Jonathan Frankle Teaching Statement

Teaching Philosophy

My favorite teaching experiences are those where I have provided students with their first exposure to an entirely new topic. During my graduate career, this has ranged from teaching hundreds of students to write their first line of Python to introducing a handful of undergraduates and early-stage graduate students to my research area and the fundamentals of scientific inquiry. Doing so is both a privilege and a responsibility. If my enthusiasm is sufficiently contagious, my teaching could alter the course of a student's career. This is also where the stakes are highest: if I leave a student underprepared or without an adequate foundation, I may hinder their progress even if I have kindled their interest.

Teaching in these contexts begins with empathy: for where each individual student is coming from, where they hope to go, and the unique strengths and apprehensions they bring to the learning process. It also requires pursuing clear teaching goals while **customizing the approach for the specific audience**. For example, teaching introductory computer science to aspiring engineers (as a master's student at Princeton) and creating a similar class for aspiring lawyers (as a fellow at Georgetown Law) required me to address precisely the same material in radically different ways to accommodate different preparation and inhibitions. In research, diversity of backgrounds and goals demands **a unique plan for every student.**

In all cases, students will struggle at times; my job as a teacher is to ensure that struggles are productive opportunities for growth and that support is available to help students and avoid disillusionment. I have found that **frequent, low-stakes milestones and assessments** make it possible for me and the students to track their individual progress and identify trouble spots while they are easily addressable. As a partner in this process, I need to track my own progress: I have found **collecting feedback from students on a frequent (e.g., weekly) basis** to be a valuable way to calibrate and course correct.

In an era of exploding computer science enrollments and remote teaching, I have experience scaling these strategies to classes numbering in the dozens or hundreds. Even when courses become too large for the instructor to get to know each student personally, students should always have a specific point of contact (usually a specific TA) who is equally dedicated to their success. I have experience both serving in that capacity and training TAs to do so.

Few experiences are more fulfilling than accompanying a student on the first steps of an intellectual journey and, years later, finding out that they began a PhD, created a new area of research, started a company, or found ways to integrate the material they learned into an entirely different career path. In my mind, this means understanding what success means to each individual student and equipping them to accomplish it.

Teaching Approach

At its heart, I see teaching as the act of organizing information so that students are able to process and synthesize it in real-time. Although each lecture must be an engaging and well-rehearsed performance in its own right, the bulk of the work takes place when writing the underlying script: assessing the dependencies of the content to be presented, structuring the content in the proper order, organizing the information into intuitive abstractions, and motivating the endeavor with compelling, practical examples. As Shafi Goldwasser, a mentor early in my PhD, once explained to me:

> *I've been teaching introductory cryptography since the 1980s, and we cover at least twice as much information now as we did then. This isn't because the students are getting smarter: it's because we've developed better abstractions that make it easier to reason about the material.*

I design my lectures to encourage critical thinking, active learning, and participation, even in the context of the large enrollments typical in introductory (and, often-times, upper-level) computer science classes. Each lecture begins with an open-ended warmup question designed to encourage discussion and debate while motivating the day's lesson (e.g., *Why does Python have types? Name one instance where you want an immutable value and one where you don't. Describe two ways of initializing a neural network that would undoubtedly lead to a bad final model.*)

During the lecture itself, I reinforce concepts with examples and exercises, assigning students to work on them in pairs and soliciting student input to help me complete the solutions. Encouraging students to willingly participate requires **creating a classroom culture from day one that enables students to take risks and lowers the stakes** so that students feel comfortable attempting to answer questions, even if they aren't absolutely certain about the answers. In this culture, interaction can help students to stay engaged in large lectures without creating anxiety.

I prefer **frequent, lower-stakes assessments** to give students opportunities to put each week's material into practice and track their progress. I view problem sets as an opportunity for additional teaching, and I structure them around applications I wish I had time to explore in lecture (e.g., an assignment to teach lists and for-loops with the conceit of finding hash-collisions, cracking passwords, and mining Bitcoin).

In upper-level classes, I see **end-of-semester projects as a valuable way to give students space for creativity**. I have found that these projects must be managed carefully to ensure students are able to take full advantage of the opportunity (rather than, for example, wandering for many weeks and rushing something unsatisfying at the last minute). This requires support for developing ideas, clear milestones, and frequent feedback on progress. In short, **it requires a willingness to dedicate substantial teaching resources to the effort**, far more than for an equivalent number of structured problem sets.

Teaching Experience

Dating back to my earliest days as an undergraduate at Princeton, I have sought out ways **to share my enthusiasm for computer science through teaching**. I took full advantage of the opportunities available to undergraduates, serving as a grader, lab TA, and mentor to students in introductory classes (and receiving a departmental award for my efforts).

I remained at Princeton for a fourth year as a master's student, which allowed me to enter the classroom in a formal way. I helped to provide students with their first introduction to computer security as a TA in an upper-level course on the subject with more than 100 students. I held office hours, contributed to assignments and tests, and led a week of well-attended review sessions in the leadup to the final. My students rewarded my efforts, lobbying the undergraduate Engineering Council to present me with **one of just three awards for graduate teaching that term across the entire school of engineering**. The following term, I taught two sections ("precepts") of the introductory computer science course, reviewing and enhancing the lecture material for four hours each week. To let my students speak for themselves:

> *"Jonathan was so enthusiastic!" "Jonathan is the best preceptor by far." "Precept was super helpful!!!" "Precept was the saving grace." "Many of my friends, who attended other precepts, repeatedly told me that Jonathan is the best preceptor." "You are the man." "Absolute best part of the class." "Literally a life-saver." "THE BEST EVER!"*

The computer science department agreed, giving me another departmental teaching award for my efforts.

After my master's, I took a gap year to work in technology policy at Georgetown Law. My mentor, Prof. Paul Ohm, was a law professor with a degree in computer science. He had always been eager to teach law students to code with the intention of enabling them to engage more closely with technical topics in law and policy. Together, we developed a pilot *Programming for Lawyers* course that covered essentially the same content as the introductory course I had taught at Princeton **but for an audience with entirely different goals, preparation, and apprehensions**. For example, a frequent joke around the law center was that students and faculty "had become lawyers because they couldn't do math." In practice, they were excellent programmers who needed help overcoming these internalized (but unfounded) inhibitions.

Together, we developed an entirely new curriculum from the ground up, focused on data types (e.g., strings, text, PDFs), examples (e.g., technology policy and "memos from the partner"), and applications (e.g., web scraping and text processing) that would resonate with this audience. After a pilot semester with 17 students, the course has grown to an annual enrollment of 50-75 students at Georgetown with seven other law schools offering versions of the class. Although I can no longer teach it on a regular basis from MIT, I flew to DC every Tuesday in the spring of 2019 to teach in a visiting capacity. The course now includes a

dozen law student TAs (who simultaneously take an intermediate *Programming for Lawyers* class), small lab sections alongside lecture, and a textbook that Prof. Ohm and I are working to publish.

The biggest lessons from this experience have been about **the work necessary to develop a new class**. Assignments, lectures, and textbook chapters each require dozens of hours to create and years to perfect based on many rounds of student feedback. It has taken five years of iterative refinement to get the course to the point where it can be taught smoothly each year with minimal revisions. **As a faculty member, I would be eager to make a similar long-term commitment to creating or adopting a class and improving it over many years, having experienced the rewards and impact of doing so first-hand.**

Advising Experience and Approach

My advising philosophy is similar to my teaching philosophy: although I have a high-level curriculum and a set of teaching objectives in mind (namely, teaching science and research practice), the manner in which that manifests has to be **customized for the strengths, goals, and interests of each individual student**. At MIT, I have formally advised two undergraduate students and three master's students. I have also greatly enjoyed working with many early-stage PhD students as a mentoring "second author."

Early on, students often struggle with creating structure for themselves: forming an answerable research question in an open-ended space and managing their time to make progress toward answering that question effectively. I have found it valuable to encourage students to produce written work-products from the very beginning, for example developing (and refining) research questions, formulating hypotheses, specifying experiments to evaluate them, and tracking results. Doing so helps students to subdivide projects into smaller, more manageable pieces and reinforces scientific best-practices.

However, these general practices must be tailored to the individual student. Some students have come into the process with a clear question in mind but have struggled to answer it. Others are effective at answering questions once specified but need help finding interesting, answerable questions. I often begin working with a student by reading several papers together in an area, which establishes background knowledge and gives a student room to discover topics and questions that speak to them.

I prefer a hands-on approach to advising, meeting with students frequently and keeping my (virtual) door open whenever possible. Early in my PhD, I personally experienced many times where I felt like I was struggling and slipping through the cracks. I strive to ensure that none of my mentees ever feel this way. I have found **frequent check-ins** (even if they are only a few minutes) to surface concerns sooner, help students get back on track more quickly when they are stuck, and reinforce that they should not hesitate to ask for support whenever needed. As students get experience and mature, they typically become more independent and these check-ins naturally become shorter and less frequent.

Teaching Interests

I am interested in teaching a wide variety of classes. I have a particular affinity for introductory courses and, with that in mind, I would be eager to teach courses in the introductory computer science and machine learning curricula. I would be especially keen to teach a general introduction to deep learning, including overviewing exciting topics of contemporary interest in such a fast-moving field. In addition, my research has spanned multiple application areas of deep learning and, although I am not a domain specialist in either of these specific topics, I would be interested in getting involved in courses on computer vision and NLP.

At the graduate level, I am especially interested in developing a course on the foundations of deep learning as I see them through my empirical, scientific lens. A key aspect of this course would be codifying and teaching empirical research methods for machine learning. As empiricism becomes an increasingly important tool for studying complex machine learning systems, it is essential that we establish standards for rigor and educate the next generation of researchers accordingly.