highlights the essential components and offers Wappler users ideas for constructing a site structure. Users of Wappler have asked for a mechanism that can transform their design into HTML code. This request is being made because the time-consuming work slows down the design process, which starts with brainstorming on a whiteboard. An image of a design is taken once it is made, and it is then manually transformed into a usable HTML wireframe that can be viewed in a web browser. Web-based software that transforms a hand-drawn user interface sketch from an image to a valid HTML markup code is what Wappler's users wish to employ artificial intelligence for.

The use of deep learning techniques in front-end development is becoming more popular. Examples include Tony Beltramelli's pix2code and Airbnb's "Sketching Interfaces" project. Applying deep learning to the Wappler platform has the potential to more effectively automate the creation of code based on designer mockups and to provide Wappler users with a more productive and efficient development experience. Wappler will be used to create a web-based interface that uses AI to generate HTML code from rough layout images or screenshots of website layouts.

This study introduces a web-based AI tool for converting website GUI screenshots into code in PyTorch. It involves understanding how to create a server-side action that uses the Wappler server to output HTML code, executes a Python-based tool, and manages a GUI screenshot of a Bootstrap-HTML-based website. The system feeds images into a ResNet-152-based encoder CNN to extract image features, and then a decoder RNN that uses word embeddings, and GRU or LSTM to create a sequence of predicted DSL tokens that are then transformed to valid HTML markup code using Compiler.

Furthermore, this study investigates the fine-tuning of pre-trained models. This method entails training the model on a smaller dataset that is tailored to the job at hand. As a result, the model can be "tuned" to be more efficient and accurate for the task at hand.

Research Report

Software Engineering Graduation

Creating responsive and user-friendly web pages is a time-consuming and hard undertaking in web development. Because of its flexibility and simplicity of use, HTML Bootstrap code is a popular choice for designing such websites. Manually creating HTML Bootstrap, on the other hand, can be a demanding undertaking needing substantial knowledge and skill.

By fine-tuning GPT-3 on Bootstrap HTML code, this difficulty and simplify of the web-building process were addressed in this paper. The research focuses on gathering data on Bootstrap HTML code and its corresponding code descriptions, which will be employed to fine-tune the GPT-3 model. The final HTML Bootstrap code generator will be extremely personalized and detailed, capable of recognizing code description texts and phrases. This would considerably reduce the time and effort required to produce responsive web pages, making web development more accessible to a wider audience.

The following goals can be achieved by Wappler's users using an AI system that can translate both images for webpage layouts and code description text into Bootstrap HTML code:

1. **Usability:** Wappler's users can construct web pages more easily, even if they do not have much knowledge of HTML coding, by employing an AI system that can convert both code descriptions text and images for webpages layouts into Bootstrap HTML code.
2. **Time-saving:** By automating the process of generating HTML code from both code descriptions text and images for webpages layouts, an AI system can help Wappler's users save time. They might then concentrate on other components of web development as a result.
3. **Productivity:** By enabling users of Wappler to generate HTML code more rapidly and effectively, AI systems can assist its users gain productivity.
4. **Accuracy and consistency:** An AI system can make sure that the HTML code created from both an image and text is correct and consistent. This might help to increase the website's overall quality and make sure it performs properly with an average of 92% accuracy
5. **Innovation:** A cutting-edge method of web development is the employment of an AI system to translate both texts and images into HTML code markups. Customers of Wappler can keep on top of trends and build more interactive, engaging websites by using cutting-edge technologies.

10.3 Techniques

The following approaches are detailed below, with examples of how they were used in this project:

[Computer Vision Techniques:](#) This section discusses digital image analysis and understanding, including colour recognition and edge detection.

[Machine Learning Techniques:](#) The focus here is on using deep learning techniques, especially Convolutional Neural Networks (CNNs) like ResNet-152, for recognizing image tasks.

[Techniques for Fine-tuning:](#) This section addresses the approaches used to improve models, such as GPT-3, through data preparation and fine-tuning for specific project goals.

10.3.1 Computer vision techniques

1. *Colour Detection*

In computer vision techniques, according to Technology Robotix society (2019, July 4), the process of discovering and recognizing colours in digital images or video frames is known as colour detection. In image processing and computer vision, it is a basic component.

The following steps are generally included in colour-detecting algorithms:

1. **Image acquisition:** The initial phase essentially takes or loads an image into memory.
2. **Preprocessing:** preprocessing is next applied to improve the image's quality, lower noise, and normalize its colours.
3. **Conversion of the image's colour space:** The image is changed from its original colour space to one that is more appropriate, such as RGB, HSV, or Lab.
4. **Colour segmentation:** Using methods like thresholding, clustering, or region growth, the image is divided into various regions based on how well its colours match.
5. **Feature extraction:** Each region is given a set of features to reflect its colour distribution, such as colour histograms or colour moments.
6. **Classification:** Each region is then classified into several colour categories or objects using the extracted features.

2. *Edge Detection*

Edge detection is a fundamental computer vision approach for identifying the borders of objects and features inside an image. Detecting the locations in an image where the brightness of adjacent pixels changes significantly, resulting in an edge, is the technique. This approach detects object boundaries, which are then used to segment an image into separate components.

10.3.2 Machine learning techniques

1. *Deep learning*

A branch of machine learning known as "deep learning" involves teaching artificial neural networks how to learn from massive amounts of complex data and predict the future. Many different domains, including computer vision, natural language processing, speech recognition, and many more, have effectively used deep learning (Yasar, 2023).

2. *Convolutional Neural Networks(encoder)*

According to Microsoft/resnet-152 on Hugging Face (n.d), the ideas of skip connections and residual learning were made more accessible by the convolutional neural network known as ResNet (Residual Network). This makes it possible to train even deeper models. ResNet model pre-trained on ImageNet-1k at resolution 224x224.

Convolutional neural networks (CNNs) with the ResNet-152 architecture which includes a combination of convolutional layers, pooling layers, and fully connected layers are frequently employed for computer vision and image recognition tasks. It is a member of a series of ResNet models that were developed to solve the issue of vanishing gradients in deep neural networks.

According to Great Learning Team (2022, October 31), the network's layer count is indicated by the "152" in ResNet-152. ResNet-152 is a highly deep neural network because it has 152 layers. Residual connections, which enable information to skip some tiers and go across the network more quickly, are

Research Report

Software Engineering Graduation

one of the major features of the ResNet architecture. As a result, ResNet-152 can be trained more successfully than previous deep neural network architectures, addressing the issue of disappearing gradients.

Before advancing to the decoder, it is necessary to remove the ResNet-152 model's last collecting layer to intercept detected features and replace it with a Linear PyTorch layer.

3. Recurrent Neural Networks(decoder)

According to Loye(2023, January 23), an RNN decoder is a form of a decoder that creates each component of the output sequence using recurrent neural networks. The decoder creates a new output element and a new hidden state at each time step by using the previous output element as input. The encoder creates a context vector that is used to update the hidden state while also using the prior hidden state.

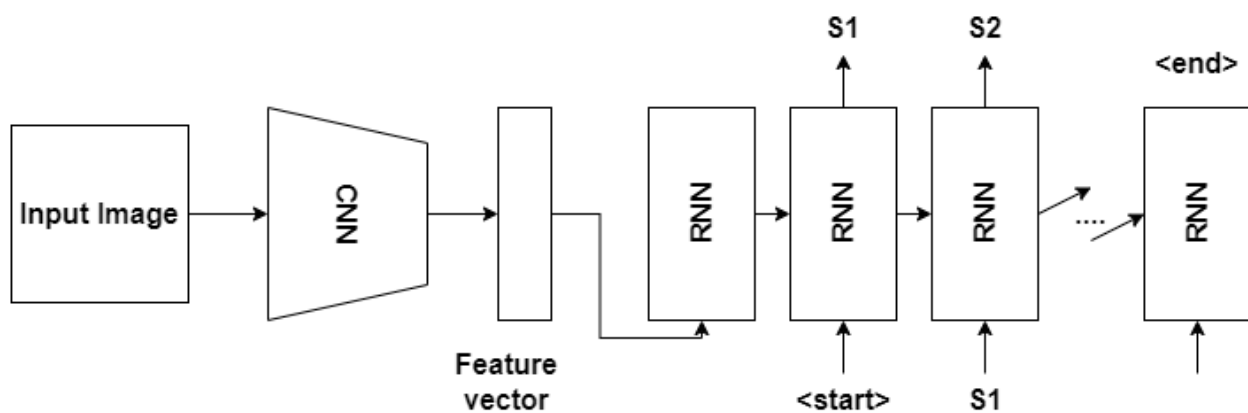


Figure 6 CNN Encoder and RNN Decoder

4. Pytorch

PyTorch is a free and open-source machine learning framework that includes a variety of tools and functions for creating and training neural networks. It is commonly used in research and industry to create cutting-edge deep learning models such as image and natural language processing models, and many more.

PyTorch also includes several high-level abstractions and APIs for designing and training neural networks, such as the (nn module) for defining layers, loss functions, and optimizers.

Furthermore, PyTorch has a big and active developer community that contributes to the framework and provides help via forums and online resources. Overall, PyTorch is a strong and flexible machine learning framework that allows developers to easily design and train cutting-edge deep learning models.

5. OpenCV Python

OpenCV (Open Source Computer Vision) is a free and open-source programming function library designed mainly for real-time computer vision applications. It was created by Intel and is now supported by Willow Garage and Itseez. The library is developed in C++, but it also offers Python and Java bindings, as well as hundreds of computer vision algorithms for image processing.

10.3.3 Fine-tuning Techniques

1. *Data Preprocessing*

It is necessary to preprocess the dataset to reduce noise, mistakes, and inconsistencies so that the GPT-3 model may be trained on clean and correct data. Cleaning up the HTML code and standardizing the text may be required.

2. *Fine-tune Methodology*

It is essential to select a suitable fine-tuning process to ensure that the GPT-3 model is trained efficiently and successfully on the dataset. Transfer learning, domain adaptation, and few-shot learning are some of the fine-tuning approaches available.

10.4 Methodology

A hybrid strategy called "image-to-code" that combines rule-based and deep learning techniques is used. Input images are processed using rule-based techniques to convert the extracted features from deep learning models into structured HTML code. The GPT-3 model (Davinci) has also been improved to produce code based on textual prompts for text-to-code conversion. Below is an explanation of the dataset, languages, and frameworks:

10.4.1 Language and frameworks

The boundaries of this project are to use the Wappler environment, so the programming will be done in **Electron** with **JavaScript** and **NodeJS** and use **Python** for the machine learning models. The research on the real-world uses of artificial intelligence (AI) in the low-code development tool Wappler outlined in this graduation assignment can be conducted using a variety of frameworks and technologies. Several examples include:

- Computer Vision frameworks such as OpenCV or TensorFlow for layout-block generation and analysis.
- Python Flask App to deploy the model.

Furthermore, GPT-3 is being fine-tuned for HTML generation. A multitude of computer languages and frameworks can be used to create Bootstrap HTML code. Python, on the other hand, is a popular choice due to its ease of use and the availability of modules for natural language processing and deep learning.

- Natural Language Processing (NLP) frameworks such as spaCy and GPT-3 for text generation and analysis.
- The OpenAI API and its different AI types: These tools can be used to access a wide range of advanced AI capabilities, including natural language processing. Examples of AI types include GPT-3.

10.4.2 Dataset and Features

To train and evaluate the hybrid image-to-code conversion technique, a dataset including pairs of images and their corresponding HTML code representations is required to train and assess the hybrid image-to-code conversion process. However, because expert-level annotations are required, obtaining such a dataset is a substantial problem. Nonetheless, in my search to solve this problem, I found an open-source dataset that will be a helpful resource for this project.

In my research, the Sketch2Code and Pix2Code datasets have emerged as the key sources of analysis. These datasets include image and Domain-Specific Language (DSL) markup pairs that are useful for training and evaluating the hybrid methodology's effectiveness. The Sketch2Code collection provides picture drawings of user interfaces together with their accompanying HTML code representations, whereas the Pix2Code dataset includes screenshots of web pages and their related HTML code.

In this study, a hybrid approach known as "image-to-code" that combines rule-based and deep learning techniques is used. The method involves converting extracted features from deep learning models into structured HTML code by processing input images using rule-based techniques. Combining these two approaches creates an extensive plan for effectively converting visual data from images into useable, well-structured HTML code. Preprocessing and feature engineering are critical steps in preparing input data for the hybrid technique.

Image Processing: Several preprocessing procedures are performed on the images before they are fed into the deep-learning model. Typically, these procedures involve scaling the images to a specific resolution and normalizing pixel values. The cv2 package for preprocessing to ensure compatibility with the pre-trained ResNet-152 model was used in this research. The dataset is rescaled from its original size of 2400×1380 pixels to a standardized resolution of 224×224 pixels in this stage. By rescaling the dataset, I match it with the needs of the ResNet-152 model, allowing for effective feature extraction and subsequent image-to-code conversion.

Feature extraction: From the preprocessed images, the deep learning model extracts relevant information. These features are derived from multiple layers of the model's architecture, and they capture both low-level and high-level visual representations.

Research Report

Software Engineering Graduation

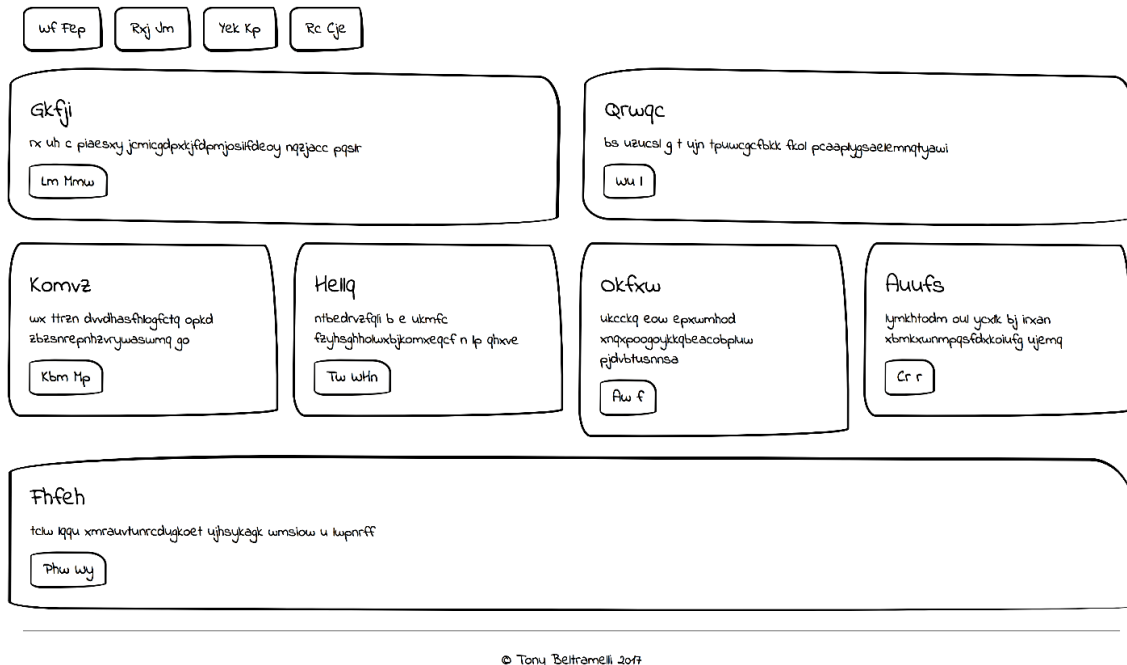


Figure 7 One Image from Sketch2Code Dataset

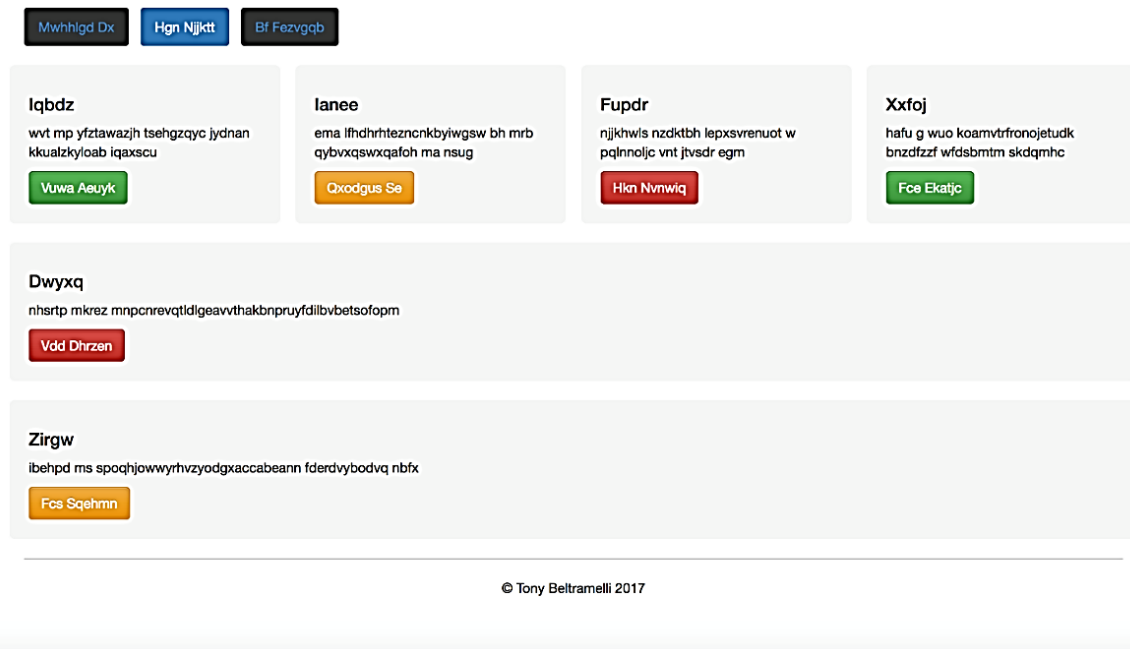


Figure 8 One Image from Pix2Code Dataset

Research Report

Software Engineering Graduation

To fine-tune GPT-3, It is critical to guarantee that the dataset used for fine-tuning GPT-3 is accurate and representative, a wide range of Bootstrap HTML code instances including different templates, styles, and components must be included. Because of this variety, the model may learn and generalize across diverse code structures and design aspects. It is also critical to provide any relevant metadata, such as template types, styles, or components, as well as any corresponding code descriptions. This new data contribute to the creation of a more informative and context-rich dataset, allowing the model to better grasp the link between code snippets and their descriptions.

Unfortunately, there is no open-source data that has HTML code and corresponding HTML code descriptions and phrases. This leads me to create a manual dataset that includes code descriptions and corresponding HTML code. First of all, a file of examples for fine-tuning, where each example is a prompt followed by the appropriate completion is needed. Also, it is important to make sure that the data set used for fine-tuning is of high quality and is representative of the task of generating HTML code using the model. The dataset consists of 1000 prompt-completion pairs. According to OpenAI API(n.d), all the prompts should end with the suffix `->`, start with the prefix `Wappler:` and all completions end with the suffix `\n###` as shown in the figure below.

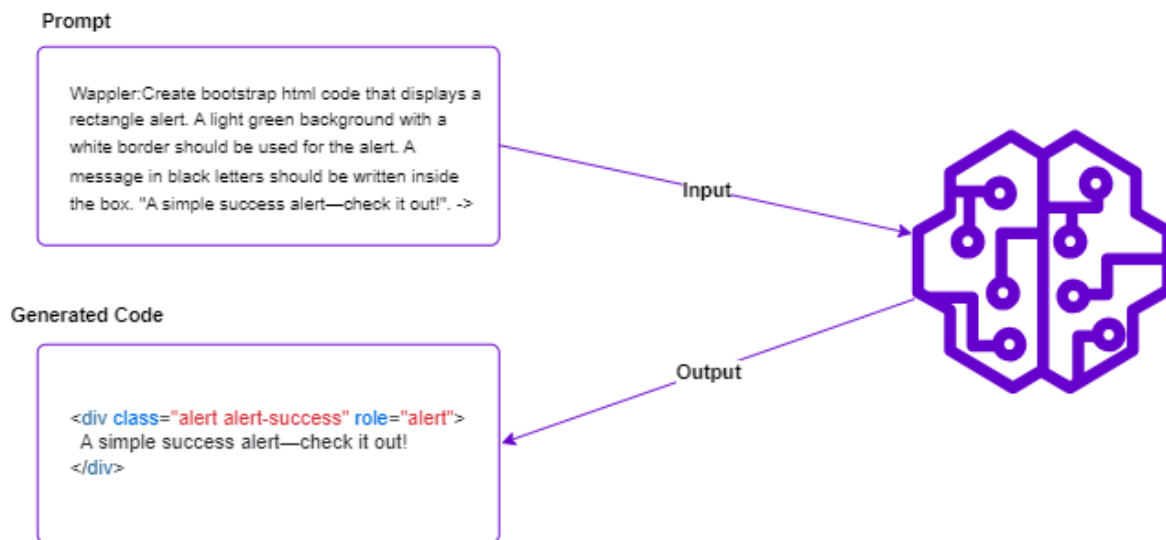


Figure 9 Input and Output of GPT-3 Finetuned model

10.5 Experiment

Experimenting with Image-to-Code Hardware, Software, and Model Parameters:

Throughout the investigation, extensive testing in a specific hardware and software environment is performed. The hardware configuration included an NVIDIA GeForce RTX 4090 GPU. Python 3.10.9, Conda list toolkit 11.8.9, torch 2.0.0. were used in the software environment, along with additional libraries like cv2 and NLTK.

Several configurations and parameters to improve the model's performance are experimented with. Modifying critical parameters such as embedding size, number of epochs, and hidden size was the focus. The dataset was divided into two parts: 2,500 samples for training and 500 samples for testing. The Convolutional Neural Network (CNN) with a variety of epoch counts ranging from 20 to 100 with a learning rate of 0.001 was trained.

To assess the model's performance, the Bilingual Evaluation Understudy (BLEU) ratings were used, a generally used grading method for image captioning models. The BLEU scores quantify the similarity between the predicted and ground truth token sequences. The values range from 0.0 to 1.0, with higher scores indicating greater similarity.

Aside from the experiment setup, a preparation process for the dataset was followed. First, a Dataset class was developed which was looped over the file names and chose those with the ".png" extension. The vocabulary was then built, beginning with the "START>" token and finishing with the "END>" token. To generate the vocabulary, a Tokenizer function was also used. The image data filenames were reviewed, resizing the matching images to satisfy the model's needs.

The experiment was divided into two parts: the Encoder CNN and the Decoder RNN. A pre-trained ResNet-152 model was used to replace the top fully connected (FC) layer in the Encoder CNN. The modified ResNet-152 was used to initialize the weights and extract image feature vectors.

In terms of the Decoder RNN, the hyperparameters were configured and built the layers. I initialized the weights for the Decoder RNN were initialized in the same way as the Encoder CNN. This component decoded the image feature vectors and created captions for the images.

Throughout the experiment, several setups were done to see how the model performed. For example, the ResNet-152 model for one batch size across 19 epochs was trained, resulting in an accuracy score of 70-90% and a loss score of 0.0716. In addition, the model with 32 mini-batches was also trained, which resulted in a pretty high loss score of roughly 17624.73. There were also experiments with tokenizing the DSL code language and making changes to the encoder layer, switching from a GRU to an LSTM. Although the model failed to predict meaningful outputs, training it in four batches for 25 epochs resulted in a poor accuracy score (0.000005) but an excellent loss score (0.00610). While increasing the number of epochs to 35 and employing 32 mini-batches resulted in a respectable loss score (0.0025), the accuracy remained poor (0.002-0.029), indicating probable overfitting.

Research Report

Software Engineering Graduation

GPT-3 Fine-Tuning for HTML Code Generation: Dataset Preparation and Model Development:

Several steps were carried out in this experiment, including data collection, dataset preparation, model construction, fine-tuning, model evaluation, and model deployment. As shown in the figure below:

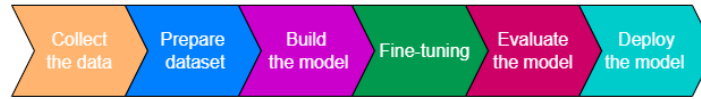


Figure 10 The Flow of Creating Text to HTML Code System

Using a carefully collected manual dataset, the GPT-3 model was fine-tuned for HTML code generation for the experiment. The dataset was created to offer a wide range of Bootstrap HTML code examples integrating various templates, styles, and components. Because no open-source dataset containing HTML code descriptions was available, a manual technique was used to construct a dataset of 648 prompt-completion pairings. Each prompt followed a certain format, beginning with the prefix "Wappler: " and finishing with the suffix "->," whereas the completions ended with the suffix "\n###." This guaranteed consistency and coherence within the dataset.

Python was chosen as the primary language to help in the fine-tuning process because of its versatility, ease of use, and availability of natural language processing (NLP) and deep learning modules. NLP frameworks such as GPT-3 were used for text creation and analysis, allowing for fast dataset preparation and modification. As well as essential libraries like Pandas and Matplotlib.

In this experiment, GPT-3 model was fine-tuned to create a (DaVinci:ft-wappler-2023-05-04-12-34-54) model. Then fine-tuned the already finetuned model to create (DaVinci:ft-wappler-2023-05-16-10-11-27) model. The models were trained using two different datasets. The first model was trained on a dataset of 648 prompt-completion pairs, while the second model was trained on a larger dataset of 1016 prompt-completion pairings. Each prompt in both datasets includes a description of the necessary HTML code, followed by the accompanying Bootstrap HTML code sample. The datasets were designed to cover a variety of components in Bootstrap 5.

Both models were developed using the GPT-3 architecture and fine-tuned using the relevant datasets. To improve performance, certain setups and characteristics were chosen for each model. The batch size was set to 1, the learning rate was set to 0.01, and the number of training iterations was set to 4. The architectures of the models, such as the number of layers and attention heads, were consistent with the fundamental GPT-3 model design.

These experiments allowed me to evaluate the model's performance in various configurations and draw educated judgments regarding the best setup for image-to-code conversion and text-to-code conversion.

10.6 Result

Optimizing Image-to-Code Conversion: Positive Results and Insights from Encoder-Decoder Models Using Pre-trained ResNet-152:

The experiment's findings showed good results after training the encoder and decoder models with the pre-trained ResNet-152 model as the basis. Notably, the model achieved a high BLEU score of 0.92 during the testing phase, showing a high level of similarity between the predicted and ground truth token sequences. This quantitative assessment indicator points to the efficiency of the image-to-code conversion approach.

Several key experiments were carried out to improve the model's performance, especially focused on differences in the number of epochs and the number of hidden layers inside the decoder. The goal was to find the best configuration that would produce the most accuracy. A consistent pattern was found across all experiments: increasing the number of hidden layers resulted in higher accuracy ratings. This indicates that the additional layers enabled advanced representations and a greater understanding of the input images.

It is important to note, however, that extending the training period beyond a certain point resulted in overfitting, which caused the model's performance to gradually degrade. Overfitting happens when a model becomes overly specialized to the training data, limiting its capacity to generalize successfully to previously unknown samples. As a result, when the model got highly fitted to the training dataset, its accuracy scores decreased.

This discovery points out the significance of achieving a balance between model complexity and generalization capabilities. The model may achieve optimal performance without overfitting by carefully selecting the proper number of hidden layers and monitoring the training duration. It is critical to locate the sweet spot where the model captures the underlying patterns and connections in the data without recalling individual examples.

Finally, the testing findings showed that the encoder and decoder models trained using the pre-trained ResNet-152 model produced outstanding results, as seen by the high BLEU score of 0.92. The following tests demonstrated the effect of varying the number of epochs and hidden layers on the model's accuracy. However, care must be taken to avoid overtraining, since an extended training duration might result in a reduction in performance. The image-to-code conversion process can produce the best results by finding the proper balance and producing accurate HTML code representations for a given input image.

Research Report

Software Engineering Graduation

Fine-tuned GPT-3 for HTML Code Generation: A Quantitative Analysis and Qualitative Insights:

This section presents the results of an experiment to fine-tune the GPT-3 model for producing HTML code from text prompts. The models were evaluated using quantitative and qualitative assessments, as well as a review of several performance measures.

Quantitative Analysis

Several criteria were used to assess the models' effectiveness in creating valid HTML code. Validation loss, training loss, validation token accuracy, and training token accuracy were among these measures.

648 prompt-completion pairings were included in the dataset used to train Model One (DaVinci:ft-wappler-2023-05-04-12-34-54); as a result, the validation loss fell from 0.642 to 0.0173 and the training loss from 0.9558 to 0.0086. To ensure optimal performance, the goal was to reduce the loss as much as possible. The training token accuracy dramatically increased during the duration of the training process, rising from a low value of 0.545 to a high value of 0.998. Similar to this, the validation token accuracy started at 0.6875 and increased over time to achieve the highest possible score of 1.0. These accuracy metrics show how the fine-tuning procedure improved the model's capacity to produce accurate and dependable HTML code.

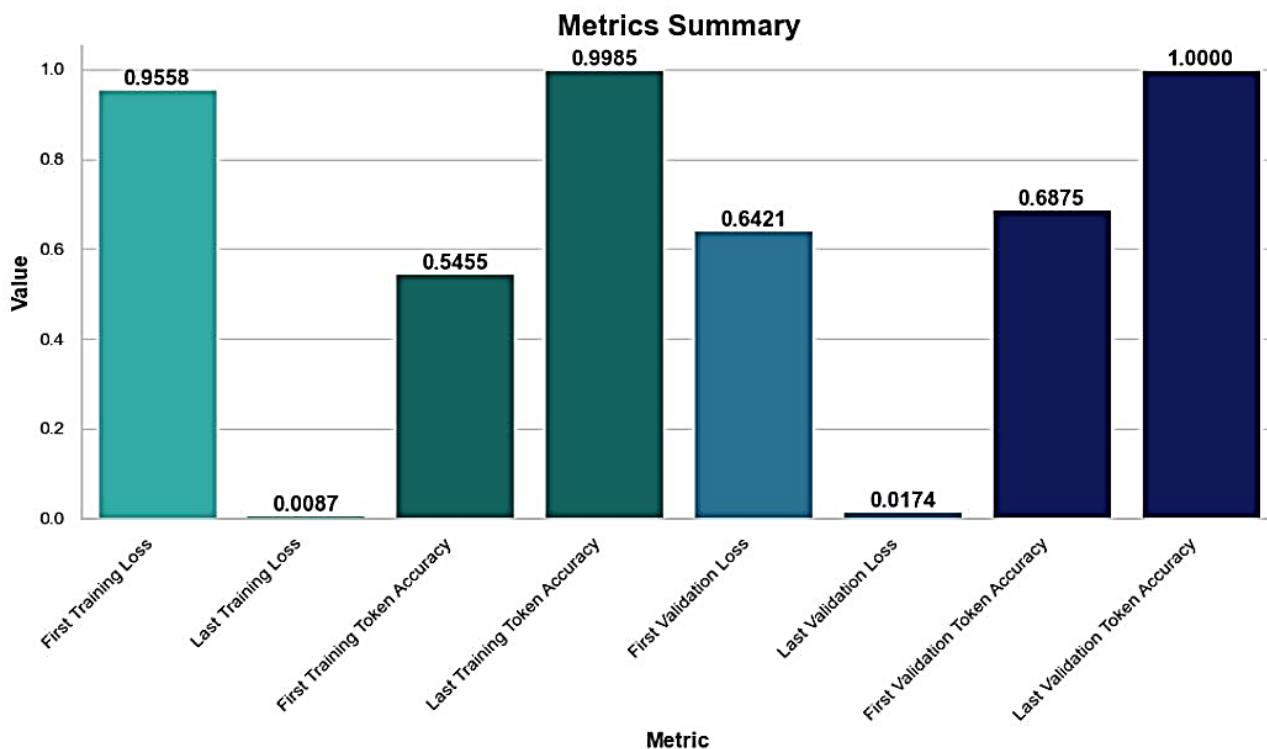


Figure 11 Model 1: Training and validation metrics of all the steps for model 1

Research Report

Software Engineering Graduation

Model two (DaVinci:ft-wappler-2023-05-16-10-11-27) performed noticeably better after being trained on a larger dataset of 1016 prompt-completion pairings. While the training loss started at 0.760 and subsequently fell to 0.0170, the validation loss started at 0.0135 and rapidly grew to 0.2321. To indicate a stronger alignment between the generated HTML code and the desired result, the goal was to decrease the loss values as much as possible. The training token accuracy showed significant growth, rising from 0.826 to a high value of 0.9933. The validation token accuracy has a similar range, beginning at 0.947 and ending at a reasonable 0.8. These results demonstrate the effectiveness of training model two on a larger dataset, resulting in increased accuracy and HTML code generation capabilities.

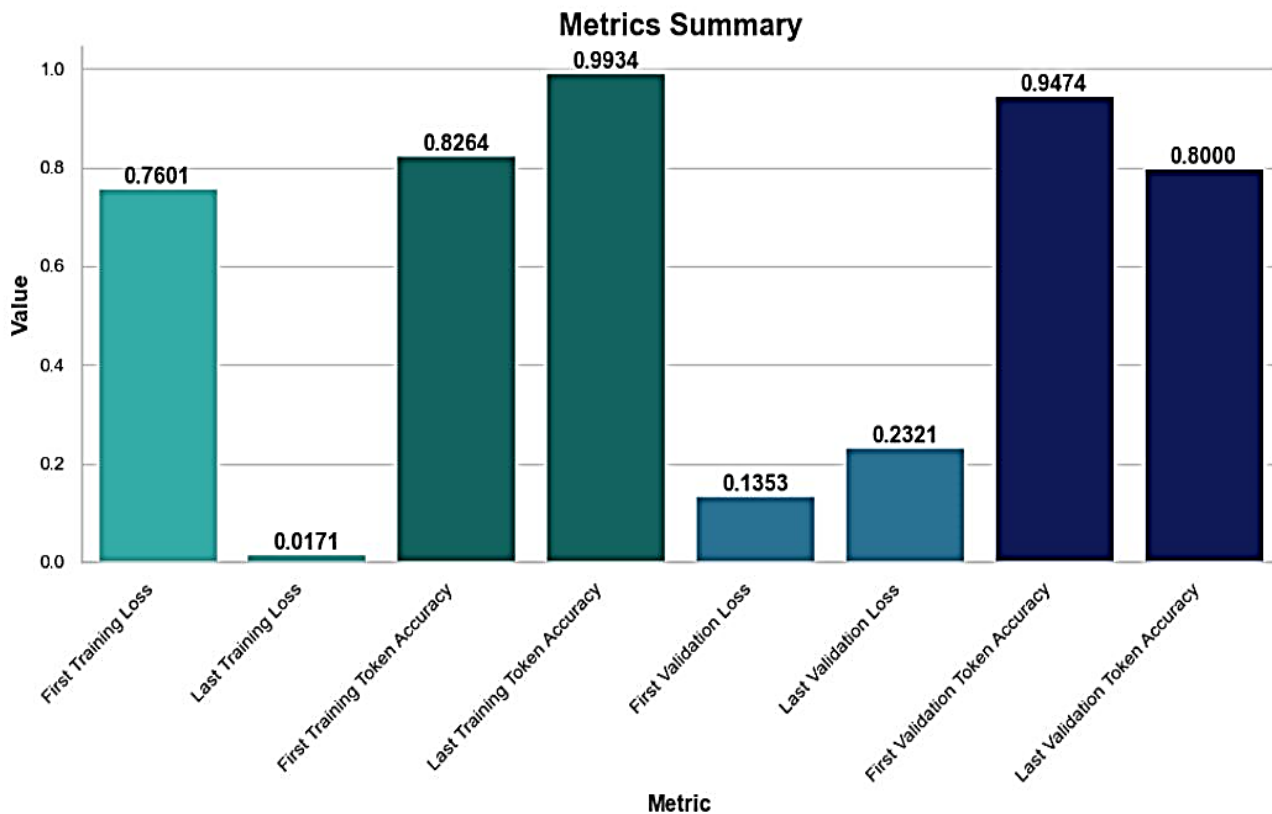


Figure 12 Model 2: Training and Validation mean metrics of all the steps for model 2

Research Report

Software Engineering Graduation

Finally, the experiment's goal was to improve the GPT-3 model's capacity to create entire HTML code from text prompts. The model improved its comprehension of HTML code structure by gathering more information and expanding the training procedure.

The results of the experiment were encouraging, as the model (DaVinci:ft-wappler-2023-05-30-15-43-41) showed an ongoing improvement in training token accuracy from 0.829 to a perfect score of 1.0. Furthermore, the training loss dramatically dropped from 0.555 to an incredibly low figure of 0.0056. Although further work was needed to address the issue of header placement uncertainty and adding more header examples solved the issue, the model showed a strong understanding of the basic HTML structure. Additionally, the validation loss showed a significant decrease from 0.409 to 0.0401, while the validation token accuracy showed a rise from 0.8982 to 0.9911.

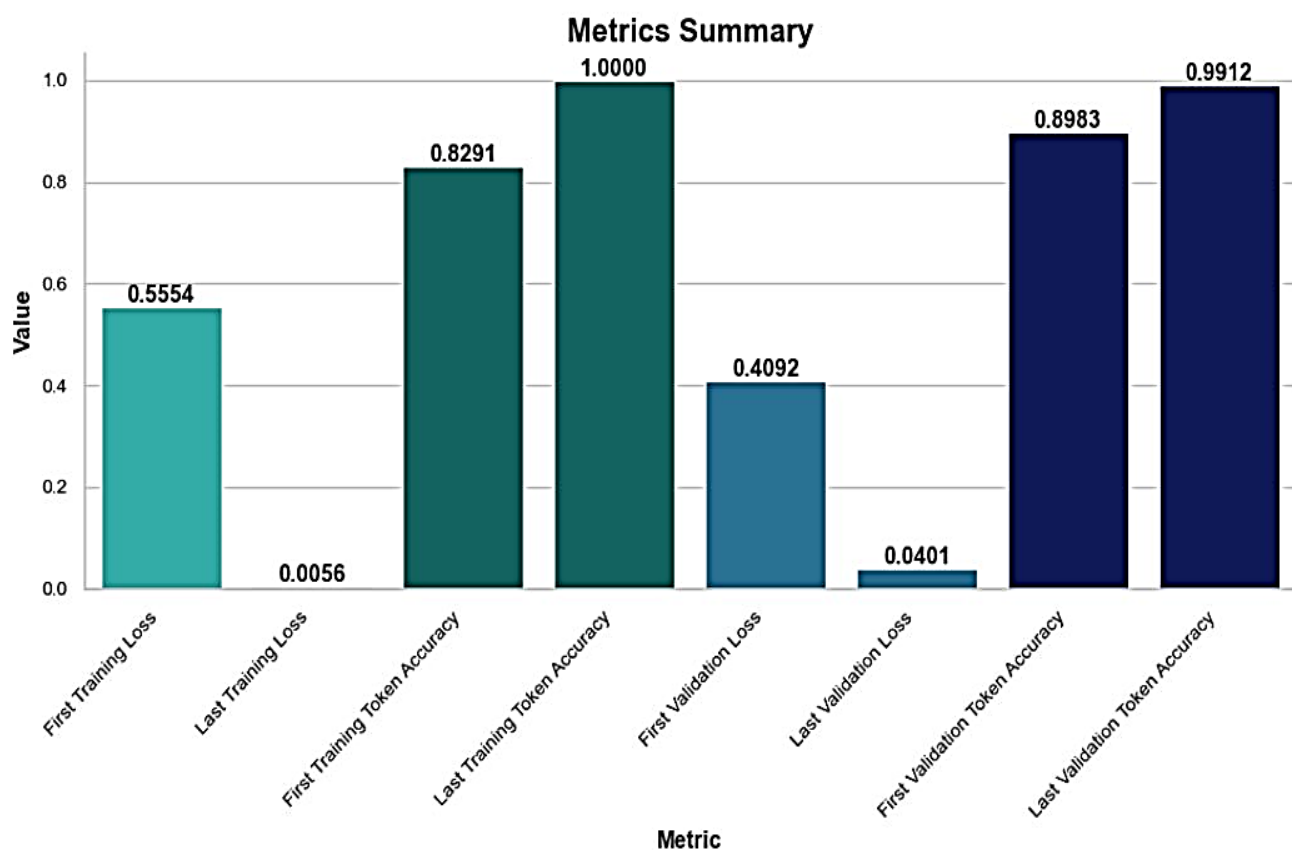


Figure 13 Model 3: Training and validation mean metrics for generate complete html page

Research Report

Software Engineering Graduation

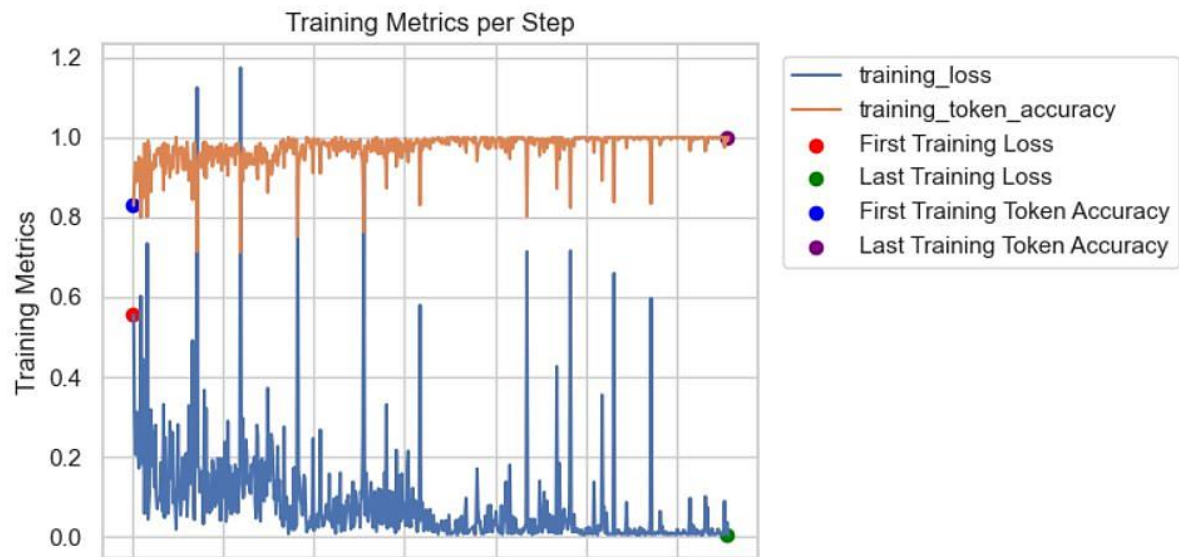


Figure 14 Shows the Improvements in the training metrics for the final model

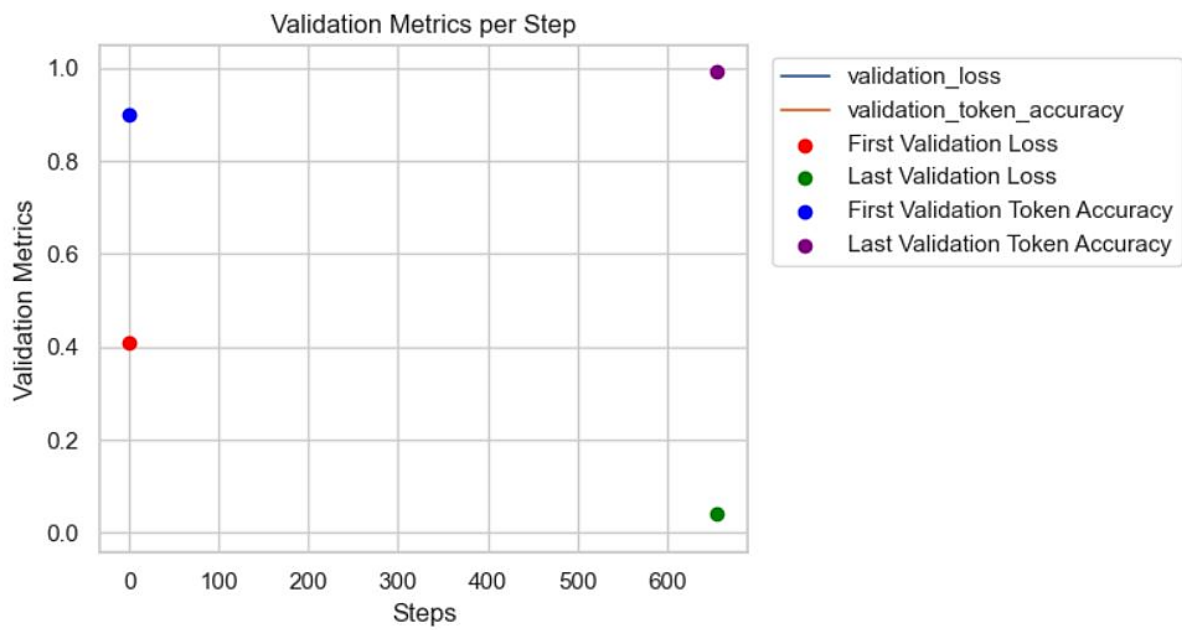


Figure 15 Shows the Improvements in the validation metrics for the final model

Research Report

Software Engineering Graduation

Significant progress has been achieved in training the model to create HTML code from text prompts. The improved training token accuracy and lower training loss show that the model can grasp and recreate the HTML structure to a decent degree.

Qualitative Analysis

A qualitative evaluation was carried out in addition to the quantitative measures by testing the models and inspecting the resulting code snippets. The evaluation focused on the created HTML code's quality, structure, and usefulness.

Model one, the first model trained on the smaller dataset, was found to have limits in its code creation capabilities. When given user input, model one created plain text rather than acceptable HTML code in several cases. This pattern implies that extracting the particular syntax and structure of HTML code from the prompts is challenging, resulting in inefficient code production. Model two performed better in terms of creating code snippets that closely mirrored the necessary Bootstrap HTML code when testing it in the open AI playground. The third and final model is the best compared to the first and second models and it can generate a full HTML web page instead of just a component.

10.7 Discussion

A Comparative Analysis of Insights and Challenges in Image-to-Code Conversion:

Comparing the generated findings to earlier efforts or baselines gives useful insights into the progress and performance of the image-to-code conversion approach. While direct comparisons may be difficult due to differences in dataset composition and experimental setups, it is worth mentioning that the success of this study may be linked to the pre-trained image model's capacity to extract information from graphical interfaces. This feature lays the groundwork for accurate image evaluation and subsequent code creation.

One interesting discovery is that any faults or inconsistencies in the output HTML code might be traced to the pre-trained model's ability to include additional elements to the image. This limitation may result in slight variations between expected and ground truth representations. It is crucial to note, however, that the general efficiency of the system is mainly due to the relatively basic nature of the DSL language used. The DSL language's simplicity helps the methodology's effectiveness in precisely capturing the structure and information of the input images. Several challenges and difficulties were experienced throughout the project. One such problem was a lack of datasets with expert-level annotations, which are required for training and assessing the image-to-code conversion model. This limitation limited the experiment's scope and required careful consideration while selecting and preparing the available datasets.

Furthermore, the complex and subtle nature of graphical interfaces made it impossible to effectively read and translate the visual elements into structured HTML code. Addressing these problems requires a mix of domain expertise, data preprocessing techniques, and model architectural changes.

Furthermore, significant biases may have occurred inside the experiment. The availability and selection of datasets, as well as the selection of pre-trained models, may create biases that impair the results' generalizability. To ensure the experiment's reliability and fairness, these biases have to be carefully considered and addressed.

Research Report

Software Engineering Graduation

Despite these limitations and difficulties, the study produced promising results in the field of image-to-code conversion. The accomplishment in correctly capturing the DSL language structure and creating HTML code from input images indicates the hybrid technique used. Future research should concentrate on overcoming the stated limitations, such as improving the pre-trained model to eliminate needless component additions and investigating additional varied datasets to improve the model's generalization capabilities.

Reviewing Model Performance and HTML Code Generation Future Directions:

The discussion that follows provides an examination of the experiment findings, emphasizing the model's capacity to create entire HTML code and addressing the collected performance metrics. In addition, a comparison with earlier research or baselines, limits and potential biases, and project problems are presented.

The refined model also includes a chat technique, which allows users to communicate with the AI tool to enhance and edit the produced code until they obtain their desired result. Users may engage with the model by including a chat element and modifying its output depending on their desires and expectations.

However, there was some confusion about where the header section should be placed inside the HTML code, and some more development and fine-tuning of the model was made to make better its knowledge and interpretation of the header's suitable place in the overall HTML structure.

Regardless of how far the experiment has progressed, it is critical to recognize specific limits and potential biases that may have impacted the results. These constraints might include biases in the gathered dataset, such as over- or under-representation of certain HTML code formats or use cases. Furthermore, the quality and variety of the dataset might affect the model's capacity to generalize to previously encountered samples.

Furthermore, the model's reliance on the quality and comprehensiveness of the training data may induce biases. Inconsistencies, mistakes, or missing information in the training dataset might impair the model's ability to output proper HTML code. Users can better grasp the model's capabilities and possible limits when using it to generate HTML code if they are aware of its limitations and biases.

10.8 Implementation Details

Making an accessible tool for creating code from hand-drawn sketch designs:

The goal of this research was to develop an accessible tool for creating code based on hand-drawn sketch designs. The key components used to achieve this goal were a computer vision model that used a Convolutional Neural Network (CNN) for image feature extraction, a language model that used a Gated Recurrent Unit (GRU) to encode DSL code tokens, and a decoder model that predicted the next token in the sequence based on the outputs of the previous two steps.

Several important conclusions came from the investigative process. In properly translating images to structured HTML code representations, the hybrid technique integrating computer vision and language models produced excellent outcomes. The model's capacity to create code sequences that closely approach the ground truth is seen in its high BLEU score of 0.92 achieved during testing. This signifies that the visual elements were successfully translated into code syntax.

However, it is important to recognize the approach's limitations. One significant disadvantage is the model's reliance on handmade features, which limits its generalizability beyond specific drawing parameters. To overcome this limitation, more studies would be needed to investigate alternate ways, such as using language models instead of procedural approaches. The ability of language models to generalize beyond defined features opens up new opportunities for greater strength and responsive code production.

Furthermore, because the model's output is very basic and the model's limitations are obvious, it acts as a proof-of-concept for the conversion process. Instead of integrating the model into Wappler, I chose to illustrate its usefulness via a Flask application as shown below. By creating the following web application, the potential of the sketch2code approach is demonstrated and offers a practical illustration of how it works. This project has provided me with invaluable insights into the fields of web development and AI-powered interfaces, improving my academic experience.

Research Report

Software Engineering Graduation

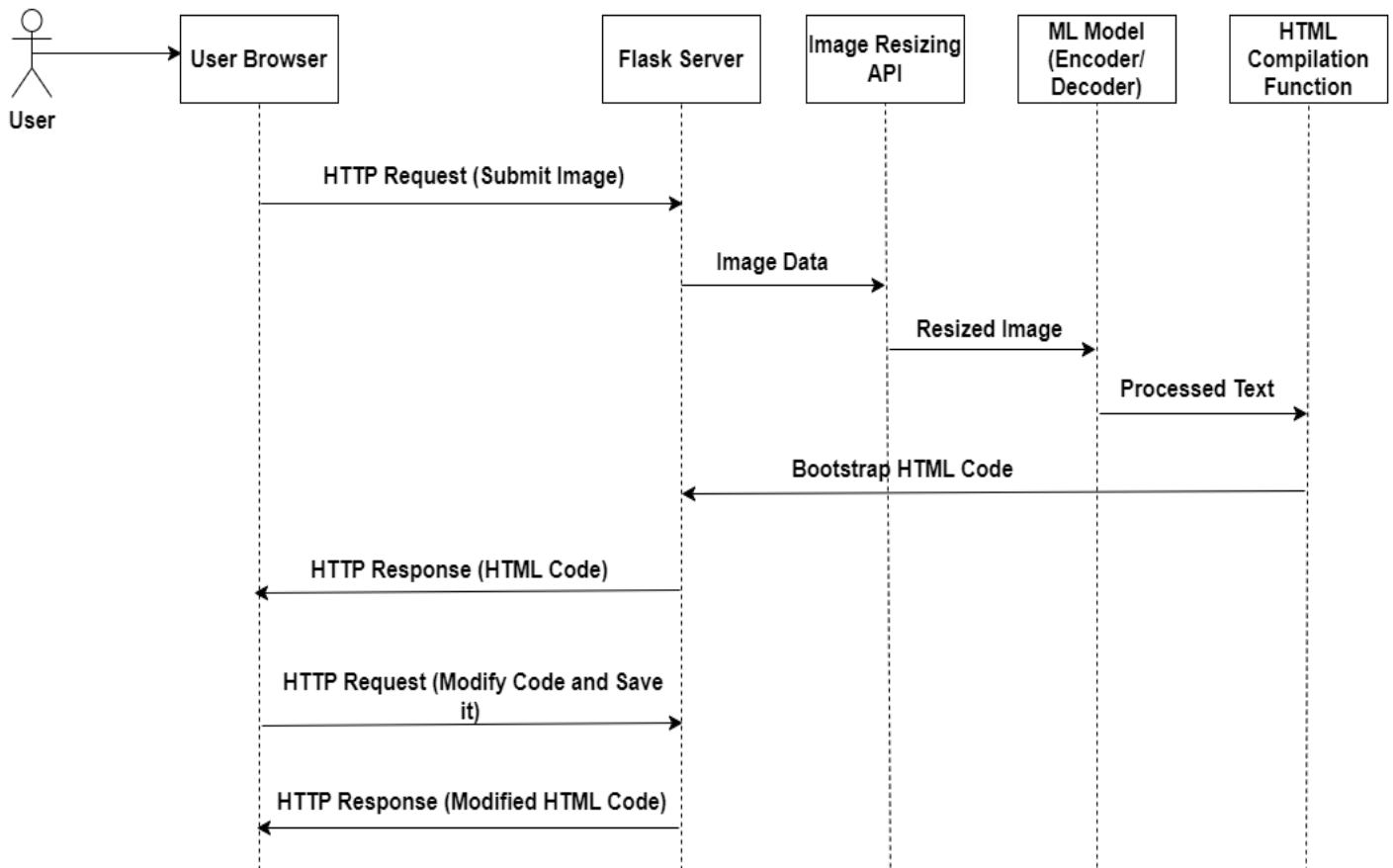


Figure 14 Sequence Diagram for Flask's Server-side

Here is how to get from an image to processed HTML: A Step-by-Step Guide to Flask's Server-side Workflow:

1. The Flask Server receives an HTTP Request (Submit Image) from the User Browser.
2. After receiving the request, the Flask Server sends the Image Data to the Image Resizing API.
3. The Image Resizing API presses the image data and sends the resized image to the ML Model (Encoder/Decoder).
4. The ML Model (Encoder/Decoder) processes the Resized Image and produces the Processed Text.
5. After that, the Processed Text is given to the HTML Compilation Function, which turns it into HTML Code.
6. The HTML Code is received by the Flask Server from the HTML Compilation Function.
7. The Flask Server responds to the User Browser with an HTTP Response (HTML Code).
8. The HTTP Response is received by the User Browser, which then displays the HTML Code, resulting in a visual representation of the processed image.
9. The User can make changes to the produced Bootstrap HTML code and send a save request.

10. The Flask Server receives an HTTP Request (Save Code) from the User Browser.
11. After receiving the request, the Flask Server sends back the modified HTML code.

HTTP Requests and Responses on the Backend:

- **GET/:** When a user reaches the root URL, the backend responds with the initial HTML template index.html.
- **POST/submitImage:** When a user uploads an image through the frontend, this request is sent out. The image file is included in the request body. The image is received by the backend, which processes it and produces a scaled image, processed text, and assembled HTML. The backend then returns a JSON response with information about the processed data, such as the URLs of the resized image and the produced HTML.
- **POST/saveChanges:** This request is done when the user clicks on the frontend's "Save Changes" button.
- **GET/output:** This request is done when the user clicks on the frontend's "Download HTML" button. The backend responds with the built HTML file as a download attachment.

HTTP Requests and Responses on the Frontend:

- **GET/:** The frontend sends a GET request to the backend to receive and render the initial HTML template (index.html) in the browser.
- **Post/resize:** This request is sent when the user selects an image and then hits the "Convert" button. The image file is sent to the backend for processing by the frontend. The backend completes the required activities and returns a JSON response including information about the processed data, such as the URLs of the resized image and the produced HTML.
- **Post/executeCode:** This request is sent when the user wants to see the output of the code editor.
- **Post/saveChanges:** This request is sent when the user did some changes and wants to save them to receive the new modified HTML code.
- **GET/output:** When a user selects the "Download HTML" button, the frontend sends a GET request to the backend to receive the created HTML file.

Research Report

Software Engineering Graduation

The following diagrams show the sequence diagram, explained above:

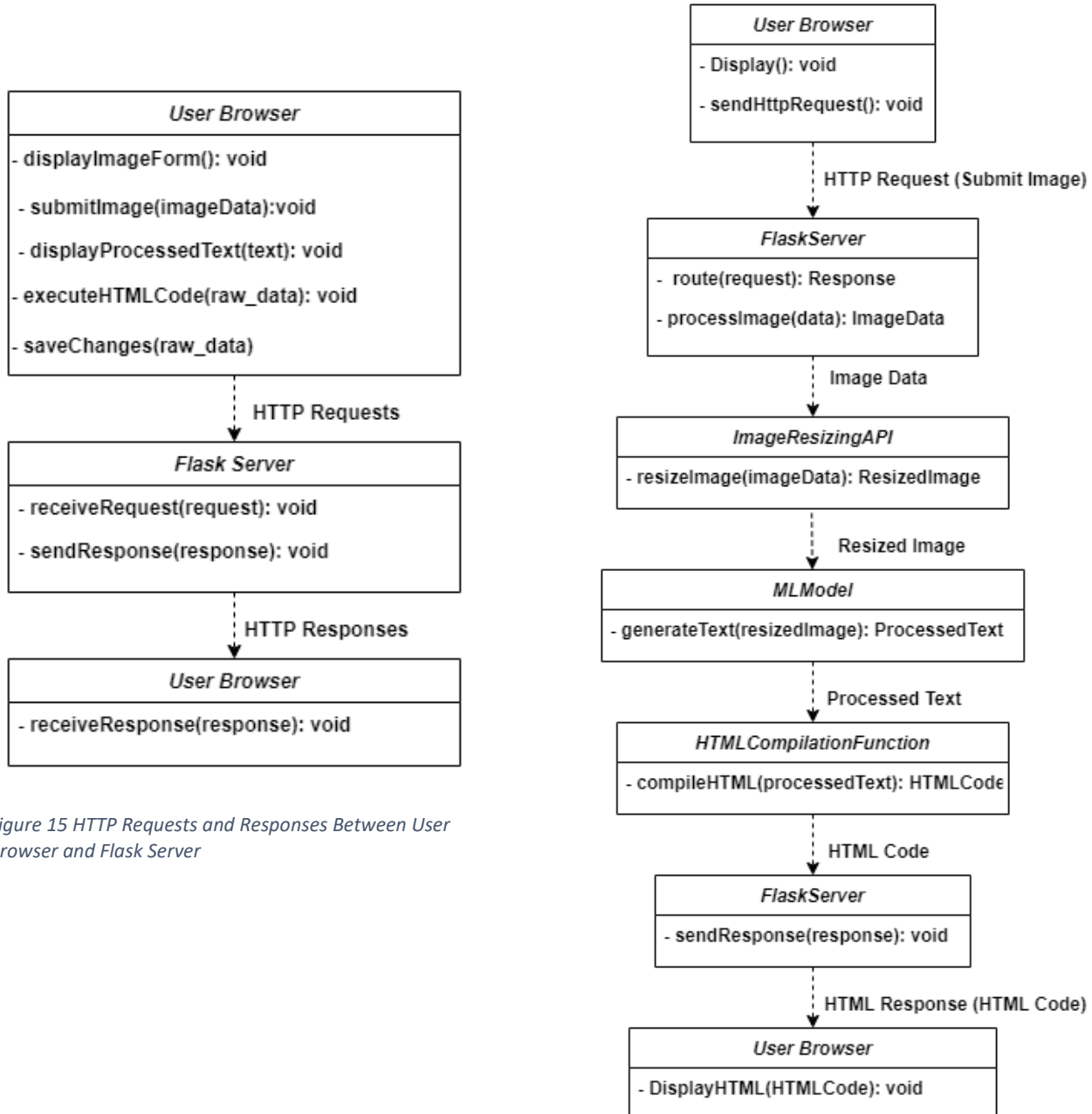


Figure 15 HTTP Requests and Responses Between User Browser and Flask Server

Figure 16 HTTP Request and Responses from Backend and Frontend

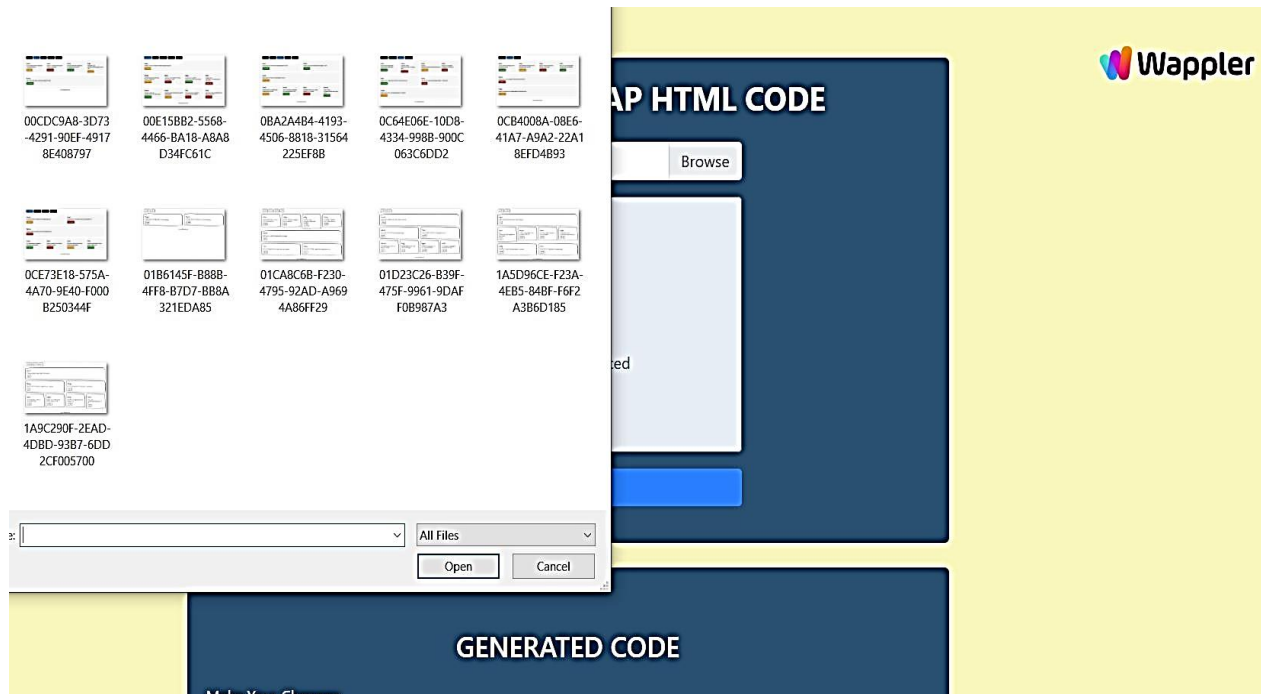
To summarize, the frontend communicates with the backend using HTTP requests to particular routes/endpoints. The backend processes the requests, executes the necessary activities, and creates HTTP replies with the requested data or information.

Research Report

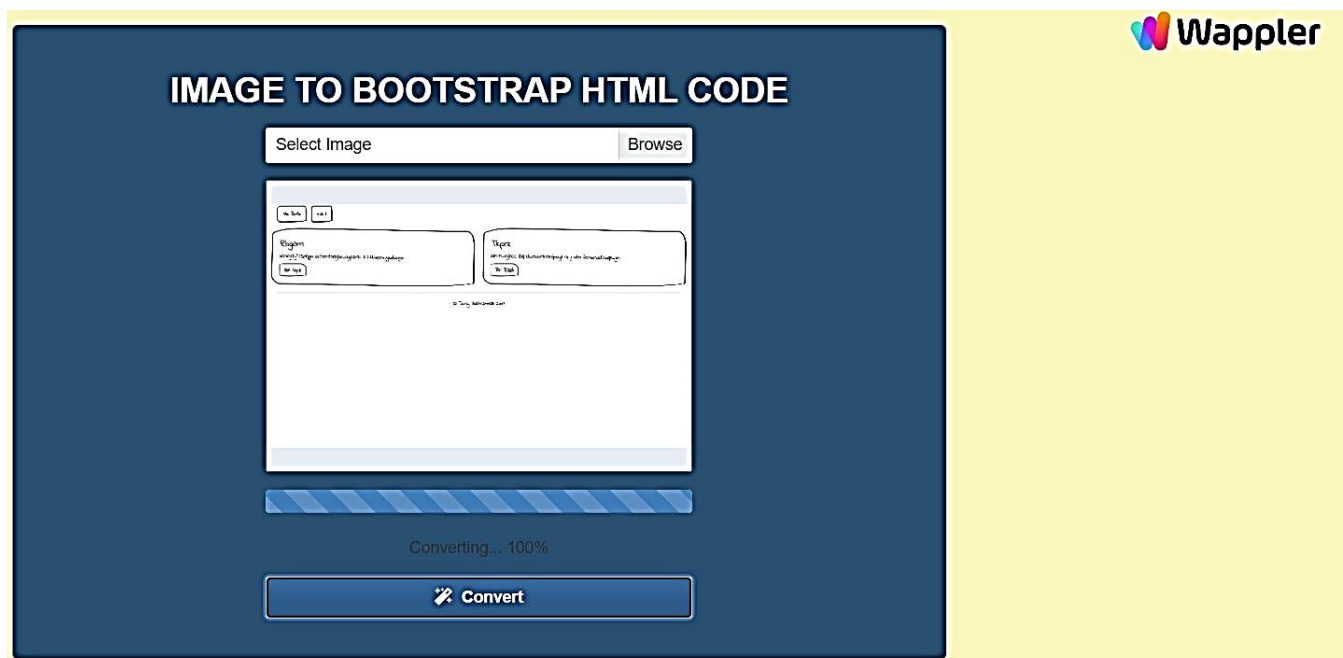
Software Engineering Graduation

On the frontend, creating a Sketch2code system using Flask involves numerous phases. Here's a brief description of the procedure:

This is the initial phase, users must browse and choose an image; if no image has been selected, the user will see the message "no image selected." If the user chooses an image, the image will be displayed to the user:



The second phase, after uploading the image and pressing the convert button, will show the user a progress bar to indicate that there is a work in progress that is preparing the image and extracting features from it to translate it to html code:



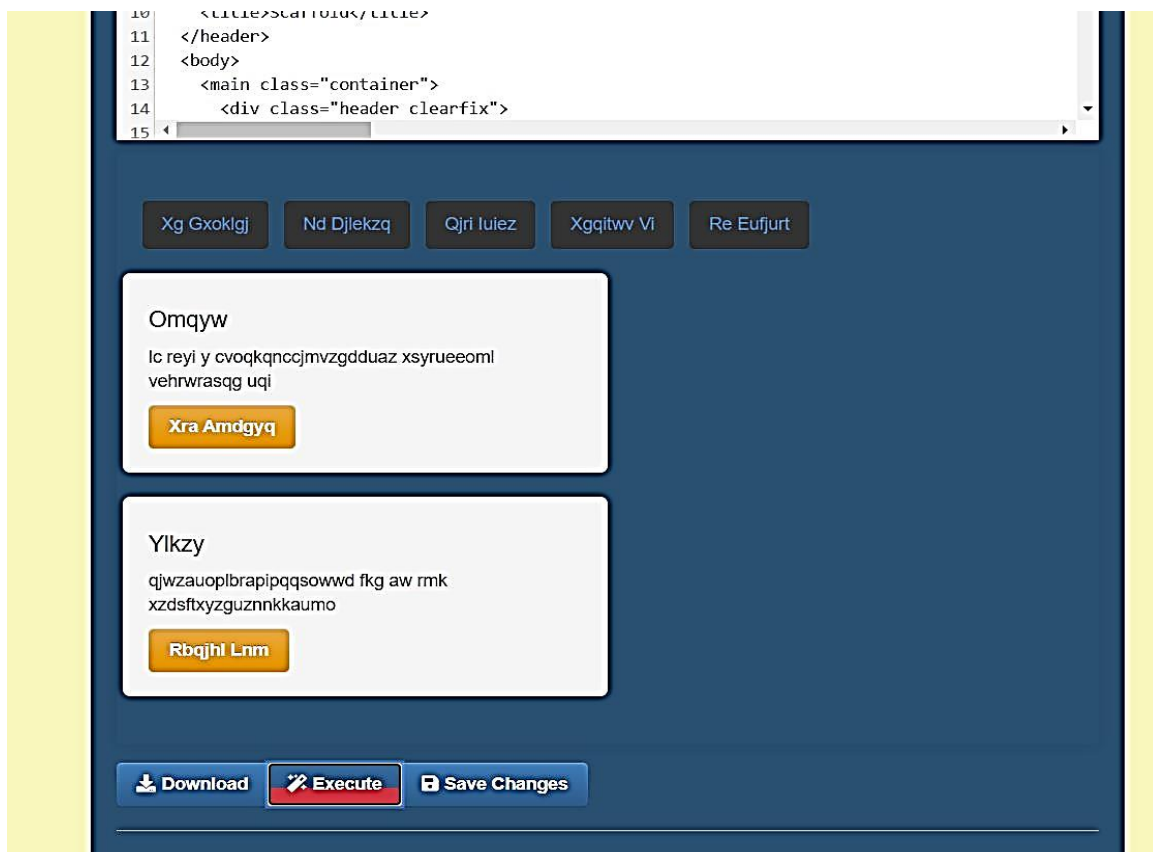
Research Report

Software Engineering Graduation

In the third phase, the generated HTML code will be displayed in the code editor so the user can modify the code and make changes to it:



In the last phase, the user can save the changes, execute the code to see the result and also download the HTML code:



Research Report

Software Engineering Graduation

Moving ahead, future approaches and improvements should concentrate on overcoming the mentioned restrictions and investigating the use of language models. The goal of the research should be to create tactics that successfully use language models, allowing for improved visual component transfer and higher generalization. It is possible to overcome present limits and enhance the image-to-code conversion process by expanding the area of research.

To summarize, this study shows the hybrid methodology's potential for image-to-code translation. While certain things may be done better, the findings show that it is possible to create an accessible tool based on hand-drawn sketch designs. Future research can pave the way for more complex and varied image-to-code conversion systems, making coding more accessible and efficient for a larger variety of users, by resolving the observed limits and getting deeper language models.

Creating an Accessible Tool for Text Prompt-Based Code Generation:

The goal of this study is to create an easy-to-use tool for producing code from code description text. The GPT-3 (Davinci model) needed to be adjusted using a dataset made up of HTML code and the descriptions that go with it. Although it has some issues with various components, the model is capable of producing Bootstrap HTML code. Despite these challenges, it is thought that integrating the model into the Wappler platform is worthwhile. The creation of the server, which requires storing data in JSON line files, will make it easier to further improve the model by allowing analysis of how well it produces HTML code. The technology stack used by the Wappler platform was followed when Node.js was chosen for implementation.

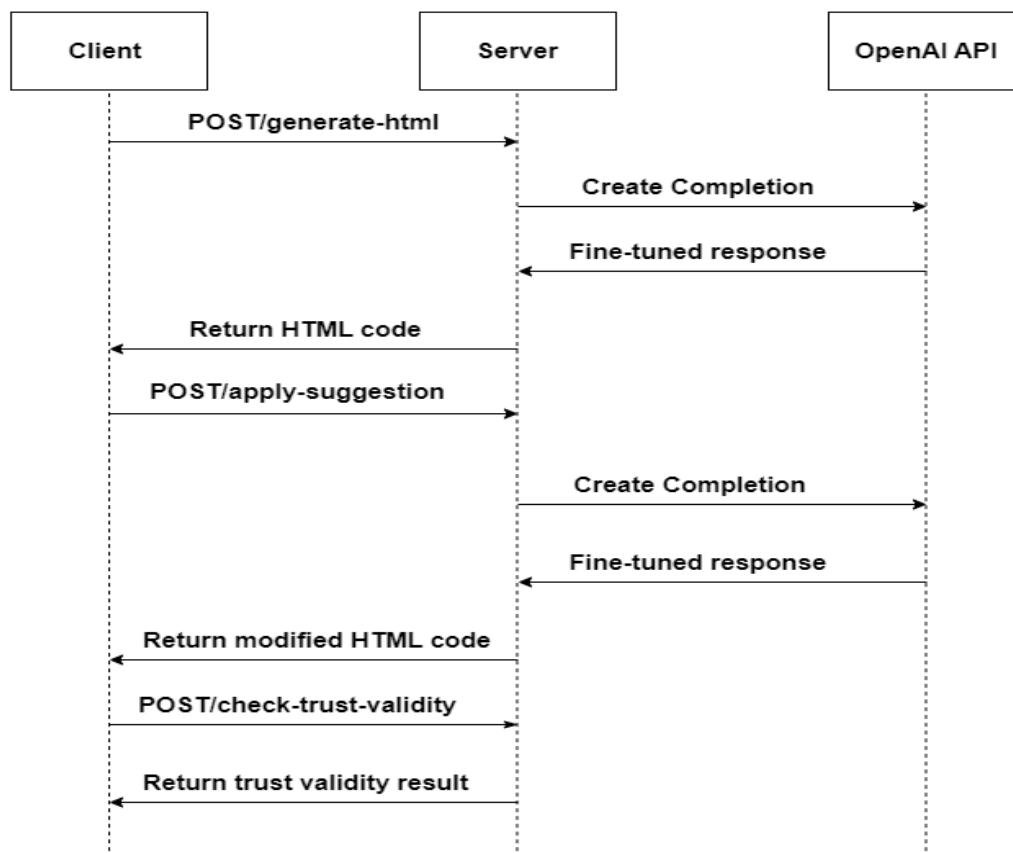


Figure 17 Sequence Diagram: Generating and Modifying HTML Code with OpenAI API

Research Report

Software Engineering Graduation

This sequence diagram shows the interactions of several actors (Client, Server, and OpenAI API) in a system including the generation and modification of HTML code using the OpenAI API. Here's a step-by-step explanation of the diagram:

1. The Client starts the interaction by submitting a POST request to the Server with the endpoint `"/generate-HTML"`. This request is designed to produce HTML code from user input.
2. The request is received by the server, and the user input is processed. It then creates HTML code based on the information supplied.
3. After creating the HTML code, the Server must use the OpenAI API to enhance or modify it. It requests the OpenAI API to construct a completion. This indicates that the Server is requesting the OpenAI API to produce HTML code.
4. The OpenAI API processes the Server's request and returns a fine-tuned response. Based on the HTML code description text given by the Server. The HTML code is generated using machine learning models and algorithms.
5. The OpenAI API returns the fine-tuned response to the Server, which receives and processes it.
6. As a response to the initial `"/generate-html"` request, the Server provides the created HTML code (which includes the recommendations or adjustments offered by the OpenAI API) back to the Client. This HTML code can now be used by the Client.
7. At some point, the Client may want to apply a specific suggestion to the HTML code. It sends a POST request to the Server with the endpoint `"/apply-suggestion"` and includes the specified recommendation as part of the request.
8. The request is received by the server, which processes the selected recommendation and applies it to the HTML code. This stage allows the Client to change the HTML code in response to particular requests or enhancements.
9. The Server, like the previous phases, submits a request to the OpenAI API, requesting completion. The request this time is to fine-tune the modified HTML code even further.
10. The OpenAI API evaluates the request, provides a customized response, and returns it to the Server.
11. The response from the OpenAI API is received by the Server, which includes a fine-tuned version of the modified HTML code.
12. Finally, in response to the `"/apply-suggestion"` request, the Server provides the updated HTML code to the Client. The modified HTML code is now available to the Client.
13. There is also another stage in which the Client can submit feedback data. It makes a POST request to the Server using the endpoint `"/check-trust-validity"`.
14. The Server gets the request and checks the feedback data for trust validity. This stage makes sure the accuracy and reliability of the input provided.
15. As a response to the `"/check-trust-validity"` request, the Server delivers the result of the trust validity checks to the Client. The outcome notifies the Client that the feedback data was correctly sent.

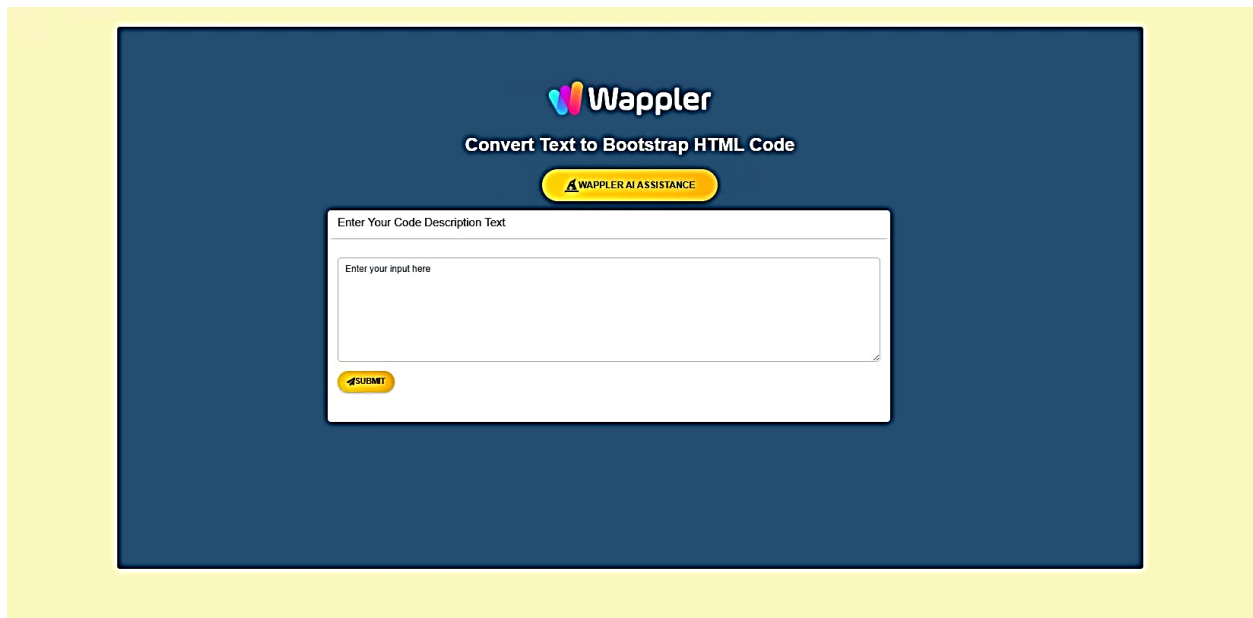
Overall, this sequence diagram shows the flow of interactions between the Client, Server, and OpenAI API when producing, updating, and verifying HTML code with the OpenAI API.

Research Report

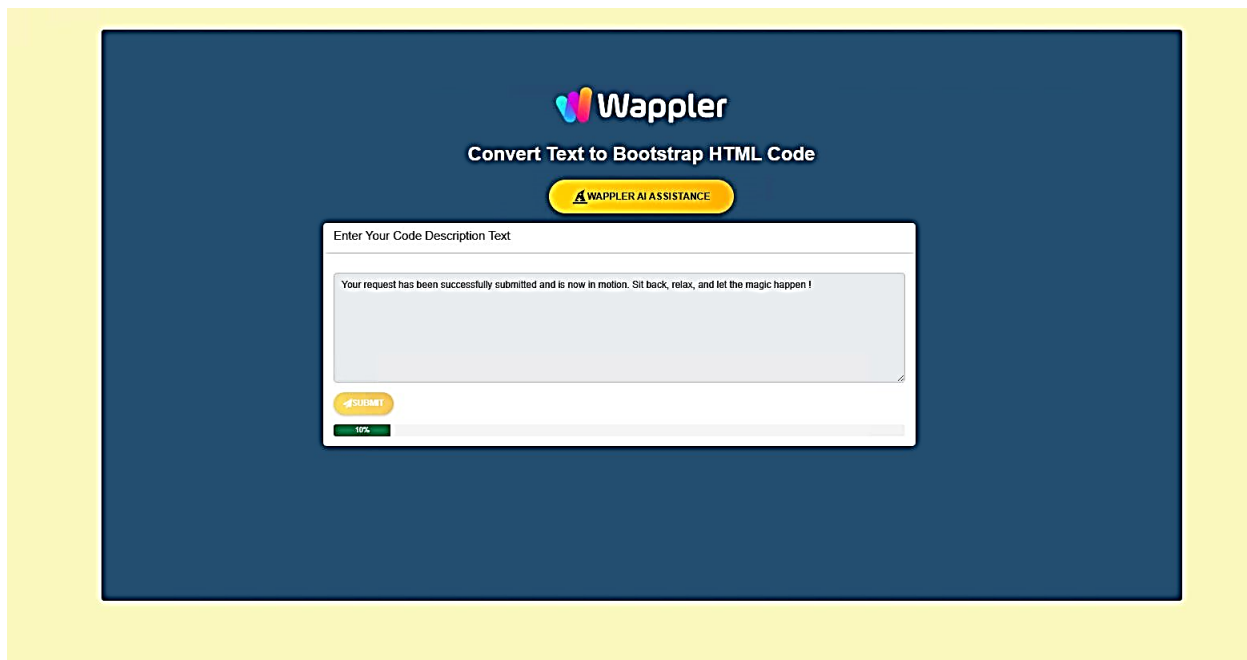
Software Engineering Graduation

Communication with the server must happen through the client. To carry out numerous tasks, including creating HTML code, implementing suggestions, and sending feedback data, it sends HTTP requests to the server. The frontend part can be seen in the following screenshots:

First, the "Wappler AI Assistance" button can be clicked by the user to show a card component whenever they come across it. This card component makes it easier for the user to provide code explanation text.



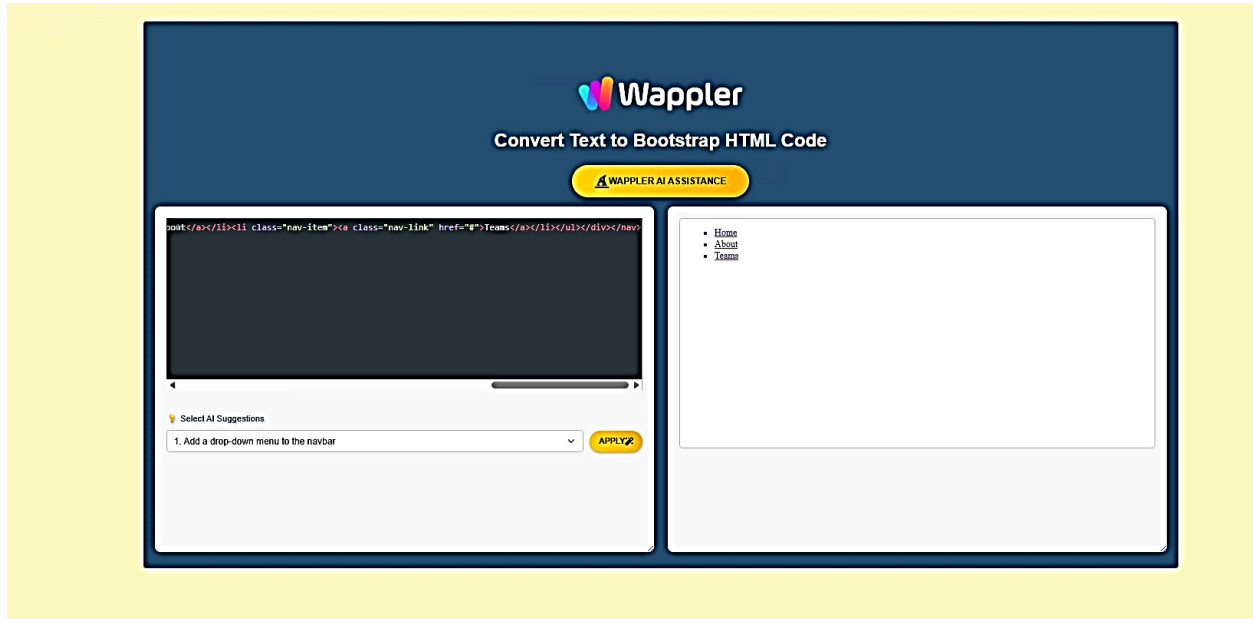
The user can click the submit button once they have finished inputting the code description. When they click, a progress bar displaying the status of their request will appear. A notification message will also be shown to the user to let them know their request was sent successfully.



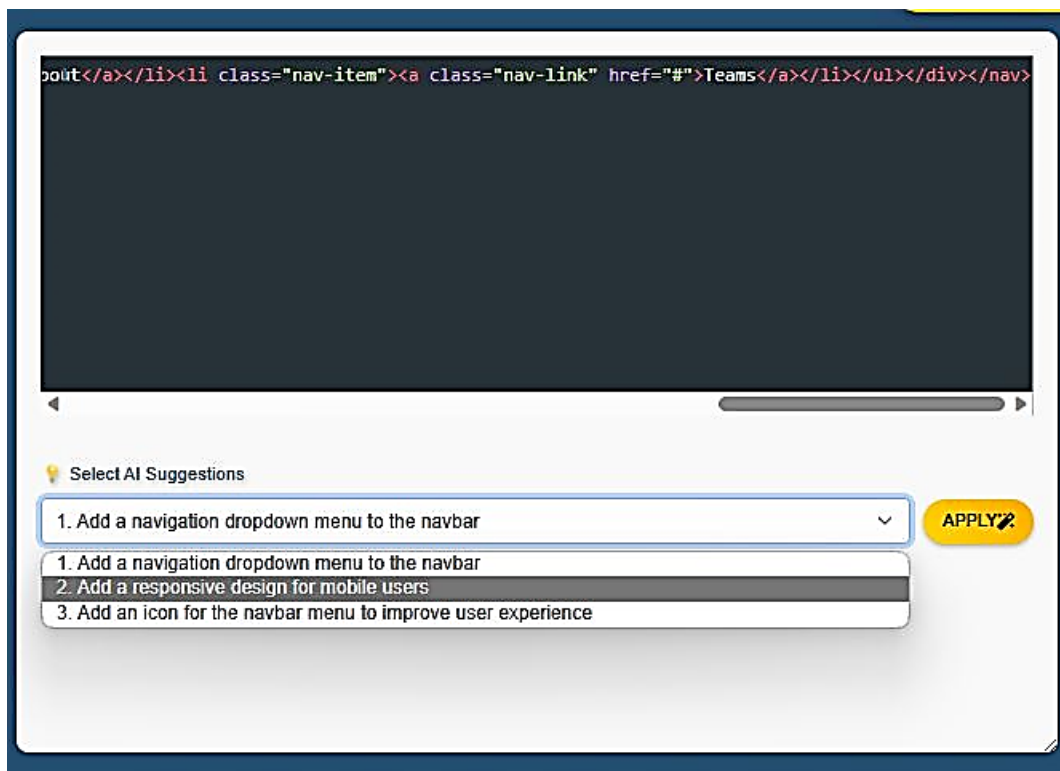
Research Report

Software Engineering Graduation

The generated code along with three recommendations for code improvement will be provided once the code has been generated by the fine-tuned model and Turbu-3.5 text model. A CodeMirror editor will be provided along with a live preview of the code to assist with future adjustments. The user may easily make modifications and immediately observe the modified code



Three recommendations for improving the code will be shown in a dropdown menu after the code is generated. Users can quickly select and explore the proposed improvements via the user-friendly UI.



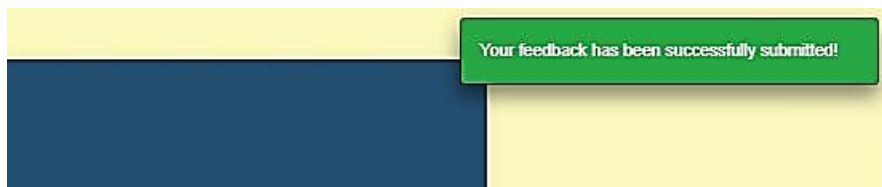
Research Report

Software Engineering Graduation

A spinner will appear to show that the work is in the process once a user chooses a suggestion from the dropdown menu and clicks the apply button.



Users have the option to leave remarks on the produced HTML when the updated code is generated and shown in the code editor. To keep track of preferred and disliked components, this feedback data is kept. The model's performance is then enhanced by refining and fine-tuning it to correspond with the user's preferences using the feedback that has been gathered. The capabilities of the model are continuously improved due to this iterative process.



11. Project Conclusion

The goal of the research project was to create a flexible solution that uses artificial intelligence to transform website layout drawings and code description text into HTML Bootstrap code, while also offering an easy and user-friendly interaction with Wappler. This final project conclusion presents the significant results and accomplishments for both the image-to-code and text-to-code tasks while addressing the central topic of how AI might be used for this purpose.

Image-to-Code conclusion: The combination of computer vision and language models produced great results when converting hand-drawn sketch designs into HTML code representations. A CNN for image feature extraction, a GRU for code token encoding, and a decoder model produced code sequences that were near the ground truth, resulting in a high BLEU score of 0.92. Based on manually constructed features, on the other hand, limits generalizability beyond certain drawing parameters. Investigating other methodologies, such as language models, can improve flexibility and code creation. The current output of the model acts as a proof-of-concept, including minimal code representations. The sketch2code approach was presented with a Flask application. Valuable insights in web development and AI-powered interfaces were obtained, improving the academic experience and demonstrating practical applications.

Text-to-Code conclusion: The experiment's goal was to enhance the GPT-3 model's HTML code generation from text prompts. The model improved its understanding of HTML structure by increasing the training procedure and collecting more data. Significant increases in the model's performance were noticed during the training and validation processes. The validation loss was reduced from 0.409 to 0.0401, demonstrating a significant decrease in errors. Similarly, the training loss fell from 0.555 to 0.0056, indicating better model prediction accuracy. Furthermore, both the validation token accuracy and the training token accuracy improved noticeably. The training token accuracy increased from 0.82 to 1.0, while the validation token accuracy climbed from 0.8982 to 0.9911. These enhancements are in line with the anticipated outcomes since lower loss values and greater accuracy values indicate improved model performance.

The text-to-code server was successfully developed using Node.js and may be used inside the Wappler software. When users connect with the server, OpenAI receives requests from the server. There has been significant progress made in testing the model within the OpenAI framework, even though Wappler's text-to-code tool is still in the final stages of development. The initial work provides the groundwork for future integration efforts and demonstrates the power of using AI to transform code description language into HTML Bootstrap code.

Finally, this study investigates the use of artificial intelligence to bridge the gap between drawings, written descriptions, and HTML code creation. The hybrid technique for image-to-code translation, as well as the training of the GPT-3 model for text-to-code creation, produced positive outcomes. The study emphasizes the potential of AI-powered solutions to improve productivity and save development time by streamlining the process of developing HTML code. The project lays the groundwork for future AI-assisted web development research and development. The incorporation of AI technology into tools such as Wappler has significant potential for empowering developers and increasing their efficiency in generating advanced and flexible web designs.

12. Reflection

This software engineering graduation project provides a demonstration of level three HBO-ICT competencies in analyzing, releasing, and designing. An explanation of how these competencies were developed is provided below:

Conclusions of an Analysis, Release, and Design Competencies:

Analyze: A significant study on text-to-code conversion and image-to-code conversion was done in the project's early stages, demonstrating a good ability to examine various strategies and relevant technology. It was possible to identify limitations in the extraction of complex features from images after a thorough evaluation of the machine learning model's performance for the conversion of images to codes. This showed a critical evaluation of the model's advantages and disadvantages. The project's focus was changed to creating a model for text-to-code translation as a result of the realization that a different strategy was required. This choice was influenced by an analysis of the limits found in the image-to-code conversion approach, demonstrating a flexible approach and the capacity to choose wisely after careful consideration.

Release: The project's release phase was a showcase of skills for creating and releasing useful software. On an open-source dataset, the machine learning model for converting images to codes was successful and achieved an amazing accuracy score of 92%. This accomplishment proved my technical skill in producing excellent software. Additionally, using a manually compiled dataset of HTML code and descriptions, I improved the GPT-3(Davinci) model for text-to-code translation, demonstrating my skills in gathering and preparing data for model training. In addition, I created a successful AI-powered application and smoothly incorporated it into the Wappler platform to produce HTML code from human descriptions. This integration served as an example of how to successfully release software products and integrate them into pre-existing environments.

Design: The server's architecture, which successfully overcame the limits of the text-to-code translation methodology when creating Bootstrap HTML code, was a clear indication of the design capabilities. My ability to create software solutions that directly address problems encountered was demonstrated by this. The creation of a user-friendly interface further demonstrated my care for user experience. The addition of a feedback mechanism to the server further demonstrated my concern for user experience. my ability to create systems that gather and use user input to improve the overall experience was demonstrated by the technique that enabled the collection of feedback from users on the generated HTML code. Additionally, by highlighting areas in the project that may be enhanced and built upon, I demonstrated my skill at spotting design chances for future development and improvement. My capacity to predict the project's growth and progress is demonstrated by my ability to recognize and take possibilities for design.

Conclusion on the process: Throughout the project, I intended to develop an AI-powered tool capable of translating images or text descriptions into HTML code, with the eventual objective of incorporating it into Wappler. I performed significant research on both image-to-code conversion and text-to-code conversion in the early phases. I successfully created a machine learning model that predicts HTML code from images using the RestNet-152 pre-trained model. On an open-source dataset, the model earned an excellent accuracy score of 92%. On closer inspection, however, I discovered that the model could only extract simple features from the images, resulting in basic and constrained layout predictions. This limitation provided a barrier to efficiently incorporating the model into Wappler.

Recognizing the need for a different approach, the focus shifted to developing a model capable of interpreting human descriptions and generating corresponding HTML code. This entailed fine-tuning the GPT-3(Davinci) model with a manually gathered dataset of HTML code and descriptions. Manual data collection proved to be time-consuming nevertheless essential to efficiently training the model. The fine-tuned model demonstrated promising skills, producing code snippets for various components and displaying an understanding of the general structure of a whole HTML page.

After lengthy research and development, an AI-powered tool that generates HTML code from human descriptions has been developed successfully. Although the tool displays promising capabilities, some areas still need attention and development. The model has difficulty creating some Bootstrap HTML code, such as the model and slider components, and has difficulty understanding the requirements for the rows and columns in a table. Despite these difficulties, the Wappler platform may easily use the current server code. The ability to use the AI tool is made available to users through this integration, which speeds up the conversion of descriptions into HTML code and greatly improves productivity and efficiency in web development inside the Wappler environment.

Conclusion on the product: A result is an AI-powered tool that converts images and text descriptions into HTML code. I have reached key milestones in the building and fine-tuning of machine learning models, identifying both strengths and opportunities for growth.

On an open-source dataset, the image-to-code conversion model initially demonstrated an amazing accuracy score of 92%. However, it became clear that the model's capacity to extract complex information from images was restricted, resulting in basic layout predictions. This was a good lesson in learning about the potential and limitations of AI models when applied to specific jobs.

To solve this restriction, the focus was moved to text-to-code translation and fine-tuned the GPT-3 model using a dataset of HTML code and descriptions that I gathered. The resulting model displayed promising skills, including the generation of code snippets for components and a grasp of the general structure of HTML pages.

The architecture of the server has been created to make it easier for Wappler to figure out areas where the model has trouble producing Bootstrap HTML code. The server gathers and maintains this information as well as a counter that keeps track of the number of likes and dislikes received for each output by allowing users to leave feedback on the generated HTML. Through this procedure, the dataset used to train and fine-tune the model can be gathered and expanded, hopefully enhancing the model's performance and overall accuracy.

Research Report

Software Engineering Graduation

As it is, the project highlights the potential for automating the process of creating HTML code from text descriptions, highlighting advances in AI and machine learning. However, it is critical to keep refining and improving the tool to assure its usability and usefulness in real-world circumstances.

Finally, the software marks a substantial advancement in the automation of HTML code production. Despite the noted flaws, it lays the groundwork for future growth and refinement. Wappler may continue to expand the product's capabilities and make it a valued asset in the field of web development by addressing the limits and recommendations provided.

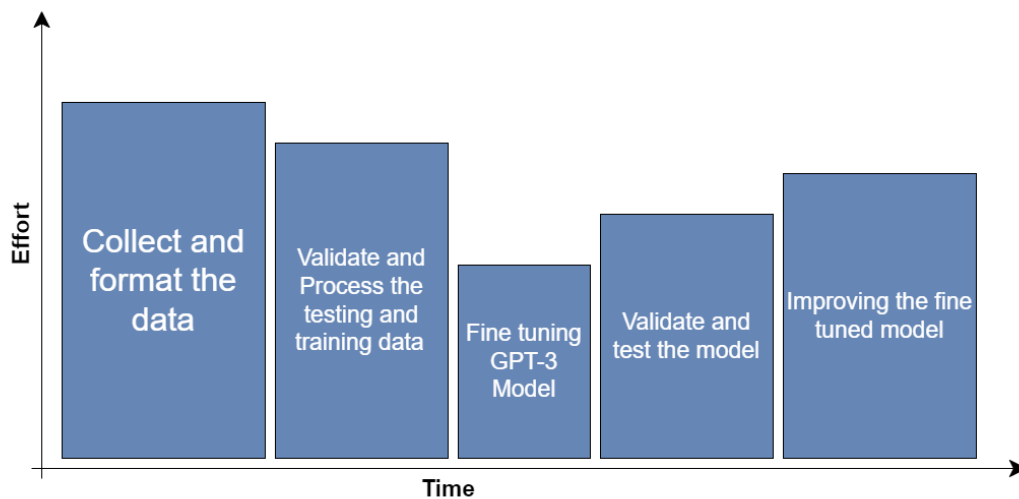


Figure 18 Displays the effort required for different tasks in relation to time for Text to Code tool

13. References

- Bhanot, K. (2019, May 24). *Color identification in images - machine learning application*. Medium. <https://towardsdatascience.com/color-identification-in-images-machine-learning-application-b26e770c4c71>
- Burns, E., & Brush, K. (2021, March 29). *What is deep learning and how does it work?*. Enterprise AI. <https://www.techtarget.com/searchenterpriseai/definition/deep-learning-deep-neural-network>
- Introduction to wappier*. Wappier Documentation. (n.d.). <https://docs.wappier.io/>
- Loye, G. (2023, January 23). *Beginner's guide on Recurrent Neural Networks with pytorch*. FloydHub Blog. <https://blog.floydhub.com/a-beginners-guide-on-recurrent-neural-networks-with-pytorch/>
- Microsoft/resnet-152 · hugging face*. microsoft/resnet-152 · Hugging Face. (n.d.). <https://huggingface.co/microsoft/resnet-152#:~:text=ResNet%2D152%20v1.,-5&text=ResNet%20model%20pre%2Dtrained%20on,by%20the%20Hugging%20Face%20team>
- OpenAI API*. OpenAI Platform. (n.d.). <https://platform.openai.com/docs/guides/fine-tuning>
- Society, T. R. (2019, July 4). *Colour detection*. Medium. <https://medium.com/image-processing-in-robotics/colour-detection-e15bc03b3f61>
- Team, G. L. (2022, October 31). *The vanishing gradient problem*. Great Learning Blog: Free Resources what Matters to shape your Career! <https://www.mygreatlearning.com/blog/the-vanishing-gradient-problem/>
- What is Computer Vision?*. IBM. (n.d.). <https://www.ibm.com/topics/computer-vision#:~:text=Resources-.What%20is%20computer%20vision%3F,recommendations%20based%20on%20that%20information>
- Yasar, K. (2023, March 15). *What is image recognition?: Definition from TechTarget*. Enterprise AI. <https://www.techtarget.com/searchenterpriseai/definition/image-recognition#:~:text=Image%20recognition%2C%20in%20the%20context,software%20to%20achieve%20image%20recognition>

14. Appendix

Image to Code:

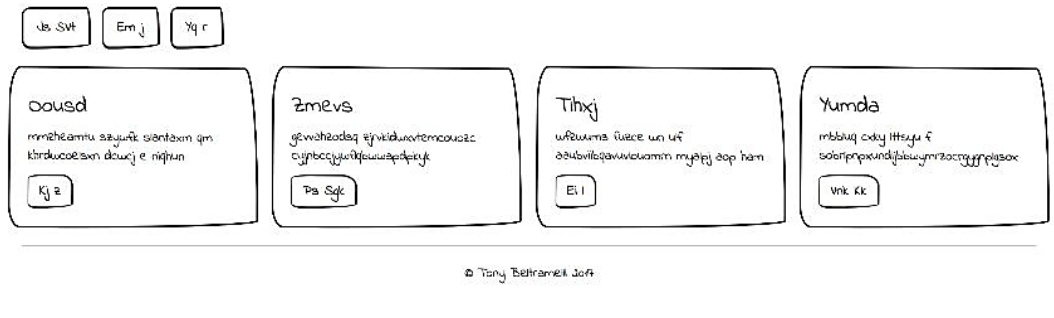


Figure 19 Example 1: Input Image (Sketch Design)

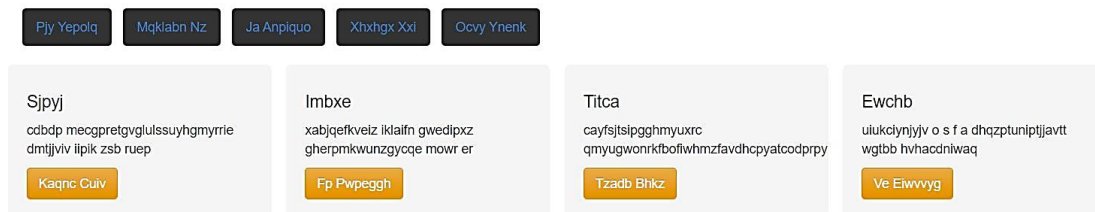


Figure 20 Example 2: Output Image (Generated Bootstrap HTML Code)

The above images are an illustration of the image-to-code generating system. The first image shows the input, and the second shows the result, which is the produced Bootstrap HTML code.

Research Report

Software Engineering Graduation

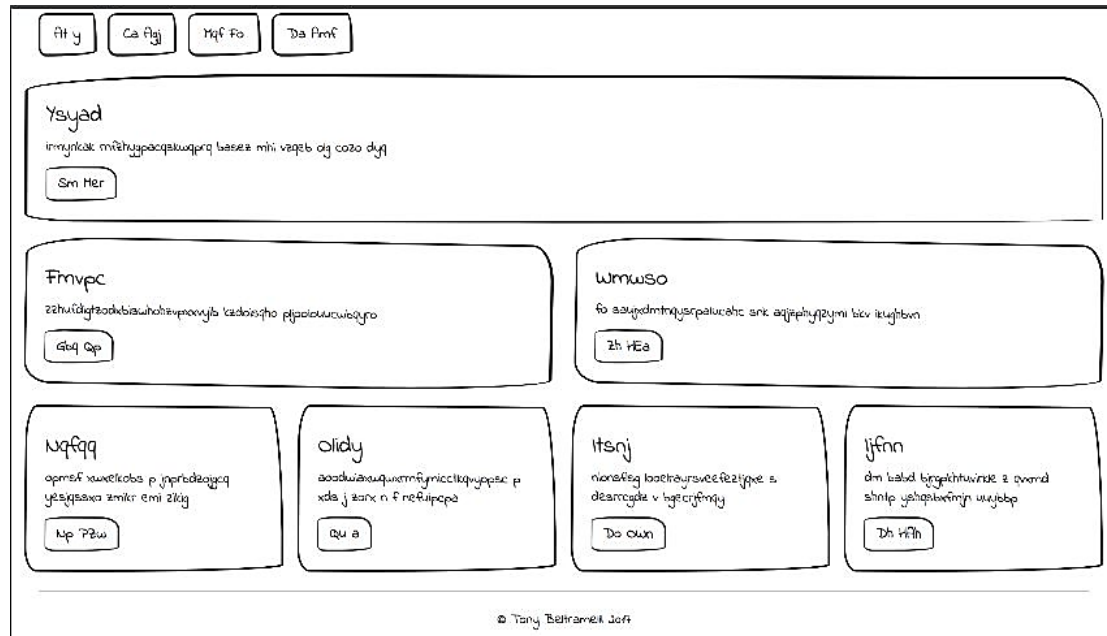


Figure 21 Example 2: Input Image (Sketch Design)

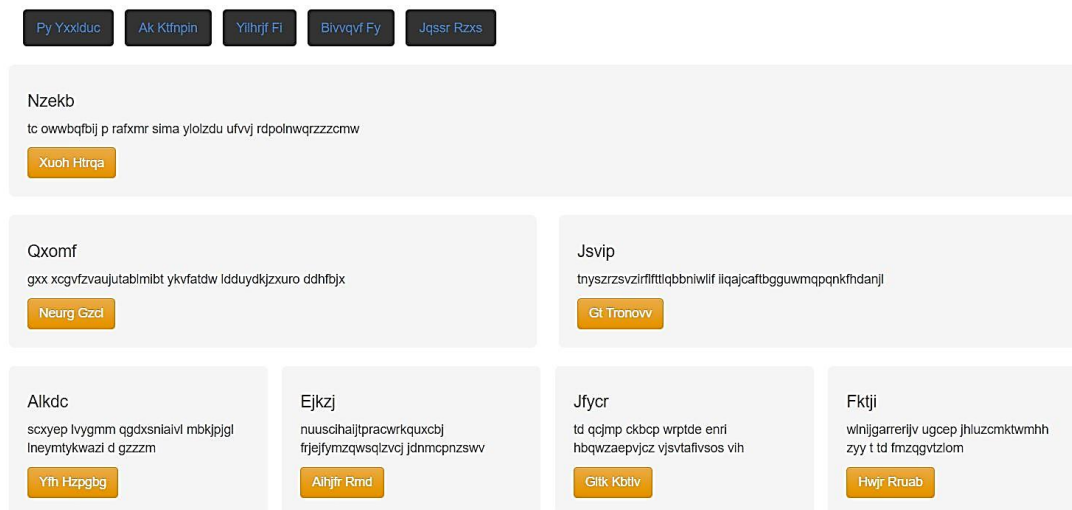


Figure 22 Example 2: Output Image (Generated Bootstrap HTML Code)

This is a second example of an image-to-code generating system. The first image shows the input, while the second shows the result, which is the generated Bootstrap HTML code.

The below image shows the full user interface for the image-to-code tool.

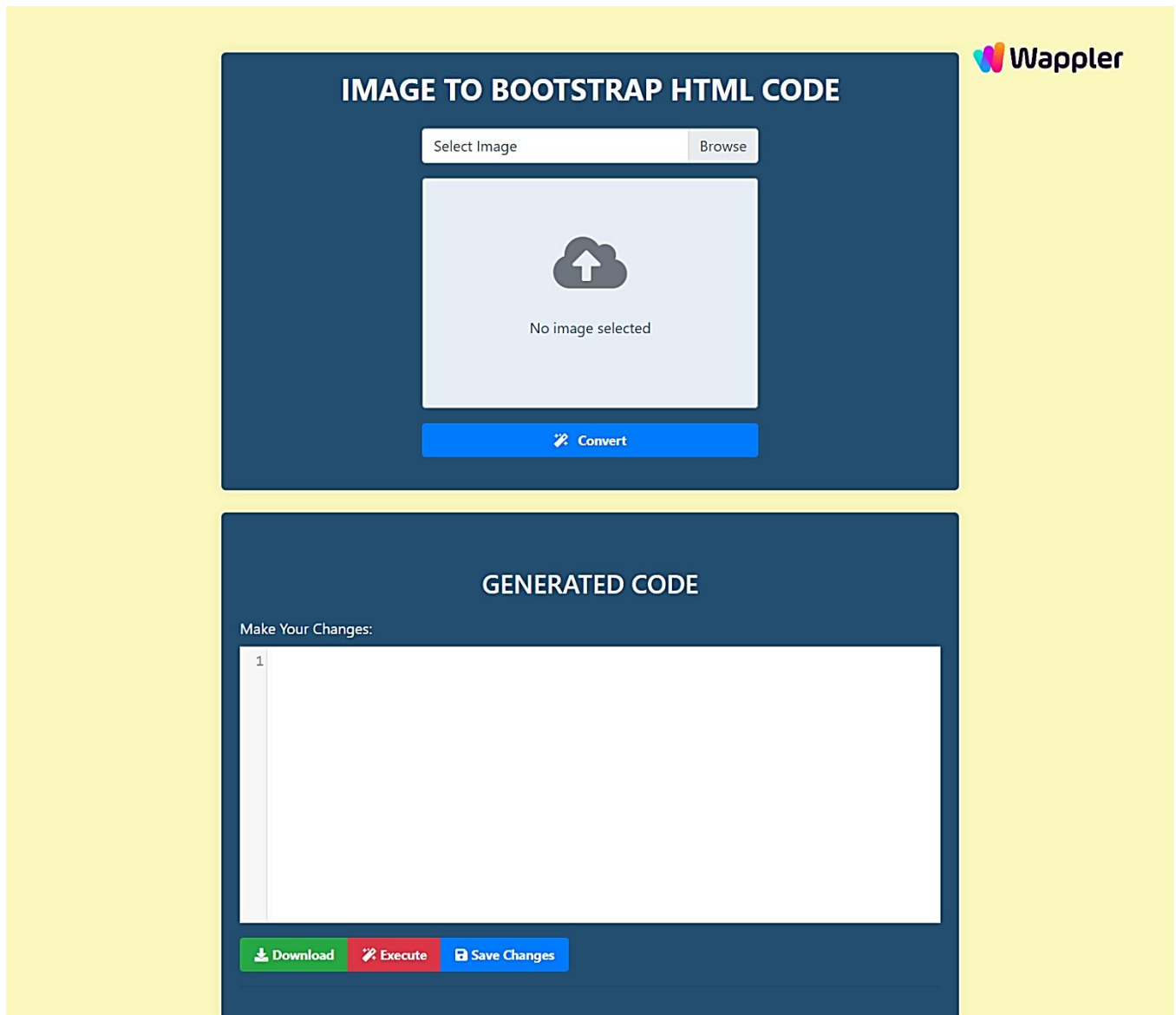


Figure 23 User-Interface for image to code tool

Research Report

Software Engineering Graduation

Text to Code:

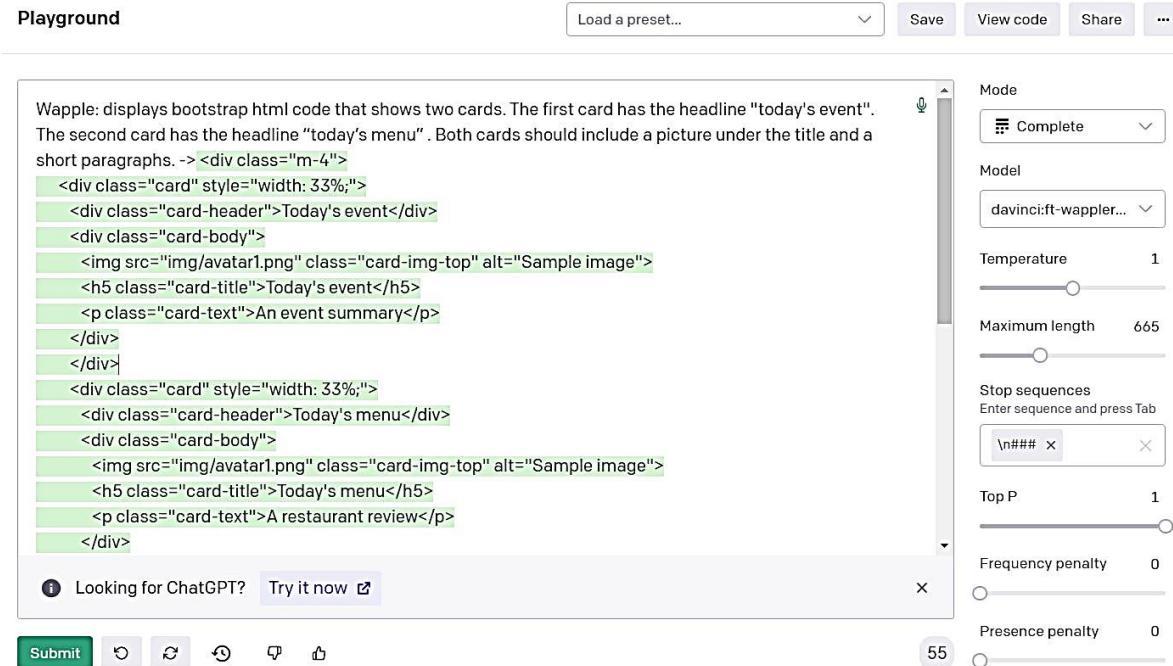


Figure 24 Text-to-Code System Output in OpenAI Playground

The image above shows the OpenAI Playground, displaying the input text and the associated code output created by the Text-to-Code system's fine-tuned model.

Research Report

Software Engineering Graduation

The image below displays a conversation between the Wappler AI and users using the fine-tuned model for the text-to-code system.

```
> Wappler: keep the same generated card and add two links before the button with the texts "link1" and "link2". ->
AI:<div class="card">
<div class="card-body">
<h4 class="card-title">Card title</h4>
<p class="card-text">Some text.</p>
<a href="#" class="card-link">Link1</a>
<a href="#" class="card-link">Link2</a>
<button type="button" class="btn btn-primary">Go</button>
</div>
</div>
> Wappler: keep the same generated card and add an image. ->
AI:<div class="card">
<div class="card-body">
<h4 class="card-title">Card title</h4>
<p class="card-text">Some text.</p>

<a href="#" class="card-link">Link1</a>
<a href="#" class="card-link">Link2</a>
<button type="button" class="btn btn-primary">Go</button>
</div>
</div>
> Wappler: keep the same generated card and add a card header and footer. ->
AI:<div class="card">
<div class="card-header">
<h4 class="card-title">Card title</h4>
</div>
<div class="card-body">
<p class="card-text">Some text.</p>

<a href="#" class="card-link">Link1</a>
<a href="#" class="card-link">Link2</a>
<button type="button" class="btn btn-primary">Go</button>
</div>
</div>
```

Figure 25 Chat between User and Wappler AI for Text-to-Code System

Research Report

Software Engineering Graduation

The image below shows the user interface that shows the created code from the text to the image tool.

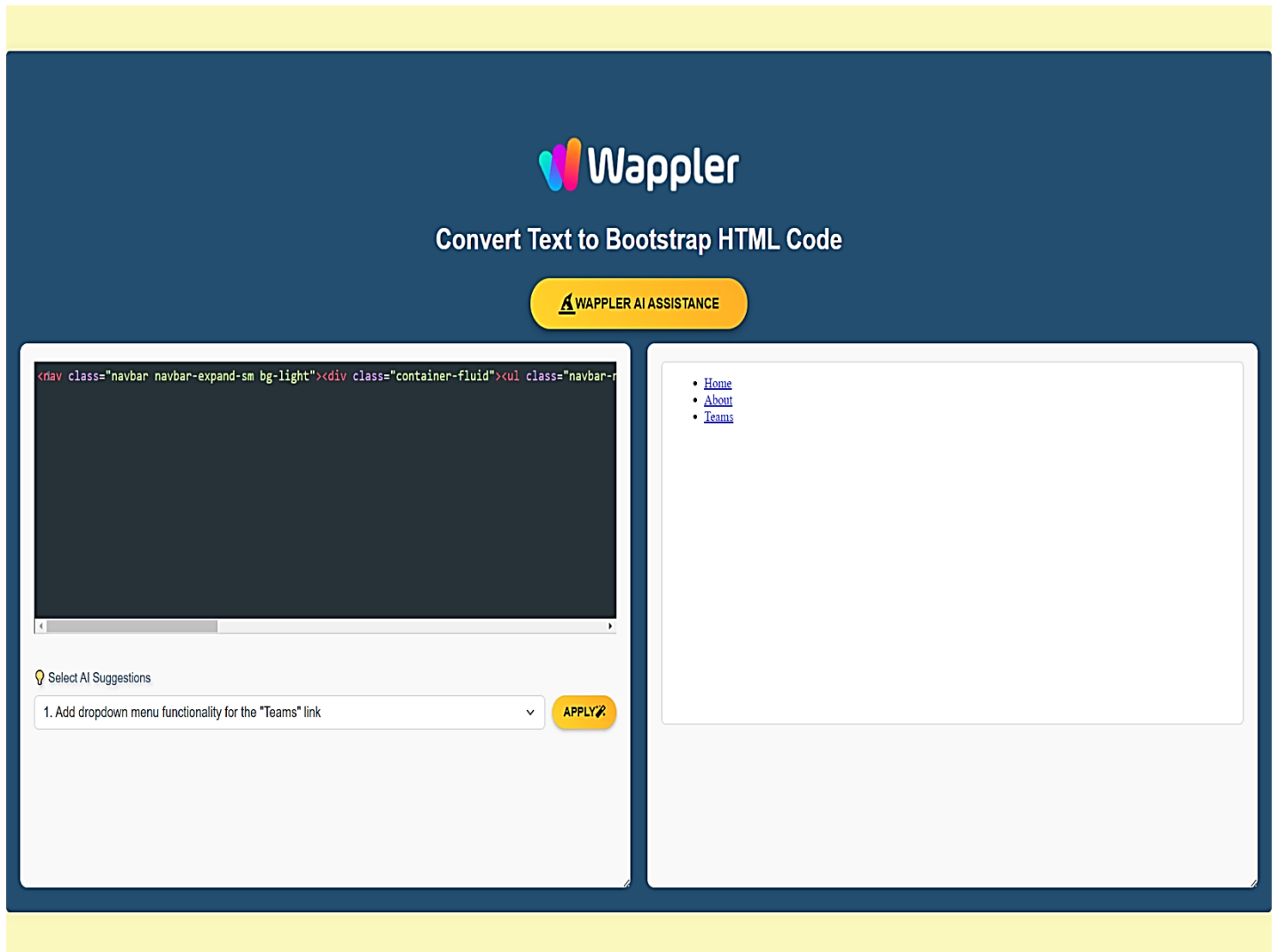


Figure 26 User-Interface for Image-To-Code Tool