Annotated version of Philip Guo's 2018 NSF CAREER proposal

For more details and links, see: `http://pgbovine.net/NSF-CAREER-proposal.htm`

In April 2019 I decided to annotate my NSF CAREER proposal (submitted in July 2018 and awarded in Dec 2018) with a bunch of margin notes. At the time of writing, I had served on three NSF review panels, but none was for CAREER. Thus, these margin notes are my own speculations about what I think worked well, but obviously I wasn't in the panel room when my proposal was discussed, so whateves.

Serving on panels is the best way to improve your proposal writing skills.

I'm making this proposal public because I think it's the most equitable when everyone has access to good examples. I hope you find it useful, but if I don't know you please *don't email me to ask for clarifications, materials, or feedback on your proposal.* This PDF and accompanying webpage has all the info you'll need.

First some high-level thoughts on the CAREER application process, which are probably most relevant to faculty in computer science, computer engineering, information science, and human-computer interaction:

- My main unsolicited advice (which other people might disagree with but whateves, this is my doc!) *. . . wait as long as you can before applying for the first time.* If you have five full years before going up for tenure, that means applying in the summer before you start Year 3. Then if you don't get it, re-apply at the start of Year 4, then Year 5. This gives you three attempts (the maximum permitted) before going up for tenure at the start of Year 6. I took this idea a bit too far: I applied for the first time at the start of Year 5. I don't recommend cutting it this close, but luckily my first attempt worked!

  In a few pages you'll find out why I suggest this. Yes I know others have gotten it earlier, but their CVs were probably stronger than yours!

- If you're eligible for NSF CRII, then apply for that super-early before you start Year 1 or maybe Year 2; I applied before Year 1 and got it on my first try. Other than the CRII or maybe EAGER, I wouldn't recommend applying for any "regular" NSF grants as a single PI before you try for CAREER. Why not? One, because your chances of success on those proposals are much lower since you'll be competing in an open field against much more experienced senior faculty and multi-PI teams; for CRII/CAREER your proposal is in a pool with only single-PI proposals from assistant professors. Two, because if you get regular NSF grants, then your CAREER reviewers will (rightly!) question how your CAREER proposal is different from your other grant(s). It's OK to tag along on grants led by senior faculty, since those topics usually won't interfere with your CAREER topic.

  I didn't annotate or upload my CRII proposal since it follows the same structure as this one, except that it's shorter.

- Submit to the program that best fits your research area and try to get the *most relevant* program officer assigned to your proposal. (You don't control these assignments, but just email them to ask.) This maximizes your chances of getting reviewers who are familiar with your work, which leads to . . .

  I submitted to CISE-IIS-CHS but originally intended to submit to a more education-focused area, which would have ended up being a worse fit for my research style. Talk to program officers to assess possible fit!

- . . . the most important tidbit: *reviewers should not be at all surprised when they read what your CAREER proposal is about*. When my reviewers saw mine, (I hope!) they probably thought, "*oh yeah, Philip Guo has a proposal about scaling up learning programming on his widely-used Python Tutor platform, sure that's what I expected! Let's see what new programmer assistance tools he's cooking up ...*" There were several other newer (and to me, more exciting!) lines of work I originally wanted to write about, but I resisted the urge to do so because then reviewers would've been like "*oh yeah, a Philip Guo proposal on ... applying virtual reality to blockchains using stochastic gradient descent?!? WHAT THE WHAT?!?*" I obviously have zero background or past publications in this new area I just made up, so any proposal I write about it would get rejected. Bottom line: write about what you're already known for. Of course, you should propose something cool and innovative along that direction, but don't write about something that reviewers would be surprised to see from you.

  One way to think about it is that reviewers are deciding to fund YOU as an early-career PI. So write something that's quintessentially YOU. This differs from "regular" NSF proposals where reviewers are deciding to fund a specific project and not necessarily a PI.

- What about budgets, data management plans, department chair letter, and other supplemental docs? Just make sure they don't stand out in any weird way (e.g., an oddly huge budget), and you'll be OK.

  Ask colleagues in your immediate field for their supplemental docs and just emulate theirs.

- Finally, remember that this is a proposal for what you want to do in the future, not a legally-binding contract for what you must do if it's funded. Nobody actually expects you to implement every single part of your proposal in the exact way that you planned out; otherwise it wouldn't be called research!

Project Summary

Overview:

Millions of people from diverse backgrounds now want to learn computer programming to prepare for careers across many fields. Decades of research have made great advances in improving how programming is taught in traditional classrooms. However, although some well-resourced schools in relatively affluent areas offer formal programming courses, the vast majority of people around the world do not have access to classroom environments. Thus, there is a critical need to bring the best aspects of these in-person environments to freely-accessible online settings.

Write your Project Summary in a text editor and paste it into the submission website. Don't use any fancy formatting.

The objective of this research is to develop interactive systems that enable large groups of people to help one another learn to code in online environments where experts are often not available. The PI is uniquely positioned to implement this research since he created Python Tutor, an online programming platform that has had over 3.5 million users from over 180 countries. Having full control over this platform gives the PI access to a diverse global population of users to evaluate the proposed systems using large-scale randomized controlled trials.

The specific aims of this research are to develop two novel systems atop the Python Tutor platform: 1) Omnitutor: a system that organizes learners to tutor one another even while they are individually working on their own code, 2) Rosetta: a system that enables learners to annotate their code and errors with hints that may benefit future learners who face similar issues.

Say in simple English exactly what you're planning to do. Being clear upfront is critical for getting your proposal into the hands of reviewers who understand your type of research. Also, when reviewers are discussing the stack of proposals at the panel meeting, they need to be able to recall what yours was about by skimming this summary while distracted by other people talking in the room.

In addition, the PI's Education Plan will tightly integrate this research with new programming education initiatives that the PI will be launching both within UC San Diego and with industry partners at Google, edX (the largest nonprofit provider of MOOCs), and Project Jupyter (a popular notebook-based scientific computing environment).

In sum, this research lays the foundation for the PI's long-term career goal to create new systems for learning programming at scale so that as many people as possible can gain access to this vital skill.

Intellectual Merit:

This research makes systems and empirical contributions to HCI. Specifically, it advances the state of knowledge regarding: 1) how to coordinate a group of learners to provide both synchronous and asynchronous help to total strangers while they are in the midst of working on their own code, 2) how to design interactive systems that achieve this goal by developing novel algorithms such as code similarity analysis, machine-assisted code simplification, and generalization of code annotations, 3) how these designs enable novices to provide high-quality assistance to strangers, as evidenced by randomized controlled trials with millions of participants, 4) how novices chat about their mental models and common novice misconceptions about programming languages (released as anonymized datasets).

What new knowledge will be gained if this project is successful? Building a system or tool by itself isn't new knowledge; what you and the world can *learn* from that system/tool being built is the knowledge. One common criticism of systems-focused proposals like mine is that it's "just engineering work." Counteract that by being clear about what generalizable knowledge will be gained via engineering.

Broader Impacts:

Computer programming is a critical skill for many kinds of modern professions. This research builds upon the PI's Python Tutor platform, which is already used by over 3.5 million people from over 180 countries to learn programming. The free peer tutoring services provided by this research will foster development of a diverse globally-competitive STEM workforce. The PI's Education Plan to use this research to create new programming curricula across the social sciences at UC San Diego will open up career opportunities for female students ($>$ 50% of student population), underrepresented minorities (21% of population), and first-generation college students (38%). The PI will also partner with XXX XXX (Director of Research at Google), XXX XXX (CEO of edX), and XXX XXX (creator of Jupyter Notebooks) to integrate this research into their open online learning materials. Finally, the PI's outreach work via creating popular blogs, podcasts, and YouTube videos with millions of combined views will increase public scientific literacy and public engagement with science and technology.

Collaborator names are anonymized in this public annotated document, but you can probably find out who they are!

Source: `http://pgbovine.net/NSF-CAREER-proposal.htm`

1 Overview and Objectives

Millions of people from diverse backgrounds now want to learn computer programming to prepare for careers in fields such as software engineering, data science, computational research, public policy, health informatics, and data-driven journalism. Decades of research have made great advances in improving how programming is taught in traditional K-12 and university classrooms [52]. However, although some well-resourced schools in relatively affluent areas offer formal programming courses [29, 52], the vast majority of people around the world—children in low-income areas [53], working adults with full-time jobs [27], the fast-growing population of older adults [48], and billions in developing countries [49]—do not have access to traditional classroom learning environments. Thus, there is a critical need to bring the best aspects of these in-person learning environments to freely-accessible online settings. Failure to do so means that critical computer programming skills remain disproportionately accessible to only those from already-privileged backgrounds [12, 51] and cannot plausibly reach the majority of the world's population.

OK, this is the main event! You have only two pages to get the reviewer on your side. If they can't understand and get excited about your proposal from these first two pages, then you've lost.

First make sure you abide by NSF rules for fonts, font sizes, margin lengths, and page limits. I know it's boring, but you don't want to get disqualified upfront due to formatting issues. (I've offset the horizontal margins in this annotated version to make room for margin notes, but otherwise it looks exactly like my actual proposal.)

My long-term goal is to create, deploy, and validate new technologies for learning programming at scale so that as many people as possible can gain access to this vital skill. ***The overall objective of this CAREER proposal, which is a critical step toward this long-term goal, is to develop interactive systems that enable large groups of learners to help one another in online environments where experts are often not available.*** If successful, this project's outcomes point toward a future where anyone around the world with an internet connection can learn to code nearly as effectively as if they had access to in-person tutoring.

This project is motivated by the fact that millions of people are learning to code online by themselves, so they do not have someone to personally guide them like those in classrooms do. Can these learners somehow help one another even while they are working alone? To find out, I will develop two complementary systems:

Specific Aim 1. Omnitutor: many-to-many tutoring. I will develop a system that organizes learners to tutor one another even while they are individually working on their own code. The main technical challenge is how to scaffold a group of novices to provide fast and effective *synchronous help* to total strangers.

Specific Aim 2. Rosetta: learner-generated code explanations. I will develop a system that enables learners to annotate their code and errors with hints that may benefit future learners who face similar issues. The main technical challenge is how to turn raw annotations into a useful corpus of *asynchronous help*.

I went back and forth a lot about whether to propose two or three Specific Aims. I ultimately went with two so that I could discuss them in greater detail but at the risk of the project seeming too "small" for a five-year timeline. I've seen others propose three aims, but more than that risks sounding too shallow.

For these kinds of systems to be rigorously validated and achieve broader impacts beyond academia, ideally they must be deployed to large numbers of users worldwide. I am uniquely positioned to accomplish this goal since I created Python Tutor [45], an online programming platform that has ***over 3.5 million users from over 180 countries***. Having full control of this platform gives me access to a diverse global population of users for my systems. Thus, I propose to build the Omnitutor and Rosetta systems on top of Python Tutor.

2 Expected Significance

One-on-one real-time tutoring from an expert is the gold standard for effective learning across diverse domains [15, 21, 91]. However, billions of people around the world do not have convenient access to such experts, so they currently miss out on these opportunities. ***This project's contributions are significant because they show how interactive systems can scaffold a group of novices to provide timely and relevant programming assistance at a much greater scale than what experts alone can achieve.*** Specifically, this project will result in the following benefits that are strongly aligned with all NSF CAREER review criteria:

For the structure and style of this proposal, I closely followed the template from *The NSF Grant Application Writer's Workbook* by GWSW. I highly recommend this book. Its style suggestions might seem corny at parts (like telling the reader here in bold why you think your contributions are significant) but I found them to be tremendously helpful.

2.1 Intellectual Merit

This project advances the state of knowledge in HCI regarding: 1) how to coordinate a group of learners to provide both synchronous and asynchronous help to total strangers while they are in the midst of working on their own code, 2) how to design interactive systems that achieve this goal by developing novel algorithms such as code similarity analysis, machine-assisted code simplification, and generalization of code annotations, 3) how these designs enable novices to provide high-quality assistance to strangers, as evidenced by

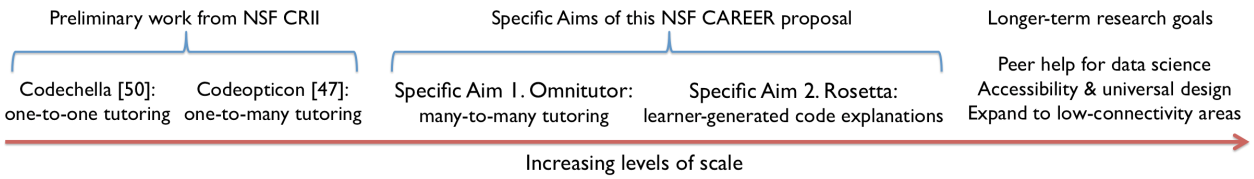

Figure 1: This proposal builds upon preliminary work and forms the foundation for my longer-term goals.

randomized controlled trials with millions of participants, 4) how novices chat about their mental models and common novice misconceptions about programming languages (released as anonymized datasets).

2.2 Broader Impacts summary (see Section 8 for details)

Section 8 describes my full broader impacts plan. In sum, my research and outreach work will contribute toward: developing a diverse globally-competitive STEM workforce, full participation of women and underrepresented minorities in STEM, improved STEM education and educator development, enhanced research and education infrastructure, increased partnerships with industry, and increased public scientific literacy.

Preview your Broader Impacts and Education Plan here to prime your reader, but devote full sections to them at the end of your proposal. These are critical for making yours stand out!

2.3 Integration of Research and Education (see Section 7 for details)

Section 7 describes my full education plan. In sum, I will use my systems to develop new curricula across the social sciences and data science at UC San Diego, which will reach a diverse campus-wide population. I will also integrate them into popular MOOCs, textbooks, and the Jupyter Notebook platform.

3 PI Qualifications and Relationship to PI's Longer-Term Career Goals

I am uniquely qualified to carry out this project due to my ability to deploy new systems atop the large user base of my Python Tutor platform [46]. This project is well-matched for my expertise in both building systems for and conducting studies on novice programmers. In my first four years as an assistant professor, my lab has built interactive systems for visualization-assisted code tutoring [47, 50, 60], remote pair programming [95], detecting novice programmer frustrations [28, 90], scaffolding the development of data science tutorials [100] and web-based UI mashups [101], improving coding tutorial videos [61], and generating step-by-step software tutorials [40, 71, 72]. We have also studied challenges faced by underrepresented groups such as women [35, 96], older adults [48], non-native English speakers [49], and those from non-technical college majors [23, 24, 93] when learning to code, along with the limitations of discussion forums [102] and digital textbooks [94] for coding. Our papers have collectively won five HCI conference Honorable Mention awards at CHI and UIST and two software engineering conference Best Paper awards.

Like I mentioned in margin notes on Page 1, the CAREER award is to fund YOU as an early-career PI, so you need to show reviewers both why your trajectory is already off to a great start and why this funding is essential for accelerating your progress even more. This is why I suggested to wait as long as possible before submitting, because then you'll have the strongest possible foundation.

I have all of the resources needed to successfully complete this project. My university has provided me with lab space and startup funding for necessary computational research infrastructure. I have also attached ***six letters of collaboration*** (Section 7.1) from colleagues who will assist with my Education Plan.

This project forms a vital foundation for my longer-term career goals of creating technologies that enable people who do not have privileged access to in-person experts to learn complex technical topics such as computer programming. Figure 1 shows how it continues the momentum I have built up over the past three years with my NSF CRII award by proposing interactive help systems that scale better than my prior one-to-one [50] and one-to-many [47] tutoring tools. If successful, this project will open the doors for me to expand my longer-term research agenda in three new directions: 1) designing analogous peer-based help systems for data science, motivated by estimates that an order of magnitude more people now need data science skills rather than standard programming skills [62, 78, 79], 2) working toward universal design [65] by making interactive learning systems more accessible to those with physical or cognitive impairments, and 3) developing ways to bring high-quality programming and data science training to the remaining billions of people around the world who do not yet have reliable computational infrastructure or internet connectivity.

Show a longer-term, big-picture vision for your career and how this proposal fits into it. This is why I suggested on Page 1 to write about a topic that you're already well-known for, not to pitch a brand-new direction.

End of page two. If the reviewer isn't cheering for you by now, then your proposal is toast!

4 Background: Review of Relevant Literature

One page feels about right for your background section, which should show that a) you're a legit researcher who knows the literature, b) there is a real need for what you're proposing, and c) what you're proposing is novel research-wise. But don't make this section too long since you want to leave more space to talk about your own work, not the work of others.

Motivation: the need for human guidance in online learning. A common criticism of online learning resources (e.g., MOOCs, YouTube videos, blogs) is that they mostly serve self-directed autodidacts: people who already know how to learn well by themselves [74]. For instance, those who successfully complete MOOCs are mostly professionals in technical fields who hold graduate-level degrees [12, 25, 73]. Critics argue that a key missing ingredient here is *human guidance to sustain motivation and engagement for less independent learners* [63]. Discussion forums and Q&A sites like Stack Overflow serve that role in principle, but in practice are often dominated by a vocal minority of elite participants [57], can feel unwelcoming to novices [34, 35], and lack real-time interactivity [25, 66], which is critical for helping someone debug their code. As an alternative to human assistance, Intelligent Tutoring Systems present learners with progressively-staged hints [11, 41, 56, 92]; however, these systems are time-consuming to build, must be specially crafted for specific problem types, and lack the flexibility and empathy of human tutors [33]. For programming in particular, code history mining (e.g., HelpMeOut [54]) and program synthesis techniques (e.g., AutoGrader [82], Refazer [76]) can generate bug-fix hints from a corpus of code and fixes; but they too lack the flexibility of actual human tutors. *My proposed work aims to fulfill this important need for human guidance in online learning by offering free peer-based help for computer programming.*

Not all systems-focused or engineering projects need to be informed by theory, but the ones that are will stand out more.

Theory: the efficacy of using learners as peer tutors. The theoretical underpinnings of this proposal come from findings in the learning sciences on the efficacy of peer tutoring. One-on-one tutoring has been shown to be more effective than group-based instruction [15], self-directed learning [63], and Intelligent Tutoring Systems [91]. Also, using learners as *peer tutors* is more cost-effective than hiring dedicated experts as tutors [43] yet still leads to comparable learning gains in settings as diverse as K-12 schools [31, 59, 75], universities [26, 32], and medical schools [58, 88]. Peer tutoring is especially effective for at-risk learners [42] with learning disabilities [18, 31], behavioral disorders [59], or low socioeconomic status [75]. Finally, peers can serve as effective tutors even when their level of expertise is not much greater than learners, since they can prompt learners to question assumptions and work together to discover solutions [21]. However, the most significant limitation here is *lack of scale*: peer tutoring requires extensive in-person coordination to find and organize peers as helpers, especially because they too may be busy learning [26].

My proposed work aims to overcome the challenges to scaling up peer tutoring by using machine assistance to scaffold a group of online learners to help one another both synchronously and asynchronously. Prior work points to the feasibility of bringing peer tutoring online in this way: In a lab study, tutoring sessions held using text-only chat led to similar learning gains as those held using face-to-face interactions [81]. Also, an anonymous online format could eliminate some of the shortcomings of face-to-face peer tutoring, such as implicit biases due to gender or race [67, 68], or the need to manage in-person impressions [26].

Systems: programming help interfaces. My proposed work builds upon the rich lineage of interactive systems that enable people to help one another on programming-related questions. For instance, synchronous help interfaces such as Collabode [39], CodePilot [95], and Codechella [50] provide scaffolding for pair programming with text-based chat. Asynchronous help interfaces such as Codeon [20] and CrowdCode [64] are IDE extensions that allow programmers to crowdsource their programming questions to experts online. Feedback amplification systems such as Overcode [38], Foobaz [36], AutoStyle [70], and MistakeBrowser/FixPropagator [55] aim to amplify expert effort by displaying their written feedback on many pieces of similarly-behaving student code. My Rosetta system is inspired by learnersourcing systems [37, 40, 97], but it does not require all learners to be annotating the *same* fixed set of code examples.

My proposed work is novel because it is, to my knowledge, the first to organize groups of anonymous learners to help one another even while they are working on their own individual code in different contexts. Prior systems had either assumed that experts were available to give feedback on a specific coding problem that everyone was working on, or lacked algorithms for matching learners with potentially relevant helpers.

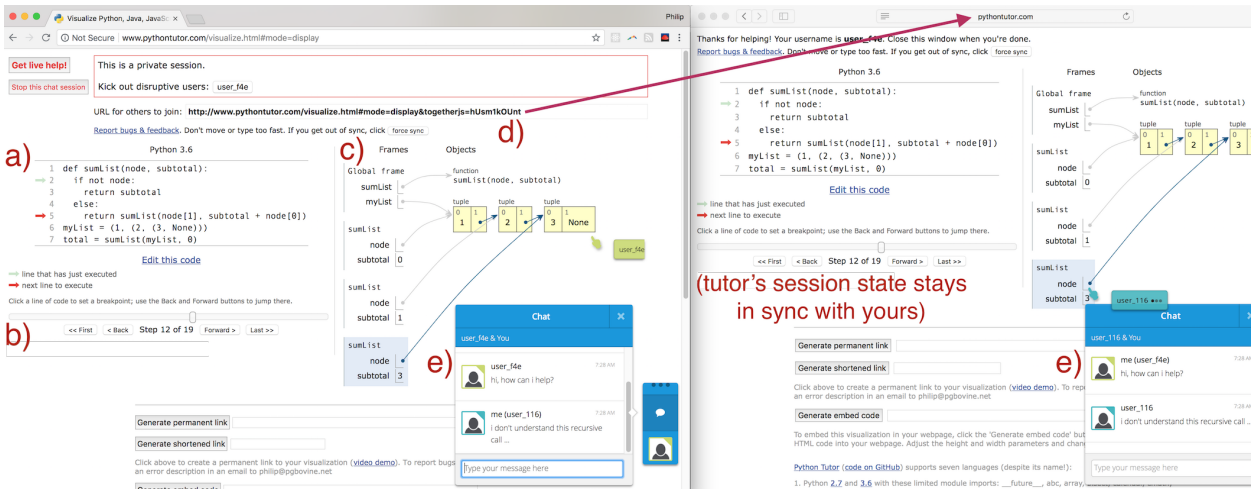

Figure 2: My Python Tutor website [45] allows users to: a) write code in Python, Java, JavaScript, Ruby, C, and C++, b) run that code and navigate both forward and backward through individual execution steps, c) see detailed visualizations of run-time program state at each step, d) send a URL to a friend or tutor start a real-time collaborative session, e) write and run code together, text chat, and see each other's mouse cursors.

5 Prior Work by the PI: The Python Tutor online programming platform

You might not need this section, but I needed it to introduce my Python Tutor platform which this proposal builds upon. If you have a body of prior work that your proposal naturally extends, then it's good to put it here so it's clearly separated from the prior work of others.

I created the Python Tutor online programming platform [45, 46] in 2010 and currently serve as its sole developer and code maintainer. From 2015–2018, my NSF CRII award funded the accelerated development of Python Tutor and several extensions that directly serve as the preliminary work for this CAREER proposal. (The NSF CRII is a new grant intended to help early-career PIs "produce sufficient preliminary results to serve as the basis for future competitive research proposals" [4] such as the NSF CAREER.)

Overview: As Figure 2 shows, Python Tutor is a free website [45] that lets users write code in ***six programming languages*** (it began with Python, but now also supports Java, JavaScript, Ruby, C, and C++), see what happens to data structures on the stack and heap when that code executes step-by-step, and collaboratively edit code and chat in real time. Its rendering engine automatically produces visualizations of nested and linked data structures commonly seen in introductory courses, which help learners overcome a fundamental barrier: understanding what happens as the computer runs each line of code [46]. Many people use Python Tutor as a visual debugging aid in conjunction with MOOCs, digital textbooks, and online coding tutorials.

Impact: Although automated program visualization tools have existed for decades [84], Python Tutor is now the most widely used of such tools [83, 84], with over 3.5 million users so far from over 180 countries. User survey results show that its user base is diverse in terms of age (41% are over 35 years old, 10% over 55 years old) and education levels (20% never attended college). Instructors in dozens of MOOCs and university courses have already incorporated it into their curricula. I attribute the widespread usage of Python Tutor in part to *long-term sustained engineering efforts* that I have put in over the past decade.

Platform for Scientific Research: Python Tutor's large international user base has turned it into a productive platform for my early-career research agenda. So far I have published 10 papers on research built upon it at venues such as CHI, UIST, TOCHI, and VL/HCC, including: new program visualization interfaces [46, 60], real-time code tutoring systems [47, 50], techniques for detecting novice programmer frustrations [28, 90], learnersourcing step-by-step code explanations [40], clustering student solutions for scaling instructor feedback [38], and studies of barriers faced by older adults [48] and non-native English speakers [49] who are learning to code. ***This track record so far shows that I am capable of successfully implementing the new research ideas described in this proposal atop my Python Tutor platform.***

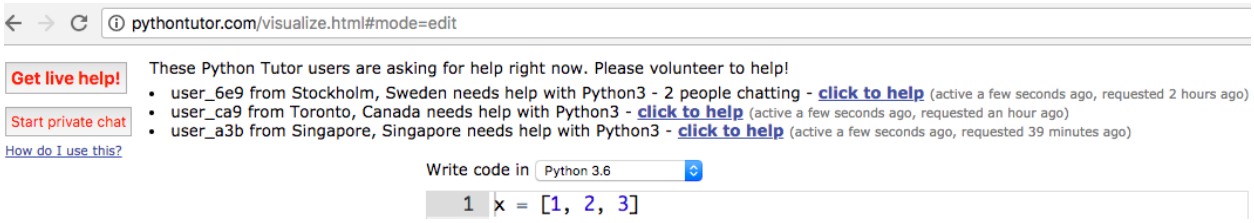

Figure 3: Clicking "Get live help!" will add a user to a public help queue at the top of the UI. Anybody using the Python Tutor website can volunteer to help, which starts a shared session like the one in Figure 2.

6 Proposed Research

Figure 1 summarizes the two Specific Aims of this proposal, which trend toward increasing levels of scale, as shown by the horizontal red arrow. My NSF CRII work developed one-to-one [50] and one-to-many tutoring [47] for programming. Specific Aim 1 scales this format up to many-to-many tutoring, allowing ad-hoc groups of learners to help each other in real time. Specific Aim 2 further increases scale by letting learners leave snippets of asynchronous coding help. Although these two aims follow a narrative arc as outlined in Figure 1, each is a standalone system that can be implemented independently of one another and result in independent research publications, thus ***minimizing the overall project risk in this proposal***.

Your relevant prior work should feed into your Specific Aims to form a coherent narrative arc.

6.1 Specific Aim 1. Omnitutor: many-to-many tutoring

Ideally everyone learning to code would receive real-time tutoring from an expert whenever they needed help. One way to approximate this ideal in an online setting where experts are scarce is to use other learners as peer tutors. To work toward this goal, I propose to build *Omnitutor*, a many-to-many tutoring system that enables groups of learners to help one another in real time on the Python Tutor website.

6.1.1 Preliminary Work

My NSF CRII work (left part of Figure 1) forms the foundation for my proposed Omnitutor system. I first created Codechella [50], an extension to Python Tutor that turns it into a multi-user environment where multiple users can simultaneously write code, explore its run-time state visualizations, and chat (Figure 2d and e). Using Codechella, a learner can send a URL to an expert in order to receive real-time one-to-one tutoring. Although this system is useful for people who know someone who can help them at the moment, its usefulness is limited since the majority of online learners do *not* personally know anyone who can help.

Each Specific Aim should start with the preliminary legwork you've already done toward this aim. Again this is why you shouldn't propose to go off on a brand-new direction, since you haven't done any credible legwork yet. The nuance here is that you should show a substantial amount of preliminary work but leave major gaps remaining for you to fill in with your proposed research plan.

To address this limitation, I then created Codeopticon [47], a one-to-many tutoring system with a dashboard interface that allows a single tutor to simultaneously help multiple learners. Using Codeopticon, a tutor can sign onto the Python Tutor website, monitor how a group of learners are progressing on their coding assignments, and assist via text chat. It amplifies a single expert's attention up to 10X and allows learners to receive real-time help if they happen to be using Python Tutor during the time that an expert is holding "virtual office hours" on the website. However, the limiting factor is that there are far fewer experts than learners in many realistic settings (e.g., a MOOC with 5 teaching assistants and 5,000 students), so even with Codeopticon it is infeasible for the few available experts to fulfill the help requests of all learners.

To overcome the aforementioned scalability limitations of Codechella and Codeopticon, in this specific aim I propose to use learners as peer tutors. The growth of Python Tutor's user base over the past decade means that ***there are now 50 to 200 learners concurrently using the website at any given time***. Unlike prior one-to-one or one-to-many tutoring systems, my proposed Omnitutor system will enable anyone currently using the Python Tutor website to freely help anyone else they want in a *many-to-many* manner.

As preliminary work for Omnitutor, I have implemented a prototype help queue that is displayed at the top of the Python Tutor website. Figure 3 shows that: a) Any user can request help by clicking the "Get live help!" button to add themselves to a public queue. b) Everyone using the Python Tutor website sees the

help queue at the top of their UI, which shows a time-ordered list of who is asking for help and where they are from; this screenshot shows users from Sweden, Canada, and Singapore, respectively. c) Anyone can volunteer to help by clicking the link next to a help request. The helper then joins a shared session with the help requester so that they can provide real-time assistance (Figure 2).

This help queue is the first step toward many-to-many tutoring. Preliminary results look promising: Between November 2017 (when this feature debuted) and July 2018 (when this proposal was written), Python Tutor users have volunteered to hold over 6,547 substantive help sessions with total strangers. We define a "substantive" session as one that contains at least 10 back-and-forth chat exchanges (20 messages); very short sessions are usually failed help attempts. In each substantive session, participants chatted for an average of 29 minutes and exchanged 76 chats, which is comparable in length to in-person tutoring sessions [50]. In sum, ***this preliminary work establishes the feasibility of many-to-many tutoring*** by showing that learners using Python Tutor are interested in and capable of volunteering to help one another in real time.

6.1.2 Research Plan

Omnitutor will extend my preliminary work—the prototype help queue in Figure 3—with three new algorithms and interactions whose collective goal is to *maximize help quality while minimizing wait times*:

When I first saw examples of funded NSF proposals, I was surprised by the amount of text that came before, well, the actual proposal of what research they were planning to do! For instance, we're already at page 6 out of 15 here when I first mention my Research Plan. Don't worry, that's normal.

Code similarity analysis: Right now all Python Tutor users see an ordered list of who needs help in the queue, but not what they need help on. We believe that someone currently on the site is more likely to volunteer to help if the code they are working on is similar to what the requester is working on. Thus, Omnitutor will continually analyze all users' code edits and summarize them into a compact representation suitable for similarity analyses. Then it will proactively notify promising helpers based on code similarity.

The main technical challenge here is finding an appropriate level of granularity. Our hunch is that taking literal syntactic or AST tree diffs will *not* be effective: Everyone's code will superficially look different since they are working on different tasks. We instead propose to summarize code by two main features: 1) what modules and API function calls it is using, and 2) what language features it is using, as drawn from programming education concept inventories [19, 89] (e.g., recursion, closures, higher-order functions).

The most common criticism of research plans is that they're not specific enough. So be specific. Be as specific as you can and as decisive as you can in your language. Don't sound like you're just going to explore a bunch of possible options. In reality everyone knows you WILL end up exploring a bunch of possible options (that's the nature of research!), but you should at least lay out a plausible plan upfront. I know it might feel weird to speak about the future with such overt decisiveness, but just trust me and do it.

Using this code similarity analysis, for each user currently on the Python Tutor website, Omnitutor will proactively highlight the requesters on the help queue whom they are best suited to help. For instance, if someone is writing code to process JSON data using Python list comprehensions, Omnitutor may suggest for them to help someone who is also working with JSON API calls or with list comprehensions.

This algorithm improves the odds of a more successful tutor match, but why not just let the help requester directly write a short public note describing what they need help on? We piloted this idea but ultimately opted not to keep it for three main reasons: 1) It requires help requesters to put in additional work, whereas our proposed code similarity analysis is fully automatic. 2) Novices often have trouble articulating their problems precisely, so our pilot tests showed that many such notes end up being uninformative or vague. 3) There is the potential for inappropriate messages to be shown to everyone on the help queue.

Machine-assisted code simplification: Our experiences with the help queue over the past year revealed two main sources of user frustration: 1) Help requesters are frustrated when they need to wait too long for someone to arrive, which can occur when there are not enough people currently on the Python Tutor website. 2) Volunteer helpers enter a session eager to help but have a hard time providing useful assistance since the requester's code is too complicated and they may not know how to articulate well-posed questions.

To simultaneously address both frustrations, we propose to create a chatbot that can help the requester reduce their problem to its root cause. After someone makes a help request, the bot exchanges chat messages with them while they are waiting for a human helper to arrive. The bot's chat script will be modeled after debugging best practices, with the goal of getting the help requester to ***simplify their code to a minimal test case*** [7, 99] that exemplifies the root of their problem. It would start by asking high-level questions such as *"Please point to where you think the problem is in your code ..."*; then as the user makes initial (probably

incorrect) guesses, the bot would hone in and request follow-up actions like, *"Comment out this part of your code, re-run it, and let me know if you still see the same problem."* And if the problem disappears, the bot could direct the user to backtrack: *"Let's undo that change and then let me know if the problem re-appears."*

The main technical challenge is designing a chat script that is general enough to cover a variety of introductory programming use cases. Our hunch is that a simple script that forces learner self-explanations [21, 22] will be an effective start, inspired by the classic ELIZA bot for psychotherapy [98] and the common practice of *rubber duck debugging* [6] (i.e., getting the learner to figure out their own problems by having them talk to a rubber duck about their code). We will also analyze the chat logs of over 6,547 help sessions from our preliminary work to determine how to design our bot to sound more like real human tutors.

Interacting with a chatbot will serve the dual purpose of both giving the help requester something to do while waiting for a helper to arrive, and also simplifying their code so that when a human does arrive, they have a greater chance of successfully helping. Finally, why not create a bot that can provide fully-automated help? As we discussed in Section 4, while there have been major advances in Intelligent Tutoring Systems, we still believe that it is hard for a bot to fully match the flexibility and empathy of a human helper.

Remember I said to be specific, but the problem is that you can't really be that detailed in your research plan since you just don't have enough space. I had only two Specific Aims and used 1.25 pages for the research plan of each one. If you have three aims, you may have only 1 page for each. Don't worry, that's OK. You can be specific without being overly detailed. Specific means telling the reader exactly what you plan to do, not that you'll maybe explore this thing or that thing or this other thing.

Impromptu study groups: Code similarity analysis and chatbot assistants can improve help quality, but there is still no guarantee that anyone will volunteer to help. To reduce wait times, Omnitutor will suggest for everyone currently on the queue who is working on similar code to join impromptu study groups.

This UI is inspired by students forming in-person study groups where everyone works on their own code individually but can chime in to help one another as needed. To simulate this experience online, Figure 4 shows that when Python Tutor users join a study group, they see their own code alongside a Codeopticon-like dashboard [47] that shows all their fellow group members' coding activities in real time. Members can chat individually with one another or broadcast their chats to everyone in the group. If a helper eventually joins a requester's session, everyone else in their study group can watch along to learn indirectly. This interaction is akin to a crowd of other students huddling around an instructor who is currently helping a student to see if they too can learn something relevant.

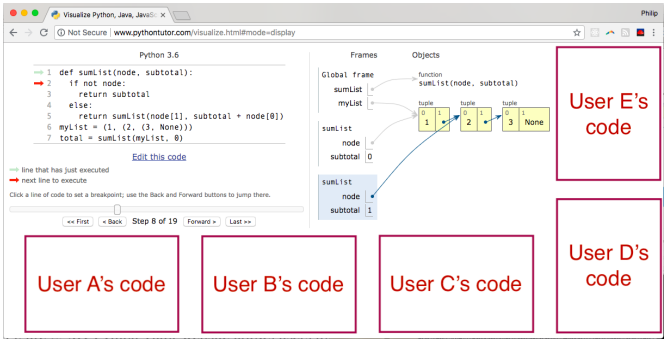

Figure 4: UI mockup of impromptu study groups where a user sees all group members' code and can chat with them.

6.1.3 Evaluation Plan

The goals of Omnitutor are to minimize wait time and to maximize help quality for learners who request help on the Python Tutor site, so that is what we plan to measure as dependent variables in our evaluation.

Wait time will be defined as the time between someone requesting help and a volunteer joining to make an earnest attempt at helping. In our experience analyzing chat logs so far, a heuristic of both parties each exchanging at least 10 chat messages back and forth is a reasonable indicator that a real attempted help session is occurring. We will also measure "dropouts" when a requester logs off without getting helped.

Help quality is more subjective and thus harder to quantify. To measure this property, we will follow a similar methodology as we did when analyzing chat logs for the Codechella paper [50]: We will get a panel of introductory programming instructors to serve as expert raters and have them look over the chat logs of sessions to assess the quality of each one. In our experience with Codechella, the easiest-to-measure indicator of quality is a binary one: *Was the helper successful in answering the requester's immediate question?* Usually the requester will explicitly acknowledge a successful helper by saying thanks. This is a start, but we want to go deeper, so we will also have the expert raters do a *close reading* [16] of the chat

The second most common criticism of research plans is that they don't come with a credible evaluation plan. Reviewers will ask, *OK you've told me what you want to do, how will we know whether it works or not?* Just like your research plan, your evaluation plan should be specific specific specific. Show you've put some serious thought into methodology and metrics.

logs. We will give raters a rubric based on Bloom's taxonomy [14] and the SOLO taxonomy [13] to quantify the level of sophistication of knowledge transfer between tutor and learner. For instance, a chat interaction where a learner was able to apply their knowledge to a new coding problem would be rated higher than one where they just remembered rote facts (Apply=Level 3 in Bloom's Taxonomy, Remember=Level 1). Note that this evaluation is inherently limited by the fact that we have no way of contacting the session participants to assess true knowledge transfer; the best we can do is to use their chat logs as a proxy.

Once establishing these metrics, we will run ***randomized controlled trials*** (RCTs) on the live Python Tutor website to measure the impact of Omnitutor's features on the dependent variables of wait time and help quality. Although imperfect, randomization is powerful because it can minimize selection bias of participants and other confounders such as the fluctuating numbers of learners using Python Tutor during different times of day. Specifically, here are the RCTs we will perform on each of the three features:

For both of my Specific Aims, I spent almost as much space on the evaluation plan (1 page) as on the research plan (1.25 pages). I felt it was worthwhile to be a bit less detailed in my research plan in order to leave more space for evaluation plan details.

Code similarity analysis: For each help request, randomly decide whether or not to perform this analysis, and measure its impact on wait time and help quality. Note that we can also use these RCTs as formative evaluations to refine our code similarity analysis by testing new heuristics versus their predecessors.

Machine-assisted code simplification: For each help request, randomly decide whether to activate the chatbot assistant, and measure its impact on help quality if a human helper eventually arrives. We will also measure whether the chatbot reduces queue dropout rates because the requesters will presumably be more engaged and less likely to get bored of waiting. Finally, we will have expert raters assess the size and quality of the code that results after the chatbot helps the requester simplify their code.

Impromptu study groups: We will randomly activate and deactivate the study groups feature every hour (e.g., active during the top half of the hour, inactive during the bottom half). Note that we cannot randomize by individual requester here since this is a globally-visible feature that affects everyone on the site. We will measure the impact of study groups on wait times, dropout rates, and help quality.

The large user base of Python Tutor makes randomized controlled trials more compelling since we regularly get over 10,000 daily active users on the site. Also, in eight months of deploying the prototype help queue so far, we have logged over 6,547 substantive help sessions (containing at least 10 back-and-forth chat exchanges), which is an average of 27 per day. Thus, by deploying these trials for one full year, ***we will likely get on the order of one million participants and 10,000 help sessions' worth of data.***

6.1.4 Expected Outcomes

- To our knowledge, the first interactive system that organizes anonymous online learners to help one another code in real time even when they are individually working on their own unique code.
- Algorithms and interaction techniques for matching requesters with potential helpers and for interactively helping requesters simplify their code to improve their odds of getting successfully helped.
- Empirical evidence from RCTs that our algorithms enable learners to get helped faster and to engage in higher-quality tutoring sessions than using a baseline help queue with no machine assistance.
- To our knowledge, the first freely-available dataset of anonymized chat logs that can provide insights into how total strangers interact when helping one another with programming problems.

6.1.5 Potential Risks and Alternative Approaches

One risk is that Python Tutor users may not be able to successfully provide help since they are also learners. However, from reading the chat logs of thousands of sessions initiated so far using our prototype help queue (Figure 3), we found that, anecdotally, users are often able to provide reasonable levels of help, similar to how peers in a classroom can help one another. An alternative approach to improving help quality is to recruit *power users* on the Python Tutor site (likely more advanced learners) to serve as helpers.

End every Specific Aim with a thoughtful discussion of risk and alternative approaches. This will demonstrate humility and counterbalance the overtly-decisive tone you had to take earlier in your research and evaluation plans. Humbly show how you're aware of what can possibly go wrong and propose ways to address those risks.

Any online interaction with strangers carries the risk of potentially offensive messages being sent. To mitigate this risk, Omnitutor will allow help requesters to kick and ban anyone from their sessions, and also

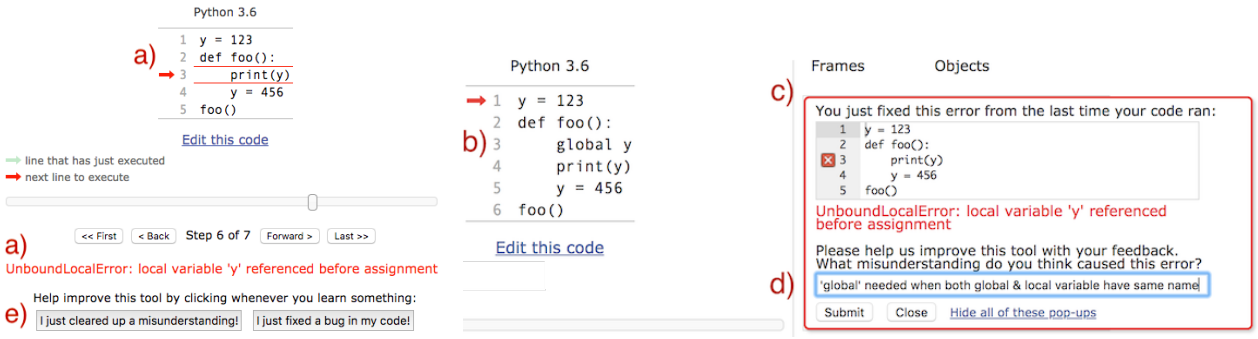

Figure 5: Inline error annotations: a) The user encounters an error while coding in Python Tutor, b) fixes their code and re-runs, c) sees a pop-up dialog showing the error they just fixed, and d) writes an annotation about why they made that error. e) The user can also click a button to make a freeform annotation.

to report problematic IP addresses to our team so that we can investigate in depth. From our experiences monitoring thousands of chat sessions over the past year, we have seen very few cases of bad behavior; since all chats are private, there is little incentive to spam or troll since those messages are never shown in public. Related, Python Tutor *does not have customizable user profiles or reputation scores* since we wanted to maximize privacy; everyone who uses the website gets a randomly-generated persistent username (e.g., `user_6e9` in Figure 3). We may add these features later if they can improve peer interactions on the site.

Finally, any help mechanism (including getting in-person help from friends) has the potential for misuse to violate academic integrity. Note that the majority of Python Tutor users are learning on their own and *not taking a formal course for grades*. That said, Python Tutor displays a clear message reminding users to abide by relevant academic integrity standards. Fortunately, from reading chat logs so far, we observed that helpers usually refuse to simply give away answers or to do someone's homework for them, which is similar to community norms on MOOC forums and Stack Overflow.

6.2 Specific Aim 2. Rosetta: learner-generated code explanations

Synchronous help is limited by the number of people available at any given moment to help. To overcome this limitation, I propose to aggregate the insights of everyone who has used Python Tutor in the past to leave asynchronous help for future users. To work toward this goal, I will build *Rosetta*, a system that lets users annotate their code with inline explanations that might benefit future users who face similar issues.

The same format as Specific Aim 1. Ideally all of your aims would be of a similar "size" so none feel exceedingly small or big. Otherwise an easy reviewer criticism is why you bothered including such a small aim or that another aim is too big to be feasibly completed.

6.2.1 Preliminary Work

Every day around 10,000 people write, run, and debug 100,000+ pieces of code on the Python Tutor website. As preliminary work for this proposed Rosetta system, I wanted to determine whether learners were willing and able to annotate their own code with notes that may be helpful to future learners. To do so, during my CRII project (but not yet published) I implemented three annotation features atop Python Tutor:

1) Syntax error annotations: Syntax errors are very common and frustrating for learners, often because compiler error messages are hard for novices to decipher [10, 54, 85]. For instance, say that a Python Tutor user tries to run the following code: "`x=[1,2,3`" and sees this cryptic message: "*SyntaxError: unexpected EOF while parsing*." They might puzzle over that error and suddenly realize that they need to add a closing bracket: "`x=[1,2,3]`" At the moment they *fix* that error, Python Tutor pops up a dialog with the prior erroneous code and a prompt that asks: "What misunderstanding do you think caused this error?" The user can write a note summarizing their insights at that moment when the error is still fresh on their mind.

2) Runtime error annotations: Figure 5 shows a similar dialog for runtime errors: a) The user sees the following when they step to line 3: "*UnboundLocalError: local variable 'y' referenced before assignment.*" b) After puzzling over this error for a while, they realize that they need to add a `global` declaration before

that line. c) After fixing and re-running, Python Tutor pops up a dialog showing the prior erroneous code and asking, "What misunderstanding do you think caused this error?" d) The user writes in: "'*global' needed when both global & local variable have same name*". This annotation captures their insights about the possible root cause of that error. Python Tutor saves its text along with both the erroneous and fixed code.

3) Freeform annotations: Finally, Figure 5e shows two buttons that the user can click any time they want while visualizing step-by-step code execution: "I just cleared up a misunderstanding!" and "I just fixed a bug in my code!" Clicking brings up an annotation prompt. Python Tutor then saves the user's annotation along with the current state of their code and visualization in order to capture the surrounding context.

Preliminary results indicate that Python Tutor users are willing to write these annotations: Between May 2017 (when this feature launched) and when this proposal was written in July 2018, users collectively submitted 14,338 annotations containing at least 5 characters each: 4,621 for syntax error fixes, 3,138 for runtime error fixes, 4,100 for "I just cleared up a misunderstanding!" and 2,479 for "I just fixed a bug in my code!" (Filtering by string length of at least 5 characters eliminated many nonsensical submissions.)

From reading through these annotations, I observed that they capture a diverse variety of learner insights that are likely more helpful to future learners than the default compiler error messages are. For instance, syntax error annotations such as one learner writing "*change '=' to '==' in if condition*" are much more informative than Python's default message of "*SyntaxError: invalid syntax*" when '=' is improperly used for comparisons. Similarly, runtime annotations such as "*I accidentally used 'append' instead of 'join'*" indicate ideas for potential fixes when Python would have simply produced a generic runtime error.

Specific examples are great in research plans to clarify your ideas and to, well, show that you're being specific. In hindsight, I should've used more examples.

In sum, ***this preliminary work establishes the feasibility of learner-generated code explanations*** by showing that Python Tutor users are capable of annotating their code with precise and actionable messages.

6.2.2 Research Plan

Rosetta will extend my preliminary work on error annotations (Figure 5) to create context-specific explanations that can help future Python Tutor users. Rosetta will contain three interface components:

Learner-augmented error messages: We will first augment the default compiler/run-time error messages in Python and other supported languages on the platform (Java, JavaScript, C, C++, and Ruby) with annotations written by learners who previously encountered those errors. Thus, the user experience for someone who encounters a coding error in Python Tutor will be to see both the default message (for consistency) along with the most relevant annotations from earlier users who had errors when writing similar code.

The main technical challenge is to generalize the code snippet that accompanies each annotation so that Rosetta can display that annotation when future users encounter errors on similar (but not exactly identical) code. For compile-time errors, the problem is that the code does not parse, so we do not even have ASTs to compare. To cope, we will use techniques from error-tolerant code parsing [17] and program synthesis [76] to try to match locally-similar partial parses. For example, if someone writes an annotation of "*change '=' to '==' in if condition*" when trying to compile "`if(x=5)`", that annotation should ideally be shown to any future user who tries to use '=' within an if-statement condition, not just an exact match for "`if(x=5)`".

Again, examples can be very effective.

Rosetta will try to match run-time errors in a similar way, except that since the code can now compile and execute, it will have both ASTs and run-time value traces to use for matching up code with annotations.

We will iteratively develop and refine our code-matching heuristics using our corpus of over 14,338 code+annotation pairs collected so far from preliminary work (Figure 5).

Voting on and iteratively refining annotations: To account for imprecisions in our heuristics for matching code with annotations, Rosetta will allow users to vote on the quality of annotations they see and to iteratively refine them. In the above example, if a user tries to compile "`while(x=5)`" and sees the "*change '=' to '==' in if condition*" annotation, they can choose to refine it to "*change '=' to '==' in while condition*" or, even better, to generalize it to "*change '=' to '==' when making comparisons*". This way, as more and more Python Tutor users view these annotations, the overall corpus will gradually improve in quality.

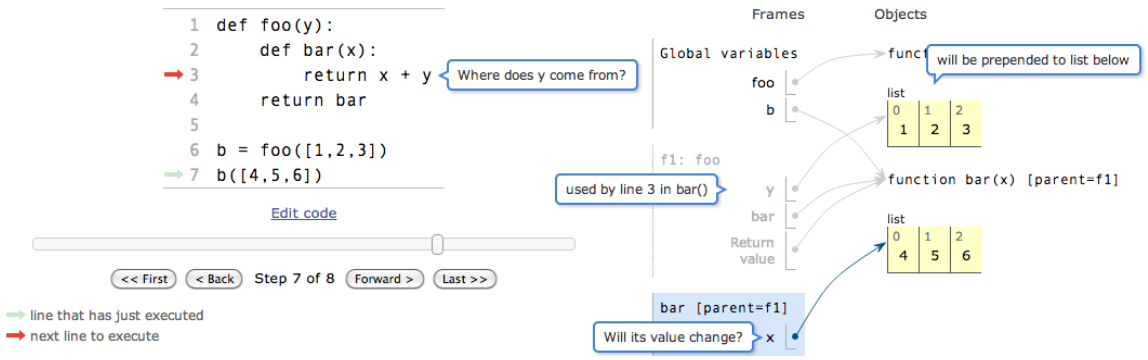

Figure 6: UI mockup of our proposed custom explanation requests interface. Users will be able to annotate both code and visualization components with precise questions and answers (shown as blue text bubbles).

In addition, 10,000 users per day seeing and potentially voting on Rosetta annotations lets more useful ones rise up to the top and grays out lower-quality ones, similar to voting on Q&A sites such as Stack Overflow.

Custom explanation requests: So far Rosetta has augmented compiler and run-time error messages, but what about learners who need help understanding parts of their code that do *not* have errors? Using Omnitutor (Specific Aim 1), they can ask a helper to explain that code to them via chat, but what if nobody is available? To allow users to make asynchronous help requests, Rosetta will let them take a snapshot of their current code and visualization, and then annotate it with a question. The UI mockup in Figure 6 shows that having a run-time state visualization allows users to ask precise questions about both their code and data.

Another way to be specific without having actually built anything yet is to create mockup UIs and diagrams to sketch out your ideas. Again this contributes toward your research plan being specific specific specific.

Rosetta will post the requester's question to the sessions of current and *future* Python Tutor users who are working on similar code, again using our code similarity analysis from Section 6.1.2 to find likely matches. Hopefully those users will be willing to glance at the visualization snapshots of relevant help requesters (e.g., Figure 6) and write a helpful explanation since they are currently working on similar code. Rosetta will send the requester an email to notify them when new explanations arrive. This asynchronous setup greatly increases the pool of users who can answer questions as long as the requester is willing to wait.

Finally, just like with compiler and run-time error annotations, Rosetta will collect these custom explanation requests and present them to future users who are working on similar code. Note that this user interface differs from a traditional discussion forum or Stack Overflow-like Q&A site because questions and answers are not shown publicly to everyone; they are shown only to Python Tutor users who are working on similar code, which drastically reduces the amount of irrelevant content visible in the interface.

6.2.3 Evaluation Plan

Similar to the evaluation plan for Specific Aim 1, we will again run randomized controlled trials. The dependent variables here will be annotation quality, error fix metrics, and learner-reported satisfaction.

We will measure ***annotation quality*** again using a panel of introductory programming instructors to serve as expert raters. We will direct raters to read each annotation and its associated code+visualizations to assess correctness (i.e., is the annotation even factually correct?), precision (does it precisely describe the core issue?), and depth (does it go beyond surface-level features to talk about deeper concepts?).

But no matter how good experts think the annotations are, ultimately what matters is whether those annotations help real learners. To evaluate this, we will design several ***error fix metrics*** to quantify how well learners actually fix their errors in response to seeing annotations made by their peers. Possible metrics include: time to fix errors, numbers of edits required to fix, and whether those errors re-occurred later.

Our final dependent variable is ***learner-reported satisfaction***, which we will measure with a short question displayed alongside each annotation. We acknowledge that self-reported data can be noisy, but it is the only practical way of assessing how learners felt about responses to custom-requested explanations when

there was not an explicit error to be fixed (e.g., *"Please tell me what this part of the code does ..."*). The ideal metric is whether the learner gained true lasting knowledge from reading the annotation, but that is hard to measure at scale so instead we will use self-reported satisfaction and error fix metrics as proxies.

We will run the following randomized controlled trials (RCTs) on the live Python Tutor website to measure the impact of Rosetta's features on the aforementioned dependent variables:

Learner-augmented error messages: We will randomly assign Python Tutor users to either see only the built-in compiler/run-time error messages or to also see a list of suggested learner-generated error messages. We can measure the impact of this intervention on error fix metrics. Note that we can also use RCTs here as formative evaluations to refine our similarity heuristics for matching learner-generated messages with code.

Voting on and iteratively refining annotations: We will randomly activate or deactivate the voting and refinement features on each annotation, and measure impact on final annotation quality, error fix metrics, and learner-reported satisfaction. Doing so will help us assess the efficacy of voting and iterative refinement.

Custom explanation requests are harder to assess via RCTs (it is not as clear what to randomize), so instead we will perform a more descriptive evaluation by assessing explanation quality, error fix metrics, and learner-reported satisfaction (both from the learner who originally asked the question and future learners who see these explanations as they are coding). We will also measure wait time before receiving responses.

In sum, again the active user base of Python Tutor makes it feasible to perform randomized controlled trials with large sample sizes. By deploying these RCTs online for one year, we estimate that we can get on the order of ***one million participants interacting with tens of thousands of unique annotations***.

6.2.4 Expected Outcomes

- To our knowledge, the first system that coordinates a group of anonymous learners to create a corpus of explanations about code and errors while they are in the midst of doing their own coding.
- Algorithms and interaction techniques for matching learner-generated error messages with relevant code, for voting on and refining annotations, and for custom explanation requests.
- Evidence from RCTs that our algorithms enable learners to generate high-quality annotations, as judged by experts, and to fix errors faster than when seeing default compiler/run-time error messages.
- A freely-available dataset of learner-generated annotations that provides insights into common novice programming misconceptions across six popular languages (Python, Java, JavaScript, Ruby, C, C++).

6.2.5 Potential Risks and Alternative Approaches

The main potential risk is that learner-generated annotations may be low-quality or contain offensive content. Note that this is a risk shared by *any* system that allows users to freely post questions and answers. Just like on sites such as Stack Overflow, Rosetta's users can downvote annotations to de-prioritize them and also flag potentially-offensive ones for the research team to inspect in detail. In addition, since we will be continually running RCTs of Rosetta's features on the live Python Tutor site, if we notice that a particular annotation consistently performs worse than baseline, then we will flag it to inspect for problems. Given the current rate of $\sim$35 new annotations being written each day (Section 6.2.1), we expect light moderation effort to be manageable by the research team for the coming few years. A more scalable alternative approach is for us to enlist trusted power users of the Python Tutor website to volunteer as annotation moderators.

6.3 Timeline of Planned Research and Education Activities

One full-time Ph.D. student + PI effort: *Year 1*: Omnitutor implementation. *Year 2*: Omnitutor year-long evaluation via online deployment; curriculum integration for Education Plan (Section 7). *Year 3*: submit Omnitutor paper to CHI or UIST; Rosetta implementation. *Year 4*: Rosetta year-long evaluation; integrate with online materials for Education Plan. *Year 5*: submit Rosetta paper to CHI or UIST; finish evaluation portions of Education Plan; submit a journal paper on longitudinal deployment experiences.

Some people include a project timeline diagram, but I think it's a waste of valuable space that's better spent on your research and evaluation plans. Make this section as short as possible while still being specific.

7 Education Plan: Tight Integration of Research and Education

My Python Tutor platform is unique in that using it for education directly leads to new research ideas, which then contribute back to improving education, which in turn inspires additional research in a positive feedback loop. With the help of my six external collaborators (see below), I will implement and evaluate the following education efforts both within my own university and beyond.

CAREER proposals have a 15-page limit. I reserved almost 3 pages for Education Plan and Broader Impacts, even at the expense of being less detailed in the research sections. That's far more space than what others devote, but I strongly stand by this decision. I believe if you have a reasonably good research plan, then showing you've thought seriously about education and broader impacts will put you over the top. This is where your proposal can really stand out amongst other similarly-strong ones in your pool.

7.1 Letters of Collaboration (attached as supplementary documents)

- XXX XXX, Director of Research at Google, co-creator of the first AI MOOC in 2011, co-author of one of the most widely-used computing textbooks: *Artificial Intelligence: A Modern Approach* [77]
- XXX XXX, co-founder and CEO of edX, the world's largest nonprofit provider of MOOCs [1]
- XXX XXX, Distinguished Professor and Dean of the Division of Social Sciences, UC San Diego
- XXX XXX, Professor and Director of the Halicioglu Data Science Institute, UC San Diego
- XXX XXX, Prof. of Education Studies, Advisor of Teaching+Learning Commons, UC San Diego
- XXX XXX, Assistant Professor at UC Berkeley, co-creator of the Jupyter notebook-based scientific computing environment [2], lead recipient of 2017 ACM Software System Award [3]

7.2 New Curriculum Development Across the Social Sciences and Data Science at UCSD

Collaborator names have been anonymized in this public annotated version.

My project period (2019–2024) coincides with two new UCSD education initiatives that I will help launch:

- The Dean of the Division of Social Sciences (which houses my primary department, Cognitive Science) is planning to create an ambitious new initiative to teach computer programming to all undergraduate students in the division. ***This is the largest academic unit at UC San Diego***, with 29% of the 28,000+ undergraduate students majoring in the social sciences [8]. As one of the few faculty members in the division whose training is in computer science (and who has worked as a professional software engineer at Google), I have been selected by the Dean to help launch this curricular effort.
- The UCSD Data Science Institute began operations in March 2018, with a large amount of undergraduate enrollment interest throughout campus. I am a founding faculty member of the Institute and will be involved in creating the educational curriculum for the new undergraduate data science major and minor, especially in using Python for introductory programming and data analysis courses.

Think creatively beyond just putting your research into new course curricula or free online materials. Talk to people at your university about how you can get involved beyond the level of individual courses. And the more tightly you can integrate your research and education plans, the better.

These activities will allow me to ***tightly integrate my research and education efforts***. Python Tutor and my proposed Omnitutor and Rosetta systems will be critical for scaling up personalized learning to the tens of thousands of students enrolled in new introductory programming courses across the Social Sciences and Data Science. I will work closely with relevant instructors to integrate my systems into their courses as needed. Then as these courses ramp up in the coming years, I will collect usage data and likely discover new research opportunities related to how students from non-engineering backgrounds are learning to code.

What I find most exciting about this plan is the opportunity to ***introduce programming skills to a highly diverse student population across the entire UC San Diego campus***, which can open career opportunities for tens of thousands of students and also inspire new research directions. The social sciences already attract a far more diverse and representative undergraduate population than computer science and related engineering majors; based on early estimates, we predict that data science will follow suit. At UC San Diego, women comprise over 50% of social sciences majors, 21% of undergraduates are from underrepresented minority groups, 33% are transfers from community colleges, and 38% are first-generation college students [8, 9].

Just like how your research plans should each come with their own evaluation plans, your Education Plan should too! Being specific about evaluation shows that you're taking educational outcomes seriously.

Evaluation Plan: We plan to formally evaluate these new educational initiatives using standard student outcome metrics such as learning, retention, progression through the major, graduation rates, and job placement statistics. The Dean of Social Sciences and the Director of the Data Science Institute will provide outside evaluator personnel; the overall effort will potentially be organized by Professor XXX XXX of Education Studies, who is an expert on educational evaluations (see their letters of collaboration).

7.3 Education Efforts Lead to Undergraduate Research Mentoring

Another wonderful benefit of using Python Tutor in my teaching is that it attracts large amounts of interest amongst undergraduate students who first see it in class and then want to get involved in working on it for research. This project has been a magnet for attracting undergraduate researchers to my lab. As an assistant professor, I have mentored 12 undergraduate students on research so far, including 6 women and 2 first-generation college students. Of those, 3 have *first-authored* research papers [40, 95, 102]; one won the nationwide Computing Research Association (CRA) Outstanding Undergraduate Researcher Award in 2015; another won Honorable Mention for that same award; 3 are now in computer science Ph.D. programs at Stanford, Berkeley, and Georgia Tech, respectively. I plan to continue this trajectory in the coming years.

Applying for CAREER later gives you more time to build up the basis for your education plan and broader impacts. If I had applied too early, I wouldn't have had this mentoring record yet.

Evaluation Plan: I will informally evaluate this effort via levels of student involvement and their post-graduation outcomes. My CAREER project should involve 5 to 10 undergraduate researchers (1–2 per year); funding will be provided by my department and potentially by REU supplements. Their work experiences with me should help them gain admissions into CS/HCI Ph.D. programs or lead to their first full-time jobs.

It's OK if your evaluation metrics are informal. Reviewers aren't expecting you to be an expert at educational evaluation like how they're expecting you to be an expert at research evaluation.

7.4 Education Beyond the University: Online AI Textbook, MOOCs, Jupyter Notebooks

The systems in this proposal can improve programming education beyond university campuses, building upon the millions of learners in over 180 countries who are already using the Python Tutor platform. I will also partner with three influential figures in online education (see letters) to integrate with their platforms:

- XXX XXX (my manager when I worked at Google) and I have been talking about porting his popular *AI: A Modern Approach* [77] textbook to an online format with interactive visual examples built upon Python Tutor and using my new proposed systems to let readers help one another learn AI.
- XXX XXX, the CEO of edX (where I worked as a visiting researcher), is committed to deeper integration of Python Tutor and the new systems in this proposal into edX MOOCs to supplement traditional discussion forums. Note that Python Tutor is already used throughout several edX MOOCs for introductory programming and data science, so this addition is very tractable.
- XXX XXX, co-creator of Project Jupyter, wants to integrate Python Tutor visualizations and peer help into the Jupyter Notebook file format so that it can be used to create more interactive educational materials. His team has the resources to host these enhanced notebooks on JupyterHub cloud servers.

Aside from providing direct educational benefits to learners around the world who access these resources online, we predict that observing how people interact with them will inspire new research directions about how to make my proposed systems more useful for AI, machine learning, and data science applications.

Lastly, think beyond the scope of your own university. Get letters from external collaborators, perhaps in industry.

Evaluation Plan: We will measure success based on log analysis of how frequently my systems are used within the context of these online resources. We will also run RCTs to compare student performance when my systems are randomly shown or not shown to users. My collaborators' respective organizations (Google, edX, and Jupyter) will provide staff and computing resources to assist with these evaluation efforts.

8 Broader Impacts

This project will achieve these Broader Impacts goals from the NSF CAREER program solicitation [5]:

I suggest directly addressing as many of the NSF's broader impacts criteria as possible, using their exact wording as subsection headings. Again, be specific! But don't seem like you're just pandering. If your project can't realistically achieve broader impacts along some dimension, then don't force it.

Development of a diverse, globally competitive STEM workforce: Python Tutor and the new Omnitutor and Rosetta systems in this proposal directly contribute toward developing a strong STEM workforce since computer programming skills are highly in-demand across many STEM fields [52]. Moreover, the free peer-based tutoring that my systems facilitate will open up career training opportunities to a more socioeconomically diverse pool of learners, including those who cannot afford to pay for tutoring or classes. User survey results indicate that Python Tutor is already reaching millions of adult learners (41% of users are over 35 years old, 10% are over 55 years old) and those without formal college educations (20% of users).

Evaluation Plan: We already have a general user survey on the Python Tutor website that asks learners about their educational backgrounds and motivations for learning to code. This survey has received over 4,000 responses so far. We will work with Prof. XXX XXX from UCSD Education Studies (our evaluation expert) to enhance this survey with more formal measures of career-related outcomes from using my systems.

Just like with your research and education plans, put in an evaluation plan for each of your broader impacts aims. Again they don't have to be overly detailed or rigorous (nobody expects you to be an expert at evaluating societal impact!), but show you've put some thought into it.

Full participation of women and underrepresented minorities in STEM: My education plan to use my proposed systems to teach programming across the Social Sciences and Data Science majors at UC San Diego (Section 7.2) will make progress toward this goal at a large public university: women comprise over 50% of our social sciences majors, 21% of our undergraduates are from underrepresented minority groups, 33% are transfers from community colleges, and 38% are first-generation college students [8, 9].

Evaluation Plan: Diversity and inclusion metrics will already be included in our formal evaluation plans for the new programming curricula in the Division of Social Sciences and the Data Science Institute at UCSD.

Improved STEM education and educator development at any level: My education plan to use my proposed systems to enhance instructional materials such as online textbooks, MOOCs, and Jupyter Notebooks (Section 7.4) will help improve STEM education by providing educators around the world with access to free, high-quality, well-tested resources to use in their curricula. We can easily augment my systems to provide private groups for individual educators' classes in order to further increase student privacy.

Evaluation Plan: We will embed user surveys within these online instructional materials to assess how educators are using those materials to aid their own teaching.

Enhanced infrastructure for research and education: My Python Tutor open-source code base provides a robust infrastructure for both programming research and education. Its code has been maintained since 2010, improved via numerous bug fixes due to usage by over 3.5 million online users, used in dozens of university courses and MOOCs, and already used as infrastructure for computing research projects outside of my own lab [30, 36, 69, 80, 86, 87, 103]. *Evaluation Plan*: Publications and courses using my software.

Increased partnerships between academia, industry, and others: I have a long history of industry partnerships, having worked as a software engineer at Google and a visiting researcher at edX and Microsoft Research. I have coauthored papers with colleagues at all of these organizations. I also have informal ties with programming curriculum developers at two other major online education providers: Coursera and Khan Academy. *Evaluation Plan*: Industry paper coauthorships and use of my open-source software in products.

Increased public scientific literacy and public engagement with science and technology: My personal website [44] contains over 300 articles and podcast episodes on research, education, and outreach topics; it receives over 750,000 page views per year. I have also created over 500 research, education, and scientific outreach videos on my personal YouTube channel, which now has 4,000+ subscribers and 500,000+ total video views. *Evaluation Plan*: Measure webpage+video viewership statistics and outreach talk invitations.

Spending one full page on broader impacts is a great idea not only for CAREER but also for other NSF proposals. That's more than what most other people write, so your proposal will stand out in the pool.

9 Results From Prior NSF Support

You might not have anything to report here, but if you do, try to keep this section short while fulfilling the requirements. Space is much better spent elsewhere.

CRII: CHS: Scaling Up Online Peer Tutoring of Computer Programming (#1660819; $175,000; 2015–2018) produced the preliminary work for this proposal. *Intellectual Merits*: Published Codechella [50] and Codeopticon [47] real-time tutoring systems; see Section 6.1.1 for details. *Broader Impacts*: Grew Python Tutor's user base from 1.5 to 3.5 million total users during this project period.

co-PI on *NRT-IGE: Augmenting, Piloting, and Scaling Computational Notebooks to Train New Graduate Researchers in Data-Centric Programming* (#1735234; $50,000 my portion; $498,751 total; 2017–2020), an educational grant that is completely unrelated to the topic of this proposal. We plan to use Jupyter Notebooks to create new curricula for training graduate students. *Intellectual Merits*: knowledge about how to use existing notebook-based technologies to conduct coding boot camps for new graduate students. *Broader Impacts*: none so far; still in first year of curriculum development so not yet deployed.

If possible make sure to clearly state how your other grants are NOT related to this proposal so reviewers won't think that this work has already been funded, which is an easy justification for rejection.

References

[1] About edX – https://www.edx.org/about-us Accessed: July 2018.

[2] About Project Jupyter – http://jupyter.org/about Accessed: July 2018.

[3] ACM Software System Award – https://awards.acm.org/software-system/award-winners Accessed: July 2018.

[4] NSF Computer and Information Science and Engineering (CISE) Research Initiation Initiative (CRII) – https://www.nsf.gov/pubs/2018/nsf18554/nsf18554.htm Accessed: July 2018.

[5] NSF Faculty Early Career Development Program (CAREER) – https://www.nsf.gov/pubs/2017/nsf17537/nsf17537.htm Accessed: July 2018.

[6] Rubber Duck Debugging: Debugging software with a rubber ducky – https://rubberduckdebugging.com/ Accessed: July 2018.

[7] Stack Overflow: How to create a Minimal, Complete, and Verifiable example – https://stackoverflow.com/help/mcve Accessed: July 2018.

[8] UC San Diego Institutional Research: Student Profile 2017–2018 – https://ir.ucsd.edu/_files/stats-data/profile/profile-2017-2018.pdf Accessed: July 2018.

[9] UC San Diego: Supporting First-Generation Students – https://srs.ucsd.edu/support/first-gen.html Accessed: July 2018.

[10] Amjad Altadmri and Neil C.C. Brown. 37 million compilations: Investigating novice programming mistakes in large-scale student data. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*, SIGCSE '15, pages 522–527, New York, NY, USA, 2015. ACM.

[11] John R. Anderson, Albert T. Corbett, Kenneth R. Koedinger, and Ray. Pelletier. Cognitive tutors: Lessons learned. *Journal of the Learning Sciences*, 4(2):167–207, 1995.

[12] Tucker Balch. MOOC Student Demographics (Spring 2013) – http://augmentedtrader.com/2013/01/27/mooc-student-demographics/ Accessed: July 2018.

[13] John B. Biggs and Kevin F. Collis. *Evaluating the Quality of Learning: The SOLO Taxonomy (Structure of the Observed Learning Outcome)*. Elsevier Academic Press, 1982.

[14] Benjamin S. Bloom. *Taxonomy of Educational Objectives, Handbook 1: Cognitive Domain*. Addison Wesley, 1956.

[15] Benjamin S. Bloom. The 2 sigma problem: The search for methods of group instruction as effective as one-to-one tutoring. *Educational Researcher*, 13(6):4–16, 1984.

[16] Nancy Boyles. Closing in on close reading. *On Developing Readers: Readings from Educational Leadership, EL Essentials*, pages 89–99, 2012.

[17] Fraser Brown, Andres Nötzli, and Dawson Engler. How to build static checking systems using orders of magnitude less code. In *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '16, pages 143–157, New York, NY, USA, 2016. ACM.

[18] Donna E. Byrd. Peer tutoring with the learning disabled: A critical review. *Journal of Educational Research*, 84(2):115–118, 1990.

[19] Ricardo Caceffo, Steve Wolfman, Kellogg S. Booth, and Rodolfo Azevedo. Developing a computer science concept inventory for introductory programming. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education*, SIGCSE '16, pages 364–369, New York, NY, USA, 2016. ACM.

[20] Yan Chen, Sang Won Lee, Yin Xie, YiWei Yang, Walter S. Lasecki, and Steve Oney. Codeon: On-demand software development assistance. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, CHI '17, pages 6220–6231, New York, NY, USA, 2017. ACM.

[21] Michelene T. H. Chi, Stephanie A. Siler, Heisawn Jeong, Takashi Yamauchi, and Robert G. Hausmann. Learning from human tutoring. *Cognitive Science*, 25(4):471–533, 2001.

[22] Michelene T.H. Chi, Nicholas Leeuw, Mei-Hung Chiu, and Christian Lavancher. Eliciting self-explanations improves understanding. *Cognitive Science*, 18(3):439–477, July 1994.

[23] Parmit K. Chilana, Celena Alcock, Shruti Dembla, Anson Ho, Ada Hurst, Brett Armstrong, and Philip J. Guo. Perceptions of non-cs majors in intro programming: The rise of the conversational programmer. In *Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, VL/HCC '15, pages 251–259, Oct 2015.

[24] Parmit K. Chilana, Rishabh Singh, and Philip J. Guo. Understanding conversational programmers: A perspective from the software industry. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, CHI '16, pages 1462–1472, New York, NY, USA, 2016. ACM.

[25] Gayle Christensen, Andrew Steinmetz, Brandon Alcorn, Amy Bennett, Deirdre Woods, and Ezekiel J. Emanuel. The MOOC Phenomenon: Who Takes Massive Open Online Courses and Why? *(working paper)*, 2013.

[26] Janet W. Colvin. Peer tutoring and social dynamics in higher education. *Mentoring & Tutoring*, 15(2):165–181, 2007.

[27] Brian Dorn and Mark Guzdial. Learning on the job: Characterizing the programming knowledge and learning strategies of web designers. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '10, pages 703–712, New York, NY, USA, 2010. ACM.

[28] Ian Drosos, Philip J. Guo, and Chris Parnin. HappyFace: Identifying and predicting frustrating obstacles for learning programming at scale. In *Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, VL/HCC '17, pages 171–179, Oct 2017.

[29] Barbara Ericson and Mark Guzdial. Measuring demographics and performance in computer science education at a nationwide scale using ap cs data. In *Proceedings of the 45th ACM Technical Symposium on Computer Science Education*, SIGCSE '14, pages 217–222, New York, NY, USA, 2014. ACM.

[30] Barbara J. Ericson, Kantwon Rogers, Miranda Parker, Briana Morrison, and Mark Guzdial. Identifying design principles for cs teacher ebooks through design-based research. In *Proceedings of the 2016 ACM Conference on International Computing Education Research*, ICER '16, pages 191–200, New York, NY, USA, 2016. ACM.

[31] John W. Fantuzzo, Gwendolyn Y. Davis, and Marika D. Ginsburg. Effects of parent involvement in isolation or in combination with peer tutoring on student self-concept and mathematics achievement. *Journal of Educational Psychology*, 87(2):272–281, 1995.

[32] John W. Fantuzzo, Linda A. Dimeff, and Shari L. Fox. Reciprocal peer tutoring: A multimodal assessment of effectiveness with college students. *Teaching of Psychology*, 16(3):133–135, 1989.

[33] Bill Ferster. *Teaching Machines: Learning from the Intersection of Education and Technology*. JHU Press, 2014.

[34] Denae Ford, Kristina Lustig, Jeremy Banks, and Chris Parnin. "we don't do that here": How collaborative editing with mentors improves engagement in social q&a communities. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, CHI '18, pages 608:1–608:12, New York, NY, USA, 2018. ACM.

[35] Denae Ford, Justin Smith, Philip J. Guo, and Chris Parnin. Paradise unplugged: Identifying barriers for female participation on Stack Overflow. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, FSE 2016, pages 846–857, New York, NY, USA, 2016. ACM.

[36] Elena L. Glassman, Lyla Fischer, Jeremy Scott, and Robert C. Miller. Foobaz: Variable name feedback for student code at scale. In *Proceedings of the 28th Annual ACM Symposium on User Interface Software and Technology*, UIST '15, pages 609–617, New York, NY, USA, 2015. ACM.

[37] Elena L. Glassman, Aaron Lin, Carrie J. Cai, and Robert C. Miller. Learnersourcing personalized hints. In *Proceedings of the 19th ACM Conference on Computer-Supported Cooperative Work & Social Computing*, CSCW '16, pages 1626–1636, New York, NY, USA, 2016. ACM.

[38] Elena L. Glassman, Jeremy Scott, Rishabh Singh, Philip J. Guo, and Robert C. Miller. Overcode: Visualizing variation in student solutions to programming problems at scale. *ACM Trans. Comput.-Hum. Interact.*, 22(2):7:1–7:35, March 2015.

[39] Max Goldman, Greg Little, and Robert C. Miller. Real-time collaborative coding in a web ide. In *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology*, UIST '11, pages 155–164, New York, NY, USA, 2011. ACM.

[40] Mitchell Gordon and Philip J. Guo. Codepourri: Creating visual coding tutorials using a volunteer crowd of learners. In *Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, VL/HCC '15, pages 13–21, Oct 2015.

[41] A. C. Graesser, P. Chipman, B. C. Haynes, and A. Olney. Autotutor: an intelligent tutoring system with mixed-initiative dialogue. *IEEE Transactions on Education*, 48(4):612–618, Nov 2005.

[42] Charles R. Greenwood. Classwide peer tutoring: Longitudinal effects on the reading, language, and mathematics achievement of at-risk students. *Journal of Reading, Writing, and Learning Disabilities International*, 7(2):105–123, 1991.

[43] C.R. Greenwood, J.J. Carta, and D. Kamps. Teacher-mediated versus peer-mediated instruction: A review of educational advantages and disadvantages. In *Children Helping Children*, pages 177–205. John Wiley and Sons, New York, 1990.

[44] Philip J. Guo. personal website. `http://www.pgbovine.net/` Accessed: July 2018.

[45] Philip J. Guo. Python Tutor website. http://pythontutor.com/, January 2010.

[46] Philip J. Guo. Online Python Tutor: Embeddable web-based program visualization for CS education. In *Proceedings of the 44th ACM Technical Symposium on Computer Science Education*, SIGCSE '13, pages 579–584, New York, NY, USA, 2013. ACM.

[47] Philip J. Guo. Codeopticon: Real-time, one-to-many human tutoring for computer programming. In *Proceedings of the 28th Annual ACM Symposium on User Interface Software and Technology*, UIST '15, pages 599–608, New York, NY, USA, 2015. ACM.

[48] Philip J. Guo. Older adults learning computer programming: Motivations, frustrations, and design opportunities. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, CHI '17, pages 7070–7083, New York, NY, USA, 2017. ACM.

[49] Philip J. Guo. Non-native english speakers learning computer programming: Barriers, desires, and design opportunities. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, CHI '18, pages 396:1–396:14, New York, NY, USA, 2018. ACM.

[50] Philip J. Guo, Jeffery White, and Renan Zanelatto. Codechella: Multi-user program visualizations for real-time tutoring and collaborative learning. In *Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, VL/HCC '15, pages 79–87, Oct 2015.

[51] Mark Guzdial. Limitations of MOOCs for Computing Education - addressing our needs: MOOCs and technology to advance learning and learning research (ubiquity symposium). *Ubiquity*, 2014(July):1:1–1:9, July 2014.

[52] Mark Guzdial. *Learner-Centered Design of Computing Education: Research on Computing for Everyone*. Morgan & Claypool, 2015.

[53] Mark Guzdial, Barbara Ericson, Tom Mcklin, and Shelly Engelman. Georgia computes! an intervention in a us state, with formal and informal education in a policy context. *Trans. Comput. Educ.*, 14(2):13:1–13:29, June 2014.

[54] Björn Hartmann, Daniel MacDougall, Joel Brandt, and Scott R. Klemmer. What would other programmers do: Suggesting solutions to error messages. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '10, pages 1019–1028, New York, NY, USA, 2010. ACM.

[55] Andrew Head, Elena Glassman, Gustavo Soares, Ryo Suzuki, Lucas Figueredo, Loris D'Antoni, and Björn Hartmann. Writing reusable code feedback at scale with mixed-initiative program synthesis. In *Proceedings of the Fourth (2017) ACM Conference on Learning @ Scale*, L@S '17, pages 89–98, New York, NY, USA, 2017. ACM.

[56] Neil T. Heffernan and Cristina Lindquist Heffernan. The assistments ecosystem: Building a platform that brings scientists and teachers together for minimally invasive research on human learning and teaching. *International Journal of Artificial Intelligence in Education*, 24(4):470–497, Dec 2014.

[57] Jonathan Huang, Anirban Dasgupta, Arpita Ghosh, Jane Manning, and Marc Sanders. Superposter behavior in MOOC forums. In *Proceedings of the First ACM Conference on Learning @ Scale Conference*, L@S '14, pages 117–126, New York, NY, USA, 2014. ACM.

[58] T. A. Jackson and D. J. R. Evans. Can medical students teach? a near-peer-led teaching program for year 1 students. *Advances in Physiology Education*, 36(3):192–196, 2012.

[59] Connie Juel. Cross-age tutoring between student athletes and at-risk children. *The Reading Teacher*, 45(3):178–186, 1991.

[60] Hyeonsu Kang and Philip J. Guo. Omnicode: A novice-oriented live programming environment with always-on run-time value visualizations. In *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology*, UIST '17, pages 737–745, New York, NY, USA, 2017. ACM.

[61] Kandarp Khandwala and Philip J. Guo. Codemotion: Expanding the design space of learner interactions with computer programming tutorial videos. In *Proceedings of the Fifth Annual ACM Conference on Learning at Scale*, L@S '18, pages 57:1–57:10, New York, NY, USA, 2018. ACM.

[62] Miryung Kim, Thomas Zimmermann, Robert DeLine, and Andrew Begel. The emerging role of data scientists on software development teams. In *Proceedings of the 38th International Conference on Software Engineering*, ICSE '16, pages 96–107, New York, NY, USA, 2016. ACM.

[63] Paul A. Kirschner and Jeroen J.G. van Merrienboer. Do learners really know best? urban legends in education. *Educational Psychologist*, 48(3):169–183, 2013.

[64] Thomas D. LaToza, W. Ben Towne, Christian M. Adriano, and André van der Hoek. Microtask programming: Building software with a crowd. In *Proceedings of the 27th Annual ACM Symposium on User Interface Software and Technology*, UIST '14, pages 43–54, New York, NY, USA, 2014. ACM.

[65] Ron Mace. What is universal design. *The Center for Universal Design at North Carolina State University* `http://www.udinstitute.org/principles.php`, 1997.

[66] Sui Fai John Mak, Roy Williams, and Jenny Mackness. Blogs and forums as communication and learning tools in a MOOC. In *Proceedings of the 7th International Conference on Networked Learning*, pages 275–284, 2010.

[67] Jane Margolis. *Stuck in the Shallow End: Education, Race, and Computing*. The MIT Press, 2008.

[68] Jane Margolis and Allan Fisher. *Unlocking the Clubhouse: Women in Computing*. The MIT Press, 2003.

[69] Djokic-Petrovic Marija, Pritchard David, Ivanovic Milos, and Cvjetkovic Vladimir. Imi python: Upgraded cs circles web-based python course. *Computer Applications in Engineering Education*, 24(3):464–480, 2016.

[70] Joseph Bahman Moghadam, Rohan Roy Choudhury, HeZheng Yin, and Armando Fox. Autostyle: Toward coding style feedback at scale. In *Proceedings of the Second (2015) ACM Conference on Learning @ Scale*, L@S '15, pages 261–266, New York, NY, USA, 2015. ACM.

[71] Alok Mysore and Philip J. Guo. Torta: Generating mixed-media gui and command-line app tutorials using operating-system-wide activity tracing. In *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology*, UIST '17, pages 703–714, New York, NY, USA, 2017. ACM.

[72] Alok Mysore and Philip J. Guo. Porta: Profiling software tutorials using operating-system-wide activity tracing. In *Proceedings of the 31st Annual ACM Symposium on User Interface Software and Technology*, UIST '18, New York, NY, USA, 2018. ACM.

[73] Adam Palin. Moocs: Young students from developing countries are still in the minority. *Financial Times*, 2014.

[74] Annie Murphy Paul. Bill Gates Is an Autodidact. You're Probably Not. Ed tech promoters need to understand how most of us learn. *Slate*, July 2014.

[75] Cynthia A. Rohrbeck, Marika D. Ginsburg-Block, John W. Fantuzzo, and Traci R. Miller. Peer-assisted learning interventions with elementary school students: A meta-analytic review. *Journal of Educational Psychology*, 95(2):240–257, 2003.

[76] Reudismam Rolim, Gustavo Soares, Loris D'Antoni, Oleksandr Polozov, Sumit Gulwani, Rohit Gheyi, Ryo Suzuki, and Björn Hartmann. Learning syntactic program transformations from examples. In *Proceedings of the 39th International Conference on Software Engineering*, ICSE '17, pages 404–415, Piscataway, NJ, USA, 2017. IEEE Press.

[77] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Pearson Education, 2 edition, 2003.

[78] Christopher Scaffidi. Counts and earnings of end-user developers – `https://www.linkedin.com/pulse/counts-earnings-end-user-developers-chris-scaffidi/`. September 2017.

[79] Christopher Scaffidi, Mary Shaw, and Brad Myers. Estimating the numbers of end users and end user programmers. In *Proceedings of the 2005 IEEE Symposium on Visual Languages and Human-Centric Computing*, VLHCC '05, pages 207–214, Washington, DC, USA, 2005. IEEE Computer Society.

[80] Eric L. Seidel, Ranjit Jhala, and Westley Weimer. Dynamic witnesses for static type errors (or, ill-typed programs usually go wrong). *Journal of Functional Programming*, 28:e13, 2018.

[81] Stephanie Ann Siler and Kurt VanLehn. Learning, interactional, and motivational outcomes in one-to-one synchronous computer-mediated versus face-to-face tutoring. *International Journal of Artificial Intelligence in Education*, 19(1):73–102, January 2009.

[82] Rishabh Singh, Sumit Gulwani, and Armando Solar-Lezama. Automated feedback generation for introductory programming assignments. In *Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '13, pages 15–26, New York, NY, USA, 2013. ACM.

[83] Juha Sorva. personal communication, January 2014.

[84] Juha Sorva, Ville Karavirta, and Lauri Malmi. A review of generic program visualization systems for introductory programming education. *ACM Transactions on Computing Education*, 13(4):15:1–15:64, November 2013.

[85] Andreas Stefik and Susanna Siebert. An empirical investigation into programming language syntax. *Trans. Comput. Educ.*, 13(4):19:1–19:40, November 2013.

[86] Ryo Suzuki, Gustavo Soares, Andrew Head, Elena Glassman, Ruan Reis, Melina Mongiovi, Loris D'Antoni, and Björn Hartmann. Tracediff: Debugging unexpected code behavior using trace divergences. In *Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, VL/HCC '17, pages 107–115, Oct 2017.

[87] Terry Tang, Scott Rixner, and Joe Warren. An environment for learning interactive programming. In *Proceedings of the 45th ACM Technical Symposium on Computer Science Education*, SIGCSE '14, pages 671–676, New York, NY, USA, 2014. ACM.

[88] Olle ten Cate, Irene van de Vorst, and Sjoukje van den Broek. Academic achievement of students tutored by near-peers. *International Journal of Medical Education*, 3:6–13, 2012.

[89] Allison Elliott Tew and Mark Guzdial. The fcs1: A language independent assessment of cs1 knowledge. In *Proceedings of the 42Nd ACM Technical Symposium on Computer Science Education*, SIGCSE '11, pages 111–116, New York, NY, USA, 2011. ACM.

[90] Kyle Thayer, Philip J. Guo, and Katharina Reinecke. The impact of culture on learner behavior in visual debuggers. In *Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, VL/HCC '18, 2018.

[91] Kurt VanLehn. The relative effectiveness of human tutoring, intelligent tutoring systems, and other tutoring systems. *Educational Psychologist*, 46(4):197–221, 2011.

[92] Kurt Vanlehn, Collin Lynch, Kay Schulze, Joel A. Shapiro, Robert Shelby, Linwood Taylor, Don Treacy, Anders Weinstein, and Mary Wintersgill. The andes physics tutoring system: Lessons learned. *Int. J. Artif. Intell. Ed.*, 15(3):147–204, August 2005.

[93] April Y. Wang, Ryan Mitts, Philip J. Guo, and Parmit K. Chilana. Mismatch of expectations: How modern learning resources fail conversational programmers. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, CHI '18, pages 511:1–511:13, New York, NY, USA, 2018. ACM.

[94] Jeremy Warner, John Doorenbos, Bradley N. Miller, and Philip J. Guo. How high school, college, and online students differentially engage with an interactive digital textbook. In *Proceedings of the International Conference on Educational Data Mining*, EDM '15, 2015.

[95] Jeremy Warner and Philip J. Guo. Codepilot: Scaffolding end-to-end collaborative software development for novice programmers. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, CHI '17, pages 1136–1141, New York, NY, USA, 2017. ACM.

[96] Jeremy Warner and Philip J. Guo. Hack.edu: Examining how college hackathons are perceived by student attendees and non-attendees. In *Proceedings of the 2017 ACM Conference on International Computing Education Research*, ICER '17, pages 254–262, New York, NY, USA, 2017. ACM.

[97] Sarah Weir, Juho Kim, Krzysztof Z. Gajos, and Robert C. Miller. Learnersourcing subgoal labels for how-to videos. In *Proceedings of the 18th ACM Conference on Computer Supported Cooperative Work & Social Computing*, CSCW '15, pages 405–416, New York, NY, USA, 2015. ACM.

[98] Joseph Weizenbaum. Eliza—a computer program for the study of natural language communication between man and machine. *Commun. ACM*, 9(1):36–45, January 1966.

[99] Andreas Zeller. *Why Programs Fail: A Guide to Systematic Debugging*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2005.

[100] Xiong Zhang and Philip J. Guo. Ds.js: Turn any webpage into an example-centric live programming environment for learning data science. In *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology*, UIST '17, pages 691–702, New York, NY, USA, 2017. ACM.

[101] Xiong Zhang and Philip J. Guo. Fusion: Opportunistic web prototyping with ui mashups. In *Proceedings of the 31st Annual ACM Symposium on User Interface Software and Technology*, UIST '18, New York, NY, USA, 2018. ACM.

[102] Joyce Zhu, Jeremy Warner, Mitchell Gordon, Jeffery White, Renan Zanelatto, and Philip J. Guo. Toward a domain-specific visual discussion forum for learning computer programming: An empirical study of a popular mooc forum. In *Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, VL/HCC '15, pages 101–109, Oct 2015.

[103] Daniel Zingaro, Yuliya Cherenkova, Olessia Karpova, and Andrew Petersen. Facilitating code-writing in pi classes. In *Proceeding of the 44th ACM Technical Symposium on Computer Science Education*, SIGCSE '13, pages 585–590, New York, NY, USA, 2013. ACM.