1. Short Answer Questions

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

How they reduce development time: AI-driven code generation tools like GitHub Copilot analyze vast codebases and suggest context-aware code snippets, autocompleting functions, or entire blocks based on developer input. This speeds up coding by reducing manual typing, suggesting optimized solutions, and providing boilerplate code or bug fixes, allowing developers to focus on higher-level logic.

Limitations: These tools may generate incorrect or insecure code, requiring manual review. They rely on training data quality, so they can miss niche or domain-specific patterns. Overreliance may reduce learning opportunities, and they may struggle with complex, novel problems or ambiguous requirements.

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

Supervised Learning: Uses labeled datasets (e.g., code with known bugs and fixes) to train models to identify and classify bugs. It excels at detecting known bug patterns with high accuracy but requires extensive labeled data and may miss novel bugs.

Unsupervised Learning: Analyzes code without labeled data, identifying anomalies or clusters of potential bugs based on patterns or deviations. It's better for detecting unknown bugs but may produce false positives and lacks the precision of supervised methods for known issues.

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

Bias mitigation is critical because AI personalization relies on user data, which can reflect historical biases (e.g., skewed demographics). Unmitigated bias leads to unfair or discriminatory recommendations, alienating users and reducing trust. It also risks ethical and legal issues, ensuring equitable experiences across diverse user groups.

2. Case Study Analysis

1.Predictive Build Failure Detection in CI/CD Pipelines:

AIOps analyzes historical build data, code changes, and test results to predict potential failures in CI/CD pipelines. It identifies risky commits (e.g., those likely to cause test failures or dependency conflicts) and suggests fixes before deployment. This reduces failed builds, minimizes debugging time, and accelerates the deployment process by ensuring stable code reaches production.

2.Automated Resource Scaling for Deployment Environments:
AIOps monitors real-time metrics like CPU, memory, and network usage during deployment, dynamically scaling cloud resources (e.g., containers or virtual machines) to match demand. This prevents resource bottlenecks, optimizes performance, and reduces costs by avoiding over-provisioning, ensuring faster and more efficient deployments.