SOFTWARE ENGINEERING EDUCATION IN THE ERA OF LARGE LANGUAGE MODELS: PROMISES AND PERILS

Massimiliano Di Penta
University of Sannio, Italy
dipenta@unisannio.it, @mdipenta

ABOUT ME

- Full professor, University of Sannio
- Research on software engineering
- Chaired the main software engineering conferences: ICSE 2023, ESEC/FSE 2021, ASE 2017, and others
- Associate Editor-in-Chief: IEEE Transactions on Software Engineering, Editor-in-Chief: Journal of Software Evolution and Process, Associate Editor: Empirical Software Engineering Journal

MY MAIN RESEARCH TOPICS

- Software analytics
- Software evolution
- Recommender systems for software engineers
- DevOps

MY TEACHING

- Advanced software engineering (Master)
- Video game development (Master)
- Natural language processing (Bachelor)

HOW MY SOFTWARE ENGINEERING TEACHING USED TO BE

Master course, so students know already about topics such as

- Analysis and design
- (Some) software configuration management and continuous integration

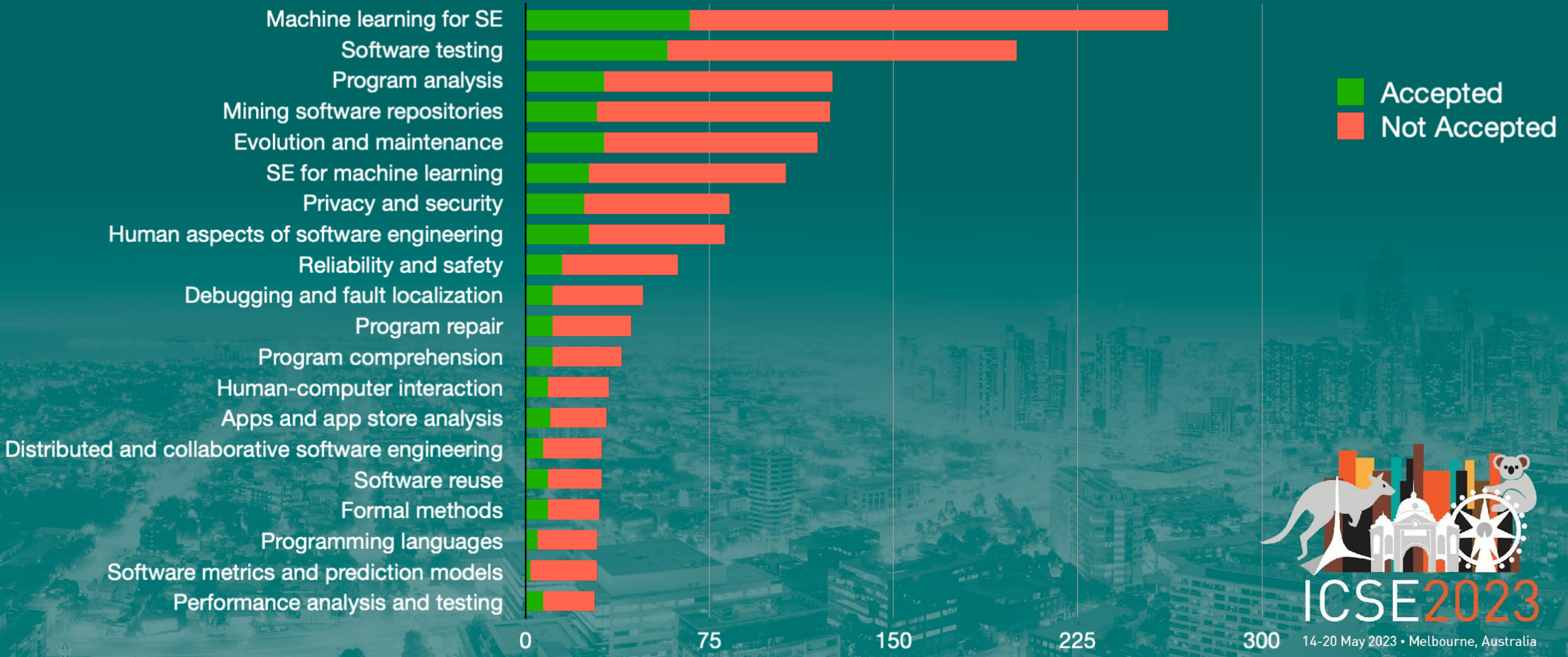
WHAT'S LEFT?

MY (OLD) COURSE GOALS

- Provide advanced software engineering contents mostly focused on software quality
- Stimulate workgroup
- Stimulate investigation on innovative topics

IN THE MEANTIME… HOW IS SOFTWARE ENGINEERING CHANGING?

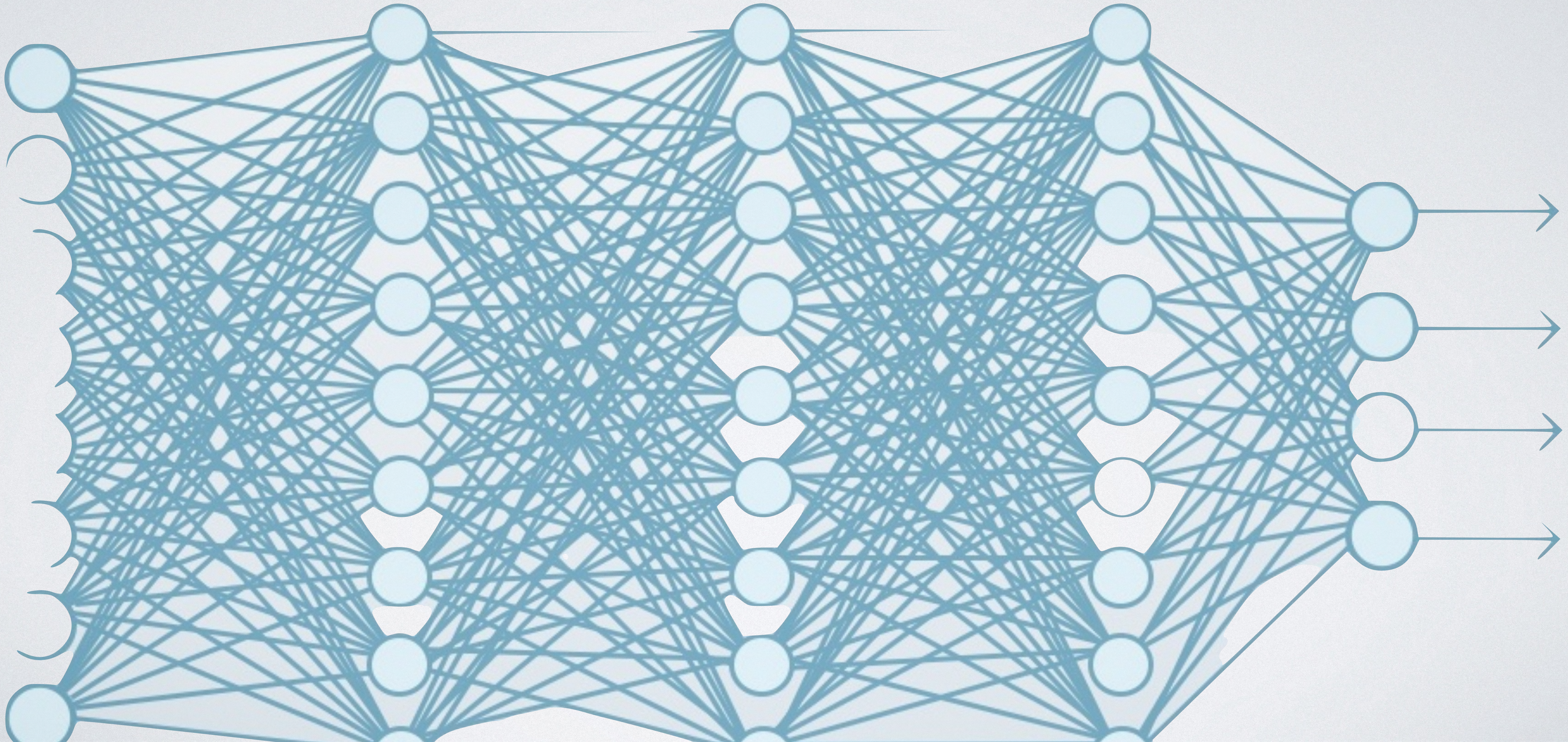
ICSE 2023 submitted/accepted papers by topic (top 20)



ICSE2023
14-20 May 2023 • Melbourne, Australia

ICSE 2023 accepted papers at a glance

THE RISE OF DEEP LEARNING APPLICATIONS TO SE…



BETTER HARDWARE…

MORE DATA...

MORE (CURATED) DATA

- Torrents, e.g., GitTorrent, SOTorrent, TravisTorrent
- Initiatives like World of Code
- Increase of sharing (large) datasets with papers
- Publication reward for dataset and reproducibility

PRETRAINED MODELS

- SE researchers can focus on
  - Relatively **small fine tuning from problem specific data**
- No need to bother with large, expensive model training
- ICSE 2023 had a full session on this

APPLICATION EXAMPLES…

An Empirical Study on the Usage of Transformer Models for Code Completion

Matteo Ciniselli, Nathan Cooper, Luca Pascarella, Antonio Mastropaolo, Emad Aghajani, Denys Poshyvanyk, Massimiliano Di Penta, and Gabriele Bavota

Abstract—Code completion aims at speeding up code writing by predicting the next code token(s) the developer is likely to write. Works in this field focused on improving the accuracy of the generated predictions, with substantial leaps forward made possible by deep learning (DL) models. However, code completion techniques are mostly evaluated in the scenario of predicting the next token to type, with few exceptions pushing the boundaries to the prediction of an entire code statement. Thus, little is known about the performance of state-of-the-art code completion approaches in more challenging scenarios in which, for example, an entire code block must be generated. We present a large-scale study exploring the capabilities of state-of-the-art Transformer-based models in supporting code completion at different granularity levels, including single tokens, one or multiple entire statements, up to entire code blocks (*e.g.,* the iterated block of a *for* loop). W
Text-To-Text Tra
particular the T5
model to guess

Index Terms—

Neural Transfer Learning for Repairing Security Vulnerabilities in C Code

Zimin Chen, Steve Kommrusch, and Martin Monperrus

Abstract—In this paper, we address the problem of automatic repair of software vulnerabilities with deep learning. The major problem with data-driven vulnerability repair is that the few existing datasets of known confirmed vulnerabilities consist of only a few thousand examples. However, training a deep learning model often requires hundreds of thousands of examples. In this work, we leverage the intuition that the bug fixing task and the vulnerability fixing task are related and that the knowledge learned from bug fixes can be transferred to fixing vulnerabilities. In the machine learning community, this technique is called transfer learning. In this paper, we
propose an approach for repa
on a large bug fix corpus and
we show that a model trained
improves the ability to repair
trained with a denoising task
well for repairing security vuln
ty fi

Using Pre-Trained Models to Boost Code Review Automation

Rosalia Tufano
SEART @ Software Institute
Università della Svizzera italiana
Switzerland

Simone Masiero
SEART @ Software Institute
Università della Svizzera italiana
Switzerland

Antonio Mastropaolo
SEART @ Software Institute
Università della Svizzera italiana
Switzerland

Luca Pascarella
SEART @ Software Institute
Università della Svizzera italiana
Switzerland

Denys Poshyvanyk
SEMERU @ Computer Science Department
William and Mary
USA

Gabriele Bavota
SEART @ Software Institute
Università della Svizzera italiana
Switzerland

ABSTRACT

Code review is a practice widely adopted in open source and industrial projects. Given the non-negligible cost of such a process, researchers started investigating the possibility of automating specific code review tasks. We recently proposed Deep Learning (DL) models targeting the automation of two tasks: the first model takes as input a code submitted for review and implements in it changes likely to be recommended by a reviewer; the second takes as input the submitted code and a reviewer comment posted in natural language and automatically implements the change required by the reviewer. While the preliminary results we achieved are encouraging, both models had been tested in rather simple code review

Given the non-negligible cost of code review, we recently proposed the automation of specific code review tasks: The goal is not to replace developers, but to help them save time in two scenarios. The first is that of a contributor (*i.e.,* the developer submitting the code for review) who wants to receive a rapid feedback about the code they wrote before submitting it for review. The feedback is provided by a Deep Learning (DL) model trained to take as input the code to submit for review C_s and provide as output a revised version of C_s (*i.e.,* C_r) implementing code changes that are likely to be recommended by a reviewer.

The second scenario concerns the reviewer(s) involved in the process: a DL model is trained to take as input (i) the code C_s

DEAR: A Novel Deep Learning-based Approach for Automated Program Repair

Yi Li
New Jersey Inst. of Technology
New Jersey, USA
yl622@njit.edu

Shaohua Wang*
New Jersey Inst. of Technology
New Jersey, USA
davidsw@njit.edu

Tien N. Nguyen
University of Texas at Dallas
Texas, USA
tien.n.nguyen@utdallas.edu

ABSTRACT

The existing deep learning (DL)-based automated program repair (APR) models are limited in fixing general software defects. We present DEAR, a DL-based approach that supports fixing for the general bugs that require dependent changes at once to one or multiple consecutive statements in one or multiple hunks of code. We first design a novel fault localization (FL) technique for multi-hunk, multi-statement fixes that combines traditional spectrum-based (SB) FL with deep learning and data-flow analysis. It takes the buggy statements returned by the SBFL model, detects the buggy hunks to be fixed at once, and expands a buggy statement s in a hunk to include other suspicious statements around s. We design a two-tier, tree-based LSTM model that incorporates cycle training and uses a divide-and-conquer strategy to learn proper code transformations for fixing multiple statements in the suitable fixing context consisting of surrounding subtrees. We conducted several experiments to evaluate DEAR on three datasets: Defects4J (395 bugs), BigFix (+26k bugs), and CPatMiner (+44k bugs). On Defects4J dataset, DEAR outperforms the baselines from 42%–683% in terms of the number of auto-fixed bugs with only the top-1 patches. On BigFix dataset,

1 INTRODUCTION

Researchers have proposed several approaches to help developers in automatically identifying and fixing the defects in software. Such approaches are referred to as *automated program repair* (APR). The APR approaches have been leveraging various techniques in the areas of *search-based software engineering, software mining, machine learning (ML)*, and *deep learning (DL).*

For *search-based approaches* [9, 10, 24, 30], a search strategy is performed in the space of potential solutions produced by mutating the buggy code via operators. Other approaches use software mining to *mine and learn fixing patterns* from prior bug fixes [15, 17, 19, 20, 27] or similar code [28, 32]. Fixing patterns are at the source code level [19, 20] or at the change level [13, 16, 40]. *Machine learning* has been used to mine fixing patterns and the candidate fixes are ranked according to their likelihoods [21, 22, 33]. While some DL-based APR approaches learn similar fixes [11, 41, 42], other ones use machine translation or neural network models with various code abstractions to generate patches [5, 6, 12, 18, 35, 38, 39].

Despite their successes, the state-of-the-art DL-based APR approaches are still limited in fixing the *general defects, which involve*

SURVEYS IN THE AREA

Large Language Models for Software Engineering: A Systematic Literature Review

XINYI HOU*, Huazhong University of Science and Technology, China
YANJIE ZHAO*, Monash University, Australia
YUE LIU, Monash University, Australia
ZHOU YANG, Singapore Management University, Singapore
KAILONG WANG, Huazhong University of Science and Technology, China
LI LI, Beihang University, China
XIAPU LUO, The Hong Kong Polytechnic University, China
DAVID LO, Singapore Management University, Singapore
JOHN GRUNDY, Monash University, Australia
HAOYU WANG†, Huazhong University of Science and Technology, China

Large Language Models (LLMs) have significantly impacted numerous domains, including Software Engineering (SE). Many recent publications have explored LLMs applied to various SE tasks. Nevertheless, a comprehensive understanding of the application, effects, and possible limitations of LLMs on SE is still in its early stages. To bridge this gap, we conducted a systematic literature review on LLM4SE, with a particular focus on understanding how LLMs can be exploited to optimize processes and outcomes. We collect and analyze 229 research papers from 2017 to 2023 to answer four key research questions (RQs). In RQ1, we categorize different LLMs that have been employed in SE tasks, characterizing their distinctive features and uses. In RQ2, we analyze the methods used in data collection, preprocessing, and application highlighting the role of well-curated datasets for successful LLM for SE implementation. RQ3 investigates the strategies employed to optimize and evaluate the performance of LLMs in SE. Finally, RQ4 examines the specific SE tasks where LLMs have shown success to date, illustrating their practical contributions to the field. From the answers to these RQs, we discuss the current state-of-the-art and trends, identifying gaps in existing research, and flagging promising areas for future study.

CCS Concepts: • **General and reference** → **Surveys and overviews**; • **Software and its engineering** → **Software development techniques**; • **Computing methodologies** → **Artificial intelligence**.

Additional Key Words and Phrases: Software Engineering, Large Language Model, Survey

Large Language Models for Software Engineering: Survey and Open Problems

Angela Fan
Generative AI Team
Meta Platforms Inc.
New York, NY, USA

Beliz Gokkaya
PyTorch Team
Meta Platforms Inc.
Menlo Park, CA, USA

Mark Harman
Instagram Product Foundation
Meta Platforms Inc.
London, UK

Mitya Lyubarskiy
Developer Infrastructure
Meta Platforms Inc.
London, UK

Shubho Sengupta
FAIR
Meta Platforms Inc.
Menlo Park, CA, USA

Shin Yoo
School of Computing
KAIST
Daejeon, Korea

Jie M. Zhang
Department of Informatics
King's College London
London, UK

***Abstract*—This paper provides a survey of the emerging area of Large Language Models (LLMs) for Software Engineering (SE). It also sets out open research challenges for the application of LLMs to technical problems faced by software engineers. LLMs' emergent properties bring novelty and creativity with applications right across the spectrum of Software Engineering activities including coding, design, requirements, repair, refactoring, performance improvement, documentation and analytics. However, these very same emergent properties also pose significant technical challenges; we need techniques that can reliably weed out incorrect solutions, such as hallucinations. Our survey reveals the pivotal role that hybrid techniques (traditional SE plus LLMs) have to play in the development and deployment of reliable, efficient and effective LLM-based SE.**

***Index Terms*—Automated Program Repair, Design, Documentation generation, Generative AI, Genetic Improvement, Human-Computer Interaction, Large Language Models, Refactoring, Requirements engineering, Search Based Software Engineering (SBSE), Software Analytics, Software Engineering Education, Software Processes, Software Maintenance and Evolution, Soft-**

trends and well-established approaches and subdisciplines within Software Engineering.

Although, overall, we find considerable grounds for optimism, there remain important technical challenges, which are likely to inform the research agenda for several years. Many authors have highlighted, both scientifically and anecdotally, that hallucination is a pervasive problem for LLMs [1] and also that it poses specific problems for LLM-based SE [2]. As with human intelligence, hallucination means that the LLM can create fictitious output. In the context of software engineering, it means that the engineering artefacts created could be incorrect, yet appear plausible; LLMs may introduce bugs.

However, unlike many other applications of LLMs, software engineers are typically blessed with automatable ground truth (software execution), against which most software engineering artefacts can be evaluated. Also, the software engineering

…EVERYTHING SEEMS TO GO SO WELL…

- Better solutions to problems handled already
- Ability to solve problems not considered in the past

NOVEMBER 30, 2022…

CODE GENERATION…

DI

Can you write a Python program that gather commits from a GithHub repository?

Yes, here's an example Python program that uses the GitHub API to gather the commit history from a repository:

python Copy code

```python
import requests

# Replace with your own values
github_username = 'your_github_username'
repo_name = 'your_repo_name'

# Set up the API endpoint and headers
api_url = f'https://api.github.com/repos/{github_username}/{repo_name}/commi
headers = {'Accept': 'application/vnd.github.v3+json'}

# Make the API request
response = requests.get(api_url, headers=headers)

# Check if the request was successful
if response.status_code == 200:
```

PROGRAM REPAIR…

DI

This function contains a bug. Can you please fix it?

def quicksort(lst, left=0, right=None):
    if right is None:
        right = len(lst) - 1
    l = left
    r = right
    if l <= r:

Sure, here's a fixed version of the quicksort function:

sql Copy code

```
def quicksort(lst, left=0, right=None):
    if right is None:
        right = len(lst) - 1
    l = left
    r = right
    if l < r:  # change condition to l < r
```

Changes:

- Change the condition in line 6 to `l < r` instead of `l <= r` to avoid an unnecessary recursive call when `l == r`.

VULNERABILITY ASSESSMENT…

V

is this code vulnerable?

```
static inline void
 batchCopyElem(batch_obj_t *pDest, batch_obj_t *pSrc) {
   memcpy(pDest, pSrc, sizeof(batch_obj_t));
 }
```

The code you provided appears to be a simple function that copies the contents of a source batch object (`pSrc`) to a destination batch object (`pDest`) using the `memcpy` function.

As written, this code alone does not seem to be inherently vulnerable. However, whether this code is vulnerable or not depends on how it is used in the larger context of the software system. Here are some things to consider:

1. If the size of `batch_obj_t` changes in the future, this function may need to be updated accordingly. If this function is not updated, it could lead to buffer overflows or other memory-related vulnerabilities.
2. The function assumes that the source and destination batch objects have the same size and structure. If this assumption is not true, it could lead to memory corruption vulnerabilities.
3. If the function is called with invalid pointers (e.g., NULL pointers), it could result in

CODE REVIEW…

DI

Can you please write a code review for this file:

```
#! /usr/bin/python3

import json
import sys
import re
import codecs
import datetime

class CommitData():
   def __init__(self,commitId,commitDate):
      self.commitId=commitId
      self.commitDate=commitDate
```

Sure, here's a code review for the given file:

- The code follows PEP8 style guide for Python code, which is great.
- The shebang `#! /usr/bin/python3` at the top of the file indicates that this script should be run with Python3, which is good to see.
- The `CommitData` class is defined, but it's not used in the code. It can be removed if it's not going to be used.

...AND EVEN MORE...

```python
def isBug(msg):
    p=re.compile("\\b(fix|fixed|fixing|bug)\\b",re.IGNORECASE)
    if p.search(msg):
            return True
    return False
```

- The `` `isBug` `` function is used to check whether a commit message is a bug fix or not. It uses a regular expression to match the words "fix", "fixed", "fixing", or "bug" in the message, which is a reasonable way to check for bug fixes. However, it could potentially match false positives if the message contains these words but not in the context of a bug fix. For example, a commit message that says "Fixed typo in documentation" would be considered a bug fix even though it's not really fixing a bug. So, it's not a perfect method for identifying bug fixes, but it's a good start.

EACH OF US HAS SURELY TRIED IT FOR MANY OTHER TASKS…

- Code completion
- Test generation
- Program translation and refactoring
- Software design and modeling
- ….

GAME OVER!

LLMS ARE INFLUENCING RESEARCH…

Novel approaches contribute by leveraging:

- Prompt tuning and chaining
- Hybrid models (e.g., Retrieval Augmented Generation)

PROMPT-TUNING IN SE

No More Fine-Tuning? An Experimental Evaluation of Prompt Tuning in Code Intelligence

Chaozheng Wang
Harbin Institute of Technology
Shenzhen, China
wangchaozheng@stu.hit.edu.cn

Yuanhang Yang
Harbin Institute of Technology
Shenzhen, China
ysngkil@gmail.com

Cuiyun Gao*
Harbin Institute of Technology
Shenzhen, China
gaocuiyun@hit.edu.cn

Yun Peng
The Chinese University of Hong Kong
Hong Kong, China
ypeng@cse.cuhk.edu.hk

Hongyu Zhang
The University of Newcastle
Newcastle, Australia
hongyu.zhang@newcastle.edu.au

Michael R. Lyu
The Chinese University of Hong Kong
Hong Kong, China
lyu@cse.cuhk.edu.hk

ABSTRACT

Pre-trained models have been shown effective in many code intelligence tasks. These models are pre-trained on large-scale unlabeled corpus and then fine-tuned in downstream tasks. However, as the inputs to pre-training and downstream tasks are in different forms, it is hard to fully explore the knowledge of pre-trained models. Besides, the performance of fine-tuning strongly relies on the amount of downstream data, while in practice, the scenarios with scarce data are common. Recent studies in the natural language processing (NLP) field show that prompt tuning, a new paradigm for tuning, alleviates the above issues and achieves promising results in various NLP tasks. In prompt tuning, the prompts inserted during tuning provide task-specific knowledge, which is especially beneficial for tasks with relatively scarce data. In this paper, we empirically evaluate the usage and effect of prompt tuning in code intelligence tasks. We conduct prompt tuning on popular pre-trained models CodeBERT and CodeT5 and experiment with three code intelligence tasks including defect prediction, code summarization, and code translation. Our experimental results show that prompt tuning

KEYWORDS

code intelligence, prompt tuning, empirical study

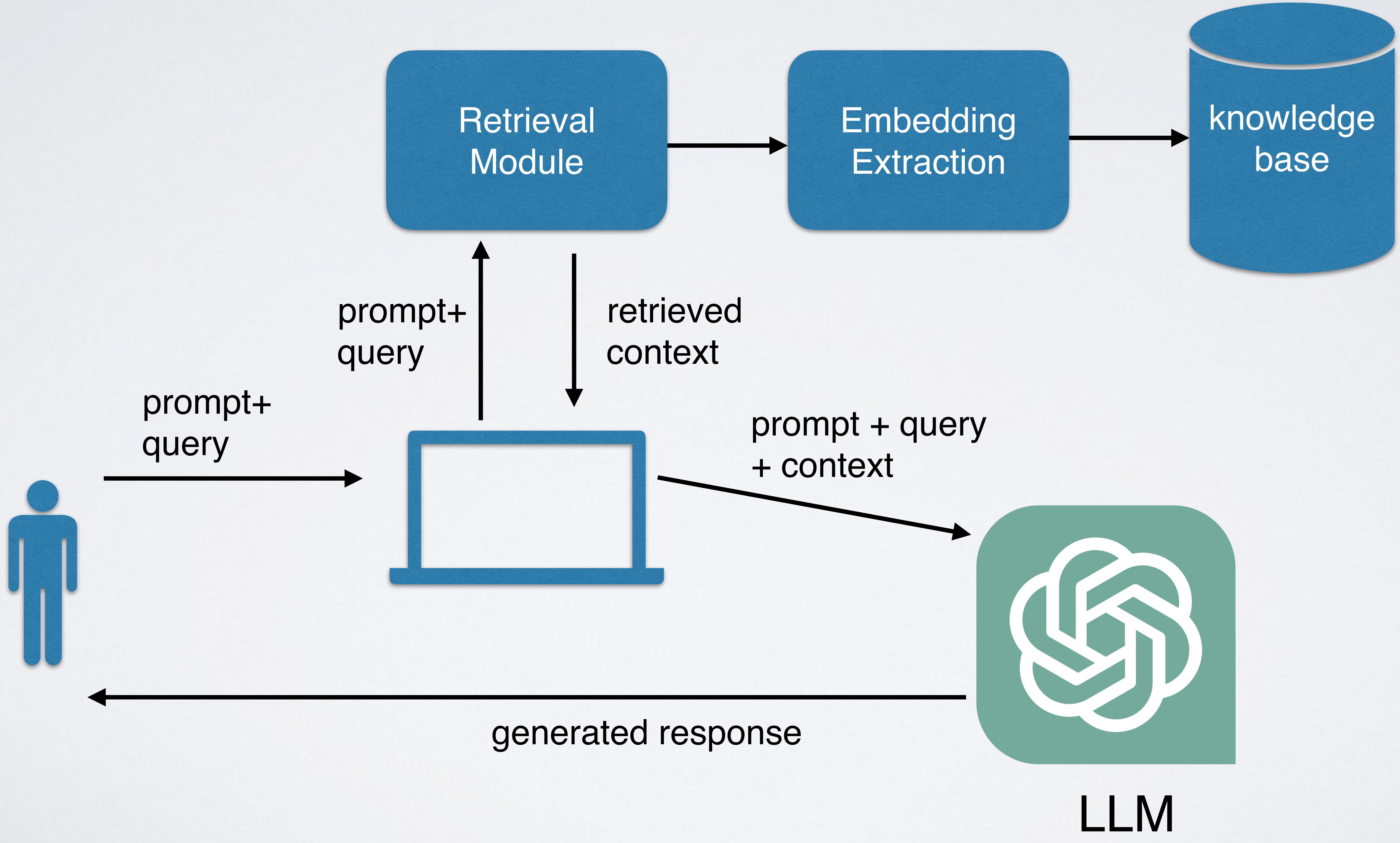
ACM Reference Format:
Chaozheng Wang, Yuanhang Yang, Cuiyun Gao, Yun Peng, Hongyu Zhang, and Michael R. Lyu. 2022. No More Fine-Tuning? An Experimental Evaluation of Prompt Tuning in Code Intelligence. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '22), November 14–18, 2022, Singapore, Singapore.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3540250.3549113

1 INTRODUCTION

Code intelligence leverages machine learning, especially deep learning (DL) techniques to mine knowledge from large-scale code corpus and build intelligent models for improving the productivity of computer programming. The state-of-the-art DL-based approaches to code intelligence exploit the *pre-training and finetuning* paradigm [1, 9, 14, 21, 56], in which language models are first pre-trained on a large unlabeled text corpora and then finetuned on downstream

- Pretrained models may originate from different types of knowledge, e.g. natural language
- Prompting reintroduce such elements during fine-tuning

RETRIEVAL AUGMENTED GENERATION (RAG)



IMPACT ON SE EDUCATION

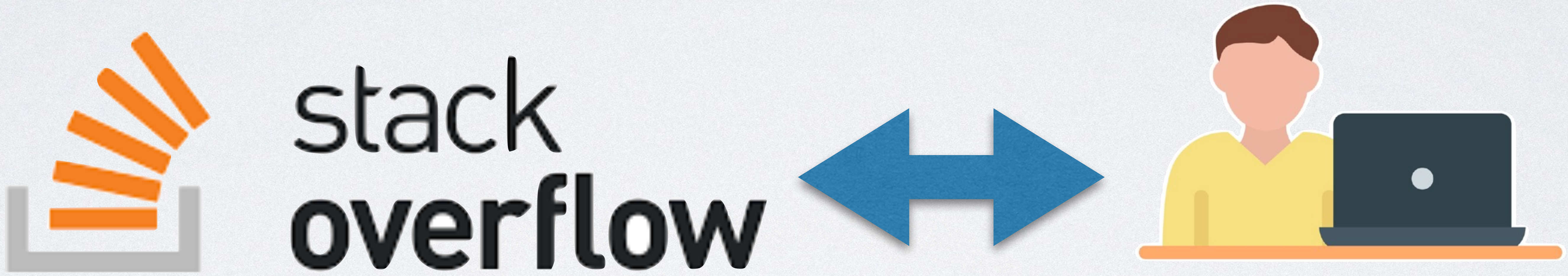

ARKING
NO
ChatGPT


ARKING
NO

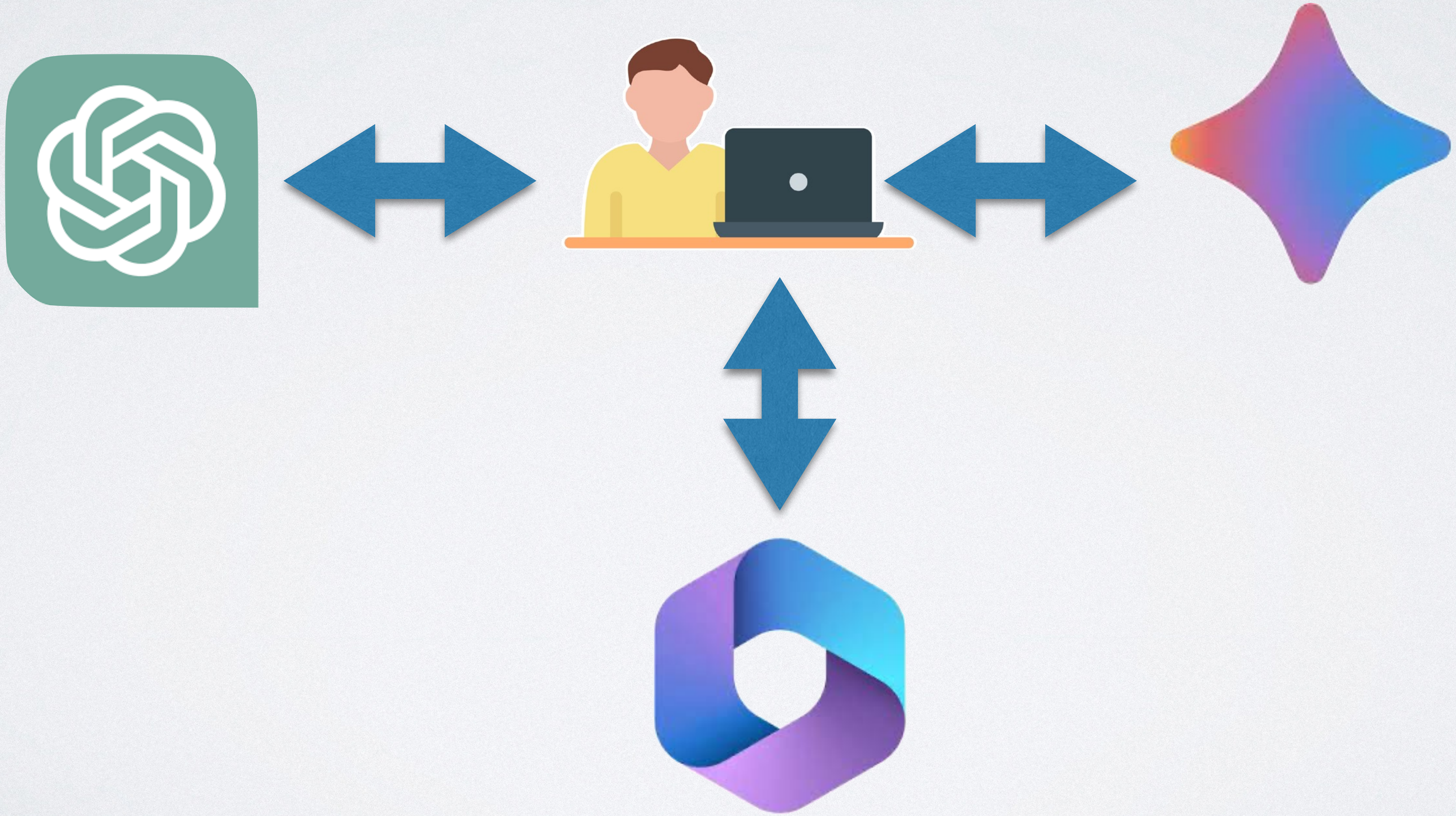
IMPACT ON SE EDUCATION

Favor a responsible, informed integration of LLMs in the software development process

FROM INTERACTING WITH STACK OVERFLOW…



TOWARD INTERACTING WITH LLMS…



INSTRUCT STUDENTS ABOUT POTENTIALS

- Applications beyond obtaining code snippets
  - Code review, refactoring, program repair, test generation, documentation generation
- Tutorials on prompt engineering and prompt chaining

- Lack of trust
- Intellectual property problems
- Hallucination

LACK OF TRUST

- Without having a source for the recommended code we can't trust it
  - This is therefore very different from conventional reuse
- We need to define suitable code review procedure for AI-generated software
  - Different from conventional code review as there's no human developer to interact with

INTELLECTUAL PROPERTY ISSUES

REUTERS®

World Business Markets Sustainability More

Lawsuit says OpenAI violated US authors' copyrights to train AI chatbot

By **Blake Brittain**

June 29, 2023 8:55 PM GMT+2 · Updated 4 months ago

Aa

https://www.reuters.com/legal/lawsuit-says-openai-violated-us-authors-copyrights-train-ai-chatbot-2023-06-29/

CONSEQUENCES WITH SOFTWARE DEVELOPMENT

- Again no traceability to recommended code
- Even if the LLM were trained with open source code with permissive licenses, lack of attribution remains a serious violation

IF I ASK CHATGPT SOME CODE…

DI

Can you please write python code to get a git log from a GitHub repository?

Sure, you can use the PyGithub library to interact with GitHub repositories and retrieve the Git log. First, you'll need to install the PyGithub library if you haven't already:

bash — Copy code

```bash
pip install PyGithub
```

Then, you can use the following Python code to get the Git log from a GitHub repository:

python — Copy code

```python
from github import Github

# Replace 'YOUR_USERNAME' and 'YOUR_ACCESS_TOKEN' with your GitHub username
# Generate a personal access token following the GitHub documentation: https
github_username = 'YOUR_USERNAME'
```

…IT HAS NO CLUE ABOUT THE LICENSE!

DI

What is the license for this code?

The code I provided is a simple example of how to interact with GitHub using the PyGithub library, and it doesn't include a specific license. Therefore, it doesn't have a separate license and falls under the default copyright laws, which means you can use it for personal or educational purposes.

If you plan to use the code in a commercial or open-source project and want to specify the terms of use, you should consider adding an appropriate license to your project. Common open-source licenses include the MIT License, Apache License, GNU General Public License (GPL), and more. You can choose a license that aligns with your project's goals and requirements.

Always ensure that you comply with the terms of use for any libraries or tools you utilize in your projects.

WHAT ABOUT SOFTWARE BILLS OF MATERIALS?

{"SPDXID":"SPDXRef-DOCUMENT","spdxVersion":"SPDX-2.3","creationInfo":{"created":"2023-10-20T07:27:34Z","creators":
["Tool: GitHub.com-Dependency-Graph"],"comment":"Exact versions could not be resolved for some packages. For
more information:
https://docs.github
.com/en/code-security/supply-chain-security/understanding-your-software-supply-chain/about-the-dependency-graph
#dependencies-included."},"name":"com.github.tensorflow/tensorflow","dataLicense":"CC0-1
.0","documentDescribes":
["SPDXRef-com.github.tensorflow-tensorflow"],"documentNamespace":"https://github
.com/tensorflow/tensorflow/dependency_graph/sbom-20811b51d6c2a1c6","packages":
[{"SPDXID":"SPDXRef-com.github.tensorflow-tensorflow","name":"com.github
.tensorflow/tensorflow","versionInfo":"","downloadLocation":"git+https://github
.com/tensorflow/tensorflow","licenseDeclared":"Apache-2
.0","filesAnalyzed":false,"supplier":"NOASSERTION","externalRefs":
[
{"referenceCategory":"PACKAGE-MANAGER","referenceType":"purl"
,"referenceLocator":"pkg:github/tensorflow/tensorflow"}]},{"SPDXID":"SPDXRef-pip-absl-py-2.0
.0","name":"pip:absl-py","versionInfo":"2.0
.0","downloadLocation":"NOASSERTION","filesAnalyzed":false,"licenseConcluded":"Apache-2
.0","supplier":"NOASSERTION","externalRefs":
[{"referenceCategory":"PACKAGE-MANAGER","referenceLocator":"pkg:pypi/absl-py@2.0
.0","referenceType":"purl"}]},{"SPDXID":"SPDXRef-pip-astor-0.7.1","name":"pip:astor","versionInfo":"0.7
.1","downloadLocation":"NOASSERTION","filesAnalyzed":false,"licenseConcluded":"BSD-2-Clause"
,"supplier":"NOASSERTION","externalRefs":
[{"referenceCategory":"PACKAGE-MANAGER","referenceLocator":"pkg:pypi/astor@0.7
.1","referenceType":"purl"}]},{"SPDXID":"SPDXRef-pip-astunparse-1.6
.3","name":"pip:astunparse","versionInfo":"1.6
.3","downloadLocation":"NOASSERTION","filesAnalyzed":false,"licenseConcluded":"BSD-2-Clause AND BSD-3-Clause
AND
Python-2.0","supplier":"NOASSERTION","externalRefs":
[{"referenceCategory":"PACKAGE-MANAGER","referenceLocator":"pkg:pypi/astunparse@1.6
.3","referenceType":"purl"}]},{"SPDXID":"SPDXRef-pip-cachetools-5.3

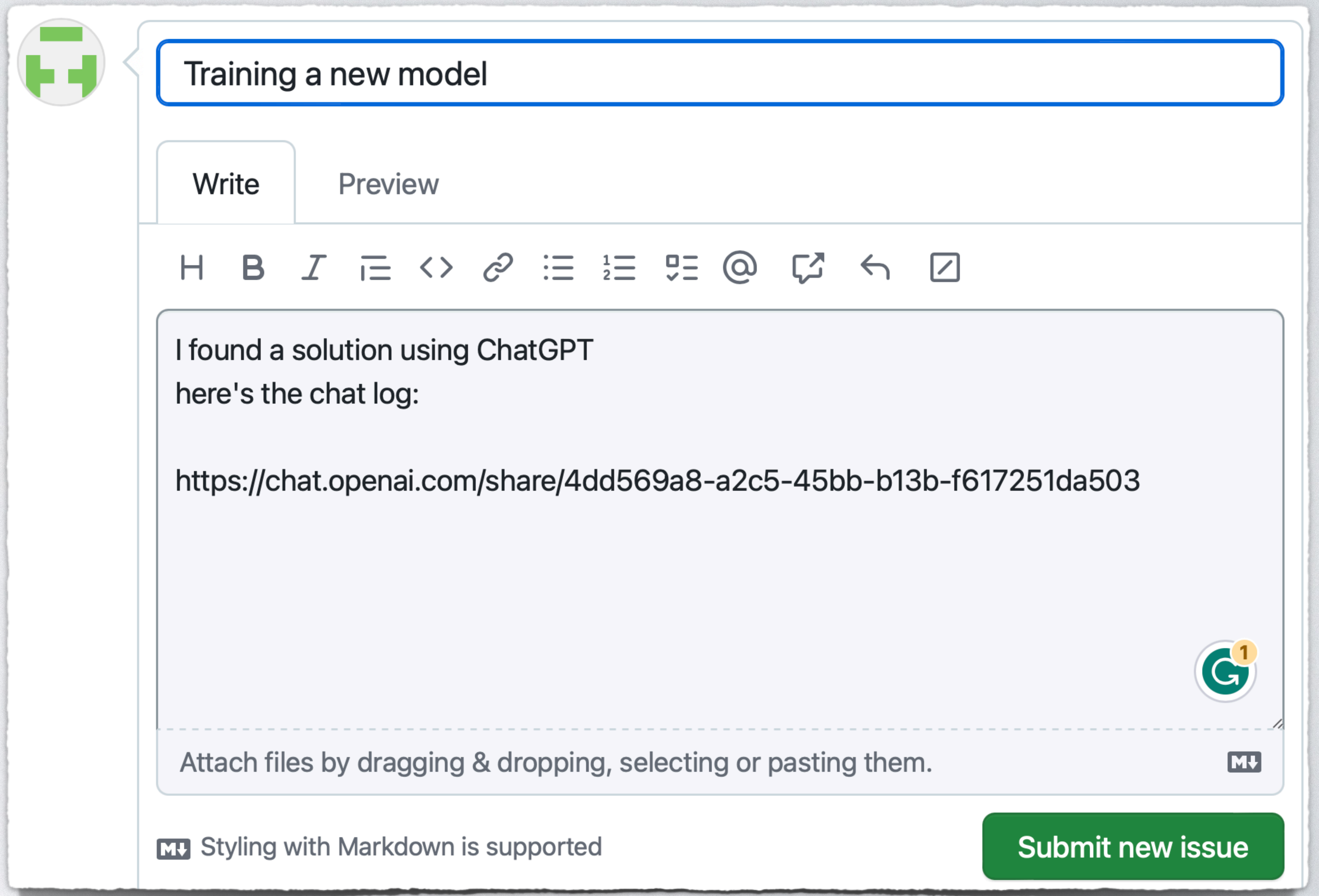

WH.GOV
MAY 12, 2021
Executive Order on Improving the Nation's Cybersecurity
BRIEFING ROOM
PRESIDENTIAL ACTIONS
By the authority vested in me as President by the Constitution and the laws of the United States of America, it is hereby ordered as follows:
Section 1. Policy. The United States faces persistent and increasingly sophisticated malicious cyber campaigns that threaten the public sector, the private sector, and ultimately the American people's security and privacy. The Federal Government must improve its efforts to identify, deter, protect against, detect, and respond to these actions and actors. The Federal Government must also carefully examine what occurred during any major cyber incident and apply lessons learned. But cybersecurity requires more than government action. Protecting our Nation from malicious cyber actors requires the Federal Government to partner with the private sector. The private sector must adapt to the continuously changing threat environment, ensure its products are built and operate securely, and partner with the Federal Government to foster a more secure cyberspace. In the end, the trust
MENU
WH.GOV
JANUARY 19, 2022
Memorandum on Improving the Cybersecurity of National Security, Department of Defense, and Intelligence Community Systems
BRIEFING ROOM
PRESIDENTIAL ACTIONS
NATIONAL SECURITY MEMORANDUM/NSM-8
MEMORANDUM FOR THE VICE PRESIDENT
THE SECRETARY OF STATE
THE SECRETARY OF THE TREASURY
THE SECRETARY OF DEFENSE
THE ATTORNEY GENERAL
THE SECRETARY OF COMMERCE
THE SECRETARY OF ENERGY
THE SECRETARY OF HOMELAND SECURITY
THE DIRECTOR OF THE OFFICE OF MANAGEMENT AND BUDGET
THE DIRECTOR OF NATIONAL INTELLIGENCE
THE DIRECTOR OF THE CENTRAL INTELLIGENCE AGENCY
MENU
the **United States Government** brought SBOMs to the forefront of digital policy

ABOUT CODE REVIEW

- Chicken and egg problem
  - LLM can review your code, but YOU should review LLM code
- Again, lack of trust
- No interaction with a human, maybe with the LLM?
- Different reviewing standards than with (expert) developers?

ALSO…

- Keep track of the LLM prompt chaiining and logs
- Possibly, using the issue tracker



…AND WE SHOULD BE AWARE ON HOW OUR STUDENTS CONDUCT LITERATURE REVIEW…

AN EMAIL I RECEIVED SOME WEEKS AGO…

Dear Professor Bavota, professor Di Penta, professor Rocco,

I hope this message finds you well. I am a second-year master's student at the University of XXX, specializing in software engineering. I am writing to request your assistance in locating one of your publications that is relevant to my master's thesis topic.

In my literature review, I came across a citation of your work titled Automated "Graph-Based Integration Testing of Web Applications". which is crucial for my research. However, despite my efforts, I have been unable to find this publication in our university's library or online databases.

All the information I was given about this publication are the following:

Bavota, G., Di Penta, M., & Oliveto, R. (2013). Graph-Based Integration Testing of Web Applications. Proceedings of the 2013 International Conference on Software Engineering, 1007–1010. https://doi.org/10.1109/ICSE.2013.6606665

I would greatly appreciate it if you could direct me to the source of this publication or share an electronic copy if available. Your guidance and expertise would significantly benefit my research.

Thank you for considering my request. I look forward to your response.

Best regards,

THE PAPER DOES NOT EXIST!

TAKEAWAYS AND CONCLUSIONS

TAKEAWAYS

TAKEAWAYS

Developers will write less and less code as the AI support is getting better

→ software engineering education has to adapt to that

TAKEAWAYS

Developers will write less and less code as the AI support is getting better
→ software engineering education has to adapt to that

Instructing SE students on AI applications, Prompt engineering and chaining, but also awareness on AI limitations

TAKEAWAYS

Developers will write less and less code as the AI support is getting better
→ software engineering education has to adapt to that

Instructing SE students on AI applications, Prompt engineering and chaining, but also awareness on AI limitations

Need to re-think the development process: A suitable Q/A is required for AI generated software artifacts

TAKEAWAYS

Developers will write less and less code as the AI support is getting better
→ software engineering education has to adapt to that

Instructing SE students on AI applications, Prompt engineering and chaining, but also awareness on AI limitations

Need to re-think the development process: A suitable Q/A is required for AI generated software artifacts

Lack of trust and traceability may create consequences also from a legal perspective: still many open challenges in this area

MY (OLD) COURSE GOALS

- Provide advanced software engineering contents mostly focused on software quality
- Stimulate workgroup
- Stimulate investigation on innovative topics

MY (OLD) COURSE GOALS

- Provide advanced software engineering contents mostly focused on software quality
- Stimulate workgroup
- Stimulate investigation on innovative topics

IMPACT ON SE EDUCATION

Favor a responsible, informed integration of LLMs in the software development process

MY (OLD) COURSE GOALS

- Provide advanced software engineering contents mostly focused on software quality
- Stimulate workgroup
- Stimulate investigation on innovative topics

IMPACT ON SE EDUCATION

Favor a responsible, informed integration of LLMs in the software development process

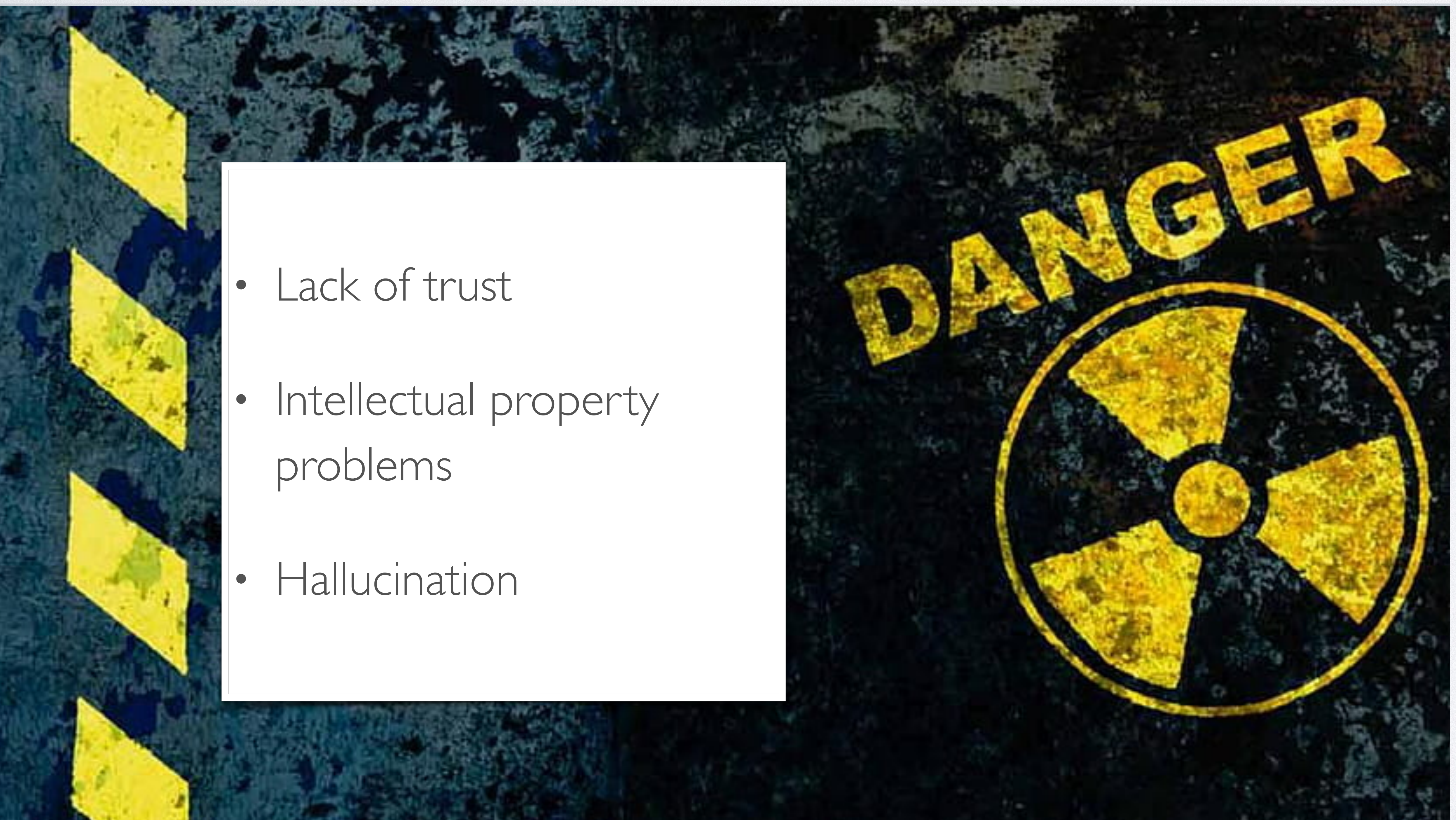
- Lack of trust
- Intellectual property problems
- Hallucination

MY (OLD) COURSE GOALS

- Provide advanced software engineering contents mostly focused on software quality
- Stimulate workgroup
- Stimulate investigation on innovative topics

IMPACT ON SE EDUCATION

Favor a responsible, informed integration of LLMs in the software development process



- Lack of trust
- Intellectual property problems
- Hallucination

TAKEAWAYS

Developers will write less and less code as the AI support is getting better
→ software engineering education has to adapt to that

Instructing SE students on AI applications, Prompt engineering and chaining, but also awareness on AI limitations

Need to re-think the development process: A suitable Q/A is required for AI generated software artifacts

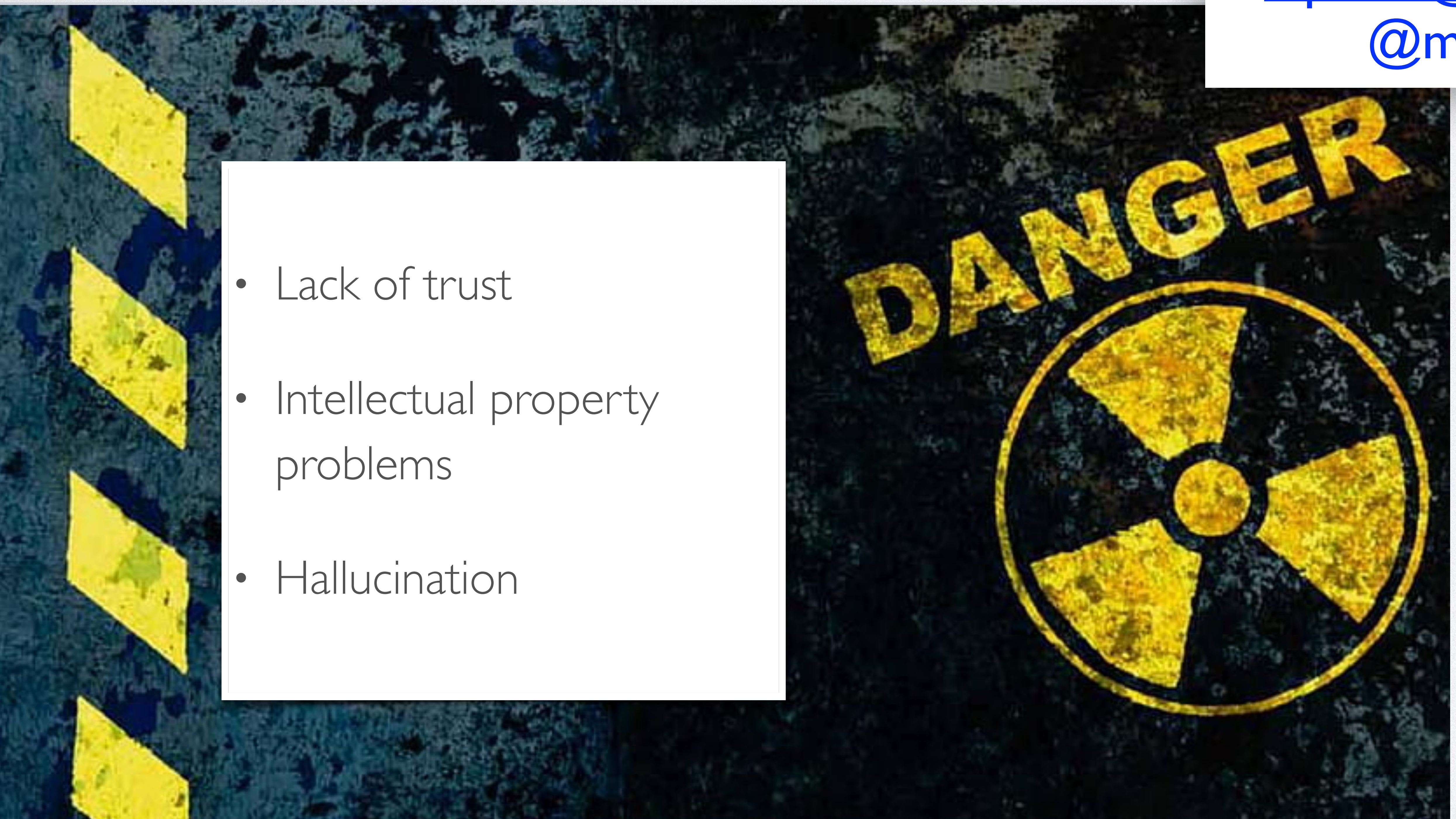
Lack of trust and traceability may create consequences also from a legal perspective: still many open challenges in this area

MY (OLD) COURSE GOALS

- Provide advanced software engineering contents mostly focused on software quality
- Stimulate workgroup
- Stimulate investigation on innovative topics

IMPACT ON SE EDUCATION

Favor a responsible, informed integration of LLMs in the software development process



- Lack of trust
- Intellectual property problems
- Hallucination

TAKEAWAYS

Developers will write less and less code as the AI support is getting better
→ software engineering education has to adapt to that

Instructing SE students on AI applications, Prompt engineering and chaining, but also awareness on AI limitations

Need to re-think the development process: A suitable Q/A is required for AI generated software artifacts

Lack of trust and traceability may create consequences also from a legal perspective: still many open challenges in this area

Questions?

dipenta@unisannio.it
@mdipenta