Q1: Explain how AI-driven code generation tools (e.g., GitHub Copilot) reduce development time. What are their limitations?

- **Reduction of boilerplate and routine work:** AI tools can autocomplete common patterns (e.g. CRUD operations, API calls), scaffolding tests, or generating documentation stubs, so developers spend less time on repetitive code and more on business logic.
- **Instant suggestions and examples:** They surface idiomatic usage, library calls, and best practices directly in the editor, reducing context-switching to search engines or docs.
- **Limitations:**
  - **Accuracy & correctness:** Generated code may compile but harbor subtle bugs or security flaws (e.g. improper input validation).
  - **Context awareness:** They often lack project-wide understanding, leading to suggestions that don't integrate cleanly with existing architecture.
  - **Bias & licensing:** Training on publicly available code can surface copyrighted snippets or perpetuate insecure patterns.

Q2: Compare supervised and unsupervised learning in the context of automated bug detection.

- **Supervised learning:** Models are trained on a labeled dataset of code snippets (buggy vs. clean). They learn explicit patterns associated with known bug types (e.g. null-pointer exceptions). Highly accurate on familiar bug categories but require large, well-labeled corpora.
- **Unsupervised learning:** Models detect anomalies or outliers in code metrics or execution traces without explicit labels (e.g. clustering rare call graphs). Better at discovering novel or unknown bug patterns but prone to false positives and require manual triage to confirm true bugs.

Q3: Why is bias mitigation critical when using AI for user experience personalization?

- **Fairness and representation:** Without mitigation, minority user groups may receive poor recommendations or be excluded from certain features, harming inclusivity.
- **Legal and ethical compliance:** Regulations (e.g., GDPR) require non-discriminatory treatment; biased personalization can expose companies to legal risk.
- **Trust and engagement:** Users who perceive unfair treatment are less likely to engage; balanced, transparent algorithms foster trust and long-term loyalty.

---

2. Case Study Analysis

Article: “AI in DevOps: Automating Deployment Pipelines”
Source: Azati Blog, February 11, 2025

How does AIOps improve software deployment efficiency? Provide two examples.

1. **Predictive CI/CD Optimization:**
   By analyzing historical build and test data, AI models predict potential build failures and prioritize the most reliable test cases to run first, reducing feedback loops and accelerating deployment. For example, CircleCI leverages ML on past test success/failure rates to select an optimal subset of tests, ensuring developers get rapid, actionable feedback and iterate more quickly Azati.
2. **Automated Rollbacks & Self-Healing:**
   Harness’s AIOps platform automatically detects a failed deployment in real time and rolls it back without human intervention, slashing mean time to recovery (MTTR) and preventing extended downtime. Similarly, AI-powered chatbots use past incident data to remediate issues within seconds, rather than hours, boosting overall pipeline resilience Azati.

art 3: Ethical Reflection (10%)

When deploying the above model to predict “issue priority” in a company, **biases** may arise if certain teams or types of issues are underrepresented in the training data—leading to systematic under- or over-prioritization. For instance, if most “high” labels came from critical infrastructure teams, issues from the R&D group might be unfairly classified as lower priority because their historical data was sparse.

Tools like **IBM AI Fairness 360** help detect and mitigate such biases. They can compute fairness metrics (e.g., disparate impact ratio) across groups and apply algorithms like **reweighting** or **adversarial debiasing** to rebalance training samples. Incorporating such fairness workflows ensures that priority predictions reflect true business impact across all teams, fostering equitable treatment and maintaining trust in automated decision-making.

Bonus Task : Innovation Challenge

Proposal: “DocuGenie” – AI-Driven Documentation Generator

1. **Purpose:**
   Automate the creation and upkeep of developer documentation (API references, architecture diagrams, usage examples) directly from code and commit history.

2. **Workflow:**
    - **Code Ingestion:** Integrate with GitHub/GitLab to scan PRs and commits.
    - **Natural Language Analysis:** Use large language models (LLMs) to summarize new functions, classes, and endpoints.
    - **Auto-Update Docs:** Push changes to a documentation site (e.g., mkdocs), including code samples and UML diagrams generated via static analysis.
    - **Review Loop:** Open PRs for docs with inline comments for developer approval, ensuring accuracy.
3. **Expected Impact:**
    - **Time Savings:** Cuts down manual doc-writing by 70%.
    - **Accuracy & Freshness:** Always in sync with the latest code.
    - **Onboarding Efficiency:** New team members can rely on up-to-date, comprehensive docs from day one.