Part 1: Theoretical Analysis (30%)

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

Answer:

AI-driven code generation tools reduce development time primarily through **context-aware code suggestion**. By analyzing the code being written, comments, and function names, these tools can automatically generate entire lines, blocks, or even functions of code.

Limitations include:

- **Security and Quality Risks:** The model may suggest code with vulnerabilities, bugs, or outdated patterns it learned from its training data
- **Lack of Deep Understanding:** It does not understand the project's broader architecture or business logic, potentially leading to functionally correct but architecturally poor code.
- **Over-reliance:** Developers might become dependent, potentially eroding their fundamental programming skills and critical thinking.

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

Feature	**Supervised Learning for Bug Detection**	**Unsupervised Learning for Bug Detection**
Approach	Learns from a labeled dataset of code that is classified as "buggy" or "clean."	Analyzes code to find inherent patterns or anomalies without pre-existing lables
Pros	Can be highly accurate if the training data is high quality and relevant	Can detect previously unseen types of bugs that are not in the labeled dataset
Cons	Requires a large, accurately labeled dataset which is expensive to create. May miss new bug patterns	Can have high false positive rate, as not all anomalies are bugs. Results can be harder to interpret

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

Answer:

Bias mitigation is critical in UX personalization because unchecked AI can create negative feedback loops and discriminatory experiences.

2. Case Study Analysis

Article: AI in DEVOPS: Automating Deployment Pipelines

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

Answer:

AI improve software deployment efficiency by enabling smarter decision-making, predictive analytics, and self-healing systems.

1. **Continuous Integration and Continuous Deployment (CI/CD)**

   With AI driven automation potential build failures can be predicted in advance and test cases optimized; meanwhile, automation helps to make successful deployment. By applying machine learning(ML) models to historical data, we use them to look for problems during the installation phase of a new system and to shape things so that they are less at risk of going down when they go live.

2. **Infrastructure as Code(IaC) Optimization**
   IaC capabilities enable teams to create machine-readable scripts that define IaC. By Providing automation and analytics underlying IaC, AI enhances settings optimization automation and configuration, security flaw detection, and even usage prediction.

Part 3: Ethical Reflection (10%)

Prompt: Your predictive model from Task 3 is deployed in a company. Discuss potential biases and how to address them.

Answer:

If the predictive model from Task 3 were deployed to prioritize IT support tickets or bug fixes, several biases could arise, primarily from the dataset itself. The original Breast Cancer Wisconsin

dataset is predominantly from a specific demographic and geographic location. Translated to a corporate context, this could manifest as:

1. **Underrepresentation of Certain Teams:** If the "cancer" data represents "system failures," the model might be biased if historical failure data comes mainly from, for example, the "Finance" department's systems. It could learn to prioritize issues from Finance over equally critical issues from an underrepresented team like "Research & Development," whose failure patterns are less common in the training data.
2. **Temporal Bias:** The model is trained on a snapshot of data. If a company rolls out a new technology stack, the old model will be ineffective and biased against recognizing issues in the new environment.

Addressing Biases with IBM AI Fairness 360 (AIF360):

IBM AIF360 is an open-source toolkit containing a comprehensive set of metrics and algorithms to detect and mitigate bias throughout the ML lifecycle.

- **Detection:** We would first use AIF360's **bias metrics**, such as **Disparate Impact** and **Average Odds Difference**, to quantify whether the model's predictions are fair across different groups (e.g., "team = Finance" vs. "team = R&D"). This would statistically prove if a bias exists.
- **Mitigation:** If bias is detected, we could employ AIF360's mitigation algorithms. For example, we could use **Reweighing** as a pre-processing technique, which would assign higher weights to training examples from the underrepresented "R&D" team to balance their influence on the model. Alternatively, we could use **Adversarial Debiasing** to build a model that explicitly learns to make accurate predictions while preventing itself from learning to discriminate based on the protected attribute (the team name).

By integrating these tools into our MLOps pipeline, we can continuously monitor drift and bias, ensuring our resource allocation model is not only accurate but also fair and equitable across the entire organization.

Bonus Task (Extra 10%)

Deliverable: 1-page proposal outlining the tool's purpose, workflow, and impact.

Proposal: "DocuGen AI" - Automated, Context-Aware Code Documentation Generator

1. Purpose:
DocuGen AI is designed to solve the pervasive software engineering problem of inadequate and outdated documentation. Developers often neglect documentation due to time constraints, leading to knowledge silos, difficult onboarding, and increased maintenance costs. DocuGen AI

automatically generates and maintains high-quality, human-readable documentation by deeply analyzing the codebase, commit history, and runtime behavior.

2. Workflow:

- **Input & Analysis:**
    - The tool is integrated into the CI/CD pipeline or run as a standalone service.
    - It ingests the source code, static analysis reports, git commit messages, and (if available) API traffic logs.
    - It uses a fine-tuned Large Language Model (LLM) to understand code structure, function purpose, and complex business logic.
- **Generation & Synthesis:**
    - **API Documentation:** Automatically generates OpenAPI/Swagger specifications by analyzing REST controller endpoints and data models.
    - **Inline Comments:** Suggests descriptive comments for complex functions and algorithms directly within the code.
    - **Architectural Overview Documents:** Creates high-level diagrams and documents explaining module interactions and data flow by building a dependency graph.
    - **Change Logs & Release Notes:** Analyzes commit messages between versions to automatically draft summaries of new features, bug fixes, and breaking changes.
- **Maintenance & Interaction:**
    - DocuGen AI includes a "Documentation Health" dashboard that flags areas where the code has changed but the documentation has not been updated.
    - It features a chatbot interface where developers can ask natural language questions like "How do I process a user payment?" and receive answers derived from the generated documentation and relevant code snippets.

3. Impact:

- **Dramatically Reduced Overhead:** Cuts documentation writing and maintenance time by over 70%, freeing developers for core feature development.
- **Improved Code Quality & Onboarding:** Comprehensive, up-to-date documentation makes codebases more understandable and maintainable, reducing the onboarding time for new engineers from weeks to days.

- **Knowledge Preservation:** Prevents critical business logic knowledge from being lost when senior developers leave the team.
- **Enhanced Collaboration:** Clear documentation bridges the gap between technical and non-technical stakeholders, ensuring everyone has a shared understanding of the system.