Teaching Statement

Rahul Yedida (Applicant for Fall 2024)

I am interested in and qualified to teach programming and machine learning topics at both the undergraduate and graduate levels, as well as software engineering. In the classroom, I aim to develop an inclusive, rigorous environment that focuses on developing strong fundamentals that students can apply and transfer to other subjects and programming languages. Throughout my studies, I have appreciated machine learning better when I understood the theoretical principles underpinning the algorithms, and programming when I learned the reasons for various design choices in programming languages and commonly accepted programming stylistic standards. As an educator, my goal is to instill those fundamentals into my students so that they are well-equipped in the career paths that they choose.

Teaching experience:

At NC State University, I have TA'd for CSC 230 (C and Software Tools, an undergraduate class focusing on teaching C, basic C++, and the use of `gdb`), CSC 510 (Software Engineering, a graduate-level course on software development best practices and the responsible use of AI in software engineering), and CSC 591/791 (Automated Software Engineering, a graduate-level course on developing ML tools to improve software developer experiences). In total, I was a TA for 927 students.

In CSC 230, which was the first course I was a TA for, I helped assess the difficulty of each assignment and project by developing solutions independently and verifying them with the other TA and the professor. Because there were only 159 students in the class, with some showing up regularly to office hours, I was able to watch them grow and learn from prior mistakes. To assist them, in their feedback, I made sure to be encouraging and tell them *why* something was incorrect. Being an early TA experience, I developed a sense of the kind of teacher I wanted to be. For example, in a fill-in-the-blanks problem for a C program, for a blank where the name of a C header file was expected, a student instead wrote (paraphrasing), "*I don't know what the name of the header file is, but we need it to access this specific function*". I urged the professor to consider partial credits since the student had clearly understood the role that that header had played. I also volunteered to teach two classes on C++, where I introduced the students to RAII[1] and the standard library classes that use the idiom. At the end of the semester, a student emailed me stating, "*you're by far the best TA I've had*".

CSC 510 (graduate-level Software Engineering) is a large class, with about 250 students on average, and I was a TA for it twice under my advisor. These experiences helped me develop strategies to handle such large classes, and to be precise with communication, both with the other instructors, and with the class. For example, in the first time (in 2022), to avoid issues with group formation, I proposed to let students either form their own groups (of 5), declare a partial group that would be merged with another, or ask to be grouped randomly. We then asked the students to fill out a Google Form stating their preferences, and I wrote a script that would automatically form groups while respecting as many groups' wishes as possible. After some bug-fixing, this script turned out to be an effective method to form groups and significantly reduced the instructors' load in finalizing project groups. As we got more experience with the subject, we were able to improve our communication about expectations and standards for course projects, which were open-ended. These standards involved project structure, the use of automated tools (linters, CI/CD, etc.), and most recently, that API keys should not be made public[2].

TA'ing for CSC 591/791 (in Spring 2023) was fun because the class is an elective that only students interested in the subject take; the distinction is that 791 is usually reserved for Ph.D. students, while 591 is open to all graduate students. Once again, the smaller class size (97 in 2023) allowed me to focus on individual students' growth better. In particular, I was interested

[1] Resource Acquisition Is Initialization, a C++ idiom that involves wrapping dynamically allocated resources in classes, acquiring the resource in the constructor, and releasing it in the destructor.

[2] We, too, were appalled that this was not obvious.

in the students' course project reports, where they would extend concepts learned from the class to a specific dataset based on a literature review. Recognizing the effort required to produce such a report, it was important to me that I also spent the time to write detailed feedback for these reports. While many of the students did come to office hours occasionally for assistance with the project, it was rewarding as an instructor to see the level of independence in these reports, especially from the Ph.D. students, some of who additionally used ideas from their own research to approach the problem in novel ways.

Teaching philosophy:

I believe in the value of strong fundamentals, especially in undergraduate courses. When students are shown *why* something works and *how* a certain approach was arrived at, I believe they are better equipped to handle unseen problems later on. This is especially true for programming and machine learning, where instead of the focus being on using clever library solutions to problems, the low-level details of the solutions set students up for success. Of course, this does not mean the material has to be dry: for example, one of my favorite textbooks is Steven Strogatz's book on nonlinear dynamics, which does an extraordinary job of explaining concepts while also discussing practical applications. To me, this bottom-up approach, where higher-level concepts build on lower-level ones, better equips students to understand more abstract ideas, and necessitates the lower-level concepts to be firm, which is one of my core teaching philosophies. We must strive towards undergraduate courses that emphasize fundamentals and spend more time on them; this "slower" pace will save time later on when concepts building on these are introduced.

At the graduate level, my teaching philosophy shifts to deeper and abstract reasoning: it is important that graduate students are exposed to the theory that hitherto was taken for granted and that concepts are taught at a more abstract level so that students are equipped to make connections between seemingly unrelated ideas. For example, at the undergraduate level, machine learning is taught with the assumption that the space is $\mathbb{R}^d$; at the graduate level, it is important for students to realize the "nice" properties of that space such as completeness and separability, that not every space has those properties, and what that implies.

A summary of my philosophy, then, is that undergraduate courses, especially those catered to freshmen and sophomores, must spend time developing strong, transferable fundamentals. This additional time spent allows us as educators to accelerate their understanding of more complex ideas later on and sets students up for success in industry or academia.

Teaching interests:

I particularly enjoy teaching core programming courses that expose the workings of computers at the lower undergraduate level. I believe that C is an excellent first language that introduces the inner workings of computer systems without hiding them behind syntactic sugar. Moreover, C is a relatively small language, which bounds the language-specific learning students need; this allows more room for hardware-level expositions to computer science and sets them up for classes on computer architecture and higher-level programming languages.

At the upper undergraduate level, I am interested in teaching courses on machine learning. It is important here to introduce students to various learning algorithms and their theoretical underpinnings without worrying about the technicalities that arise, such as convergence guarantees of gradient descent or learning in spaces outside $\mathbb{R}^d$. Programming assignments at this level would disallow libraries such as `sklearn`, and ask students to implement from scratch, the algorithms that we had discussed. I also want to teach courses on software engineering, with some emphasis on design. This course, while teaching students about standard programming practices, would also urge students to form their own opinions about standards, patterns, and language design.

At the graduate level, I would be interested in teaching two distinct kinds of courses. One would focus on theoretical machine learning and delve deep into convergence guarantees, generalization, and spaces outside $\mathbb{R}^d$; assignments in this course would primarily be proofs. I would also be interested in teaching a seminar-style, research-focused course that illuminates advances in the field and ends with open problems. Striking a balance between depth and breadth here is important, and I think that will be an interesting challenge for me as a professor.