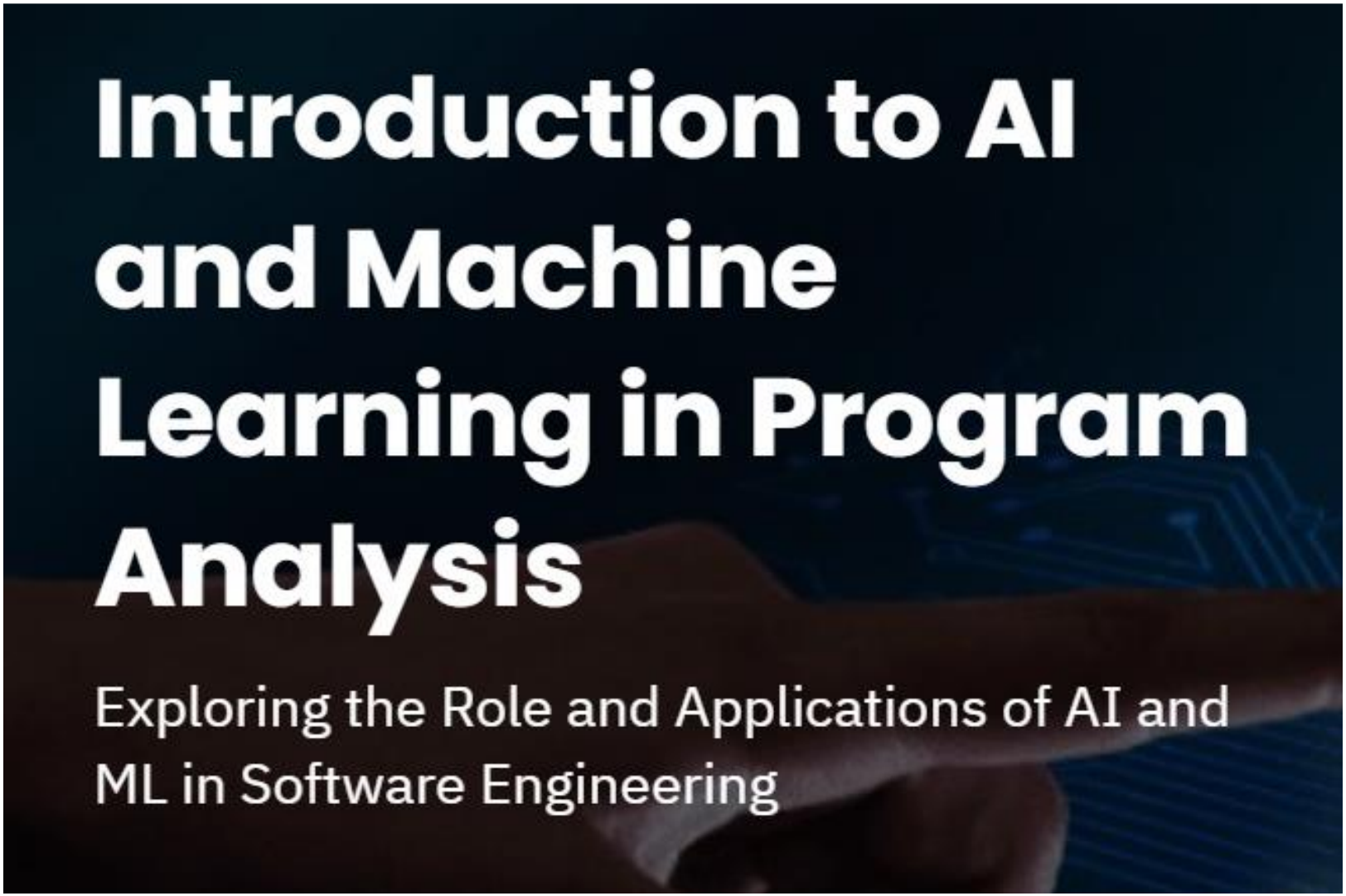
IT5450- NGUYÊN LÝ VÀ KỸ THUẬT PHÂN TÍCH CHƯƠNG TRÌNH

(Principle and Technique of Program Analysis)

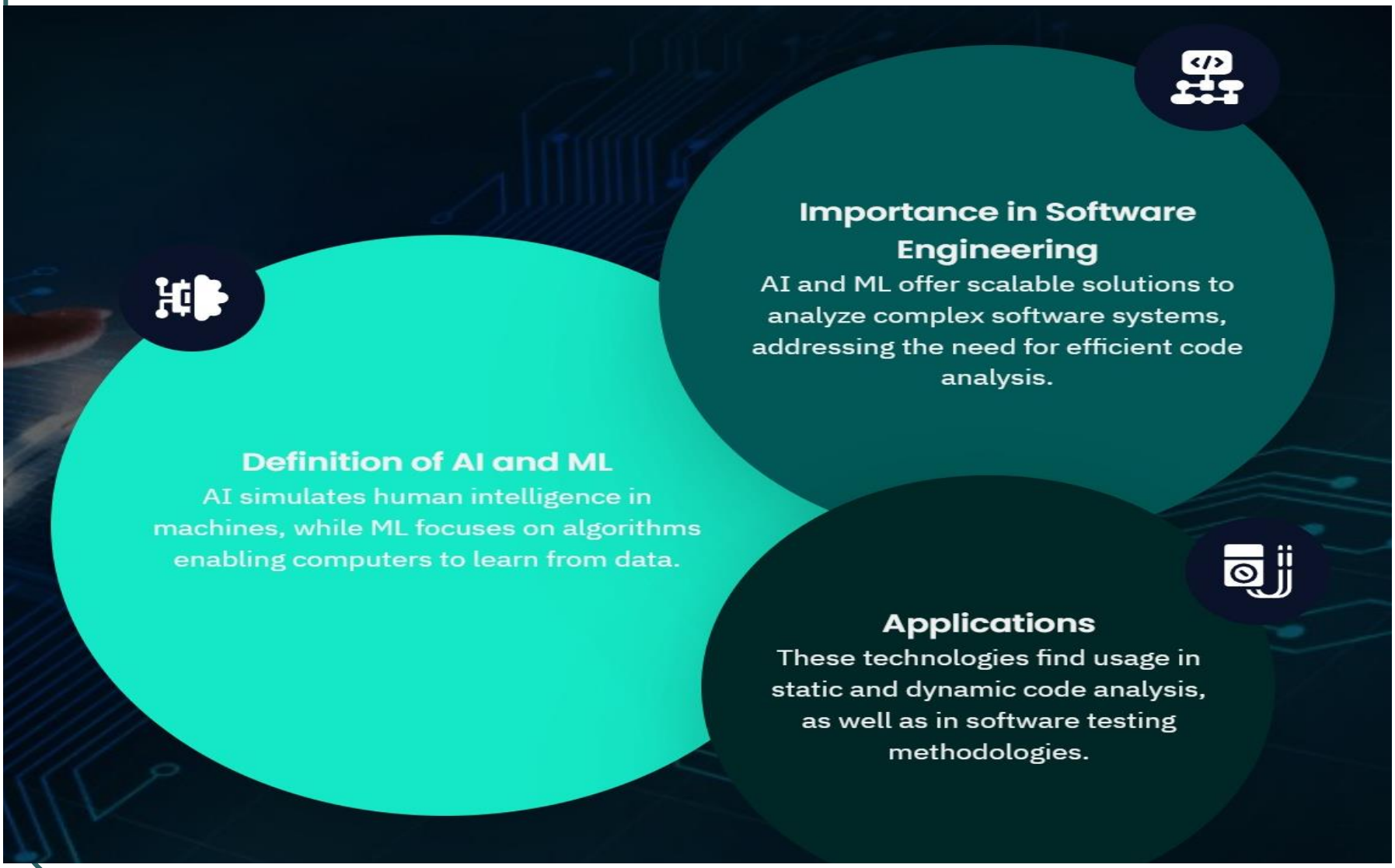
AY 2025-2026

Giảng viên: PGS. TS.Huỳnh Quyết Thắng
Khoa Khoa học máy tính
Trường Công nghệ thông tin và Truyền thông
www.soict.hust.edu.vn/~thanghq

Chapter 4


Introduction to AI and Machine Learning in Program Analysis
Exploring the Role and Applications of AI and ML in Software Engineering

Chapter 4. AI and ML in Program Analysis



AI and Machine Learning in Program Analysis

- Bug Detection & Vulnerability Analysis
  - Goal: Detect software bugs or security flaws automatically.
  - ML Techniques:
    - Supervised learning using labeled bug datasets (e.g., CodeXGLUE, Juliet).
    - Graph Neural Networks (GNNs) on program representations (ASTs, CFGs, PDGs).
    - Transformer-based models (e.g., CodeBERT, GraphCodeBERT, CodeT5, GPT-4) trained on source code.Examples:DeepBugs (Deep Learning for detecting common bug patterns).VulDeePecker, SySeVR, or ReVeal (for vulnerability detection).
- Code Representation LearningGoal:
  - Represent source code in a vector space capturing syntax & semantics.
  - Approaches:
    - Token-level embeddings (Word2Vec, FastText on code).
    - Structural embeddings from ASTs or graphs (e.g., GGNN, Code2Vec, Code2Seq).
    - Pretrained large language models (LLMs) like CodeBERT, Codex, or GPT-4.

Chapter 4. AI and ML in Program Analysis

The Role of Machine Learning in Static Code Analysis

01 **Automating Detection**

ML algorithms can identify code smells, vulnerabilities, and potential bugs by learning from historical data.

02 **Improving Accuracy**

Techniques like supervised learning help create models that predict issues more accurately than traditional methods.

03 **Case Study**

A study by Microsoft showed a 50% reduction in false positives in static analysis when integrating ML techniques.

AI and Machine Learning in Program Analysis

- Program Repair and Synthesis

  Goal: Automatically fix or generate code.

  Techniques:
  - Neural program repair: learning from pairs of buggy and fixed code.
  - LLMs for automatic code generation and correction (e.g., Codex, AlphaCode).

  Examples:
  - Sequence-to-sequence repair (e.g., Tufano et al., 2019).
  - LLM-based tools like GitHub Copilot or ChatGPT's code completion.

- Performance and Energy Optimization

  Goal: Learn to optimize compilation, memory, or performance parameters.

  Techniques:
  - Reinforcement learning for compiler optimization (e.g., DeepTune, NeuroVectorizer).
  - Predictive models for choosing optimization flags.ML-based autotuners in systems like TensorFlow XLA, LLVM.

Chapter 4. AI and ML in Program Analysis

Dynamic Analysis and Machine Learning Techniques

Leveraging real-time monitoring and predictive maintenance in AI-driven program analysis.

01

Real-time Monitoring

ML can analyze application behavior in real time, identifying anomalies that indicate bugs or security threats.

02

Predictive Maintenance

By learning from user interactions, ML models can predict when components are likely to fail.

03

Example: Google

Google uses ML techniques for dynamic analysis to enhance their cloud services, resulting in improved uptime.

Chapter 4. AI and ML in Program Analysis

Enhancing Code Reviews with AI

Leveraging AI for Improved Code Quality and Efficiency

Automated Suggestions

AI provides developers with code suggestions based on best practices learned from vast repositories.

Learning from Reviews

ML algorithms analyze previous code reviews to enhance future suggestions and identify common pitfalls.

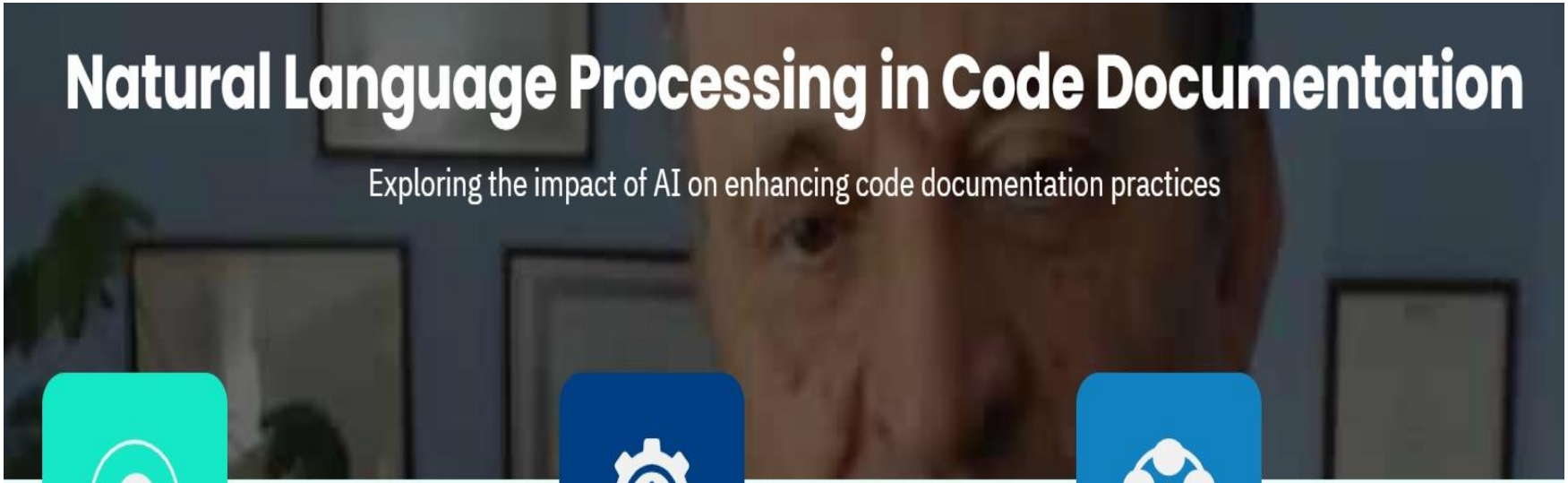
Example: GitHub's Copilot

GitHub's Copilot utilizes AI to suggest code snippets, improving developer efficiency by up to 30%.

AI and Machine Learning in Program Analysis

- Software Maintenance and Comprehension
  - Tasks: Code summarization, documentation, refactoring suggestion.
  - Approaches:
    - Transformer-based summarization (CodeT5, CodeBERT).
    - ML models to predict code quality, readability, or maintainability.
- Dynamic Analysis with AI
  - Goal: Improve fuzzing, runtime monitoring, and test generation.
  - Techniques:
    - Deep reinforcement learning for smarter fuzzing (e.g., AFL++, NEUZZ).
    - ML-guided input prioritization for dynamic testing.
    - Predictive models for crash triaging and exploitability assessment.

Chapter 4. AI and ML in Program Analysis



Understanding Code Context

NLP algorithms analyze comments and documentation to ensure alignment with actual code functionality.

Automated Generation

AI tools facilitate the generation of documentation directly from the codebase, alleviating the documentation workload for developers.

Case Study

A recent study revealed that automated documentation tools enhanced documentation quality by 40%, demonstrating significant improvement.

Chapter 4. AI and ML in Program Analysis

AI-Powered Testing Strategies

Leveraging AI and ML for Enhanced Software Testing Efficiency

01

Test Case Generation

ML can create test cases based on user behavior patterns and code changes.

02

Predictive Analysis for Testing

AI models can predict which parts of the software are likely to fail, focusing testing efforts where they are most needed.

03

Example: Facebook

Facebook uses AI-driven testing strategies, resulting in a 20% increase in bug detection.

Chapter 4. AI and ML in Program Analysis

Automating Routine Tasks

AI tools can handle repetitive maintenance tasks, freeing developers for higher-value work.

Predictive Maintenance

ML models can predict when maintenance is needed, reducing downtime and costs.

Case Study: IBM's Watson

IBM's Watson has been utilized in maintaining enterprise applications, leading to a 40% reduction in maintenance costs.

Chapter 4. AI and ML in Program Analysis

Security Enhancements through AI and ML

Leveraging AI and Machine Learning for Improved Security Measures

01

Anomaly Detection

ML algorithms learn to identify unusual patterns indicating potential security breaches.

02

Vulnerability Assessment

AI analyzes code for known vulnerabilities, providing suggestions for effective fixes.

Chapter 4. AI and ML in Program Analysis

Future Trends in AI and Machine Learning for Program Analysis

Increased Automation

More processes will be automated, leading to faster development cycles.

Enhanced Collaboration Tools

AI will facilitate better collaboration among developers through intelligent suggestions.

Ethical AI Development

As AI use grows, there will be a focus on developing ethical guidelines for AI in software engineering.

AI and Machine Learning in Program Analysis

- **Challenges**
  - **Data Quality & Labeling** – Hard to get large, accurate labeled datasets for bugs or vulnerabilities.
  - **Generalization** – Models trained on certain projects often fail on unseen codebases.
  - **Explainability** – ML models often behave as “black boxes”; explainable AI (XAI) for code is an active area.
  - **Integration with Formal Methods** – Balancing soundness (formal guarantees) with ML’s flexibility.
  - **Scalability** – Learning-based models must handle large codebases and complex control/data flows.

- **Promising Future Directions**
  - **Hybrid Analysis:** Combine symbolic execution + neural reasoning (Neuro-symbolic program analysis).
  - **AI for Secure Software Development Lifecycles (SSDLC)** – embedding ML tools directly in CI/CD pipelines.
  - **LLM-as-an-Analyzer:** Using powerful models (like GPT-5 or specialized code LLMs) to reason about semantics and security.
  - **Learning from Execution Traces:** Using runtime data to improve static predictions.
  - **Formal Verification + LLM:** Leveraging LLMs to assist theorem provers or constraint solvers.

Top 15 recent papers (2024–2025)

- **AutoCodeRover: Autonomous Program Improvement** — LLM + code-search + test-based fault localization to autonomously solve GitHub issues (program repair & feature addition); reports good efficacy and low cost on SWE-bench-lite. arXiv
- **OpenCodeInterpreter: Integrating Code Generation with Execution and Refinement** — open-source "code interpreter" family: generate → execute → refine loop with a Code-Feedback dataset; closes gap with proprietary code interpreters. arXiv
- **LDB — Debug like a Human: A Large Language Model Debugger via Verifying Runtime Execution Step-by-step** — LLM debugger that segments execution into basic blocks and uses runtime traces to verify & iteratively fix code; improves debugging on standard benchmarks. arXiv
- **AutoSpec (Enchanting Program Specification Synthesis by LLMs using Static Analysis & Verification)** — uses static analysis + LLMs to synthesize specifications (loop invariants, etc.) and validate them incrementally to enable automated verification of tricky programs. arXiv
- **LLM4PR — Towards Large Language Model Aided Program Refinement** — framework combining refinement calculus/formal methods with LLM prompting to generate code that can be verified against refinement conditions. arXiv
- **Towards AI-Assisted Synthesis of Verified Dafny Methods** — experiments and prompting strategies showing LLMs (GPT-4, PaLM-2) can be harnessed to synthesize verified Dafny methods; retrieval-augmented CoT boosts verified synthesis rates. arXiv

Top 15 recent papers (2024–2025)

- **Fixing 162 out of 337 Bugs for $0.42 Each using ChatGPT (ISSTA 2024)** — empirical study showing conversational use of ChatGPT can produce many correct patches cheaply; documents strengths and failure modes of LLM-driven APR. Lingming Zhang's Homepage
- **Towards Reliable Evaluation of Neural Program Repair with Natural Robustness Testing** — shows common APR robustness tests include unnatural transformations; proposes human-validated “naturalness” testing and an LLM metric to scale robustness checks. arXiv
- **A Systematic Literature Review on Large Language Models for Automated Program Repair (Zhang et al., 2024)** — a comprehensive SLR (2020–2024) summarizing 127 papers: usage patterns, benchmarks, open challenges and best practices for LLM-based APR. arXiv
- **RePair: Automated Program Repair with Process-based Feedback (ACL Findings 2024)** — process-supervision & iterative feedback approach (reward model + stepwise generation) that helps smaller LMs match performance of large closed-source models on APR. ACL Anthology
- **Peer-aided Repairer (PaR): Empowering LLMs to Repair Advanced Student Assignments** — uses peer-solution selection + multi-source prompts to significantly improve repair rates on advanced assignment datasets (Defects4DS). arXiv

Top 15 recent papers (2024–2025)

- **Revisiting Unnaturalness for Automated Program Repair in the Era of LLMs** — proposes using LLM entropy ("naturalness") as a complementary signal for fault localization, patch ranking and to mitigate test-suite overfitting / data-leakage concerns. arXiv
- **Graph in Graph Neural Network (GIG)** — a 2024 advance in GNN design that can process graphs whose nodes are themselves graphs (useful for richer program / AST/PDG representations). Potentially impactful for program-analysis GNN models. arXiv
- **Automated Program Repair: Emerging trends pose and expose problems for benchmarks (Renzullo et al., 2024)** — critical analysis of APR benchmarks and evaluation methodology in the ML era; argues for better benchmarks and evaluation practices. arXiv
- **Towards Practical and Useful Automated Program Repair (2024 survey/position)** — a broad treatment of APR's practical limitations and recommendations to make APR tools more usable/realistic in developer workflows. (useful synthesis for applied research). arXiv