Summary. I mentored graduate and undergraduate students during my graduate and postdoctoral years, which deepened my appreciation for the challenge and reward of mentorship. The thrill of guiding them into new subject areas, helping them prepare for qualifying exams or understand core background materials. Ultimately the most satisfying moment, is when you watch them make the hidden connection between the interconnected knowledge of the courses or when they finally understand the new topic. I strive to help students achieve hands-on, real-world constraint-driven experience of theoretical topics. My philosophy is rooted in the belief that abstract concepts only gain meaning when paired with concrete, applied projects. By guiding students through system-level explorations, hands-on coding, debugging, and collaborative discussions, I aim to help them internalize theory through practice.

Teaching Philosophy. *My central belief is that knowledge is internalized only when students build and break real systems.* Teaching, to me, is not only about transmitting knowledge but also about ensuring it is received, internalized, and applied with clarity and purpose. I enjoy teaching computer security and core CS courses through hands-on, constraint-driven projects. Teaching is a collaborative and ever-evolving practice; I keep learning how to teach by experimenting with new methods and reflecting on what truly improves students' understanding. My goal is to develop an inclusive, well-equioped, state-of-the-art technology-driven classroom that builds confidence to help tackle the new ever-changing computer science landscape and also career-readiness (i.e., interview questions, LeetCode, System Design). I also aim to teach the fundamentals as well as the SOTA methods since this continual refinement is essential in security, where new and shifting paradigms reshape both defenses and attacks that either make old knowledge (i.e., signature-based defense) obsolete in light of new knowledge (new stealthy attacks) or where sometimes a simple defense (i.e., blocking Command and Control server) can stop a sophisticated multi-prong attack (i.e., Lockbit Ransomware [2]).

Table 1: Prior teaching experience. G: Graduate Coursework and UG: Undergraduate Coursework.

Course	**G/UG**
University of Texas at Dallas, Texas, USA	
Information Security	G
System Security & Malicious Code Analysis	G
Cybersecurity Attacks & Defenses Lab	G
Operating System Concepts	G/UG
Object-Oriented Analysis & Design	UG
Organization of Programming Languages	UG
University of Evansville, Indiana, USA	
Calculus II	UG
Calculus III	UG
Differential Equations	UG
Linear Algebra	UG

My goal is to help prepare students for academic or industry career: publications, internships, and roles at leading labs and companies. Students often feel time-pressured and unsure how a course advances their professional trajectory. I address this by making the "why" explicit and the "how" tangible. For example, in a graduate *Operating Systems* course, several students questioned the relevance of "classical" topics. I worked with the instructor to introduce a hands-on lab in which students implemented minimal OS components (i.e., JosOS [1]) and progressed from a monolithic kernel to a microkernel-style design with user-space services. Along the way, they saw how foundational ideas (e.g., interrupts, scheduling, synchronization) are the basic building blocks and how new considerations such as concurrency bugs, race conditions, isolation boundaries, and security policies, become more acute as designs modularize. The hands-on lab concretized the theoretical knowledge and showed how traditional knowledge evolved to its current state, which turned skepticism into curiosity and improved engagement in the course.

With the rise in Agentic Systems and more recently Agentic Browsers, I observed a gap in students' knowledge of core CS concepts, especially lower-level systems knowledge such as how programming languages interact with the hardware through compilation toolchains. One of the core security courses, *System Security and Malicious Code Analysis* expects fluency in skills such as program execution, binary instrumentation,

and assembly debugging as it aims to teach students to reverse-engineer programs and exploit vulnerabilities. Unfortunately, more than half of the student population lacked the necessary foundation knowledge. To close the gap, I ran two supplemental sessions: one in the afternoon for students and one in the evening, primarily for part-time professionals. I wanted to ensure everyone had an equal opportunity to develop their skills. Teaching this intensive session focused that on the lower-level assembly and program execution was demanding as I had to create targeted labs and lectures to cover the underlying topics that are pertinent to system security, but this taught me how to customize courses for different skill levels and to build a well-received course for a diverse audience. These sessions improved students' class performance, and many noted that the lessons learned directly translate to other CS classes.

Courses I Plan to Teach. I am eager to contribute across the security and systems curriculum. I can teach core CS courses such as Operating Systems, Networks, Machine Learning, and Software Engineering. In security, I look forward to teaching Information Security, Application Security, System Security & Malicious Code Analysis, Network Security and Cyber-Physical System (CPS) Security. I am also excited to offer advanced, research-driven seminars that connect students to modern research in the domain of Agentic AI, Trustworthy AI/ML, and ML for Cyber-Security.

Course Design. My courses are designed with a set of measurable outcomes: what students should be able to analyze, build, and explain; and then align lectures, labs, and assessments so that skills are introduced, reinforced, and demonstrated in context. Because backgrounds vary, I built in optional bridge modules and recitations, so that students can reinforce their background while assimilating new knowledge. Assessment will include labs that emphasize design decisions, correctness, test case creation and a capstone project. The coarse goal is where students can explain, defend, and deploy into production (or pitch their project to startups) what they have built, leaving them prepared for both advanced study and real-world engineering. It is futile to think that students will not use AI and agentic systems for their coursework. Therefore, it is critical that I teach them the risk of using these systems and how to use them responsible.

Student Mentorship. During my Ph.D. and postdoctoral years, I learned that mentorship is less about directing and more about traveling with students. I have worked with undergraduates taking their first research course, master's students balancing part-time internships and research deadlines, and Ph.D. students wrestling with open-ended problems. With each group, I tried to be the kind of mentor I wished I had when I was beginning my journey: present in the lab, available after hours, and willing to open a blank page and draft research questions, discuss ideas, potential blockers alongside them. The commitment I brought to those students is the commitment I will bring as faculty member: patient, detail-oriented, attentive, and anchored in the belief that people flourish when they feel seen, supported, and challenged.

I want to teach my students how academic research differs from coursework. In coursework, objectives are defined; in research, goals shift as evidence accumulates. That moving target can be frustrating, especially for students used to crisp specifications and fixed deadlines. My approach is to demystify the process: developing a hypothesis, surveying the current methods, developing procedure to gather evidence, and selecting or designing evaluation metrics. When the evidence changes, we update the plan together, with the lesson that uncertainty is not failure; it's the raw material of discovery.

To make research work sustainable, I want to invest time in developing communication skills, teamwork, and habits that prevent burnout. I will run weekly one-on-ones and small-group meetings with clear agendas: progress since last week, blockers, next experiments, and assistance required in the current stage. I will keep issue trackers and a shared milestone board so accountability is transparent and fair, while ensuring their mental and physical health are not compromised. As a faculty mentor, I want students to work on problems that interest and excite them. I want my students to see me as a collaborator who offers guidance, but most importantly, helps them become independent thinkers. My measure of success will not only be papers, but it will also be the growth in people. I want students to grow, find their voice, define their research direction, and carry rigorous, cutting-edge, reproducible research practices wherever they go.

Critical Thinking and Adversarial Mindset. The central theme in my teaching is training students to think like both system designers and adversaries. Whether we are implementing a network protocol, analyzing an algorithm, or reasoning about an ML-based defense, my goal is not simply that students can produce a working solution, but that they can articulate when it fails, why it fails, and how a dedicated adversary will exploit those failures. In systems-oriented courses (e.g., Operating Systems, Networks), I emphasize multi-component projects where no single LLM (or agent) can safely generate the entire solution in one shot. If the students tried using Agents, they would get an incomplete solution, so to make the solution work, the students would have to critically think about the design, the current error, and the mitigations. When they are doing these, they will inadvertently have to think critically about the material. LLMs may help with boilerplate code, but students must still design the architecture, articulate invariants, and argue about correctness and robustness.

For theory-heavy courses (e.g., Algorithms, Programming Languages, Malicious Code Analysis), I design problems that are not trivially answered by LLMs. Rather than asking for a standard proof or textbook-style derivation, I ask students to: construct counterexamples where a known algorithm or heuristic fails, characterize edge cases where a security guarantee no longer holds, or compare two formal models and explain what each one cannot express. Students might be asked to critique an LLM-generated proof or explanation, locate the subtle flaw, and repair the argument. This shifts the role of AI from "answer generator" to "fallible collaborator," and it incentivizes students to engage critically, which rewards deeper reasoning over pattern-matching rather than passively consuming.

Across both systems and theory courses, I treat failure analysis as a core learning outcome. Assignments routinely include a critical-thinking component: students describe one bug, misconception, edge case, or adversarial angle they discovered (in their own work or the LLM's output), explain why it arose, and state how they would detect or prevent it next time. Over time, students internalize that security, algorithms, and AI robustness all share a common skill: the disciplined habit of looking for edge cases, understanding threat models, and reasoning carefully under domain constraints. My goal is to train students who are not just proficient coders or theorem-solvers, but critical, adversarial thinkers prepared for a computing landscape where powerful but fallible AI systems are everywhere.

References

[1] M. Frans Kaashoek and Robert Morris. 6.828: Operating systems engineering — 2018 overview. https://pdos.csail.mit.edu/6.828/2018/overview.html, 2018. MIT CSAIL PDOS Group. Accessed: October 27, 2025.

[2] Microsoft Threat Intelligence. Stopping c2 communications in human-operated ransomware through network protection. https://www.microsoft.com/en-us/security/blog/2022/11/03/stopping-c2-communications-in-human-operated-ransomware-through-network-protection/, November 2022. Accessed: October 28, 2025.