Walchand College of Engineering
(Government Aided Autonomous Institution)
Vishrambag, Sangli, 416415

1947

Department of Computer Science & Engineering

Presentation on

INSPECT-AI

AY:2025-26

Under the guidance of
Dr. B.F.Momin

Date: 18/09/2025

Team Members:
1. 22510103 : Shreeharsh Shinde
2. 22510090 : Pranav Surve
3. 22510092 : Vrushali Sangale
4. 22510008 : Shravani Joshi

Agenda

Abstract

Software code reviews are essential for ensuring quality, but traditional processes are often slow, repetitive, and inconsistent. Existing static analysis tools can detect rule-based errors but fail to provide adaptive, context-aware feedback. Similarly, current AI-powered assistants focus more on code generation than structured review. This project, InspectAI, proposes an AI-driven early code review assistant that integrates static analysis, large language models (LLMs), and reinforcement learning directly into the developer's IDE. By analyzing code changes in real time, InspectAI generates human-like, team-specific review comments, classifies issues by severity, and continuously improves through developer feedback. The system also integrates with Jira for seamless task tracking and accountability. Through early defect detection, adaptive learning, and IDE-level integration, InspectAI aims to reduce trivial review overhead, minimize CI/CD pipeline failures, and enhance developer productivity while allowing human reviewers to focus on higher-level software design and architecture

Problem statement

Traditional code reviews are slow and inconsistent, and existing tools provide limited or generic feedback. InspectAI offers real-time, AI-driven, IDE-integrated reviews that detect issues early, prioritize by severity, and adapt to team practices.

Proposed Solution

InspectAI, is an AI-driven code review assistant that integrates directly into the developer's IDE. It combines static analysis, large language models (LLMs), and reinforcement learning to analyze code changes in real time, generate human-like, team-specific review comments, classify issues by severity, and continuously improve through developer feedback. Additionally, it integrates with Jira for seamless task tracking, enabling early defect detection, reducing trivial review overhead, and enhancing developer productivity while allowing human reviewers to focus on higher-level design and architecture.

Objectives

1. To automate detection of code issues (lint errors, bugs, style mismatches) before commit, reducing pipeline failures.
2. To provide human-like, team-adapted review comments by learning from past pull requests and feedback.
3. To use AI for classifying issues by severity, suggesting fixes, and continuously improving through reviewer feedback.
4. To implement seamless Jira integration for efficient task tracking and project management.
5. To carry out comparative study of implemented system with existing system.

Methodology

Requirement Analysis
Identify challenges in traditional code reviews: slow feedback, inconsistency, and trivial overhead.
Gather team-specific coding standards, review workflows, and integration needs (IDE, Jira).

System Design
Design the architecture combining Static Analysis, LLMs, and Reinforcement Learning.
Plan IDE integration for real-time inline feedback and Jira integration for task tracking.
Define data flow for code changes → AI review → developer feedback → model improvement.

Implementation (Coding & Integration)

- Develop modules for:
  - Real-time code monitoring in IDE.
  - Static analysis engine integration.
  - AI-powered comment generation and issue classification.
  - Reinforcement learning to adapt based on developer feedback.
  - Jira API integration for issue tracking.

Methodology

Testing

Unit testing for individual modules (code detection, static analysis, AI feedback).
Integration testing within IDE to ensure seamless, real-time review suggestions.
User acceptance testing with developers to validate relevance and usability of AI-generated comments.

Deployment

Deploy InspectAI as an IDE plugin or extension.
Configure Jira integration for live task tracking.

Maintenance & Continuous Improvement

Monitor AI performance and developer feedback.
Update models using reinforcement learning to improve context-awareness and team-specific relevance.
Refine static analysis rules and workflow integration as coding standards evolve.

Tech Stack

- **Languages & Frameworks:**
  - Python 3.10+ (Core development & AI integration), FastAPI (High-performance REST API backend)
- **Machine Learning / AI:**
  - PyTorch (Model training & inference), Hugging Face Transformers (Pre-trained model support), CodeT5 / CodeBERT / Code LLaMA (Code understanding & AI-based review generation)
- **Tools:**
  - flake8, pylint (Python code quality), ESLint (JS/TS linting), Bandit (Security checks), Docker (Containerization & scalability), GitHub/GitLab + Actions (Version control & CI/CD), PyGithub & Jira API wrappers (Integration & automation)
- **Storage & Deployment:**
  - MongoDB (Review data & feedback storage), AWS EC2 / Google Cloud Run / Azure (Cloud deployment), pandas & NumPy (Data preprocessing & analytics)

Algorithms

1. Transformer-based architectures (CodeBERT, CodeT5, Code LLaMA) for code understanding
2. Retrieval-Augmented Generation (RAG) for contextual responses
3. Reinforcement Learning (with developer feedback loop) for model fine-tuning
4. Classification algorithms for issue severity (Critical / Warning / Suggestion)

Techniques

- Requirement analysis and benchmarking of static analyzers.
- Hybrid reporting (combining AI-generated insights with static analysis outputs).
- Fine-tuning of pre-trained transformer models on curated PR comments and code diffs.
- Contextual feedback generation (logic flaws, naming inconsistencies, security risks).
- Continuous learning via feedback loop (developer acceptance/rejection of suggestions).
- Cross-project validation to ensure adaptability across languages.
- CI/CD integration for automated linting and AI-based reviews.

Functional Block Diagram

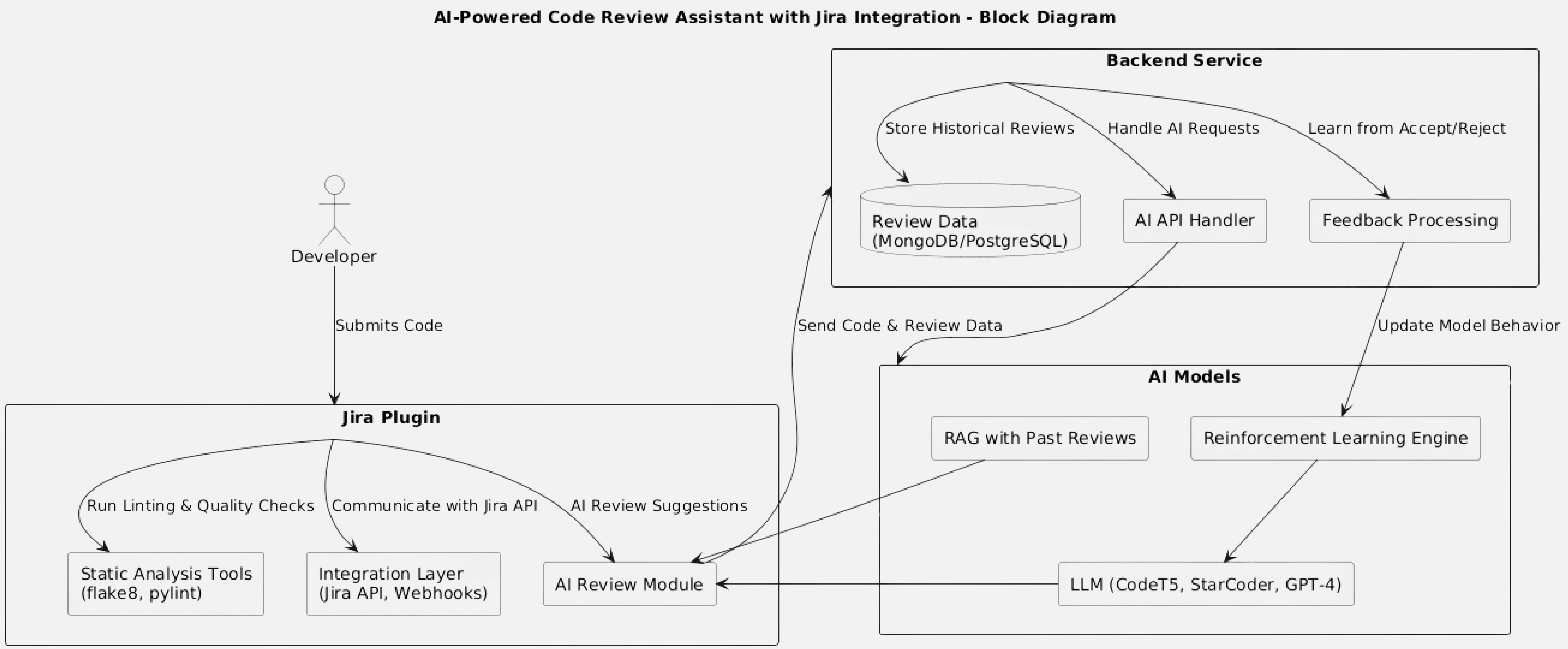
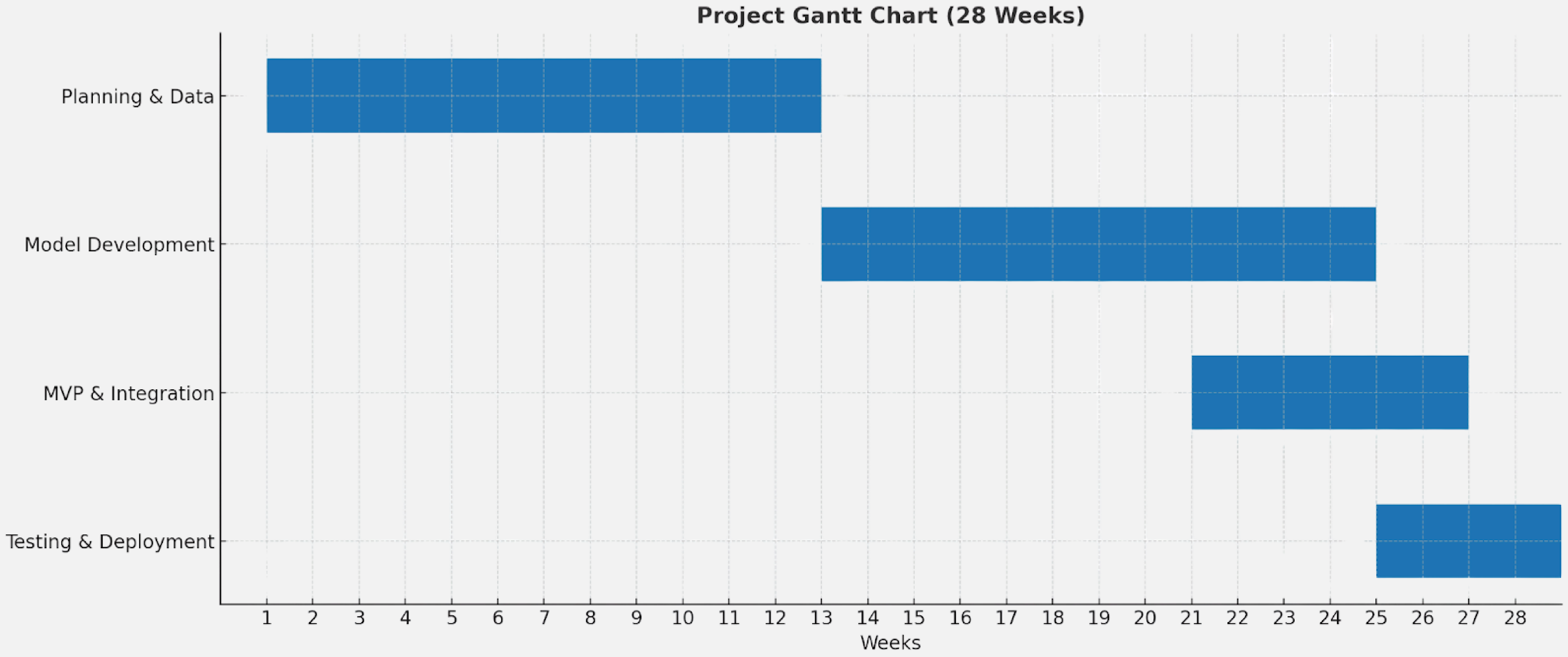

AI-Powered Code Review Assistant with Jira Integration - Block Diagram
Backend Service
Store Historical Reviews
Handle AI Requests
Learn from Accept/Reject
Review Data (MongoDB/PostgreSQL)
AI API Handler
Feedback Processing
Developer
Submits Code
Send Code & Review Data
Update Model Behavior
AI Models
RAG with Past Reviews
Reinforcement Learning Engine
LLM (CodeT5, StarCoder, GPT-4)
Jira Plugin
Run Linting & Quality Checks
Communicate with Jira API
AI Review Suggestions
Static Analysis Tools (flake8, pylint)
Integration Layer (Jira API, Webhooks)
AI Review Module

Figure: Block diagram

Project Gantt Chart


Project Gantt Chart (28 Weeks)
Planning & Data
Model Development
MVP & Integration
Testing & Deployment
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
Weeks

Conclusion

- **AI-powered code reviews**: Automates and enhances the review process using static analysis + LLMs
- **Real-time feedback**: Provides context-aware suggestions directly in the IDE
- **Reduced overhead**: Minimizes trivial review efforts and CI/CD pipeline failures
- **Developer productivity**: Streamlines workflows and saves time
- **Continuous learning**: Adapts based on developer interactions for better accuracy
- **Jira integration**: Ensures accountability and smooth issue tracking
- **Team collaboration**: Strengthens communication and review consistency
- **AI-driven automation**: Streamlines modern software development lifecycles

References

[1] Chen, M., Tworek, J., Jun, H., Yuan, Q., de Oliveira Pinto, H. P., Kaplan, J., ... & Zaremba, W. (2021). Evaluating large language models trained on code. arXiv preprint arXiv:2107.03374. https://arxiv.org/abs/2107.03374
[2] Johnson, B., Song, Y., Murphy-Hill, E., & Bowdidge, R. (2013). Why don't software developers use static analysis tools to find bugs?. Proceedings of the 2013 International Conference on Software Engineering, 672–681. IEEE. https://doi.org/10.1109/ICSE.2013.6606613
[3] Le Goues, C., Brun, Y., Devanbu, P., Ernst, M. D., Just, R., & Smyth, C. (2019). Effectiveness of automated program repair: A report on the repairnator project. Communications of the ACM, 62(12), 56–65. https://doi.org/10.1145/3318162
[4] Liu, K., Wang, Y., Chen, Y., & Lou, J. (2022). Active learning for AI-assisted code review. arXiv preprint arXiv:2206.12345. https://arxiv.org/abs/2206.12345
[5] Rozière, B., Lachaux, M. A., Chanussot, L., & Lample, G. (2023). Code LLaMA: Open foundation models for code. arXiv preprint arXiv:2308.12950. https://arxiv.org/abs/2308.12950
[6] Vaithilingam, P., Zhang, T., & Glassman, E. L. (2022). Expectations vs. experience: Evaluating the usability of code generation tools powered by large language models. Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems, 1–12. ACM. https://doi.org/10.1145/3491102.3517707
[7] Zampetti, F., Di Penta, M., & Oliveto, R. (2017). How developers use the review process in GitHub. Proceedings of the 2017 IEEE International Conference on Software Maintenance and Evolution (ICSME), 46–56. IEEE.
https://doi.org/10.1109/ICSME.2017.14

Thank You