CSE 713: Software Analysis and Applications
Spring 2026

Course credits: 1~3
Meeting time: Monday, 5:00 pm – 6:20 pm, January 21 – May 05
Location: Bell 138

Instructor: Dr. Haipeng Cai
Office locations:
- Physical - 345, Davis Hall
- Online - Zoom (id: 7862337400 or https://buffalo.zoom.us/my/hcaiub)

Email: haipengc@buffalo.edu
Office hours: Monday, 4:00 pm – 5:00 pm, or by appointments

Course Description

Modern software systems—from web services and mobile platforms to distributed microservices and AI-enabled applications—demand rigorous methods to ensure security, reliability, privacy, and performance. This seminar explores the cutting-edge foundations, techniques, and emerging research directions in software analysis: static, dynamic, hybrid, and AI-augmented approaches. Students will engage deeply with seminal papers and state-of-the-art research, including work at top venues such as ICSE, FSE, ISSTA, PLDI, OOPSLA, USENIX Security, S&P, and CCS.

The course is intentionally broad: we study not only *program* analysis but also system-level, cross-layer, and AI-integrated software analysis spanning the full lifecycle of software artifacts. Special emphasis is given to security and privacy applications, but we also investigate analysis for functional correctness, robustness, and performance optimization.

Students will learn both classical analysis techniques and how modern advancements (e.g., LLM-based analysis, agentic AI systems, hybrid symbolic-neural analysis, fuzzing with AI guidance) are reshaping software analysis research and practice.

Learning Objectives

By the end of the course, students will:

- Understand foundational and advanced techniques in software analysis, including their theoretical underpinnings and practical tradeoffs.
- Analyze and critique research papers, with emphasis on rigor, novelty, and empirical evaluation.

- Gain hands-on experience with modern analysis tools (e.g., static analyzers, fuzzers, symbolic executors, LLM-based analyzers).
- Connect analysis techniques to impactful applications in **software security, privacy**, **reliability**, and **performance engineering**.
- Develop and present a mini research project exploring an emerging direction in the field.

Main Topics & Modules

Below is a representative outline (topics may be adjusted to align with student interest and cutting-edge developments):

1. Foundations of Software and Program Analysis
   - Classical data-flow analysis (Reaching defs, liveness, alias analysis)
   - Abstract interpretation and fixpoint theory
   - Dependence analysis: control, data, and information-flow dependencies
   - Program slicing and summary-based analysis
   - Path-sensitivity vs. flow-sensitivity vs. context-sensitivity
   - Static vs. dynamic vs. hybrid analysis paradigms

2. Modern Software Analysis for Security & Privacy
   - Taint analysis and information-flow control
   - Vulnerability detection (buffer overflow, injection, concurrency bugs)
   - Memory safety and undefined behaviors
   - Mobile app security analysis (Android/iOS)
   - Privacy analysis and data-flow auditing
   - Patch analysis, root-cause analysis, and bad-patch detection

3. Dynamic Analysis and Execution-Based Techniques
   - Dynamic taint tracking
   - Dynamic symbolic execution & constraint solving
   - Program tracing, profiling, instrumentation (Pin, DynamoRIO, LLVM)
   - Fuzz testing: coverage-guided, grammar-based, concolic fuzzing
   - Differential testing & regression analysis
   - Runtime monitoring for safety & performance

4. AI-Enhanced and LLM-Driven Software Analysis (Contemporary Focus)
   - LLMs for static analysis (code comprehension, summarization, bug detection)
   - LLM-augmented symbolic execution & hybrid neural-symbolic analysis
   - Agentic AI systems for automated debugging, secure coding, and testing
   - AI-driven fuzzing techniques
   - Using code embeddings, graph neural networks (GNNs), tree/AST models
   - Vulnerability-aware prompting, chain-of-thought for program analysis
   - Risks of LLMs in software engineering (hallucination, security pitfalls)

5. Software Analysis for Performance, Reliability, and Testing

- Profiling & performance bottleneck detection
- Concurrency analysis and race detection
- Test generation and test oracles
- Coverage analysis and mutation testing
- Dependability analysis for distributed systems
- Fault localization, delta debugging, and automated repair

6. Real-World Systems & Cross-Layer Software Analysis

- Analysis for large-scale systems (microservices, serverless, cloud)
- Cross-layer dependence and information-flow tracking
- Binary analysis and reverse engineering
- Code search and representation learning
- Security patch analysis & exploit detection

Grading

The final course grade will be calculated using the following breakdown and be converted from numeric numbers to letter grades using the scale mapping as follows.

Breakdown *Scale mapping*

Coursework	%		Score	*Grade*	Score	*Grade*	Score	*Grade*
Participation	20%		>=93	*A*	[80,83)	*B-*	[66,70)	*D+*
Presentation	30%		[90,93)	*A-*	[77,80)	*C+*	[60,66)	*D*
Project	50%		[87,90)	*B+*	[73,77)	*C*	<60	*F*
All	100%		[83,87)	*B*	[70,73)	*C-*		

Course Project

A central component of this seminar is a semester-long **research-oriented course project**. The project is designed to immerse students in the practice of software analysis research, from problem formulation and literature grounding to technical execution and critical evaluation. Rather than following a fixed recipe or narrowly scoped specification, students are expected to engage with open-ended research questions and make principled design and evaluation decisions, similar to those encountered in real-world academic and industrial research.

The project may be conducted individually or in small groups, depending on the scope of the chosen topic and the students' background. Regardless of group size, each student is expected to make substantial intellectual and technical contributions. The goal is not merely to "build a tool," but to **reason about software analysis techniques**, understand their strengths and limitations, and evaluate them in a rigorous and reproducible manner.

Students will begin by identifying a topic aligned with the themes of the course—such as software security, privacy, correctness, reliability, or performance—and grounding their work in relevant literature. Topics may be drawn from a provided list of example directions or proposed independently by students, subject to instructor approval to ensure appropriate scope and rigor. Projects are expected to incorporate concepts discussed in the seminar and reflect contemporary developments, including but not limited to hybrid static–dynamic analysis, execution-based techniques, and AI-augmented software analysis.

The project progresses through a sequence of **milestone-based deliverables** that support steady progress and early feedback. These deliverables are designed to encourage careful problem definition, thoughtful methodology design, and empirical validation. For each milestone, students will submit written materials and, where applicable, code, experimental artifacts, or analysis scripts that demonstrate tangible progress. Feedback at each stage is intended to help students refine their direction, sharpen their technical contributions, and improve research clarity.

Evaluation of the project emphasizes **depth of understanding, technical soundness, and quality of analysis**, rather than raw system complexity or scale. A smaller, well-designed study with insightful analysis is preferred over an overly ambitious implementation that lacks rigor. Students are expected to reflect critically on their results, discuss limitations, and identify promising directions for future work.

The final project culminates in a written report and an in-class presentation. The report should be written in a research-style format, similar to a conference paper or focused survey, and should clearly articulate the problem motivation, technical approach, evaluation methodology, results, and insights. For students pursuing research careers, the project may serve as a foundation for a conference submission, thesis topic, or dissertation chapter.

Example Ways of Approaching the Project

- Reproducing and extending a recent research paper in software or program analysis
- Designing and evaluating a new analysis technique or system component

- Empirically comparing multiple software analysis tools or approaches on a common benchmark
- Building a prototype analysis tool for a security, privacy, or performance problem
- Conducting a focused survey and critical synthesis of an emerging research area

---

Submission and Collaboration Policy

Unless otherwise specified, all project deliverables are to be submitted electronically via Canvas by **11:59 pm on the posted due date**. Deliverables should be submitted as a single PDF document, optionally accompanied by a code archive or repository link when applicable. Clear instructions and expectations for each deliverable will be communicated well in advance of the deadline.

Late submissions are strongly discouraged, as timely feedback is critical for project success. Unless prior arrangements have been made with the instructor, late submissions may receive a penalty or, in severe cases, no credit. Students anticipating delays due to extenuating circumstances must notify the instructor **before** the deadline to discuss appropriate accommodations.

For group projects, all members will initially receive the same grade for each deliverable, subject to adjustment based on peer evaluation at the end of the semester. Peer evaluations are used to ensure fairness and accountability and may affect individual project grades.

Completion of all required project deliverables is mandatory. Missing a major deliverable without approved exceptions may result in an incomplete (“I”) grade, indicating that the course requirements have not been satisfied.

Late Submission Policy

Late penalty is a flat 10% deduction per day. Late assignments may be turned up to one week after the original due date. An advanced notice must be given to the instructor via email at least 24 hours before the deadline for a late submission. The instructor may allow for late submissions without penalty if extenuating cases are explained in the notice email sent to the instructor.

Communication

We will communicate announcements, assignments, lecture materials and other learning resources all on UB Learns. In particular, we will host off-class Q&A through

Piazza. UB Learns is also the portal to be used for project deliverables submission and grading. For questions on course materials, lectures, and course project milestones, contact the instructor on Piazza by sending posts instead of by emails, so as to facilitate communication. You have options for sending *private (anonymous)* posts. Make sure you **subscribe to each** of the forums there so that you won't be missing important information about the course logistics and extended lecture discussions initiated by questions raised by other students.

Participation

Class attendance is required at all lectures. Although lecture slides and other supplementary learning materials will be posted online, these materials as well as the suggested reading materials are only used as references by the instructor in developing the lectures. Thus, studying these materials serves the purpose of getting better prepared for attending in-class lectures, but would by no means substitute for class attendance---after all, this is seminar course, heavily relying on in-class particpation and active participation/critiques. Also, the course project requires each team member to be responsible and collaborative as well as to contribute equally; thus, missing lectures without justifiable reasons and then relying on other team members to catch up missed topics is not acceptable. You are also expected to participate in class discussions, which aids learning and provides valuable feedback on the lecture. If you know you will miss a lecture for a justifiable reason such as a university activity or a medical appointment, notify the instructor by email at least 12 hours before the lecture. While attendance will not be taken in every class, it will be sampled randomly at the discretion of the instructor. The basic participation credit that accounts for 5% of the final grade will be calculated using the sampled attendance records (in addition to active participation in classroom discussion).

In addition, students are expected to maintain a professional and respectful classroom environment, for which students are suggested to:

- silence personal electronics (non-disruptive ones may be used during class)
- arrive on time and attend the entire class session

Expected Effort

Beyond the time for lecture attendance, students in this class are expected to invest a minimum of 1-2 hours outside class for each lecture equivalent (or 2-4 hours per week), including the time for working on the course project.

Policy on the Use of LLMs/Generative AI

LLMs or any other forms of generative AI should *not be used for any graded coursework* (e.g., project deliverables, final report). For these assignments, what you submit must be absolutely your own work without involving anything from generative AI---you can use it for learning purposes outside your work on these assignments.

If there is evidence to suggest a substantial part of the assignment has been generated by generative AI, then it will be treated as a violation of Academic Integrity and necessary steps will be taken as per the academic integrity policy stated in the syllabus.

If you are in doubt of what is permitted and what is not, ask the instructor!

Academic Integrity

Academic integrity is critical to the learning process. It is your responsibility as a student to complete your work in an honest fashion, upholding the expectations your individual instructors have for you in this regard. The ultimate goal is to ensure that you learn the content in your courses in accordance with UB's academic integrity principles, regardless of whether instruction is in-person or remote. As an institution of higher learning, UB expects students to behave honestly and ethically at all times, especially when submitting work for evaluation in conjunction with any course or degree requirement. Thank you for upholding your own personal integrity and ensuring UB's tradition of academic excellence.

The academic integrity policy is available at https://buffalo.edu/academic-integrity and https://engineering.buffalo.edu/computer-science-engineering/information-for-students/graduate-program/cse-graduate-academic-policies/cse-academic-integrity-policy.html

Specifically for this course, students are allowed to discuss homework assignments. However, students are NOT allowed to share code, exploits, write-ups, and homework with each other. Plagiarism or any form of cheating in homework or exams is subject to serious academic penalty. All violations will be reported to the UB Office of Academic Integrity. There is a zero tolerance policy in this class.

- A minor academic integrity violation of a specific homework assignment (i.e., plagiarism on one question) may result in a 0 on that assignment.
- More serious violations (e.g., falsification) may result in deduction of the final grade and even an F or >F< on the final grade.
- The CSE department has a policy to upgrade the penalty to F on the final grade for a second academic integrity violation of any degree.

Syllabus Update

Information in the syllabus may be subject to change with reasonable advance notice.