Teaching Statement

Akshay Narayan

I believe teaching, interpreted broadly to include not only formal classes but also mentorship, scientific communication, and interdisciplinary collaboration, is a crucial part of practicing computer science research. Especially given the impact my mentors have had on my research career, I know firsthand the positive impact a community of mentors can have.

Teaching Philosophy. I argue that teaching systems courses should include not only technical instruction on algorithms, protocols, and interfaces, but also *technical writing skills*, especially comparatively evaluating systems quantitatively and qualitatively. For example, in a computer networking class focusing on congestion control, I would pose a hypothetical situation to students with an application (say, a file transfer server running in AWS's us-east-1 datacenter) and ask them to write an essay arguing for the adoption of one of a list of congestion control algorithms using both quantitative evidence and qualitative analysis. This type of assignment is a valuable and under-used complement to traditional implementation-focused assignments that canonically focus on understanding *how* an algorithm works. Rather, understanding that each system and protocol comes with trade-offs that are appropriate in different scenarios is an important concept for computer scientists to learn.

Further, my research observes that networks increasingly provide specialized features, and I thus believe that teaching systems and networking should reflect this. While traditional architectural principles (layering, the end-to-end principle, etc) are important concepts for students to understand, it is also important to understand the way modern systems depart from the conventional model. To this end, project-based learning helps students understand how layering and abstractions are a core part of building modern applications. To become effective computer scientists, students today must learn how to decompose complex applications into individual sub-systems, and understand how to approach building each sub-system by composing available libraries and new code. Similarly, students should learn how to develop libraries and services that compose well with other components as parts of a larger application.

Teaching Plans. At the undergraduate level, I would teach an undergraduate networking course focusing on the Internet architecture, design decisions, and core protocols. At the graduate level in networking, I would teach two classes: one on bandwidth allocation systems, covering congestion control, adaptive bitrate algorithms, traffic engineering, virtual networking, and capacity planning, and another on network stacks, covering implementing transport protocols efficiently, middleboxes, and datacenter networking. I would additionally be comfortable teaching classes on cloud computing and networked systems at both the undergraduate and graduate level. My undergraduate-level class on this topic would teach students how to design protocols and abstractions for both usability and performance (e.g., by collaboratively implementing a microservice application as a class project), how to instrument and analyze systems to determine their performance bottlenecks, how and when to use parallelism and concurrency primitives and implementation patterns in systems programs, and how to analyze tail performance separately from median performance. These are abstract technical skills that future application developers would readily put into practice.

Classroom Experience. I have been a TA for multiple computer networks classes between UC Berkeley, MIT, and IIT Madras. The Berkeley class was senior-undergraduate level, the MIT class was graduate level, and the IIT Madras class was a monthlong research seminar. At Berkeley, I taught weekly discussion sections for 20-30 students (in addition to traditional office hours), which included diving deeper into concepts discussed in lecture and working through a practice exercise to deepen students' understanding that the TAs took turns creating. For both the Berkeley and MIT classes, I helped develop and evaluate

homework assignments and exams. For the IIT Madras seminar, I was responsible for preparing students to give class presentations about each week's papers–ensuring that they understood each paper's core concepts–and helping lead classroom discussions.

Comparing and contrasting the two semester-long classroom teaching experiences was edifying for me: I came to appreciate the impact of small-scale learning, not only in office hours but also in discussion section. I also came to appreciate the value of hands-on "active" learning through course projects. These helped students grasp the underlying principles more strongly than lectures or paper reading and discussion. For example, for the graduate networking course at MIT I designed a project in which I used my research system, CCP, to expose students to actually implementing congestion control algorithms that could run on real networks. The key challenge in designing this assignment was ensuring that students could reason about the specific teaching objective–designing their own congestion control algorithm–in isolation, without dealing with other confounding factors. I also asked students to write a report describing their CCA and its tradeoffs. I believe writing and running real code and seeing its impact on the instrumentation I designed gave students a better understanding of how congestion control works than simply listening to the abstact models we must present in lectures.

Mentorship. There are four types of mentorship I believe are important and have engaged in. The first is traditional long-term mentoring of junior researchers. Through my experience mentoring multiple such individuals, I have found that the best policy is to follow my colleague's interests to ensure they are passionate about their work. For example, one student I worked with from her third year of her undergraduate degree until she finished her MEng degree initially started by working on a well-defined small project: building an eBPF-based datapath for CCP. Over time, we realized together that this project (which required significant work with the internals of the Linux networking stack) was not best aligned with her interests, and I worked with her to find a new project with another student in the group, and joined that project to continue to advise her progress. In the end, her master's thesis focused on measuring the quality of drones' cellular network connections. Working with this colleague gave me valuable new exposure to a new area of research I had little prior experience with. Ultimately, this student joined a quantitative trading firm.

While it is important to follow a mentee's interests, it is important to teach scientific rigor. For example, when the onset of the pandemic interrupted our experiments, my mentee's inclination was to end the project with the data we had already collected, but I convinced her that changing the focus of the project slightly would allow us to complete it without compromising the thesis's results. For another graudate student mentee struggling to make progress on a different project, I emphasized that it is important to understand how each experiment fits into the larger goal of a project.

The second form of mentorship I have enjoyed is cross-disciplinary collaboration. For example, I have contributed to a project led by a junior graduate student from the programming languages community which focuses on synthesizing congestion control algorithms. I enjoyed mentoring this colleague and teaching congestion control concepts, and similarly learning about state-of-the-art program synthesis techniques from her. I will encourage my own students to pursue this form of collaboration.

The third form of mentorship is outreach within the academic community, which targets not only junior researchers but everyone in the field. In my case, after winning the EuroSys best artifact award I took the opportunity to publish a post on the SIGOPS blog detailing how to create great research artifacts.

Finally, I believe short-form mentorship can help students become accustomed to implicit expectations present in our field. To this end I have participated in and helped run MIT EECS's student-led Graduate Application Assistance Program (GAAP), in which applicants from under-represented groups can receive one-on-one feedback on their graduate school application package. Throughout the semesterlong program, I taught my mentee how to tell a coherent story about her research experience. She eventually joined a top graduate program in EECS. I joined the program's executive team in its second year, and it has since flourished into a well-regarded source of graduate application feedback in EECS.