Teaching Statement

Teaching Students to Bridge Foundational Principles and Real-World Practice

Inho Choi

Bringing research insights into the classroom and classroom clarity back into research creates a powerful synergy. My own understanding of computing principles deepened most when I encountered them firsthand while building and breaking systems. Observing how abstractions like concurrency, consistency, and performance play out in a running environment taught me far more than any textbook. I want my students to develop that same depth through experiential learning. This philosophy shapes how I teach students at every level.

1 Teaching Approach

I have been drawn to teaching since before my PhD, and I actively sought out classroom opportunities from my first semester. I have served as a teaching assistant for five semesters: *Programming Methodology*, *Introduction to Operating Systems*, *Distributed Systems* (in two semesters), and *Cryptography Theory and Practice*. My roles spanned tutorials, office hours, and design and grading of exam questions and programming assignments. These experiences have shaped a core principle that guides how I design and teach courses.

Build conceptual intuition through concrete experience. I believe abstractions stick best when students first encounter the concrete problems those concepts were built to solve. Whether introducing low-level hardware constraints or high-level software abstractions, I anchor computer science principles to observable, real-world behaviors rather than static definitions. While teaching operating systems, for instance, I introduced concurrency by running a non-deterministic program whose output changed unpredictably with each run, challenging students to diagnose the root cause before explaining the formal mechanisms. Similarly, in distributed systems, I anchored abstract consensus protocols to real-world applications like Google Drive and guided students with tracing execution traces inside a datacenter simulator during injected network failures. By confronting these anomalies firsthand, students shifted from memorizing formulas to actively troubleshooting systems, developing a rigorous engineering intuition that serves as a foundation for next challenges.

2 Teaching Interests

My teaching interests center on undergraduate and graduate courses in operating systems, distributed systems, and computer networks, all of which I have either TA'd or worked in directly through my research. I also look forward to developing three new graduate seminars:

- *Programmable Network Hardware in Modern Datacenters* — for senior undergraduates and early PhD students; a seminar bridging classical computing foundations with recent advances in networking, paired with practical hardware-or-simulator exercises.
- *ML-Native Operating Systems* — graduate seminar bridging systems and ML; review of learned-systems paper, final mini-project integrating a learned component into an OS dataplane.
- *AI Infrastructure Across Heterogeneous Accelerators* — graduate course for students entering AI systems research; data movement and coordination across heterogeneous devices such as GPUs, programmable hardware, and emerging accelerators, with a project profiling a real cluster workload.

At the undergraduate level, my teaching core covers introductory programming and systems courses. I also aim to bring the same teaching principles to other foundational courses such as data structures, algorithms, and systems programming. I would also integrate short modules from my own research into the later weeks of these courses, such as a kernel-bypass receive-side handler at the end of the OS course, or a small datacenter-network trace exercise in the networks course. The goal is to show undergraduates how foundational principles connect to both current systems research and the real-world systems they use every day.

3 Mentoring

Throughout my PhD, I have devoted significant time and care to mentoring, especially working closely with four research students at NUS. On Capybara, a project I led to publication at SIGCOMM '26 in collaboration with Microsoft Research, I guided a research assistant in implementing and debugging parts of the inter-server TCP migration mechanism. I also led two undergraduate researchers to identify critical optimization opportunities in Microsoft's networking OS, and mentored a Master's student whose thesis built a software-only version of my Hydra system (NSDI '23) for deployments without programmable hardware.

Systems research has a steep entry cost: unfamiliar codebases, non-deterministic bugs, and little visible progress early on. To lower this barrier, I structure initial milestones into small, accessible sub-problems, allowing each student to own a single functional component end-to-end before scaling up. One debugging session captures this approach: instead of directly diving into more engineering and tests, I guided my mentee to reconstruct the protocol's state machine on a whiteboard until they could predict the failure themselves.

I also maintained shared, interactive documents with each mentee to continuously discuss their research progress and any technical roadblocks they encounter. Rather than simply reviewing results, this ongoing dialogue encourages students to move past a surface-level "I tried X and it failed" into articulating why they expected a certain result and why it diverged. This practice naturally builds rigorous analytical habits in students while enabling me to ensure they stay on the right path, rather than just monitoring isolated outcomes. As they gain fluency, my role shifts from breaking problems down to guiding them to choose the next challenge, moving them toward independent research.

Beyond formal mentorship, I actively support junior PhD colleagues in their research, and have advised many students considering graduate studies with an honest perspective on both the rewards and costs of a PhD. One such student later enrolled in our doctoral program; we still meet frequently to discuss research directions and graduate life. These broader interactions have reinforced my conviction that true mentorship is about cultivating a student's long-term independence, helping them navigate not just immediate technical roadblocks but their broader development as professionals.

4 Looking Forward

What makes advanced computer science difficult to enter is rarely a lack of talent. Instead, it is an invisible barrier that turns away capable students of varied preparation and backgrounds before they discover they can belong. My experience teaching and mentoring students from diverse backgrounds has shown that every student, regardless of their initial preparation, can grow into successful contributors. I have learned that the key to unlocking this potential is breaking complex objectives into structured milestones, ensuring each student builds critical intuition through direct experiences and gains full ownership over a single component from the start. I will carry this approach into my own teaching: scaffolded early assignments, projects where every student owns a functional component of a real system early on, and office hours framed as a place to come before getting stuck rather than after. Establishing this early confidence grounds students at every level, empowering them to transition from passive observers to active creators who know that advancing a system is entirely within their own hands. Ultimately, I will approach teaching and mentoring with the same deliberate practice as my research, knowing that the truest measure of my success is the long-term independence of the students I mentor.