Distribution Agreement

In presenting this thesis as a partial fulfillment of the requirements for a degree from Emory University, I hereby grant to Emory University and its agents the non-exclusive license to archive, make accessible, and display my thesis in whole or in part in all forms of media, now or hereafter now, including display on the World Wide Web. I understand that I may select some access restrictions as part of the online submission of this thesis. I retain all ownership rights to the copyright of the thesis. I also retain the right to use in future works (such as articles or books) all or part of this thesis.

Yijun Liu April 10, 2024

CodeCuddle: Automating Inclusive Web Design with Large Language Model

by

Yijun Liu

Chinmay Kulkarni
Adviser

Computer Science

Chinmay Kulkarni

Adviser

Emily Wall

Committee Member

Joyce C. Ho

Committee Member

2024

CodeCuddle: Automating Inclusive Web Design with Large Language Model

By

Yijun Liu

Chinmay Kulkarni

Adviser

An abstract of
a thesis submitted to the Faculty of Emory College of Arts and Sciences
of Emory University in partial fulfillment
of the requirements of the degree of
Bachelor of Science with Honors

Computer Science

2024

Abstract

CodeCuddle: Automating Inclusive Web Design with Large Language Model
By Yijun Liu

This thesis presents a novel approach to enhancing social inclusiveness in collaborative websites by leveraging the capabilities of Large Language Models (LLMs). In this work, we developed a system, CodeCuddle, which utilizes LLMs to generate socially inclusive features and auto-implement these features via LLM code completion. The primary objective is to automate the process of improving inclusivity in existing web pages.

The system design integrates the generation of socially inclusive features, code generation, and code integration using the advanced GPT-4 Turbo Chat Completion function from OpenAI. Users can input descriptions of their existing websites, select generated inclusive features, and upload their codebase. The system then intelligently generates functional web code that incorporates the selected features into the user's original code.

Experiments were conducted to evaluate the system's performance in two representative test cases: a scheduling app and a Q\&A platform. The results showed that the Q\&A platform had slightly better performance in terms of generating socially inclusive features. The Q\&A platform also has a higher number of features that met both inclusiveness and feasibility criteria, while the scheduling app had a higher feasibility score. The code generation evaluation revealed that while the LLM can generate code that compiles successfully, there are still challenges in ensuring the generated code accurately implements the intended functionality.

This thesis presents preliminary findings in the capacities of automated LLM-aided web improvement to enhance social inclusiveness. Future research directions include conducting qualitative studies to thoroughly assess the current state of social inclusiveness across websites and enabling users to upload their own code for a more comprehensive evaluation of the system's performance and adaptability.

CodeCuddle: Automating Inclusive Web Design with Large Language Model

By

Yijun Liu

Chinmay Kulkarni

Adviser

A thesis submitted to the Faculty of Emory College of Arts and Sciences
of Emory University in partial fulfillment
of the requirements of the degree of
Bachelor of Science with Honors

Computer Science

2024

Acknowledgments

I would like to express my gratitude to my advisors, Dr. Chinmay Kulkarni and Dr. Soya Park for guiding me through my thesis and providing me with the opportunity to explore the field of Human-Computer Interaction (HCI) and Human-AI Interaction. The comprehensive and insightful Human-AI Collaboration class taught by Dr. Chinmay Kulkarni has shaped my decision to continue exploring the field during my time at graduate school.

I am also thankful to Dr. Emily Wall and Dr. Joyce Ho for their contributions to my academic journey. I deeply appreciate Dr. Emily Wall for introducing me to the field of HCI and helping me to build essential toolkits in HCI research both through her classes (HCI and InfoVis) and the research project. I have developed design, system development, and qualitative and quantitative analysis skills that are essential for my future journey. Without Dr. Emily Wall, I would not have been able to get into the field of HCI. Dr. Joyce Ho has thoroughly guided me in a machine-learning research project. Particularly, I am super appreciative of her extra time in helping me to build better performing models by understanding feature and patient subgroups when the task was challenging.

In addition to my committee members, I would also like to thank Dr. Ting Li, Mengyu Chen, Dr. Vicki Hertzberg, Dr. Liang Zhao, Dr. Kathryn Wood, and Dr. Daniel Votipka for their mentoring and support in my undergraduate CS academic journey. Their guidance has been invaluable in shaping my path as a computer science student and researcher. Moreover, I would like to thank Dr. Soya Park again for all her guidance and insights on this project. This project is based on her PhD thesis works.

Lastly, I would like to thank Mingxuan Liu for his help in the writing of this thesis.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background Motivation

As collaborative technology increasingly embeds itself into our daily interactions, its technical robustness often overshadows a critical shortfall: inclusiveness. Despite their technical prowess, many collaborative tools fail to cater to diverse user groups, inadvertently fostering socially awkward situations within the tools. This oversight has been highlighted in several studies, noting a reluctance among varied user demographics to engage with these technologies due to a lack of inclusive design considerations for their groups.[16][7] In this thesis, we aim to focus on *social inclusiveness*, defined as avoiding social anxiety, awkwardness, and isolation, as well as synchronizing users from a variety of backgrounds in using online collaborative technologies.

With the growing popularity and advancement of Large Language Models (LLMs), we observe their ability in tasks ranging from natural language understanding and ideation to data analysis and code generation. Prior research demonstrates LLMs' proficiency in generating webpages and fostering creative ideation.[14][10][9] This study aims to utilize these capabilities of LLMs in automating the enhancement of inclusivity in collaborative websites. In this paper, we propose a novel approach where

LLMs generate ideas for inclusive features, auto-implement these features, and refine them based on user feedback.

1.2 Thesis Objective and Contributions

The goal of this thesis is to find a solution in designing and building an LLM-powered system that can help users improve their existing webpages' inclusivity as well as understand the efficacy of the design. The potential benefits of this study include the following:

1. Enhancing Inclusivity in Collaborative Websites: We aim to develop a tool that inputs existing websites and employs LLM-generated ideas to introduce missing inclusiveness features, thereby automating the enhancement process. This not only minimizes human effort but also capitalizes on LLMs' ability to ideate inclusively, potentially revolutionizing the design process in this domain.

2. Streamlining Web Development for Developers: Current practices often require manual integration of LLM-generated code. Our study proposes to streamline this process, significantly enhancing development efficiency. Users may also use this tool alone to improve efficiency in any web development code generation.

3. Redefining Web Design and Prototyping Paradigms: Traditional web development often involves initial prototyping to test effectiveness. With LLM-facilitated code generation, we explore the possibility of bypassing this conventional step, potentially transforming the standard workflow in web design and development.

1.3 Significance

We have identified the following stakeholders that will be benefitted from this study:

1. Web Developers and Designers: These professionals will directly benefit from the automated process of integrating inclusivity features into existing web pages, enhancing their efficiency and effectiveness in web development.

2. Low-status workers and Underrepresented User Groups: As the primary beneficiaries of increased inclusiveness in collaborative technologies, they will have a more smooth and welcoming experience in using the systems.

3. UI/UX Experts: The system may be able to be enhanced to be able to add user-written features come up by the UI/UX experts.

1.4 Thesis Organization

Chapter 2 provides an overview of the existing works in social inclusiveness and LLM web code generating, which are the two building blocks of this thesis. Chapter 3 introduces the system design for the interface, the prompting methods, as well as the algorithms for code generation and combination. Chapter 4 discusses the evaluation methods for this system and Chapter 5 presents the results. Chapter 6 concludes the thesis and Chapter 7 shows the limitations and future directions of this thesis.

Chapter 2

Background

2.1 Inclusiveness in Collaborative Technology

As collaborative technologies continue to evolve and become increasingly integrated into our daily lives, reaching a broader spectrum of user groups, a significant shortcoming in the current state of collaborative technology has emerged: the lack of emphasis on inclusiveness. Inclusiveness, in this context, refers to the practice of providing equal access to resources and opportunities for all individuals. Within the scope of this thesis, the focus is specifically on *social inclusiveness*, which is defined as the prevention of social anxiety, awkwardness, and isolation, while fostering synchronization among users from diverse backgrounds in their use of online collaborative technologies.

Park highlights the heightened social anxiety and reluctance of low-status workers to use collaborative systems in her thesis, emphasizing the need for interfaces that are more attuned to the challenges these users face when interacting with higher-status individuals.[16] Several existing works also focus on the current practices of social inclusiveness, such as implementing a self-reflection system for team practices and a language technology for multilingual conferences.[17][7]

The primary objective of this study is to enhance the social inclusiveness of

collaborative technology by leveraging the potential of Large Language Models (LLMs) generated ideas. By focusing on practical, implementable features for each system, we aim to contribute to the development of collaborative technologies that are more accessible, user-friendly, and inclusive for all users, regardless of their social status or background. Through the incorporation of LLM-generated ideas, we seek to identify and address the specific needs and challenges faced by diverse user groups, ultimately fostering a more equitable and inclusive collaborative environment.

2.2 LLMs and Web Code Generation

Large Language Model (LLM) code generation, also known as text-to-code, has emerged as a thriving and rapidly advancing field of research in both academia and industry.[4][11][13][1][15][14][20] Prominent models such as OpenAI GPT, OpenAI Codex, and Tabnine have gained significant popularity, particularly in the domain of web front-end code generation. These models assist front-end developers in quickly producing usable and readable HTML and CSS codes, streamlining the development process and enhancing productivity.[15][14][20]

The effectiveness of LLM code generation lies in its ability to understand and interpret natural language input, enabling developers to express their desired webpage structure and styling using intuitive, human-readable instructions. By leveraging the vast knowledge and patterns learned from extensive training on diverse codebases, these models can generate syntactically correct and semantically meaningful code snippets, reducing the time and effort required for manual coding.[3]

In the following sections of this thesis, we will delve deeper into the current state-of-the-art technologies in using LLM for webpage code generation, examining their strengths, limitations, and potential future directions. By providing a comprehensive overview of this rapidly advancing field, we aim to contribute to the ongoing research

and development efforts on LLM web code generation.

2.2.1 Commercial Projects

Several existing companies have successfully harnessed the power of Large Language Models (LLMs) to generate customized websites for their customers. Durable is a web tool that enables business owners to create their websites by simply providing their company's name and industry, streamlining the initial setup process.[6] Similarly, TeleportHQ offers an intuitive platform that generates entire websites based on a single line of description from the customer.[21]

One of the key advantages of these tools is their flexibility in allowing users to modify and customize the generated web pages according to their specific needs and preferences. Both Durable and TeleportHQ provide user-friendly interfaces and preset toolkits that enable users to easily adjust various design elements, such as font colors, background pictures, links, and icons. This level of customization empowers users with limited technical expertise to take control of their website's appearance and branding. Moreover, these platforms leverage the capabilities of LLMs to automate content generation, further simplifying the website creation process.

However, it is important to note that while these tools excel in generating visually appealing and content-rich static web pages, they cannot currently incorporate interactive functionalities. The generated websites serve primarily as informational or brochure-style pages, without the dynamic features and user interactivity that many modern websites require.

2.2.2 Academic Projects

End-user development (EUD) has emerged as a popular and extensively researched technique that empowers users to develop websites without the need to write complex code.[2][18][12] Numerous tools have been created to enable users to create websites by

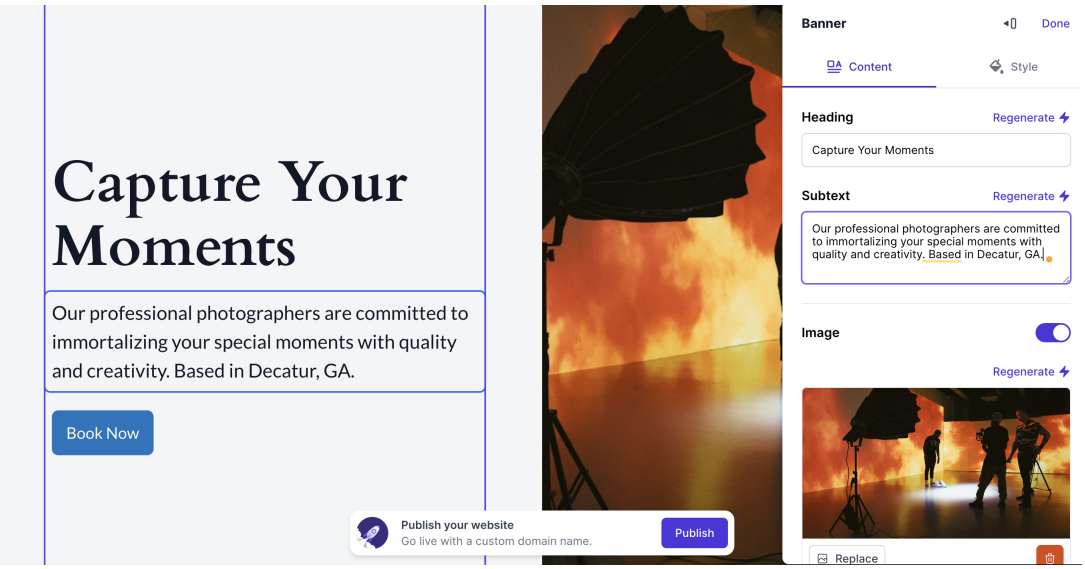

Figure 2.1: Durable Website Generation Post Modification. A recent example of an LLM-generated website. The website including the text and images is generated by LLMs. The text and pictures are editable and movable.

arranging preset elements (such as buttons and text boxes) within templates, requiring limited or no coding expertise.[5][22] The use of Large Language Models (LLMs) to develop websites represents a novel approach in EUD, aiming to provide greater flexibility and ease of use compared to traditional methods of arranging elements.[8][3]

Several studies have focused on leveraging LLMs to streamline the process of end-user development. In 2021, Huang et al. proposed a method to generate user interfaces based on textual descriptions provided by users via LLMs.[8] Calò and Russi introduced a new framework for prompting, enabling end-users to modify website elements through prompts instead of relying on preset toolkits.[3] Additionally, Li et al. developed a tool that utilizes an LLM-powered website extension to alter the user interface of websites.[10] These works collectively demonstrate the promising potential of LLMs in assisting users without coding expertise to build or modify websites. However, similar to commercial projects on LLMs in code generation, these studies primarily concentrate on static web pages. Stocco's paper in 2019 also acknowledges the possibility of AI generating interfaces but does not delve into the implementation

of dynamic features.[19]

Our study aims to elevate the application of LLMs beyond the foundational work of generating static webpages from prompts. We advance past the approaches explored by Li et al. and Calò and Russis, who focused on using LLMs for customizing webpage styles and generating HTML and CSS files from user prompts. Instead, our focus lies in enhancing existing, fully functional webpages by integrating dynamic, functional features through the use of LLMs. These features are implemented using JavaScript, enabling a more interactive and engaging user experience.

By leveraging LLMs to generate JavaScript code, our approach not only broadens the scope of LLM applications in web development but also introduces a novel dimension in upgrading and enriching user interaction and social inclusiveness on already established websites.

Chapter 3

System Design and Implementation

In this study, we developed an innovative system (CodeCuddle) that integrates generating socially inclusive features, code generation, and code integration using Large Language Models (LLMs). The primary objective of this system is to allow users to select socially inclusive features generated by LLMs and then upload their existing codebase. The system then intelligently generates functional web code that incorporates the selected features into the user's original code. To ensure cutting-edge performance and capabilities, we designed this system using the most advanced OpenAI LLM API, specifically the GPT-4 Turbo Chat Completion function.

By leveraging the power of LLMs, our system streamlines the process of enhancing web applications with socially inclusive features. Users can easily explore and select from a range of generated features that promote social inclusivity. The system then seamlessly merges these features with the user's existing codes, minimizing manual effort and ensuring a smooth integration process.

3.1 Web Feature Prompting

To generate new socially inclusive features to improve users' websites, we require user input to describe their existing websites. In this study, we ask users three

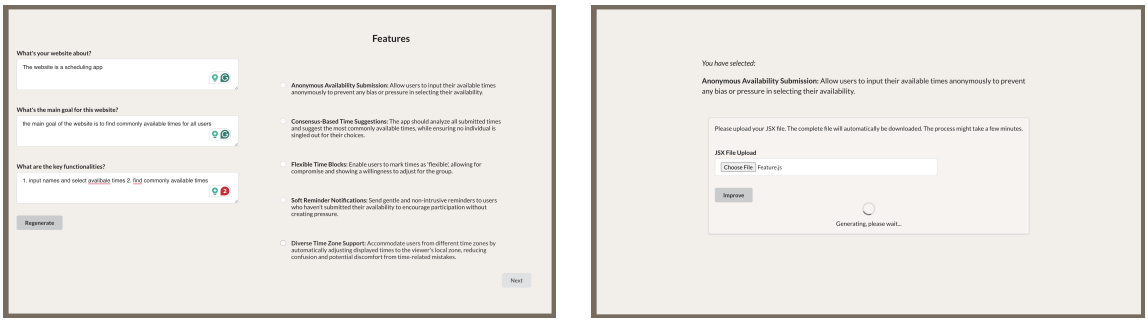

Figure 3.1: System Interface. The left image shows the text boxes for users to input and describe their existing websites and the inclusive features generated. The right image shows the interface for users to upload their existing code.

questions to understand their websites, aiming to cover a clear description of the main functionalities of their websites:

- **What's your website about?** We hope to receive the topic of the website and a few keywords on what the website is about. The interface provides an example stating that "it's a scheduling app...".
- **What's the main goal of this website?** We hope to answer the question of why users would use the tool. The interface provides an example stating that "The main goal is to find a commonly available time for all users".
- **What are the key functionalities?** This question is to ensure that the users must discuss the key functionalities in case they were not mentioned in the previous two questions as well as understand the specific functionalities of the website. The interface provides an example stating that "1. input user names and times 2. find commonly available times".

The system requires users to fill in all of the information. After obtaining the user information, we use the following prompt to obtain the list of features:

```
Prompt 1:
I currently have a website about [question 1 & 2 answer].
```

```
It has these functionalities [question 3 answer].
Now, I want to make the webpage more inclusive by avoiding socially awkward
situations, reducing the anxiety for users to use the website, and bringing
all users together regardless of their backgrounds such as language,
ethnicity, gender, and educational levels by making them feel comfortable
while using the website. Some examples include using nudges and social signalings.
Output the top seven important features in a JSON format like
{"features":[{"feature": "feature name", "description": "feature description"}]}
```

There will be a function to then check the return of this prompt to ensure that 1) it is in the correct JSON format and 2) returns 6 functionalities. After the initial generation of these functionalities will be re-checked. Specifically, we ask the LLMs to rate on the features based on the social inclusiveness and feasibility of the features, trying to provide more quantified bases on their selection process. Here is the prompt:

```
Prompt 2:
I have a list of features that I hope to implement on my website to improve
the social inclusiveness of my website on [question 1 answer].
Social inclusiveness is defined as avoiding socially awkward situations,
reducing the anxiety for users to use the website, and bringing all users
together regardless of their backgrounds such as language, ethnicity, gender,
and educational levels by making them feel comfortable while using the website.
The features are [list returned from the last prompt].
For each feature, please rate them based on these two evaluation metrics:
Social Inclusiveness: This metric evaluates whether the generated features are
directly related to social inclusiveness rather than other forms of
inclusiveness, such as accessibility or UI design. Features that specifically
address social inclusion is considered successful.
Feasibility: This metric assesses the complexity and practicality of
```

```
implementing the generated features. Features that are straightforward to
implement and do not require complex systems are considered feasible.
Features that are overly complicated and impractical to implement such as
AI-related systems or real-time systems are regarded as infeasible.
Please rate each feature on a scale of 1 to 5, where 1 indicates the lowest
rating and 5 indicates the highest rating. For each feature, provide two ratings.
Now, please only return the top 5 features based on the ratings' sum for
each feature, return in this JSON format:
{{"features":[{{"feature":"feature name", "description":"feature description",
"social_inclusiveness":"1", "feasibility":"1"}}]}}
```

Finally, these features will be sent to the front end to display and for users to select from.

3.2 LLM Code Integration

This section discusses how this thesis integrates the Large Language Model (LLM) generated code into users' existing code. Specifically, it will discuss the code type we chose and the integration algorithm we adopted.

3.2.1 Code Type

Caló and Russis's study introduces a novel approach to generating new code and modifying existing code based on prompts.[3] In the context of webpage generation, their system allows the LLMs to generate multiple HTML pages and a single CSS file.[3] When users need to modify the webpage, the prompt instructs the LLM to specify the file name, change type (add or replace), and the specific lines where the

changes should occur.[3] This modification approach effectively eliminates the risk of the LLM inadvertently altering or removing existing user input codes.

While Caló and Russis's approach is effective for static webpage generation and modification, our system aims to focus more on interactive features, which require the integration of JavaScript. Dealing with multiple files and file types (HTML, CSS, JavaScript) simultaneously can introduce complexity and potentially reduce the effectiveness of the LLM in code generation.

To address this challenge, our approach is to adapt React.js, a popular JavaScript library for building user interfaces. React.js allows for the combination of HTML and JavaScript within a single file format called JSX (JavaScript XML). JSX enables developers to write HTML-like code directly within JavaScript, simplifying the structure and organization of the codebase. Additionally, React.js provides the option to include CSS styles within the same JSX file, further streamlining the development process by combining all types of front-end codes.

By leveraging React.js and the JSX file format, our system simplifies the code generation process for the LLM. Instead of managing multiple separate files for HTML, CSS, and JavaScript, the LLM can generate code within a single JSX file, which encapsulates all the necessary components for rendering and interactivity. This approach reduces the complexity of dealing with multiple file types and allows the LLM to focus on generating cohesive and functional code snippets.

3.2.2 LLM Code Generation and Integration Algorithm

Similar to Caló and Russis's study, we leverage the power of Large Language Models (LLMs) to generate web code and utilize prompts to guide the necessary changes. The prompt we employ to generate the webpage is as follows:

```
Prompt 3:
You are tasked with modifying an existing JSX file that is a website
```

```
about [question 1 & 2 answer]. When modifying the JSX, please output
the changes in the following JSON format, accommodating two change
types (add and replace):
{"CodeChanges":[
    {"change_type": "add", "lines": "(12)", "code":"<JSX code here>"},
    {"change_type": "replace", "lines": "(1,10)", "code":"<JSX code here>"}]
}
When adding a new code snippet, please specify only the lines where the code
should be inserted; when replacing existing code, please indicate the starting
and ending lines of the code to be replaced in the format (1,10).
The feature I would like you to implement is: [selected feature].
And here's the existing JSX code: [user code].
Please add only the necessary code snippets required to realize the desired
functionality, keeping the code concise. Be cautious when replacing any lines,
and ensure that you do not comment out any code that is still needed.
Additionally, please be mindful to import packages in the first line;
please make sure all the variables used are defined prior to using them.
```

In additional to the prompt to obtain the changes in the code, the following prompt is used to ensure that the code generated is able to compile:

```
Prompt 4:
Please make this code below functional and able to compile.
[code from the previous prompt].
Return in the following JSON format {{"Code":"<Python code here>"}}
```

To handle the potential presence of commented original code, we employ an additional prompt:

```
Prompt 5:
I have noticed that there are commented lines in the code snippet
indicating the inclusion of the original code. Here is the new code snippet:
[code returned from the last prompt with numbered lines]. And here is my
original code: [user code]. Could you please replace the commented lines
with the necessary code that needs to be added?
Please return the modified code in the following JSON format:
{"CodeChanges":[
    {"change_type": "replace", "lines": "(1,10)",
    "code":"<new, uncommented JSX code>"}]
}
```

By utilizing these carefully crafted prompts, we enable the LLM to generate and modify web code effectively. The JSON format provides a structured approach to communicate the required changes, specifying the change type (add or replace), the affected lines, and the corresponding code snippet.

The initial prompt 3 clearly conveys the context of the existing website, the specific feature to be implemented, and the current JSX code. This comprehensive information allows the LLM to generate code snippets that are tailored to the website's specific requirements as well as provide LLM with an understanding of the website.

The emphasis on code conciseness and careful replacement ensures that the generated code is efficient, readable, and maintains the existing functionality of the website. By cautioning the LLM about replacing lines and preserving necessary code, we minimize the risk of inadvertently breaking the website's functionality during the modification process. Prompt 4 ensures that the code should be able to compile.

The follow-up prompt 5 addresses the potential inclusion of commented original code by the LLM. By providing the LLM with the new code snippet containing numbered lines and the original user code, we enable the LLM to replace the commented

lines with the necessary code that needs to be added. This step ensures that the generated code is clean, uncommented, and ready for integration.

Algorithm 1 shows how these prompts are utilized.

3.3 Interface Design

3.3.1 System Workflow

The system workflow, illustrated in Figure 3.2, is divided into two main stages: the ideation stage and the code generation stage. The ideation stage, primarily described in Section 3.1 and depicted on the left side of Figure 3.1, involves user interaction and feature selection. Once the user has chosen the desired features, the system proceeds to the code generation and integration stage, detailed in Section 3.2. During this stage, the system generates the necessary code based on the user's feature selection and integrates it with the existing codebase. Upon completion, the updated code is automatically downloaded to the user's device, streamlining the process. However, the system offers flexibility, allowing users to navigate back to the previous page with saved website descriptions, select additional or alternative features, and re-upload their web codes for further modifications.

3.3.2 Minimizing Coding Efforts

As discussed in Section 2.2.2, this system also focused on the effort of reducing the need to read and write codes, which aligns with the goals of many end-user development research projects. The main objective is to simplify the process of code modification and generation, making it more accessible to users with limited programming expertise.

In the proposed system, users will only need to upload their existing code file, and the new, generated code file will be automatically downloaded upon completion. This process eliminates the need for users to manually read through the code and make

Algorithm 1 LLM Code Integration

Input: Original code C, feature F
Output: Modified code C_{modified}
procedure CombineCode(C, F)
    $JSON \leftarrow$ Generate JSON code instructions based on C and F via LLM prompting
    $attempt \leftarrow 1$
    $maxAttempts \leftarrow 3$
    $offsetLines \leftarrow 0$
    for each *instruction* in $JSON.instructions$ **do**
        if *instruction.type* is *replace* **then**
            if ContainsPhrases(*instruction*) **or** IsShorter(*instruction.lines*, *C.lines*) **then**
                while ($attempt \leq maxAttempts$) **do**
                    instruction $\leftarrow$ Prompt again with Prompt 5 to fill the replace lines
                    $attempt \leftarrow attempt + 1$
                end while
                $attempt \leftarrow 1$
            end if
            $C_{\text{new}} \leftarrow$ Replace lines in C according to *instruction*
        else if *instruction.type* is *add* **then**
            $C_{\text{new}} \leftarrow$ Add lines to C according to *instruction*
        end if
    end for
     $C_{\text{final}} \leftarrow$ Prompt with C Again with Prompt 4
    return C_{final}
end procedure
function ContainsPhrases(C)
    if C contains phrases “fill”, “your code”, “rest of”, “existing code”, “original”. **then**
        return true
    end if
    return false
end function
function IsShorter(C_{new}, C)
    if number of lines in C_{new} ¡ number of lines in C **then**
        return true
    end if
    return false
end function

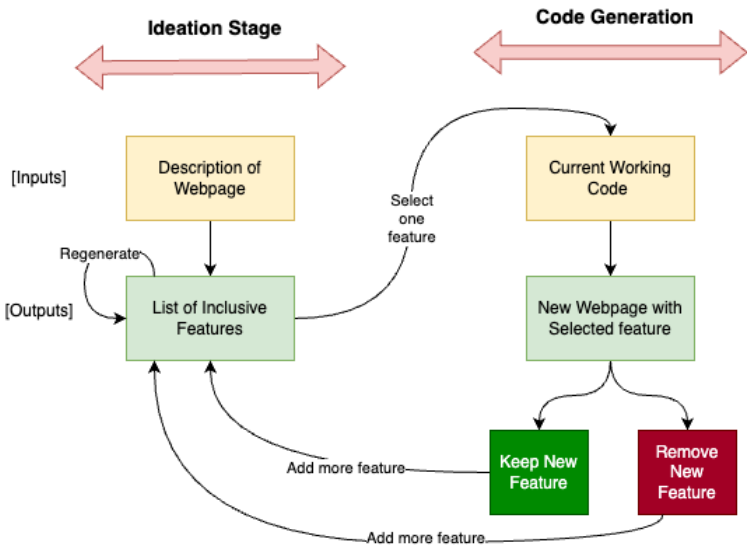

Figure 3.2: Workflow for CodeCuddle

changes themselves, saving time and effort.

Chapter 4

Evaluation

The evaluation of this system focuses on two primary aspects: inclusiveness improvement and code generation and integration. To assess the functionality and effectiveness of the system, we employed two simple test cases as the target websites: a scheduling app and a Q&A platform.

The scheduling app serves as a representative example of a collaborative tool that requires user input and coordination. The core functionality of the app involves collecting the availability of each user and identifying commonly available time slots. The Q&A platform, on the other hand, represents a different type of collaborative environment where users engage in knowledge sharing and problem-solving. The platform enables users to post questions and provide answers. These test cases allow us to evaluate how well the system generates and integrates code snippets that enhance inclusiveness features.

4.1 Inclusiveness Improvement

LLMs have shown remarkable abilities in generating human-like text and ideas, but their inherent randomness can lead to challenges when applied to specific domains. In the context of generating features related to social inclusiveness, LLMs struggle

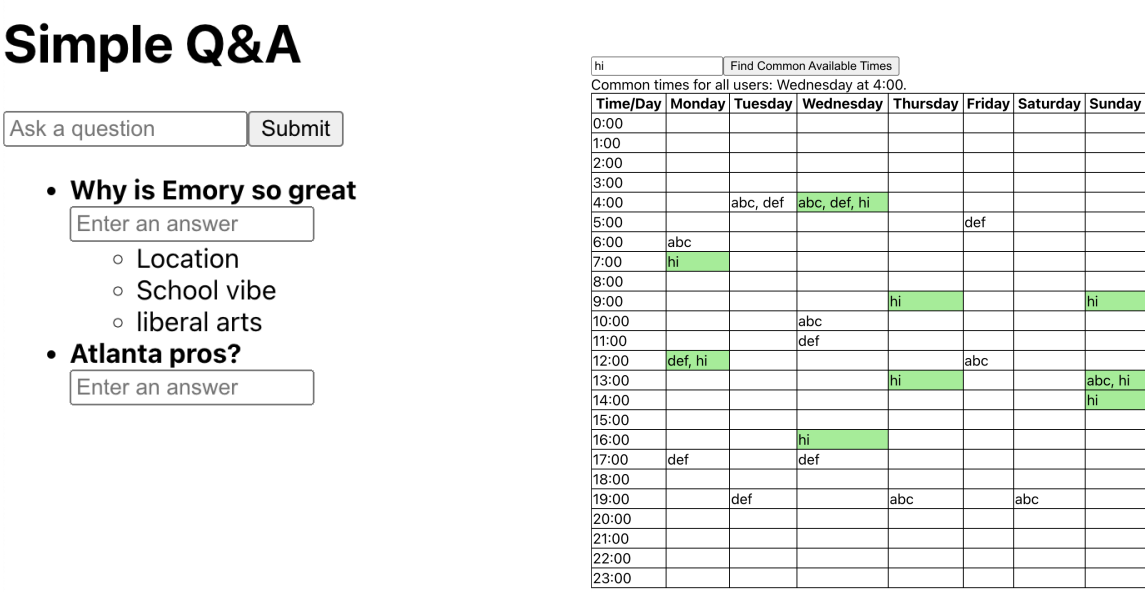

Figure 4.1: Testing Systems. The left image shows the Q&A platform. The right image shows the scheduling app.

to differentiate between social inclusiveness and other forms of inclusiveness, such as accessibility and UI design. Moreover, the generated features can sometimes be overly complex, making them difficult to implement in one step, especially when they involve AI systems, real-time modifications, or collaboration. This section aims to evaluate the effectiveness of prompts in guiding LLMs to generate socially inclusive and feasible features.

In the recent work by Lee et al., the researchers explore various methods for evaluating human-LLM interaction across different LLM models.[9] In their study, one task they focused on was evaluating the creative task of metaphor generation.[9] To assess the output of this creative task, they defined a metric called "acceptance," which measures the degree to which the model's output is accepted by the users.[9]

Similarly, our study aims to evaluate the inclusive features generated by the LLM. While this task shares similarities with metaphor generation, it is more complex. Generating inclusive features requires the model to not only demonstrate creativity

but also to understand and incorporate principles of social inclusiveness. To "accept" the feature, the feature has to satisfy 1) it complies with the social inclusiveness definition and 2) it is feasible to be implemented. Therefore, the evaluation process involves assessing the generated features based on two key criteria: their relevance to social inclusiveness and their feasibility for implementation. Specifically, we define social inclusiveness and feasibility as follows:

1. Social Inclusiveness: This metric evaluates whether the generated features are directly related to social inclusiveness rather than other forms of inclusiveness, such as accessibility or UI design. Features that specifically address social inclusion are considered successful.

2. Feasibility: This metric assesses the complexity and practicality of implementing the generated features. Features that are straightforward to implement and do not require complex systems such as changes in JavaScript that do not require external packages are considered feasible. Features that are overly complicated or impractical to implement such as AI or real-time modifications are regarded as infeasible.

In each run, a set of five features will be evaluated. The features will be assessed as a group (the five features together) for their social inclusiveness and feasibility. Each feature will be assigned a score of either 0 or 1 based on whether it satisfies the criteria for social inclusiveness and feasibility. The total score for each category will range from 0 (lowest) to 5 (highest). The evaluation process will be repeated for 10 runs in each task (Q&A and scheduling app).

In addition to the binary labeled ratings, we will also disclose the GPT-generated ratings for the features. In prompt 2, when we attempt to refine the features, we use a prompt that includes the definitions of social inclusiveness and feasibility to generate ratings for each feature. In the GPT-generated ratings, each feature will be rated on

a scale from 1 to 5. We will report the average GPT-generated ratings for the same features that were labeled by the human labeler, allowing for a comparison between the human and AI-generated assessments.

4.2 Code Generation

To evaluate the code generation aspect of the system, we focus on assessing whether the generated code complies with the specified requirements and successfully realizes the intended functionality. This evaluation is carried out by testing each of the two sample websites, the scheduling app and the Q&A platform, with ten randomly selected representative features that were labeled both 1 for feasibility and social inclusiveness in the previous 4.1 section.

During the evaluation process, we systematically execute the test cases on each website and compare the actual behavior of the generated code with the expected outcomes. This involves manually inputting the website, selecting the features, and verifying whether the output complies and performs the desirable outcome.

The evaluation metrics for code generation include two parts:

1. Functional Correctness: This metric assesses whether the generated code accurately implements the specified functionality for each feature. It involves verifying that the code produces the expected output or behavior when given valid inputs. The functional correctness success score is binary, where a score of 1 indicates successful implementation of the function, and a score of 0 indicates a failure.

2. Code Compilation Success: This metric evaluates whether the generated code can be successfully compiled without any syntax errors or compilation issues. If the code compiles successfully without any errors, it indicates that the generated code is syntactically correct and follows the language's grammar and structure.

The code compilation success score is binary, where a score of 1 indicates successful compilation and a score of 0 indicates compilation failure.

Chapter 5

Results

We conducted experiments to evaluate the performance of a system designed to generate features and code snippets that enhance inclusiveness in two representative test cases: a scheduling app and a Q&A platform. The scheduling app aims to help users find commonly available times for all participants within a week, while the Q&A platform enables users to collaboratively ask and answer questions anonymously without restrictions.

For each test case, we need to answer the following questions to describe the interfaces, as previously stated in 3.1 Web Feature Prompting:

1. What's your website about?
2. What's the main goal of this website?
3. What are the key functionalities?

For the scheduling app, we provided the following information:

1. The website is a scheduling app.
2. The main goal is to find commonly available times for all users in a week.
3. 1) Input names and available times. 2) Find commonly available times.

For the Q&A platform, we provided the following details:

1. The website is a Q&A platform.
2. The main goal is to collaboratively ask and answer questions.
3. Users can input questions and answer them without any restrictions.

The experiments focused on two primary aspects: inclusiveness improvement and code generation and integration. We evaluated the system's ability to generate features that enhance social inclusiveness and assessed the functionality and effectiveness of the generated code snippets.

In the following sections, we will present the results of our experiments.

5.1 Inclusiveness Features

The section aimed to evaluate the effectiveness of prompts in guiding LLMs to generate socially inclusive and feasible features for two distinct tasks: a Q&A platform and a scheduling app. The generated features were rated on two criteria: social inclusiveness and feasibility, with a binary score of 0 and 1 for each feature for a generation (five features for each generation). The scores are added up for each generation for social inclusiveness and feasibility, resulting in a maximum score of 5 and a minimum score of 0. Additionally, this section also provides the results from GPT-based evaluation, which ranks each feature from 1-5. The results for GPT-based evaluation are averages of the feature scores.

The results in Table 5.1 show that for the scheduling app, in each generation, an average of 3.6 out of 5 features were socially inclusive, 2.8 were feasible, and 1.4 satisfied both criteria. In comparison, the Q&A platform had slightly higher scores, with an average of 3.8 out of 5 features being socially inclusive, 2.4 being feasible, and 2.2 satisfying both criteria in each generation. These findings suggest that the

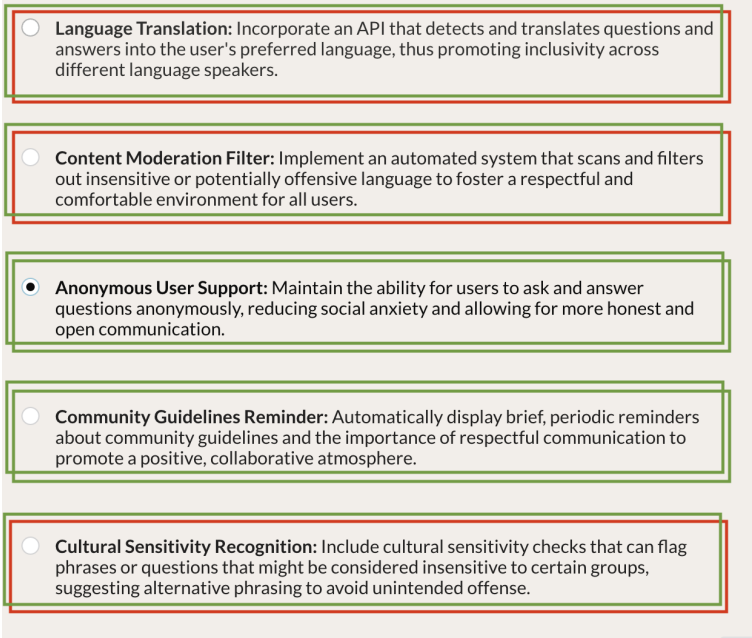

Figure 5.1: Exemplary Features. The boxes on the top represent Social Inclusiveness and the boxes on the bottom represent Feasibility. Red represents a 0 and green represents a 1.

Q&A platform had slightly better performance in terms of social inclusiveness and the number of features that met both criteria, while the scheduling app had a higher feasibility score. The "Satisfy Both" scores were important because they demonstrated the number of features that could be in fact implemented.

Table 5.1: Inclusiveness Features Performance

Website	**Social Inclusiveness**	**Feasibility**	**Satisfy Both**
Scheduling App	3.6	2.8	1.4
Q&A Platform	3.8	2.4	2.2

Table 5.2 presents the GPT-generated ratings for social inclusiveness and feasibility. The GPT-generated ratings for social inclusiveness were 4.28 for the scheduling app and 4.4 for the Q&A platform, indicating that the LLM considered the generated features to be highly socially inclusive. On the other hand, the GPT-generated ratings for feasibility were 3.44 for the scheduling app and 3.52 for the Q&A platform, suggesting that the LLM regarded the features as moderately feasible.

Table 5.2: GPT Inclusiveness Rating

Website	**Social Inclusiveness**	**Feasibility**
Scheduling App	4.28	3.44
Q&A Platform	4.4	3.52

Although the GPT-generated ratings have different scoring criteria, they show similar trends in social inclusiveness as the binary evaluation outcomes. In both cases, the social inclusiveness criteria score higher in the Q&A platform. On the other hand, feasibility shows that the two methods are different.

5.2 Code Generation

The results of the code generation evaluation for the scheduling app and Q&A platform are presented in Table 5.1. The features selected have scored both 1s in social inclusiveness and feasibility.

Table 5.3: Code Generation Performance

Website	**Code Compilation**	**Functional Correctness**
Scheduling App	80%	60%
Q&A Platform	90%	70%

For the scheduling app, the code compilation success rate was 80%, indicating that 8 out of the 10 selected features had code that was successfully compiled without any syntax errors or compilation issues. The functional correctness rate for the scheduling app was 60%, suggesting that 6 out of the 10 features had generated code that accurately implemented the specified functionality. All the features that failed to be compiled included features involving local times or language supports, where the code generated includes some fictional packages. Among those who were compiled, two of them failed to realize the desired functionalities. These two features are the Anonymized Feedback Collection and Bad Word filter, where the generated code only modified the front end for feedback collection and the package used for bad word does

not work.

In the case of the Q&A platform, the code compilation success rate was 80% and the functional correctness was 70%. This means that 9 out of the 10 selected features had code that was successfully compiled and 7 out of 10 have been accurately implemented the intended functionality. The feature that failed to compile was Anonymity Assurance Prompt, and the features that failed to be functionally correct were user name modification and inclusivity reminder text, where the text was empty.

In both cases, the functional correctness rates were lower, at 60% for the scheduling app and 70% for the Q&A platform comparing their code completion percentages. This suggests that while the LLM can generate code that compiles successfully, there are still challenges in ensuring that the generated code accurately implements the intended functionality. There are still questions about whether it was the features that were not well-defined so that when implemented, they did not function or it was the LLM that failed to generate feasible code for that function.

Chapter 6

Conclusion

This thesis presents a novel approach to enhancing social inclusiveness in collaborative websites by leveraging the capabilities of Large Language Models (LLMs). We develop a system that uses LLMs to generate ideas, auto-implement features, and refine them based on user feedback. We aim to automate the process of improving inclusivity in existing web pages.

Building upon the existing work in social inclusiveness and LLM web code generation, our study takes a step further by focusing on the integration of dynamic, functional features using JavaScript. This advancement goes beyond the generation of static web pages and enables a more interactive and engaging user experience.

The evaluation of this system focuses on how the system can enhance inclusiveness and integrate code for a scheduling app and a Q&A platform. It examines the features generated by the system as well as the code generation and integration process. The study measures the features' inclusiveness by assessing whether features promote social inclusion, and feasibility by their simplicity and practicality for implementation. It found that both platforms performed reasonably well in generating socially inclusive features, but they had varying success with feasibility and meeting both criteria simultaneously. For the Q%A platform, on average, only 2.2 out of 5 features are

both feasible and satisfy the social inclusiveness definition; for the scheduling app, the number drops to 1.4. In terms of code generation and integration, both of the test cases (scheduling app and Q&A platform) performs well for generating codes that are able to compile, but yet only around 60%-70% of times the features are functionally implemented.

Chapter 7

Limitations and Future Direction

This work only presents a preliminary finding in the capacities of automated LLM-aided web improvement to improve social inclusiveness. Many potential future directions could be explored. Particularly, this study suffers from two major limitations: the lack of qualitative studies on social inclusiveness and large user experiments. Below are the potential future works that could be done on these two topics.

1. A qualitative study in understanding social inclusiveness in online collaborative platforms. The findings from this thesis suggest that the quality of the generated features was inconsistent and the quality was not well. In future research, I believe that it is important to move beyond refining the prompts used to generate features and instead focus on thoroughly assessing the current state of social inclusiveness across a wide range of websites. By conducting a rigorous quantitative analysis, future studies on this topic can establish a baseline understanding of the prevalence and effectiveness of existing inclusiveness features. This foundational knowledge will serve as a valuable benchmark for evaluating the social inclusiveness on websites and therefore how to improve the current state of web development.

2. Crowdsourced user experiment allowing users to upload their own code. The

current study relied on simple and easy template code, which may not adequately represent the complexity and diversity of real-world websites. In future research, I believe it is important to enable users to upload their own code for their own systems, allowing to assess the LLM-powered system's ability to generate inclusive features for a broader range of websites. This approach will provide more comprehensive insights into the system's performance and adaptability.

Appendix A

Appendix

The interface is openly published at Heroku.

Bibliography

[1] Anthropic. Claude. https://www.anthropic.com, 2023.

[2] Barbara Rita Barricelli, Fabio Cassano, Daniela Fogli, and Antonio Piccinno. End-user development, end-user programming and end-user software engineering: A systematic mapping study. *Journal of Systems and Software*, 149:101–137, 2019. ISSN 0164-1212. doi: 10.1016/j.jss.2018.11.041. URL https://doi.org/10.1016/j.jss.2018.11.041.

[3] Tommaso Calò and Luigi De Russis. Leveraging large language models for end-user website generation. In Lucio Davide Spano, Albrecht Schmidt, Carmen Santoro, and Simone Stumpf, editors, *End-User Development*, volume 13917 of *Lecture Notes in Computer Science*, pages 58–79, Cham, 2023. Springer. ISBN 978-3-031-34433-6. doi: 10.1007/978-3-031-34433-6_4. URL https://doi.org/10.1007/978-3-031-34433-6_4.

[4] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss,

Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code, 2021.

[5] Davide Di Ruscio, Dimitrios Kolovos, Juan de Lara, Alfonso Pierantonio, Massimo Tisi, and Manuel Wimmer. Low-code development and model-driven engineering: Two sides of the same coin? *Software and Systems Modeling*, 21(2):437–446, 2022. ISSN 1619-1374. doi: 10.1007/s10270-021-00970-2. URL `https://doi.org/10.1007/s10270-021-00970-2`.

[6] Durable. Durable: Ai website builder. `https://durable.co/`, 2023.

[7] Ge Gao, Naomi Yamashita, Ari M.J. Hautasaari, and Susan R. Fussell. Improving multilingual collaboration by displaying how non-native speakers use automated transcripts and bilingual dictionaries. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, CHI '15, page 3463–3472, New York, NY, USA, 2015. Association for Computing Machinery. ISBN 9781450331456. doi: 10.1145/2702123.2702498. URL `https://doi.org/10.1145/2702123.2702498`.

[8] Forrest Huang, Eldon Li, Xuechen Zhou, John F. Canny, and Yang Li. Creating user interface mock-ups from high-level text descriptions with deep-learning models, 2021. URL `https://arxiv.org/abs/2110.07775`.

[9] Mina Lee, Megha Srivastava, Amelia Hardy, John Thickstun, Esin Durmus, Ashwin Paranjape, Ines Gerard-Ursin, Xiang Lisa Li, Faisal Ladhak, Frieda Rong, Rose E. Wang, Minae Kwon, Joon Sung Park, Hancheng Cao, Tony Lee, Rishi

Bommasani, Michael Bernstein, and Percy Liang. Evaluating human-language model interaction, 2024.

[10] Amanda Li, Jason Wu, and Jeffrey P Bigham. Using llms to customize the ui of webpages. In *Adjunct Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*, UIST '23 Adjunct, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9798400700965. doi: 10.1145/3586182.3616671. URL https://doi.org/10.1145/3586182.3616671.

[11] Ziyang Luo, Can Xu, Pu Zhao, Qingfeng Sun, Xiubo Geng, Wenxiang Hu, Chongyang Tao, Jing Ma, Qingwei Lin, and Daxin Jiang. Wizardcoder: Empowering code large language models with evol-instruct, 2023.

[12] Abdallah Namoun, Athanasia Daskalopoulou, Nikolay Mehandjiev, and Zhang Xun. Exploring mobile end user development: Existing use and design factors. *IEEE Transactions on Software Engineering*, 42(10):960–976, 2016. ISSN 0098-5589. doi: 10.1109/TSE.2016.2532873. URL https://doi.org/10.1109/TSE.2016.2532873.

[13] Erik Nijkamp, Bo Pang, Hiroaki Hayashi, Lifu Tu, Huan Wang, Yingbo Zhou, Silvio Savarese, and Caiming Xiong. Codegen: An open large language model for code with multi-turn program synthesis, 2023.

[14] OpenAI. Gpt-3. https://openai.com/blog/gpt-3-apps/, 2020.

[15] OpenAI. Openai codex. https://openai.com/blog/openai-codex/, 2021.

[16] Soya Park. *Designing Inclusiveness in Collaborative Technology.* PhD thesis, MIT, 2023.

[17] Soya Park and Chinmay Kulkarni. Retrospector: Rapid collaborative reflection to improve collaborative practices. *Proceedings of the ACM on Human-Computer*

Interaction, 7(CSCW2):293, October 2023. ISSN 2573-0142. doi: 10.1145/3610084. URL https://doi.org/10.1145/3610084.

[18] Jennifer Rode, Mary Beth Rosson, and Manuel A. Pérez Quinones. End user development of web applications. In Henry Lieberman, Fabio Paterno, and Volker Wulf, editors, *End User Development*, volume 9 of *Human-Computer Interaction Series*, pages 161–182. Springer, Dordrecht, 2006. ISBN 978-1-4020-5386-3. doi: 10.1007/1-4020-5386-X_8. URL https://doi.org/10.1007/1-4020-5386-X_8.

[19] Andrea Stocco. How artificial intelligence can improve web development and testing. In *Companion Proceedings of the 3rd International Conference on the Art, Science, and Engineering of Programming*, Programming '19, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450362573. doi: 10.1145/3328433.3328447. URL https://doi.org/10.1145/3328433.3328447.

[20] Tabnine. Tabnine. https://www.tabnine.com/, 2023.

[21] TeleportHQ. Ai website builder. https://teleporthq.io/ai-website-builder, 2023.

[22] Miranda Woo. The rise of no/low code software development–no experience needed? *Engineering*, 6(9):960–961, 2020. ISSN 2095-8099. doi: 10.1016/j.eng.2020.07.007. URL https://doi.org/10.1016/j.eng.2020.07.007.