AI FOR SOFTWARE ENGINEERING

POWER LEARN PROJECT

WEEK 4 ASSIGNMENT

AI IN SOFTWARE ENGINEERING

THEME: BUILDING INTELLIGENT SOFTWARE SOLUTIONS

GROUP 35

ACTIVE MEMBERS:

- LATHITHA VENA
- JOHN BROWN OUMA

Part 1: Theoretical Analysis (30%)

1. Short Answer Questions

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

AI-driven code generations tools like GitHub accelerate development by suggesting context-aware code snippets, autocompletion functions, and generating boilerplate code based on natural language prompts or partial code. They leverage large language models trained on vast codebases, enabling developers to focus on high-level logic rather than repetitive tasks. For example, Copilot can generate a sorting function from a comment like "sort list of dictionaries by keys", saving time on syntax and implementation.

Limitations:

- **Accuracy**: Suggestions may be incorrect or suboptimal, requiring manual review.
- **Context Misinterpretation**: The tool may misinterpret intent, generating irrelevant code.
- **Dependency on Training Data:** It May replicate biases or outdated practices from its training data.
- **Security Risks**: Generated code may include vulnerabilities if not vetted.

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

Supervised Learning: Uses labelled datasets e.g., code snippets marked as "buggy" or "clean" to train models like Random Forest or Neural Networks. It excels in detecting known bug patterns with high accuracy but requires extensive labelled data, which is time consuming to create.

Example: Classifying code as buggy based on historical bug reports.

Unsupervised Learning: Identifies anomalies in code without labelled data, using clustering or anomaly detection e.g., DBSCAN. It's useful for discovering novel bugs but may miss subtle issues or generate false positives.

Example: Detecting unusual code patterns in a repository.

Supervised is better for known Issues whereas Unsupervised suits exploratory bug detection.

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

Bias in AI personalization e.g., recommending features based on user data can lead to unfair outcomes, such as excluding underrepresented groups or reinforcing stereotypes. For instance, if training data overrepresents certain demographics, the AI may prioritize their preferences, alienating others. Mitigating bias ensures inclusivity, improves user trust, and complies with ethical standards. Tools like fairness-aware algorithms can reweight data or adjust model outputs to promote equitable personalization.

2. Case Study Analysis

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

AIOP enhances deployment efficiency by automating repetitive tasks, predicting issues, and optimizing resource allocation using AI. It analyses logs, metrics, and patterns to streamline CI/CD.

Example 1: AIOP tools like Dynatrace monitor application performance in real-time, detecting anomalies e.g., memory leaks before deployment, reducing rollbacks frequency.

Example 2: Tools like Jenkins with AI predict build failures by analysing historical data, enabling pre-emptive fixes and faster release cycles.

Part 3: Ethical Reflection (10%)

Potential biases in the dataset (e.g., underrepresented teams)

The Kaggle Breast Cancer dataset, used in task 3, may contain biases if certain patient demographics (e.g., age, ethnicity, or socioeconomics status) are underrepresented as it's based on historical medical records. For Instance, if the dataset predominantly includes data from one ethnic group, the model may perform poorly for others, leading to inequitable resource allocation in a company deploying this model. This could prioritize issues incorrectly for underrepresented groups, exacerbating disparities.

How fairness tools like IBM AI Fairness 360 could address these biases.

IBM AI Fairness 360 offers tools to detect and mitigate bias. Its **reweighing algorithm** adjusts sample weights to balance representation across groups, ensuring the model learns fairly from all demographics. **The adversarial Debiasing** technique trains a model to predict outcomes while minimizing bias in sensitive attributes (e.g., ethnicity). By integrating these tools, the company can monitor bias metrics (e.g., disparate impact) and retrain the model to ensure fair issues prioritization, fostering equitable software development.

Bonus Task: Innovation Challenge (Extra 10%)

Proposal: AI Tool for automated Documentation Generation (DocGenAI – Intelligent Code Documentation Generator)

Purpose

Developers often neglect or delay writing documentation due to time constraints, leading to poor code maintainability and onboarding challenges. DocGenAI aims to automate the generation of clear, structured, and standardized documentation from source code using advanced AI models. It empowers developers to focus on building software while ensuring comprehensive documentation is consistently maintained.

Workflow

1. **Input: Source Code + Comments**
    - DocGenAI integrates with repositories (e.g., GitHub/ Gitlab).
    - It scans code written in languages like python, java, or JavaScript.
    - NLP models (e.g,. BERT, CodeBert) parse fuction/ method signatures, docstrings and inline comments.
2. **Processing: AI-Powered Documentation**

- A transformer-based engine (e.g., GPT or T5 fine-tuned on technical writing) Generates:

Function/class description

Parameter definitions

Return values

Example Usage

- Converts technical content into human-readable Markdown or HTML

3. **Output: Ready-to-use Documentation**

- Automatically produces:
  README.md
  API reference pages
  Swagger-compatible outputs (for RESTful APIs)

- Optional web-hosted docs or integration with static site generators (e.g., MkDocs, Sphinx).

4. **Validation: Human-in-the-Loop Editing**

- Developers can refine outputs via a user-friendly web interface.
- AI Offers suggestions (e.g., clarify wording, add missing examples).
- Git-based versioning of changes ensures traceability.

Impact

- **Time Savings**: Reduces documentation time by up to 70%, improving developer productivity.
- **Consistency:** Enforces uniform documentation standards across contributors and teams.
- **Scalability:** Handles large, complex codebases efficiently.
- **Career Relevance**: Promotes better code quality and collaboration, key for modern software engineers.

DocGenAI bridges the gap between code and communication, making professional-grade documentation effortless and intelligent.