**Q1: AI-Driven Code Generation (GitHub Copilot)**

**Answer:**

AI-driven code generation tools like GitHub Copilot significantly reduce development time by offering intelligent code suggestions based on the developer’s current context. These tools leverage large language models trained on vast code repositories to autocomplete functions, suggest syntax, or even generate entire modules. This boosts productivity, especially for repetitive or boilerplate code, allowing developers to focus on high-level logic and design.

**Limitations include:**

* **Context awareness gaps:** Copilot may not fully grasp the broader architecture of your application, leading to less optimal suggestions.
* **Security risks:** It can suggest insecure or deprecated practices.
* **Lack of originality:** Copilot might regurgitate code from public repositories, which could pose licensing issues.
* **Debugging time:** Developers may spend extra time validating and understanding AI-suggested code.

While these tools are powerful aids, they should not replacee human judgment and code review.

**Q2: Supervised vs. Unsupervised Learning in Bug Detection**

**Answer:**

In automated bug detection, **supervised learning** relies on labelled data such as code examples annotated as “buggy” or “clean” to train a model to classify or predict bugs. It is effective when there is a large, high-quality dataset with clear labels. Examples include static code analysis tools that flag common coding errors using historical data.

In contrast, **unsupervised learning** doesn’t require labelled data. It identifies anomalies or unusual patterns in code behaviour or metrics (e.g., memory usage). Clustering or anomaly detection models are often used to highlight potential bugs that deviate from typical program behaviour.

**Comparison:**

|  |  |  |
| --- | --- | --- |
| **Feature** | **Supervised Learning** | **Unsupervised Learning** |
| Data | Labelled | Unlabelled |
| Accuracy | High if data is quality | May find unknown issues |
| Use Case | Known bugs | Novel or unknown bugs |

**Q3: Bias Mitigation in Personalization**

**Answer:**

Bias mitigation is critical in AI-based user experience personalization to ensure inclusive and fair software systems. Personalization algorithms often rely on user data like demographics, behaviour, or preferences. If this data is skewed underrepresenting certain user groups, AI systems may personalize the experience in a way that excludes or misrepresents them.

For instance, if a feature recommendation system is trained on predominantly male user data, it may fail to offer relevant suggestions to female users.

**Bias mitigation** ensures that personalization benefits all users equally. Techniques include:

* Balancing datasets during training
* Auditing personalization outcomes
* Using fairness-aware algorithms

Failure to mitigate bias can lead to legal issues, user dissatisfaction, and reputational harm.