AI-Augmented Feature to Edit and Design Mobile Applications

Ashley Granquist
David Y.J. Kim
Evan Patton
ashleymg@mit.edu
dyjkim@mit.edu
ewpatton@mit.edu
Massachusetts Institute of Technology
Cambridge, MA, USA

ABSTRACT

We are developing an AI assistance platform that enables individuals with a limited technical background, such as children, to create mobile applications from natural language input. The platform is based on MIT App Inventor and allows users to easily edit the interface and functionality of the components of their app using textual commands. The goal of the platform is to empower children and others without a background in coding with the ability to create their own mobile applications and foster their creativity and problem-solving skills in a fun and interactive way.

CCS CONCEPTS

• **Human-centered computing → Ubiquitous and mobile computing systems and tools**.

KEYWORDS

Human-Computer Interaction, Mobile Application, MIT App Inventor

ACM Reference Format:
Ashley Granquist, David Y.J. Kim, and Evan Patton. 2023. AI-Augmented Feature to Edit and Design Mobile Applications. In *25th International Conference on Mobile Human-Computer Interaction (MobileHCI '23 Companion), September 26–29, 2023, Athens, Greece.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3565066.3608248

1 INTRODUCTION

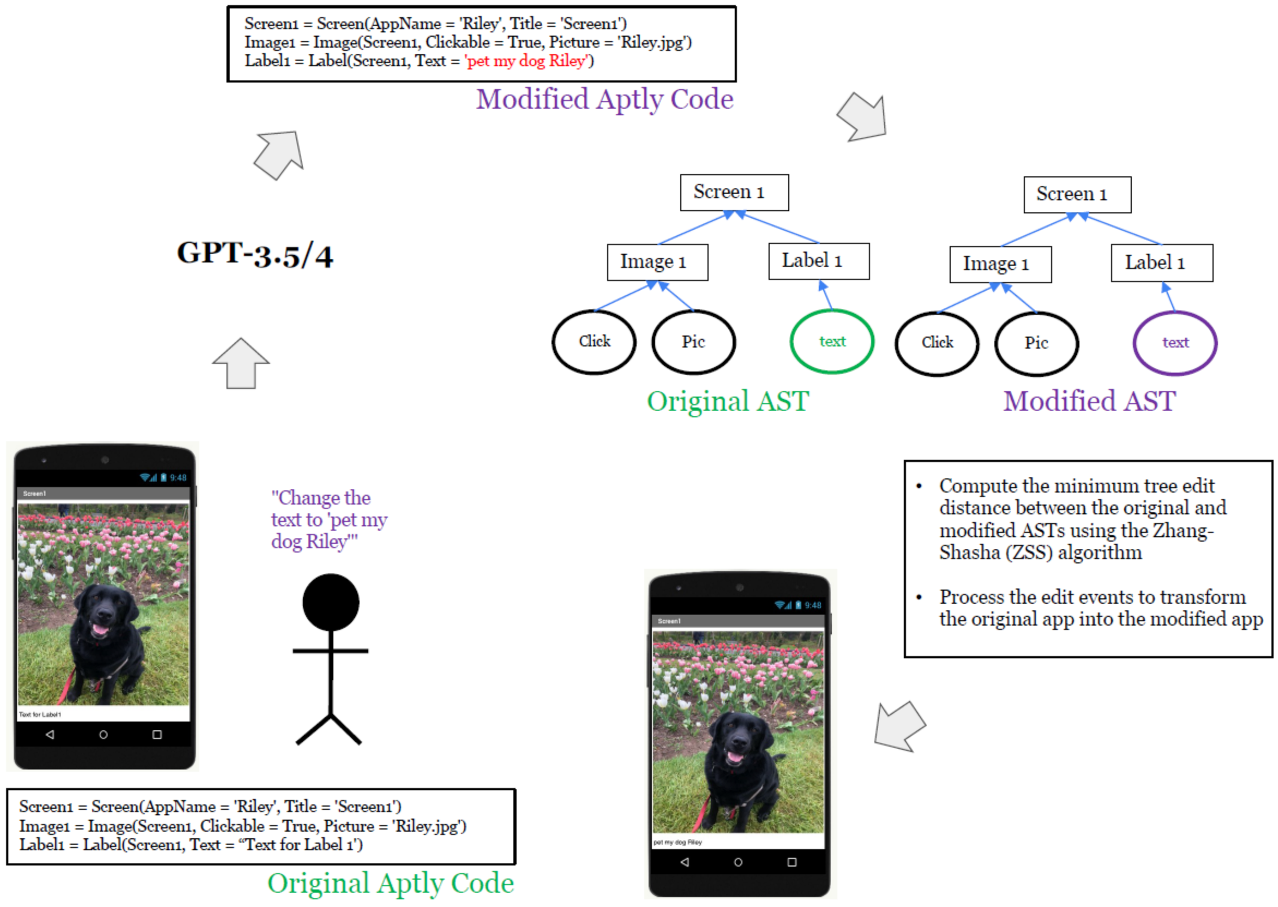
The recent development of AI has been nothing short of revolutionary, drastically changing the way we live and interact with technology [9]. While this has caused a degree of apprehension [6], it has also presented us with an incredible opportunity: the chance to collaborate with AI and leverage its strengths to unlock solutions to complex problems and achieve our goals in ways that were previously unimaginable [16, 17]. Through this collaboration, AI and humans can work together to augment each other's capabilities and create far more efficient and effective outcomes.

AI-human collaboration refers to the use of AI technology in combination with human expertise to achieve a common goal [3, 12, 14]. In the context of programming and software development, AI-human collaboration can take many forms, such as text-to-code models that generate code based on human input or low-code platforms that allow non-technical users to build applications with the help of AI-powered drag-and-drop interfaces [1, 2, 10]. There are several benefits to AI-human collaboration in programming and software development. For example, AI-powered code completion tools which suggest code snippets can help programmers write code more accurately and efficiently by reducing syntax errors and automatically completing code.

In this paper, we introduce the development of a new feature for the MIT App Inventor platform [18]. This feature will allow users to edit and customize their mobile applications by providing a simple textual description of their desired changes. With this feature, users will no longer have to rely solely on programming elements to make changes to their apps. Instead, they can easily modify their apps using plain language, making the app creation process more accessible and user-friendly. This is a major step forward in our goal to democratize mobile app development and empower users to create their own apps with ease [15].

2 RELATED WORK

Text-to-code models are AI-powered models that can generate code from natural language inputs, such as written descriptions or verbal commands [4]. These models use machine learning algorithms to understand the meaning of the input and generate code that implements the desired functionality. A major example is Copilot [4], an AI-powered code completion tool that helps programmers write code more efficiently and accurately. Text-to-code models can be used in various applications, such as code generation for web and mobile apps, software prototyping, and data analysis. They can also be used to automate repetitive tasks in software development, such as generating boilerplate code for common use cases [5]. One of the benefits of text-to-code models is that they can lower the barrier to entry for people who are interested in learning how to code but have limited programming experience. By allowing users to describe what they want to achieve in natural language, text-to-code models can help individuals quickly generate working code without having to learn the intricacies of a particular programming language. Recently, the MIT App Inventor team has been developing a new platform called Aptly [8]. Aptly lets anyone – even young students – create original apps for tablets and smartphones



```
Screen1 = Screen(AppName = 'Riley', Title = 'Screen1')
Image1 = Image(Screen1, Clickable = True, Picture = 'Riley.jpg')
Label1 = Label(Screen1, Text = 'pet my dog Riley')
```

- Compute the minimum tree edit distance between the original and modified ASTs using the Zhang-Shasha (ZSS) algorithm
- Process the edit events to transform the original app into the modified app

```
Screen1 = Screen(AppName = 'Riley', Title = 'Screen1')
Image1 = Image(Screen1, Clickable = True, Picture = 'Riley.jpg')
Label1 = Label(Screen1, Text = "Text for Label 1')
```

Figure 1: Our editing process begins when a user requests a change, such as "Change the text to 'pet my dog Riley'." We feed the original code and the user's request into one of OpenAI's GPT-3.5 or GPT-4 models [11], which outputs the modified code. We then use the ZSS algorithm to compare the original code's AST with the modified code's AST and identify the series of changes needed to make the edit. Once we have processed the edits, we arrive at the target app.

by writing in natural language. For example, a user might type in or speak: "Create an app with a few buttons corresponding to various languages. When I press a button, translate what I say into the language for that button and speak the result." Using the verbal prompt, the platform creates a fully functional app on the user's mobile device.

3 METHODS

The editing feature uses the technology of large language models to automatically generate Aptly code representing MIT App Inventor mobile apps. The performance depends on the input given to OpenAI's GPT-3.5 or GPT-4 [11], referred to as the "prompt." We automatically craft a prompt by providing a set of examples (the original code, the textual description of the desired edit, and the modified code) along with current app code and the user's requested textual edit. Such prompt engineering is referred to as few-shot learning [13].

The user begins with an in-progress app that they have created in MIT App Inventor, either by hand or by providing a natural language description to Aptly. The user could also begin with a blank MIT App Inventor project and create the app from scratch using a series of edits.

When the user requests an edit for their app, the process is as follows: The user provides a natural language description of their desired edit, such as "Add another label that says 'Welcome to my app!' " or "When the button is clicked, change the background color to blue."

The Aptly server then sends the original app's Aptly code along with the edit description and several examples to GPT-3.5. To determine which examples to provide to GPT-3.5, we choose the examples from our example bank that are most similar to the requested edit, where similarity is measured using the cosine similarity of the embedding of each example's edit description and the current user's edit description.

GPT-3.5 then generates updated Aptly code representing the new app, much like how GPT-3.5 is used to generate

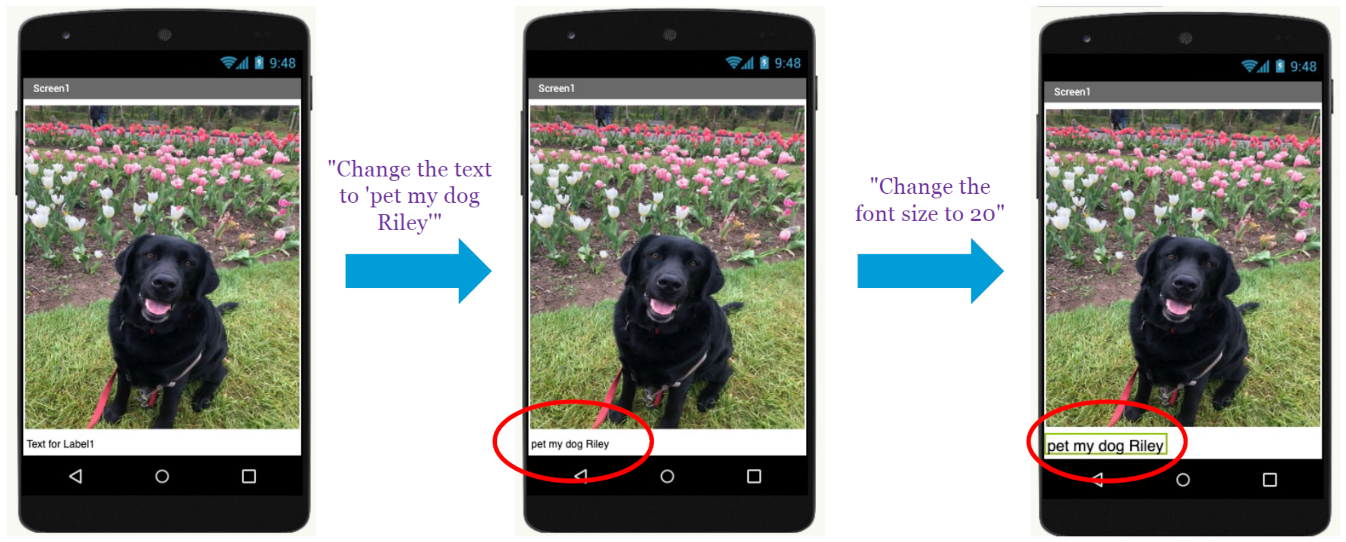

9:48
Screen1
Text for Label1
"Change the text to 'pet my dog Riley'"
pet my dog Riley
"Change the font size to 20"
pet my dog Riley

Figure 2: Examples of editing the user interface of the mobile application. In the first command, the user asks to change the label text to "pet my dog Riley", and in the second command, the user asks to change the font size of the text to be 20 (from 14). Note that programmatic language for the edit description is not required; for the second command, a similar result could have been achieved by saying "make the text bigger."

Aptly code when users create projects from natural language descriptions.

Given the updated Aptly code, the Aptly server then computes a sequence of edits needed to efficiently transform the original app into the updated one. Specifically, we compute the minimum tree edit distance between the abstract syntax tree representing the current app and the abstract syntax tree representing the modified app using the Zhang-Shasha (ZSS) algorithm [19]. "Edits" can mean inserting, updating, or deleting components or blocks (or simply keeping them the same) between the original and modified program.

Having computed the most efficient way to transform the original program into the modified program through a sequence of insertions, updates, and deletions, we begin processing those edit events over several stages. We begin by checking whether all of our proposed insertions of components or blocks should actually be insertions. ZSS may have suggested deleting a component, creating a new version of that same component, and then inserting the new version elsewhere. In these cases, we could simply move the original component (i.e. update its parent) for increased efficiency. Next, we process deletions, checking each proposed deletion against our proposed insertions to ensure we move components rather than deleting and re-inserting them where possible. We then process updates such as moving components around or changing their properties, and finally, we process insertions of new components and blocks. Having finalized the sequence of edit events, the Aptly server sends those events to MIT App Inventor through the Real-Time Collaboration (RTC) [7] server, and MIT App Inventor processes those RTC operations to modify the app.

4 RESULTS

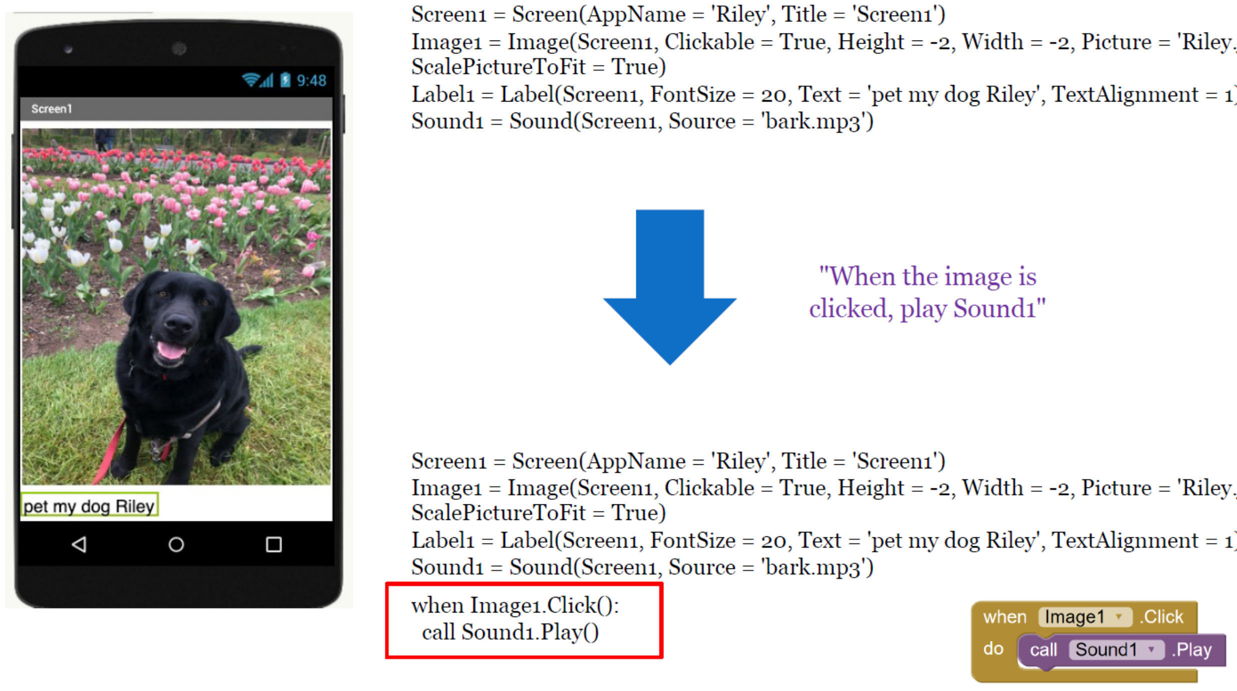
A mobile application can be broken down into the user interface of the application and the functionality of each component within the application. Our editing feature allows for the editing of both the user interface and functionality of mobile applications.

Figure 2 shows some simple examples of editing the user interface of the application. As an example, the user can ask to change a label's text to "pet my dog Riley" with a single command. They can also adjust the font size of the text with another simple command. These quick and straightforward updates demonstrate the ease with which users can manipulate their app's visual appearance to match their preferences.

Figure 3 shows some simple examples of editing the functionality of the components within an application. For example, a user might change the behavior of an image of a dog. Initially, clicking the image produces no effect, but with a simple command, the user can make a barking sound trigger whenever the image is clicked. The corresponding code blocks are automatically generated to implement the desired behavior change.

5 CONCLUSION & DISCUSSION

In this paper, we present our work on developing a cutting-edge platform that combines the simplicity of blocks-based coding with the power of AI. Our aim is to provide students with a user-friendly environment where they can easily edit and modify their block code



```
Screen1 = Screen(AppName = 'Riley', Title = 'Screen1')
Image1 = Image(Screen1, Clickable = True, Height = -2, Width = -2, Picture = 'Riley.jpg',
ScalePictureToFit = True)
Label1 = Label(Screen1, FontSize = 20, Text = 'pet my dog Riley', TextAlignment = 1)
Sound1 = Sound(Screen1, Source = 'bark.mp3')
```

"When the image is clicked, play Sound1"

```
Screen1 = Screen(AppName = 'Riley', Title = 'Screen1')
Image1 = Image(Screen1, Clickable = True, Height = -2, Width = -2, Picture = 'Riley.jpg',
ScalePictureToFit = True)
Label1 = Label(Screen1, FontSize = 20, Text = 'pet my dog Riley', TextAlignment = 1)
Sound1 = Sound(Screen1, Source = 'bark.mp3')
when Image1.Click():
  call Sound1.Play()
```

Figure 3: An example where the user changes the functionality of the app, specifically what happens when the image is clicked. Before the edit, nothing happens when the user clicks the image of a dog. The user adds a barking sound and then commands, "When the image is clicked, play Sound1." The change is represented in the Aptly code and implemented via the creation of new code blocks, as shown on the right.

with the aid of an AI-powered feature. This feature will streamline the coding process and provide students with suggestions and recommendations as they code, making the development process faster and more efficient. By leveraging the power of AI, we believe that we can empower students to take their coding skills to the next level and create more sophisticated and innovative applications. This work represents an important step forward in the field of educational technology and we are excited to see the impact it will have on students and the future of app development. Going forward, we aim to expand the editing capability to enable users to edit more diverse aspects of their mobile applications. In addition, we plan to conduct extensive case studies to demonstrate the impact of AI-student collaboration in the creation of mobile apps and how this collaboration can empower students to achieve great results. Our goal is to unlock the full potential of this collaboration, providing a platform for students to build their skills and create innovative, cutting-edge apps.

ACKNOWLEDGMENTS

We thank the MIT App Inventor team for their help and support throughout this project.

REFERENCES

[1] Simone Bales. 2018. Build Android apps without Coding: Get started with Android apps using Thunkable-MIT app Inventor.
[2] Alexander C Bock and Ulrich Frank. 2021. Low-code platform. *Business & Information Systems Engineering* 63 (2021), 733–740.
[3] Charlynne Bolton, Veronika Machová, Maria Kovacova, and Katarina Valaskova. 2018. The power of human–machine collaboration: Artificial intelligence, business automation, and the smart economy. *Economics, Management, and Financial Markets* 13, 4 (2018), 51–56.
[4] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374* (2021).
[5] Arghavan Moradi Dakhel, Vahid Majdinasab, Amin Nikanjam, Foutse Khomh, Michel C Desmarais, Zhen Ming, et al. 2022. GitHub Copilot AI pair programmer: Asset or Liability? *arXiv preprint arXiv:2206.15331* (2022).
[6] Thomas G Dietterich and Eric J Horvitz. 2015. Rise of concerns about AI: reflections and directions. *Commun. ACM* 58, 10 (2015), 38–40.
[7] Ting-Chia Hsu, Hal Abelson, Evan Patton, Shih-Chu Chen, and Hsuan-Ning Chang. 2021. Self-efficacy and behavior patterns of learners using a real-time collaboration system developed for group programming. *International Journal of Computer-Supported Collaborative Learning* (2021), 1–24.
[8] David YJ Kim, Ashley Granquist, Evan Patton, Mark Friedman, and Hal Abelson. 2022. SPEAK YOUR MIND: INTRODUCING APTLY, THE SOFTWARE PLATFORM THAT TURNS IDEAS INTO WORKING APPS. In *ICERI2022 Proceedings*. IATED, 1653–1660.
[9] Spyros Makridakis. 2017. The forthcoming Artificial Intelligence (AI) revolution: Its impact on society and firms. *Futures* 90 (2017), 46–60.
[10] John Maloney, Mitchel Resnick, Natalie Rusk, Brian Silverman, and Evelyn Eastmond. 2010. The scratch programming language and environment. *ACM Transactions on Computing Education (TOCE)* 10, 4 (2010), 1–15.
[11] OpenAI. 2023. GPT-4 Technical Report. arXiv:2303.08774 [cs.CL]

[12] Sun Young Park, Pei-Yi Kuo, Andrea Barbarin, Elizabeth Kaziunas, Astrid Chow, Karandeep Singh, Lauren Wilcox, and Walter S Lasecki. 2019. Identifying challenges and opportunities in human-AI collaboration in healthcare. In *Conference Companion Publication of the 2019 on Computer Supported Cooperative Work and Social Computing*. 506–510.

[13] Jasmine L Shone, Robin Liu, Evan Patton, and David Young-Jae Kim. EasyChair, 2022. Design and Optimization of an Automatic Mobile Application Generating Learning Platform. EasyChair Preprint no. 9136.

[14] Konrad Sowa, Aleksandra Przegalinska, and Leon Ciechanowski. 2021. Cobots in knowledge work: Human–AI collaboration in managerial professions. *Journal of Business Research* 125 (2021), 135–142.

[15] Mike Tissenbaum, Josh Sheldon, and Hal Abelson. 2019. From computational thinking to computational action. *Commun. ACM* 62, 3 (2019), 34–36.

[16] Dakuo Wang, Elizabeth Churchill, Pattie Maes, Xiangmin Fan, Ben Shneiderman, Yuanchun Shi, and Qianying Wang. 2020. From human-human collaboration to Human-AI collaboration: Designing AI systems that can work together with people. In *Extended abstracts of the 2020 CHI conference on human factors in computing systems*. 1–6.

[17] Dakuo Wang, Justin D Weisz, Michael Muller, Parikshit Ram, Werner Geyer, Casey Dugan, Yla Tausczik, Horst Samulowitz, and Alexander Gray. 2019. Human-AI collaboration in data science: Exploring data scientists' perceptions of automated AI. *Proceedings of the ACM on human-computer interaction* 3, CSCW (2019), 1–24.

[18] David Wolber, Hal Abelson, Ellen Spertus, and Liz Looney. 2011. *App inventor*. " O'Reilly Media, Inc.".

[19] Kaizhong Zhang and Dennis Shasha. 1989. Simple fast algorithms for the editing distance between trees and related problems. *SIAM journal on computing* 18, 6 (1989), 1245–1262.