COMS4995

AI for Software Security

Lecture 1 · Course overview + why AI4Sec now

Zhuo Zhang (ZZ) · Columbia CS
Email: zz@cs.columbia.edu

Who am I? Three hats, one instructor

Three perspectives you will see repeatedly in this course

Security research

Bug bounty mindset

Open source builder

Spoiler: this course lives in the overlap.

Perspective #1

Security research: make claims you can defend with evidence

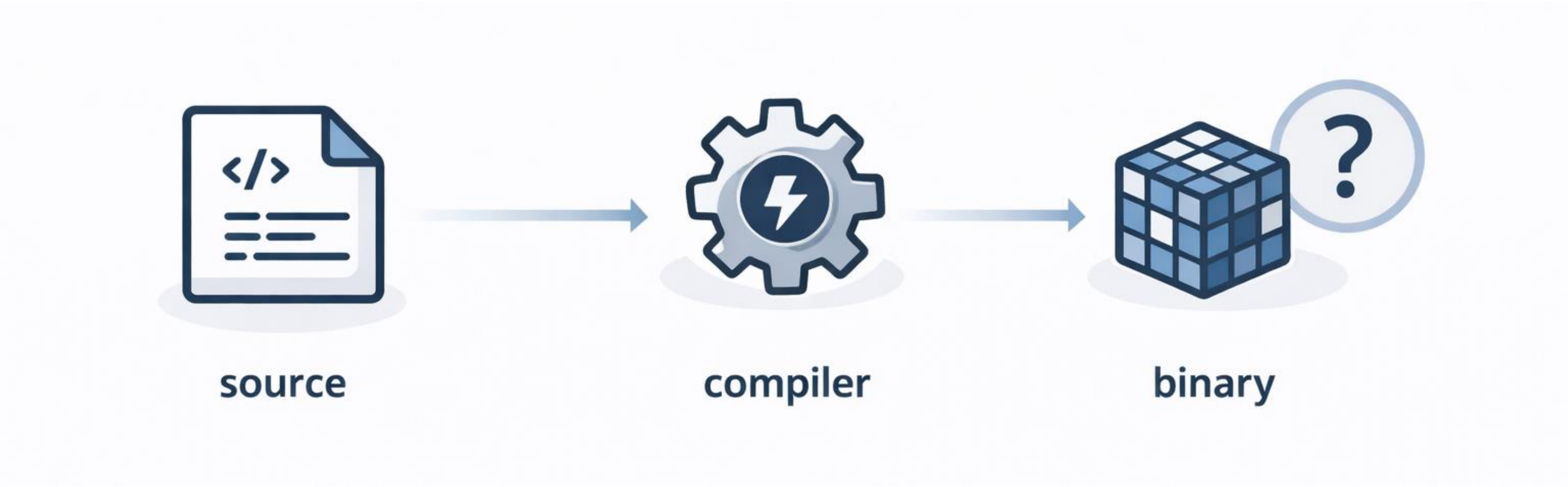
Research perspective: what I work on

Security as a design discipline + a scientific question

- Goal: make software hard to break and easy to trust.
- Engineering safety into development (not just patching after incidents).
- ***Expose defects with precise + scalable auditing.***

Example: binary analysis is uncertainty management

- Compilation removes names, types, and structure.
- We still want source-level reasoning from low-level artifacts.
- Key idea: treat recovery as probabilistic inference (not guesswork).



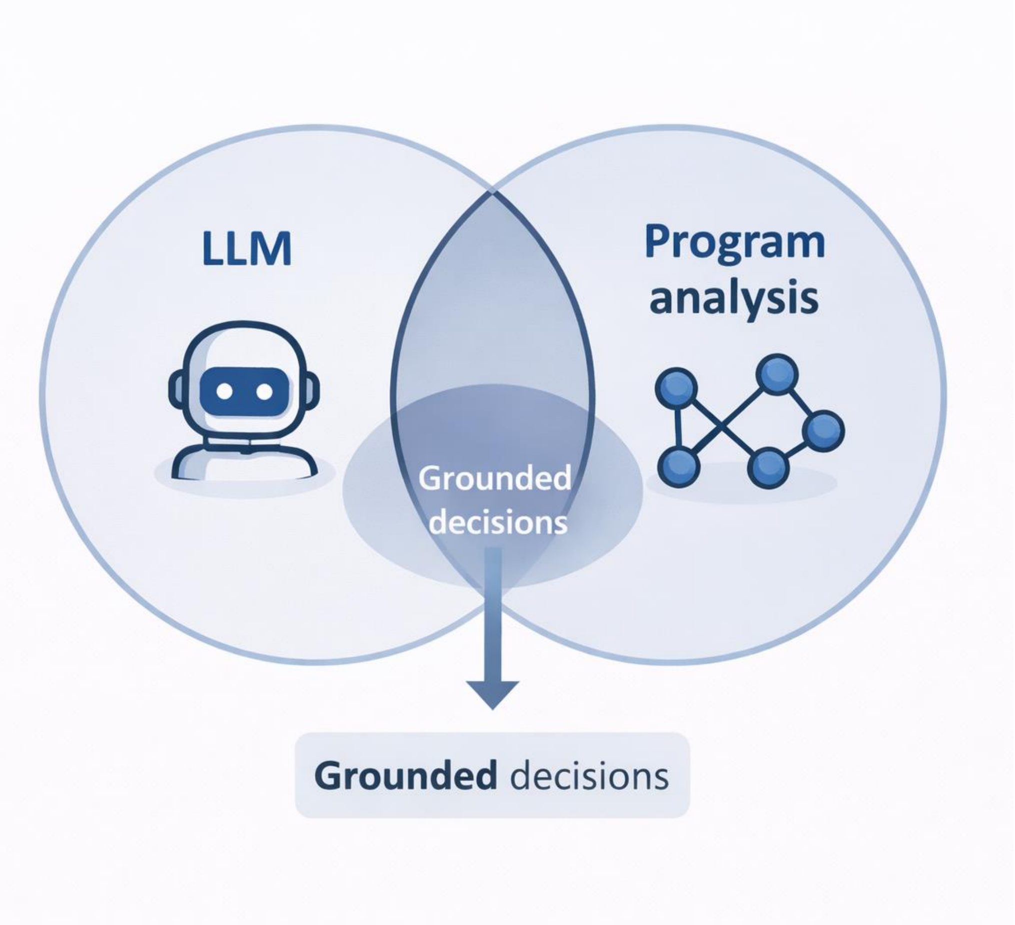
Example: LLMs + program analysis = better than either alone

- LLMs are strong at local reasoning and summarization…

- …but limited context and can hallucinate.

- Program analysis provides structure: graphs, constraints, invariants.

- Together: AI reasons with evidence instead of guessing.

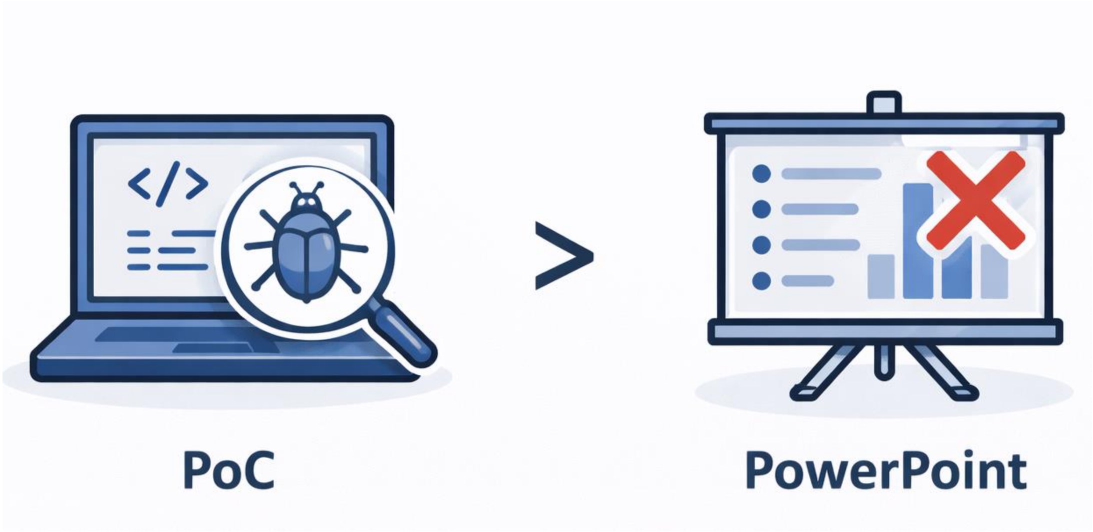


Perspective #2

Bug bounty / hacking mindset: impact, clarity, reproducibility

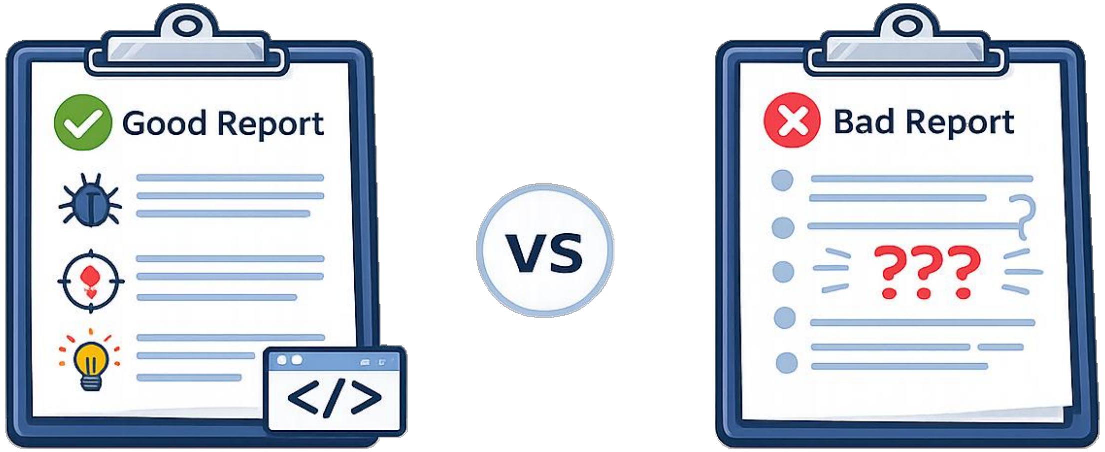
Bug bounty mindset: proof beats persuasion

- Reproduction steps > opinions.
- A tiny PoC that triggers the bug > a long argument.
- Your output should be actionable for maintainers.
- If you cannot reproduce it twice, it is a rumor.



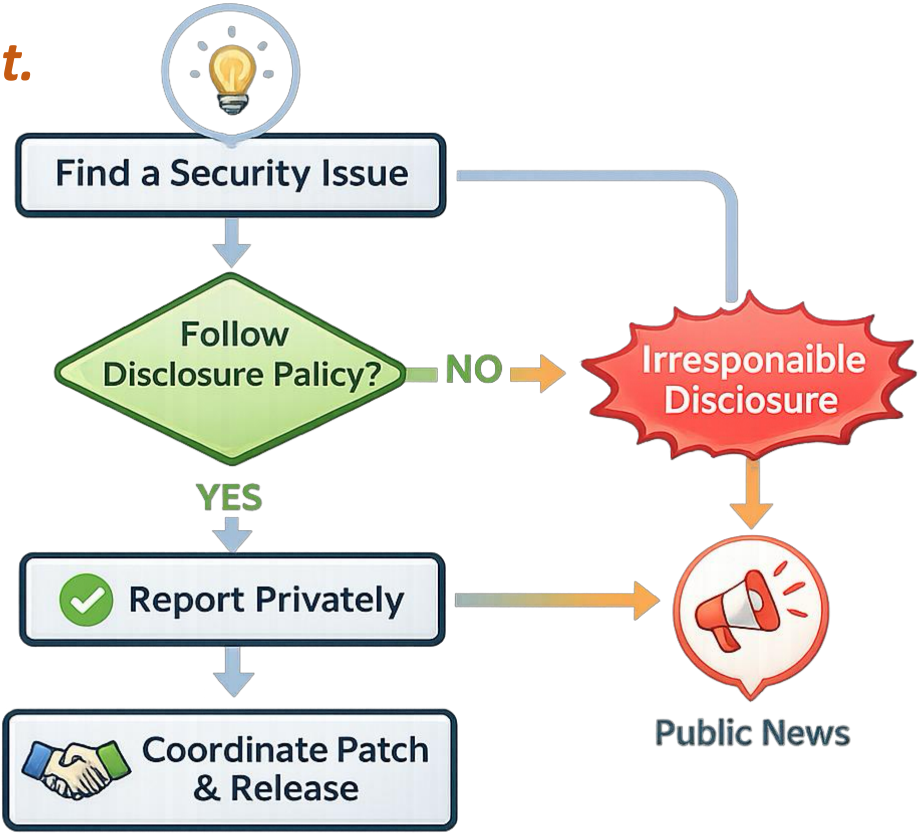
Bug bounty mindset: be useful to maintainers

- Write reports like you want to receive them.
- Include: impact, exact enviroment, prerequisites, and mitigation ideas.
- Prefer minimal, deterministic repro steps.
- Bonus: add a regression test if you can.



Ethics & disclosure: don’t be the headline

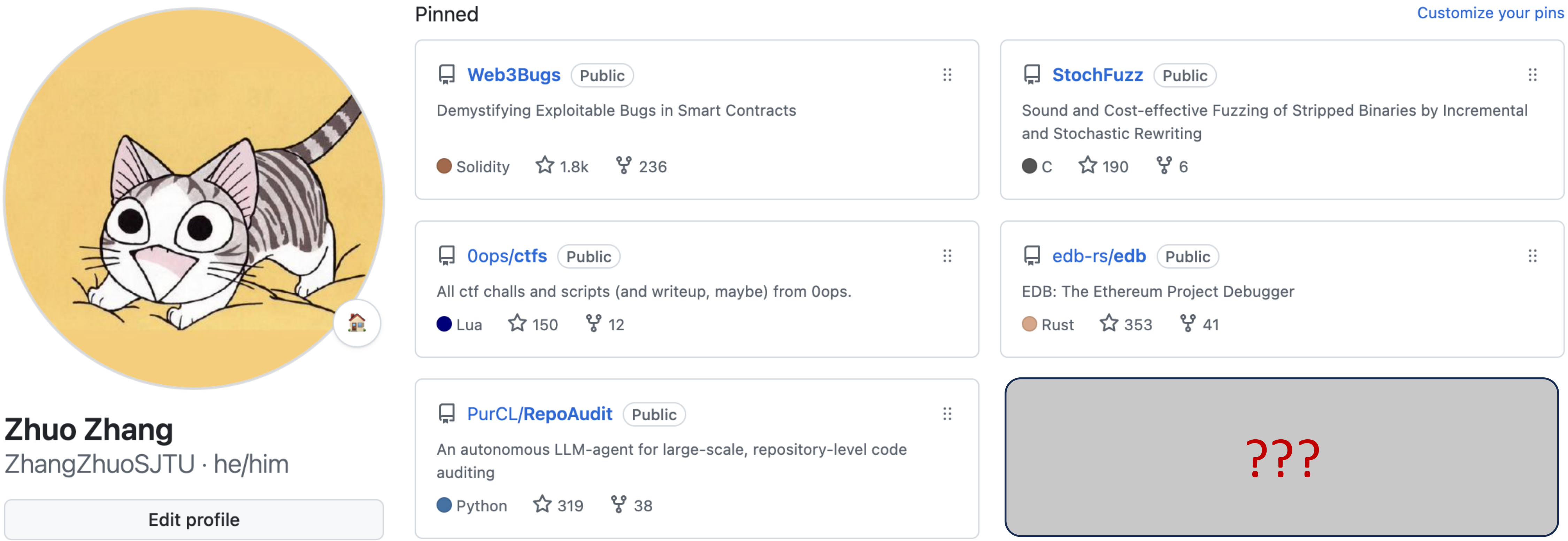
- Follow each project’s security policy (and responsible disclosure norms).
- ***Do not test on systems you do not have permission to test.***
- ***Do not publicly discuss any bugs before being fixed.***
- We want maintained software, not chaos.
- If in doubt: ask before acting.



Perspective #3

Open source builder: ship, maintain, collaborate

Open source: tools I maintain (examples)



Zhuo Zhang

ZhangZhuoSJTU · he/him

Edit profile

Pinned

Customize your pins

Web3Bugs Public

Demystifying Exploitable Bugs in Smart Contracts

● Solidity ☆ 1.8k 236

StochFuzz Public

Sound and Cost-effective Fuzzing of Stripped Binaries by Incremental and Stochastic Rewriting

● C ☆ 190 6

0ops/ctfs Public

All ctf challs and scripts (and writeup, maybe) from 0ops.

● Lua ☆ 150 12

edb-rs/edb Public

EDB: The Ethereum Project Debugger

● Rust ☆ 353 41

PurCL/RepoAudit Public

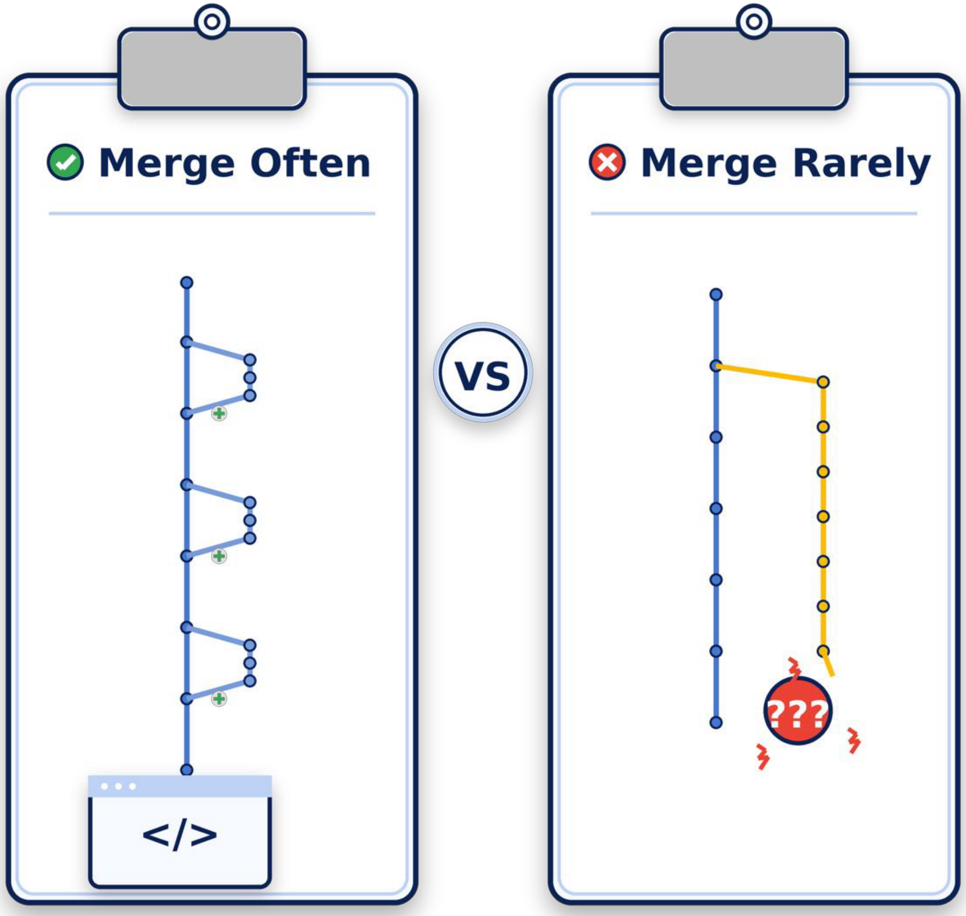
An autonomous LLM-agent for large-scale, repository-level code auditing

● Python ☆ 319 38

???

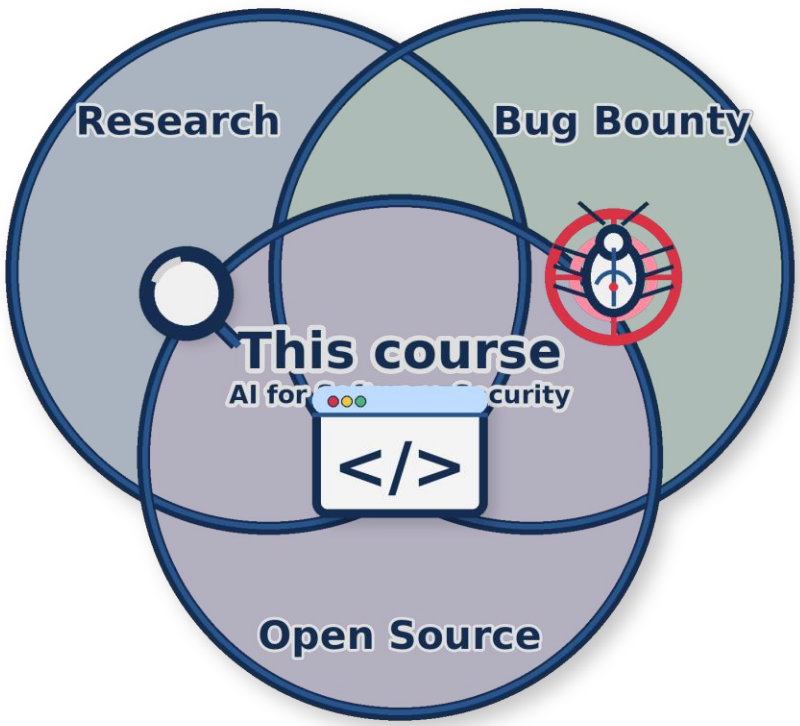
What open source teaches (that classes usually don't)

- Readable code is a feature.
- Tests are part of the product.
- Docs are empathy at scale.
- Small changes merged often > giant changes never merged.



This course lives in the overlap

Research rigor + real-world impact + open-source collaboration



What you will “feel” weekly

- We build in a shared repo (like a real team).
- We measure claims (not just demos).
- We chase real bugs when possible.
- We write tools others can run.

Today's menu

1. Quick intro (me + course vibe)
2. What this lecture covers (agenda)
3. Classroom rules (late class survival guide)
4. Prerequisite check (diagnostic only)
5. Course structure deep dive (the big chunk)
6. To-dos (GitHub org + next steps)
7. Next lecture guest speaker
8. Traditional security analysis + challenges
9. How AI can help (and how it can fail)

Classroom rules (5:50–6:55 PM edition)

- No homework, minimal attendance check

- Food is allowed (with constraints).

- If you are late, enter quietly (do not interrupt).

- Ask questions

Food policy: yes, but…

We all want dinner. We also want to hear each other.

Good examples

- Salad
- Bread / sandwich
- Quiet snacks
- Low-smell food
- No “crinkle symphony” packaging

Bad examples

- Hot pot (yes, this is a real example)
- Anything with strong smell
- Anything that requires “assembly”
- Anything that becomes loud later

Rule of thumb: if someone 3 seats away can smell it, it’s too strong.

If you’re late: be a ninja

- Do not knock.
- Do not announce yourself.
- Just walk in quietly and take a seat.

Course staff & communication

Fast help requires clear channels

Course Assistant (CA)

Sungjun Lee · sl5778@columbia.edu

Email rule (important)

- When you email the CA, please always CC me: zz@cs.columbia.edu
- Use a clear subject line: [AI4Sec] <topic>
- If it is about code/issues: include a GitHub link.

Slido check-in

Slido (you will set it up — placeholder only)

Quick diagnostic (no judgment): where are we starting?

- You will answer a few polls so I can calibrate pace and assumptions.
- If you are missing a prerequisite, ***this course could be hard for you***.

Do not edit
How to change the design

Python comfort level?

ⓘ The Slido app must be installed on every computer you’re presenting from

slido

Program analysis basics (CFG/DFG/taint)?

Git/GitHub comfort (branches/PRs/conflicts)?

Do not edit
How to change the design

Security background?

ⓘ The Slido app must be installed on every computer you're presenting from

slido

By the end of this semester, I want to…

Do not edit
How to change the design

ⓘ The Slido app must be installed on every computer you're presenting from

slido

Course structure

How we’ll run this class (and how to earn points without pain)

Course at a glance

- Meeting time: Tue/Thu 5:40–6:55 PM
- Location: 601B Sherman Fairchild Life Sciences Building
- Teams: 1–2 students
- Project-first: build AI-assisted auditors that run on real codebases
- Bug bounty: maintainer-confirmed extra points

Course at a glance

1. Tue/Thu 5:50–6:55 PM
2. 601B Sherman Fairchild Life Sciences Building
3. 1–2 students
4. Build AI-assisted auditors
5. Bug bounty: extra points

Why this course (in 5 bullets)

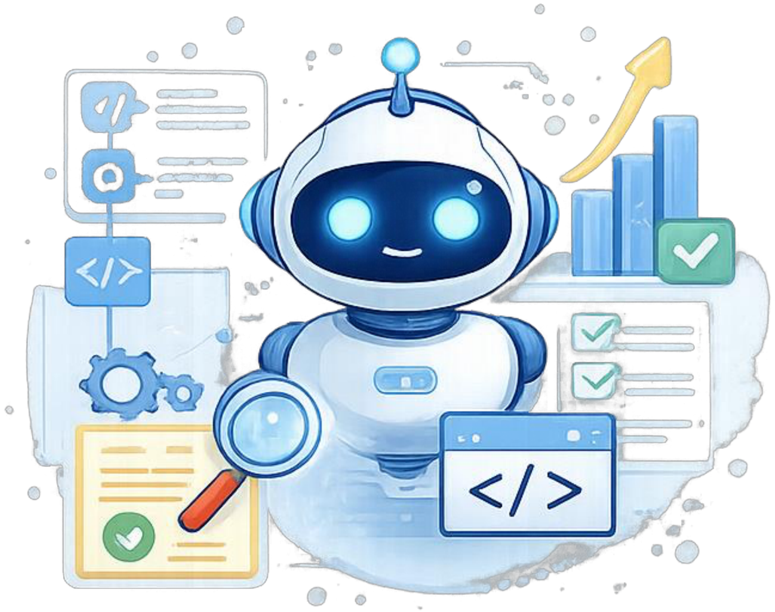
- Modern systems are too large and fast-moving for purely manual auditing.
- Static + dynamic analysis are powerful… but hit limits in precision, scalability, and engineering cost.
- AI can help with reasoning, triage, explanation, and workflow automation…
- …but AI can hallucinate, lose grounding, or generalize poorly.

The course thesis (what we will practice)

- AI should not guess — it should reason with evidence.

- We will ground AI decisions in program structure.

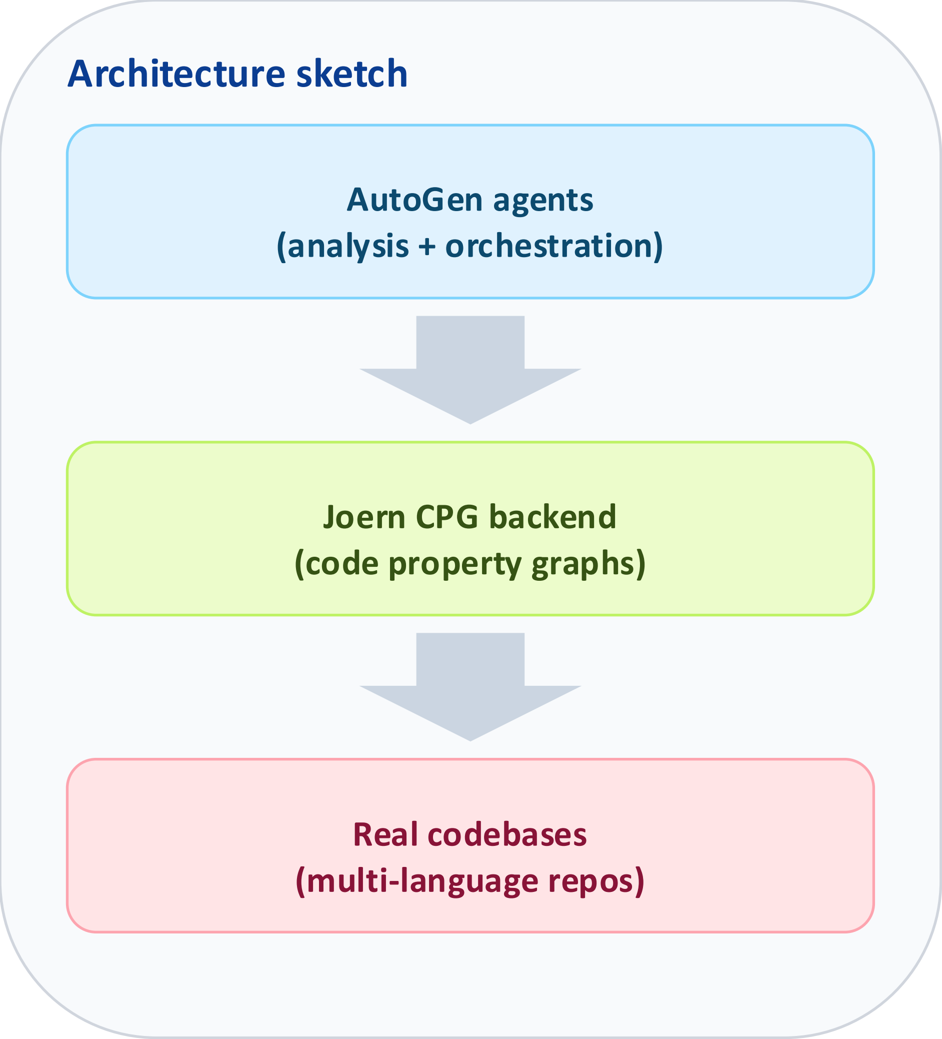
- We will ship code that others can run.



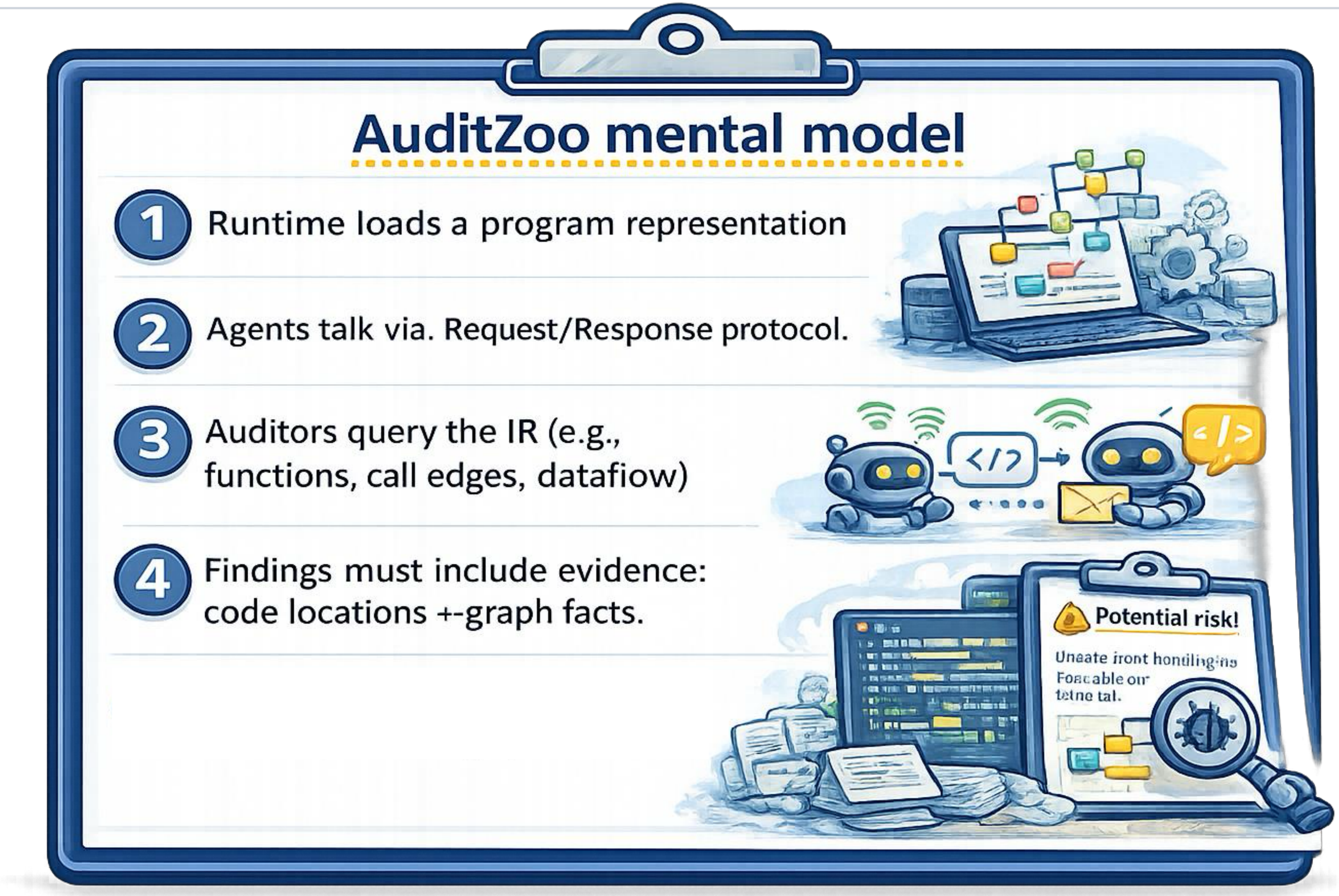
AuditZoo in one slide

AutoGen agents + Code Property Graphs (CPG)

- AuditZoo is the course backbone: a CPG-centered, agent-based program analysis framework.
- Built on: Joern (CPG) + AutoGen-Core (agent runtime).
- Gives us: a unified IR + protocol so auditors can query code structure consistently.
- Your auditors become plug-ins: same interface, different vulnerability targets.



AuditZoo mental model



AuditZoo mental model

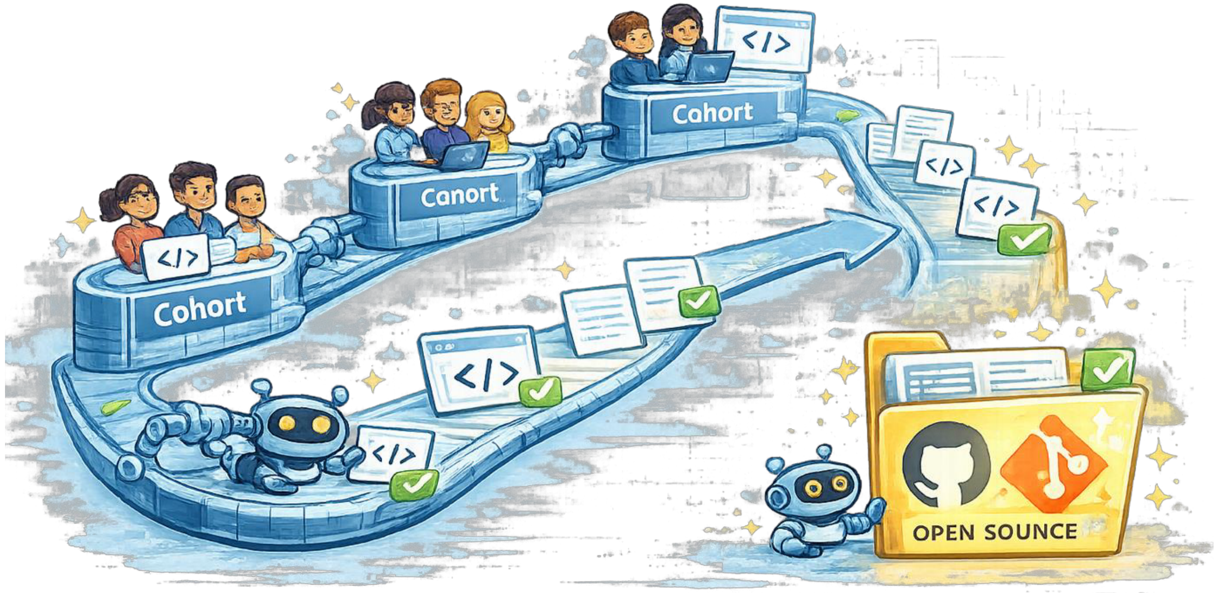
1. Runtime loads a program representation
2. Agents talk via. Request/Response protocol.
3. Auditors query the IR (e.g., functions, call edges, datafiow)
4. Findings must include evidence: code locations +-graph facts.

Not one-off projects

- We collaborate around shared infrastructure so work accumulates across cohorts.

- Projects live in a shared private AuditZoo repo during the semester.

- If students want: ***contributions can be merged into a public open-source version.***



Two things you learn simultaneously

This is not only an AI class; it is also an engineering-collaboration class

1) AI for software security

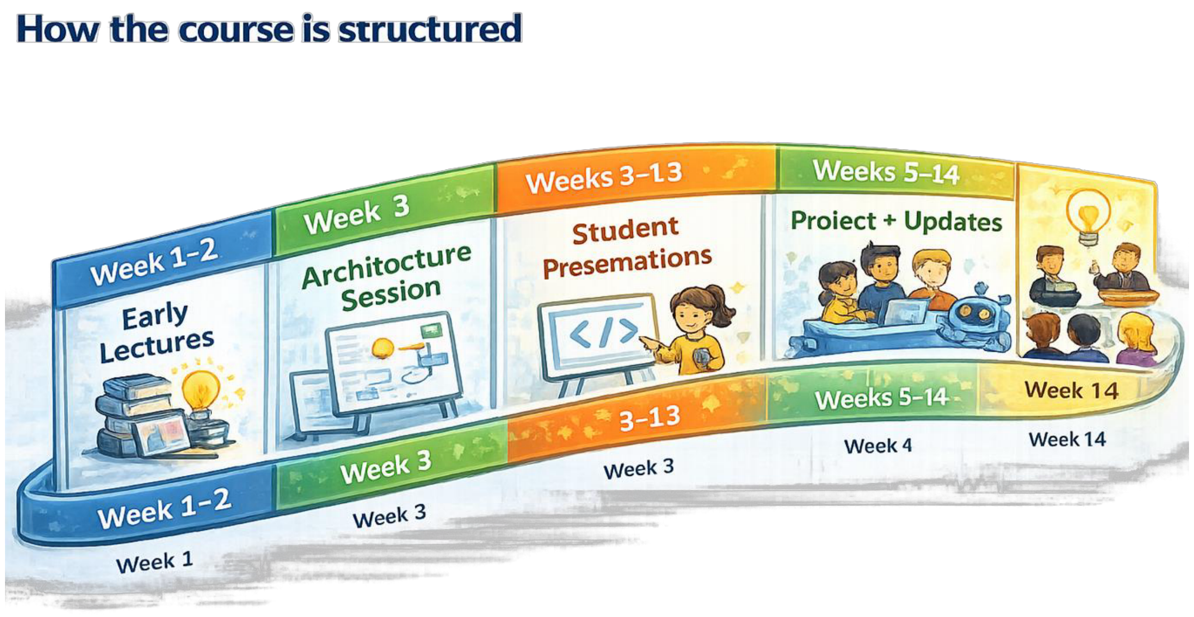
- What works / what does not
- How to ground AI in structure
- How to evaluate honestly

2) Engineering collaboration

- Shared repo workflow
- PRs + code review discipline
- Integration over hero coding

How the course is structured

- Early instructor-led lectures (first two weeks).
- AuditZoo architecture session (end of week 2).
- Student paper presentations (starting week 3): 20 min + 10 min Q&A.
- Semester-long project (teams of 1–2).
- AuditZoo updates + Q&A (weekly).
- Industry talks (up to 4).



Key dates you should put in your calendar

All deadlines are 11:59 PM unless noted

Date	Deadline / milestone
Thu Jan 29 (end of class)	Paper sign-up deadline
Thu Feb 5 (end of class)	Team formation deadline (1–2 students)
Thu Feb 19, 11:59 PM	Project proposal due (PDF + GitHub Discussion)
Fri Feb 27, 11:59 PM	Monthly project update (GitHub Discussion)
Tue Mar 31, 11:59 PM	Monthly project update (GitHub Discussion)
Tue Apr 14, 11:59 PM	Monthly project update (GitHub Discussion)
Mon May 4, 11:59 PM	Final report + final submission; bug bounty cutoff

Course rhythm (high-level)

- Weeks 1–2: instructor-led lectures + foundations.
- End of week 2: AuditZoo architecture + GitHub setup walkthrough.
- Weeks 3–? : student paper presentations (ongoing).
- Weekly: AuditZoo update/Q&A (unblock contributors).
- Midterm: short progress presentation (5–10 min).
- End: final presentation + final report + evaluation.

Paper presentations: format & feedback

- 20 minutes talk + 10 minutes Q&A.
- Pick a paper in scope (some will be pre-approved).
- Audience submits anonymous rating (1–10) + optional comments.
- Ratings provide structured feedback and contribute to scoring (normalized).

Paper sign-up (GitHub Discussions) + early bonus

- Sign up for a date in GitHub Discussions (first-come-first-confirm-first-in).
- Deadline: Thu Jan 29 (end of class).
- Bonus: ***+1 course point*** for paper talks delivered in the first two weeks of presentations.
- If you want the bonus: sign up early and prepare early.

Projects: two tracks (teams of 1–2)

- Track A (recommended): build an AI auditor for one vulnerability class / defect pattern.
- Track B: extend AuditZoo infrastructure itself.
- All projects live in the shared private AuditZoo repo during the semester.

Minimum background (sanity checklist)

- Python programming.
- Basic program analysis: CFG, DFG, taint analysis, and related concepts.
- Git and GitHub.

What you will learn (outcomes)

- How to design AI-assisted auditors grounded in program structure.
- How to turn a research idea into an implementable approach (scope, threat model, failure modes).
- How to evaluate a security tool with metrics, test cases, and honest limitations.
- How to work like an engineering team in a shared repo with PRs and integration discipline.
- How to communicate work: paper talks, proposals, updates, final presentations.

Grading breakdown

Bug bounty points are extra (can only help)

Component	Weight
Attendance	5% (light-touch)
Paper presentation (individual)	15%
Project proposal (team)	10%
Midterm progress presentation (team)	10%
Final project presentation (team)	20%
Final report + evaluation (team)	40%
Bug bounty	Uncapped extra points (maintainer-confirmed)

Late policy (written deliverables)

- Each team has 2 guaranteed late days for written deliverables.
- Late days do NOT apply to scheduled presentations.
- Beyond that, we'll be flexible if it doesn't disrupt scheduling/coordination.
- Pro tip: don't spend late days on avoidable Git conflicts.

Track A

Auditor projects (strongly recommended)

Track A: build an AI auditor agent

- Specialize in one vulnerability class or defect pattern.
- Your auditor must run on real codebases and produce evidence.
- Goal: high-signal findings, not giant hallucinated reports.
- Integration into AuditZoo is part of “done”.

Track A examples (web / systems)

- SQL injection in a Python web app (PostgreSQL/MySQL).
- Inconsistent specification-to-code mapping in go-ethereum (geth) or other clients.
- Path traversal + unsafe file handling in document processing services.
- SSRF patterns in cloud-integrated services.
- Unsafe deserialization in Java/Kotlin microservices.
- You may either implement ***ideas from existing papers*** in AuditZoo or explore your ***own novel ideas***.

Track A: what “done” looks like

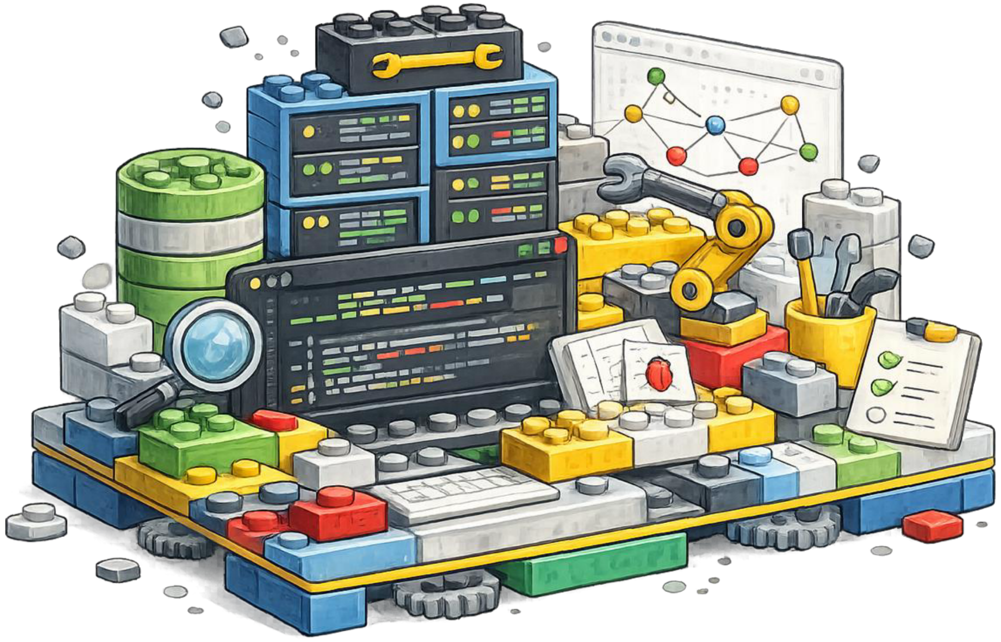
- A working auditor integrated into AuditZoo (others can run it).
- Evaluation section in the final report (metrics + honest limitations).

Track B

Infrastructure projects (extend AuditZoo itself)

Track B examples (infrastructure)

- Add a CodeQL backend or strengthen integrations.
- Add tree-sitter parsing to support more languages (e.g., Ada).
- Extend abstraction layers with new graph queries or IR adapters.
- Improve scalability + automation (docs, tests) that enable Track A auditors.



Track B: what “done” looks like

- A working infrastructure feature integrated into AuditZoo.
- Evaluation: what capability it enables + what constraints remain.
- More frequent PR merges to stay in sync with main (Track B touches core).

Project deliverables (team)

- Project proposal: 1–2 pages (IEEE S&P format).
- Monthly project updates: short GitHub Discussion posts.
- Midterm progress presentation: 5–10 minutes.
- Final presentation: 20–30 minutes (demo encouraged).
- Final report + evaluation: your main deliverable.

Monthly updates (GitHub Discussions)

Short, consistent updates keep the shared repo moving

Suggested update template

- What we shipped since last update (links to PRs).
- What we learned (including failures).
- Current blockers / help needed.
- Plan for next period (1–3 concrete goals).

Why we do this

- It prevents duplicated effort across teams.
- It makes it easier to help you when you are stuck.
- It keeps AuditZoo stable as many teams contribute.

GitHub workflow

How we collaborate (the “shared repo” survival guide)

GitHub is the system of record

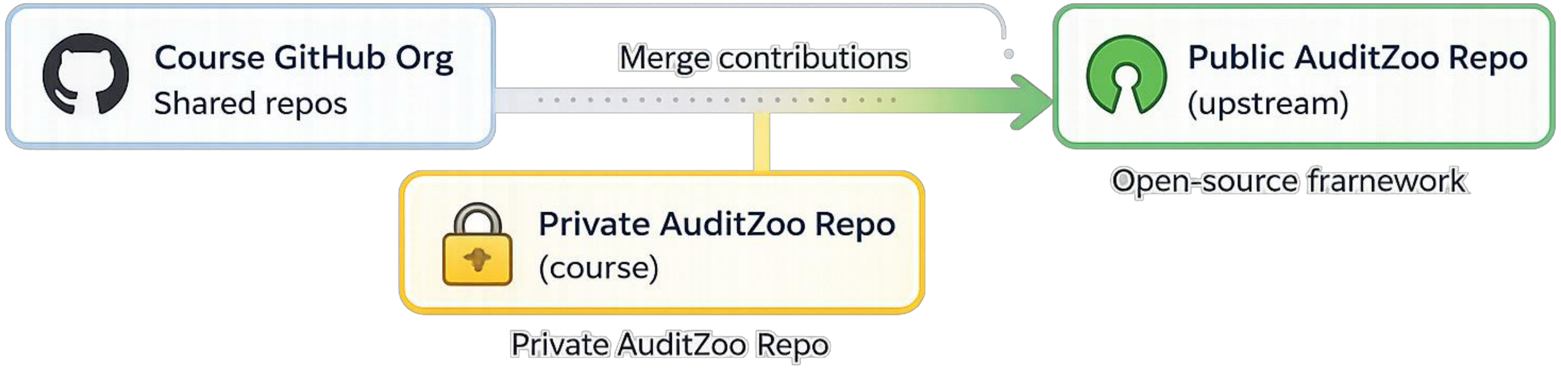
- Coordination: what are we doing?
- Collaboration: how do changes land safely?
- Communication: questions, updates, sign-ups.
- If it is not on GitHub, it is easy to lose.

Three GitHub tools, three jobs

- Issues → tracking work (bugs, features, requests).
- Pull Requests → integration (code review + tests).
- Discussions → Q&A + sign-ups + monthly updates.

Where things live

- Course GitHub org: shared repos.
- Shared private AuditZoo repo: where you build during the semester, and where you post discussions and issues.
- Public AuditZoo repo: upstream open-source framework.
- Optional: merge course contributions into the public version after the semester.



Branching model (required)

- Each team works on its own branch (on the private repo).
- Naming convention: team-name/proposal-name (keep it readable).
- Do not push directly to main.
- Small PRs, merged often.
- Never use "***git push -f***"

Sync with main (at least weekly)

- Main will evolve as other team (and I) contribute.
- Weekly sync prevents “end-of-semester merge disasters”.
- It also makes it easier to review your PRs.
- Analogy: flossing. Boring, effective, non-optional.

Pull Requests: quality bar

How to get your work merged without drama

PR checklist (short version)

- Clear title + description (what/why/how to test).
- Small scope (reviewable).
- Add/adjust tests where possible.
- Do not mix core infrastructure changes with analysis implementations.
- Link related Issues/Discussions.

Why we ban “mixed PRs”

Because before the end of semester, other teams would not be interested in your analysis/auditor implementation, but need the infrastructure updates.

To do (tonight): get GitHub access

- Apply to join the course GitHub org: send email to CA and CC’ed me, with your GitHub handle.
- Confirm you can access the shared private repo.
- Skim the GitHub guide (“GitHub in this course”).
- If anything blocks you: post in Discussions early.

To do (this week): paper + team + idea

- Pick a paper you are excited to present.
- Find a teammate (or decide to work solo).
- Choose Track A or Track B.
- Start a project proposal discussion thread early
  - Name conventions: [Team Name]: [Proposal Title].
  - The earlier you posed, the more feedback you will receive.

To do (before Jan 29): commit to momentum

- Sign up for a paper date (earlier = more options).
- Set up your dev environment and run a basic AuditZoo example.
- If you want the early +1 point: aim for one of the first two weeks of paper talks.

Next lecture: guest talk

AI has changed the security world — what does that look like in practice?

- Guest: Hari (@hrkrshnn)
- From Cantina / Spearbit (security + smart contract ecosystem).
- Title: ***Tales from building AI security agents***
- Focus: how AI is affecting real-world security workflows.

Who is Hari?

- CEO and co-founder of Cantina and Spearbit.
- Previously built the Solidity language at the Ethereum Foundation.
- Background in pure mathematics; also worked on a linear programming solver (simplex).
- Perspective: security research + industry operations + smart contract ecosystem.