My research interests lie in designing and developing programming tools for better data science while applying techniques from Human-Computer Interaction (HCI), Software Engineering (SE), and Machine Learning (ML). I am particularly interested in making programming tools intuitive and accessible, especially to underrepresented students and non-technical people. I'm currently working on collaborative programming tools with Prof. Amy Zhang at UW. Previously, I built machine learning systems for multiverse analysis in data science and wrote two papers once as the first author with Prof. Tim Althoff at UW.

With an exponential increase in available data and a growing culture of "data-driven discovery", analysis tools have enjoyed widespread innovation and adoption. However, robust processes to guide technical and non-technical people using these systems remain in relatively short supply. My research experience is related to two problems in data science programming: 1) scientific reproducibility and 2) data science collaboration. I began studying programming tools for reproducible data science in the summer of 2019 when, after my third year at Peking University, I interned at The University of Washington, advised by Prof. Tim Althoff.

As data scientists typically make many decisions in their analyses, such as determining features and selecting a model type, where to make alternative analysis decisions and what the alternatives could look like are considered part of the analysis process. These decisions form many different solution paths to consider in parallel during analysis. Different solution paths might result in different empirical data interpretations, even leading to conflicting conclusions. However, these parallel paths are not always well reflected in data science practice and require data science expertise. Automated tools that leverage existing code repositories could help technical and non-technical users brainstorm potential solution paths for robust and reproducible data science.

My projects with Prof. Tim Althoff aim to build an ML system that can automatically detect decision points and suggest alternatives for data science programs. Since there is no commonly-agreed upon definition of what decision points and alternatives should be, we started by building an ML system (CORAL) to label computational notebook cells as stages in the data analysis process from data import to wrangling, exploration, modeling, and evaluation. I built CORAL, a multitask, weakly supervised, graph-based transformer architecture for code classification, which jointly models natural language (such as markdown headers) and general-purpose source code. We also provided an extension to a corpus of online Jupyter Notebooks: a new dataset of expert annotations of stages in the data analysis process for 1,840 code cells in 100 notebooks, which we use exclusively for evaluation. I also built an annotation tool where users can upload their Jupyter Notebook and annotate stages for code blocks or view CORAL's predictions. With the CORAL model, we presented the largest ever study of data science source code in-depth. In contrast, previous studies have been limited to simple summaries such as the number or nature of imported libraries, total line counts, or the fraction of lines used for comments. This work is in submission to EPJ Data Science. I learned more about ML for code from this project and gained a better understanding of multiverse analysis in data science. It also intrigued me to dig deeper into tools that support automatic multiverse analysis.

With CORAL labeling computational notebook cells as a data science stage, we then built an ML system that detects decision points and suggests alternatives in computational notebooks, narrowing the definition of decision points and alternatives to API replacement in code. This second project targeted mining collective data science knowledge from code on Kaggle competitions to suggest alternative data analysis approaches. We formulated the tasks of identifying decision points and suggesting alternatives as a classification task and a sequence-to-sequence prediction task. We proposed a model, MORAY, which leverages information on relationships within libraries through neural graph representation learning in a multitask learning framework. Since this was a very new task, and there was no standard definition of decision points and alternatives, I built an annotation tool to collect people's thoughts on decision points and alternatives. This annotation tool allows users to upload notebooks, brush and select code that is concerned as decision points, and specify alternatives for each decision point. I also embed the trained MORAY model to the annotation tool for users to interact and evaluate with the model. This project is joint work with a Ph.D. student at UW, and is insubmission to WWW 2021. I learned about applying ML to multiverse statistical analysis in data science and gleaning insights from large-scale open-source code from this project. I also presented this work to several groups at UW and gained communication skills.

While these two projects led me to scientific reproducibility problems in data science, I found some limitations when applying ML to analyze data science code. First of all, there is a lack of high-quality datasets impeding large-scale, in-depth analysis of data science code. Although open-source code is plentiful online, most of the code is raw text without standard labels. Besides, hand-labeling a dataset is time-consuming and also requires expert domain knowledge. Second, the lack of hand-labeled datasets limits task types of analyzing data science code. Most ML models for code representation learning focus on code completion (i.e., method name

suggestion) or code classification (i.e., algorithm classification). These limitations in ML for code point to problems with existing programming tools. Take the multiverse analysis project as an example. Because the linear format of computational notebooks does not support clean and organized multiverse authoring, programmers' decision-making process is not well reflected in computational notebooks.

Therefore, I decided to dig deeper into research on programming tools to support better data science cognition. I started interning at the Social Futures Lab at The University of Washington, advised by Prof. Amy Zhang. We worked on building Jupyter Notebook extensions to support two common cases of collaboration in data science. One is pair authoring, where person A does most of the programming work, and person B gives A ideas about what decisions to make. The other one is divide and conquer, where collaborators each take part in the programming work (i.e., A working on data preprocessing and B working on modeling).

To better support collaborative cognition in these two cases, we designed and developed an extension to Jupyter Notebook that helps users split work and create branches upon original code. First, we propose to modularize notebooks by adding wrappers outside cells so that users can split work based on modules. For each wrapper, users can specify input and output variable types in advance to avoid conflicts in collaborative editing, such as modifying shared variables without making copies. Second, we allow users to branch over the original code by subdividing cells horizontally in each wrapper, where they can explore alternatives easily in a clear and organized view without writing much explanation. Besides, to better support computational notebooks' presentation nature, users can also manually collapse wrappers or hide branches to tailor the view for their needs. This extension provides easy navigation and clear presentation views for both technical and non-technical people. The modularization and branching features help programmers split work and control versions strategically in collaboration. With this project, I learned to build extensions for computational notebooks and improve user experience in programming. I also shifted my research focus to the system-building aspects of HCI.

My work at UW has been rewarding and has also led me to a broader scope of problems with current programming tools. As data science becomes more prevalent in research and industry, how do people tackle their increasing experiments? How can programming tools better visualize variance between all these experiment results? As collaboration is widespread in the industry, how do people deal with editing synchronization and debugging conflicts between each other's code? While most current programming tools are designed for programmers, how can we make these tools friendly to non-technical people? Moreover, data and algorithms are not perfect and often embed bias, such as gender inequality. Designing interactive programming tools to help users realize and prevent damages from these biases is also essential. These questions ignited my interest in building programming tools that are accessible and intuitive. By accessible, I mean usable by and inclusive of a wide variety of people, especially people underrepresented in computer science and non-technical people such as doctors, business owners, and project managers. By intuitive, I mean intimately connected with human cognition, relying on interactive graphical user interfaces and new formats of layout to provide easier navigation.

To me, a Ph.D. would mean the opportunity to branch out in new areas and develop the cognitive programming tools I need to answer these questions. I look forward to learning how to draw on my prior research experiences and related work to explore these problems further and produce novel solutions to them. Further, building programming tools for data science is not merely about programming language and software engineering but also requires a deep understanding of user interface, cognition process, CS education, and exploratory data analysis. Therefore, I want to pursue interdisciplinary research during my PhD. A research-based education would be an excellent opportunity for me to work with faculty from many different areas and tackle long-shot problems.