

# GenAI for Software Engineering 2025

## Limitations and the Future

Jorge Melegati

# SWOT ANALYSIS

Talked a lot so far!



In this lecture, we will focus on these!

# Limitations

- Hallucinations: statements that sound plausible but are not true
- Catastrophic interference (forgetting)
- Limited size of the input
- Concerns on IP
- Lack of Explainability

# Example of hallucination:

Imagine that you're programming in the Ruby language and want to use a library (which Ruby programmers refer to as a "gem") to perform AES encryption. The Ruby aes gem has encrypt and decrypt functionality, but maybe you want some expert advice on how to use these methods. So you might enter the following prompt into ChatGPT.

Prompt:

```
aes ruby gem AES.encrypt
```

Sure, here's an example of how to use the 'AES.encrypt' method from the 'aes' Ruby gem to encrypt data with AES-256 encryption:

```
require 'aes'

key = 'mysecretkey12345'
iv = AES.iv(:AES_256_CBC)
plaintext = 'my secret data'

ciphertext = AES.encrypt(plaintext, key, iv: iv)
```

In this example, we generate a random initialization vector (IV) using the 'AES.iv' method with the ':AES\_256\_CBC' option to specify the CBC mode of operation. We then call the 'AES.encrypt' method with the plaintext data, the key, and the IV to encrypt the data.

To decrypt the encrypted data, you can use the following code:

```
require 'aes'

key = 'mysecretkey12345'
iv = ciphertext[0..15] # extract the first 16 bytes as the IV
encrypted_data = ciphertext[16..-1] # extract the rest as the encrypted data

plaintext = AES.decrypt(encrypted_data, key, iv: iv)
```

In this code, we extract the IV and encrypted data from the ciphertext, and then call the 'AES.decrypt' method with the encrypted data, the key, and the IV to decrypt the data.

Note that the 'AES.encrypt' method uses CBC mode by default, which can be vulnerable to certain types of attacks. If you need to use a more secure mode of operation, such as GCM, you can pass additional options to the 'encrypt' and 'decrypt' methods. For example:

```
require 'aes'

key = 'mysecretkey12345'
iv = AES.iv(:AES_256_GCM)
plaintext = 'my secret data'

ciphertext, tag = AES.encrypt(plaintext, key, iv: iv, tag: true)
decrypted_data = AES.decrypt(ciphertext, key, iv: iv, tag: tag)
```

In this example, we use the ':AES\_256\_GCM' option when generating the random IV to specify the GCM mode of operation. We also pass the 'tag: true' option to the 'AES.encrypt' method to request the authentication tag, and then pass the tag to the 'AES.decrypt' method to verify the integrity of the decrypted data.

ChatGPT appears authoritative. It even warns about the vulnerability of the default CBC encryption mode to certain kinds of attacks. But ChatGPT missed [a greater danger](#): This gem applies the Ruby hex function on the key. So a programmer using it must take care that the key employed is a string that can be interpreted as a series of hexadecimal numbers. Ruby's hex function applied to the key given in ChatGPT's example code would turn it into zero. Using such a key would result in the encryption [being completely insecure](#).

# Another example of hallucinations on security

## **AI hallucinates software packages and devs download them – even if potentially poisoned with malware**

- [https://www.theregister.com/2024/03/28/ai\\_bots\\_hallucinate\\_software\\_packages/](https://www.theregister.com/2024/03/28/ai_bots_hallucinate_software_packages/)
- ChatGPT gives an answer recommending the use of a software library, package, or framework that doesn't exist
- An attacker can upload a malicious package with the same name to the registries and wait for people to download the packages

# Hallucinations and generation of flaky tests

- Flaky tests are tests that produce inconsistent results
- Automatic test generation tools have a similar ratio of flaky tests are human-created
- But they are different
  - Human: caused more by concurrency and networking
  - Generated: more the result of randomness and unspecified behavior

## Do Automatic Test Generation Tools Generate Flaky Tests?

Martin Gruber\*  
BMW Group, University of Passau  
Munich, Germany  
martin.gr.gruber@bmw

Muhammad Firhard Roslan\*  
University of Sheffield  
Sheffield, United Kingdom  
mfroslan2@sheffield.ac.uk

Owain Parry  
University of Sheffield  
Sheffield, United Kingdom  
oparry1@sheffield.ac.uk

Fabian Scharnböck  
University of Passau  
Passau, Germany  
scharn05@ads.uni-passau.de

Phil McMinn  
University of Sheffield  
Sheffield, United Kingdom  
p.mcminn@sheffield.ac.uk

Gordon Fraser  
University of Passau  
Passau, Germany  
gordon.fraser@uni-passau.de

### ABSTRACT

Non-deterministic test behavior, or flakiness, is common and dreaded among developers. Researchers have studied the issue and proposed approaches to mitigate it. However, the vast majority of previous work has only considered developer-written tests. The prevalence and nature of flaky tests produced by test generation tools remain largely unknown. We ask whether such tools also produce flaky tests and how these differ from developer-written ones. Furthermore, we evaluate mechanisms that suppress flaky test generation. We sample 6 356 projects written in Java or Python. For each project, we generate tests using EvoSuite (Java) and Pynguin (Python), and execute each test 200 times, looking for inconsistent outcomes. Our results show that flakiness is at least as common in generated tests as in developer-written tests. Nevertheless, existing flakiness suppression mechanisms implemented in EvoSuite are effective in alleviating this issue (71.7 % fewer flaky tests). Compared to developer-written flaky tests, the causes of generated flaky tests are distributed differently. Their non-deterministic behavior is more frequently caused by randomness, rather than by networking and concurrency. Using flakiness suppression, the remaining flaky tests differ significantly from any flakiness previously reported, where most are attributable to runtime optimizations and EvoSuite-internal resource thresholds. These insights, with the accompanying dataset, can help maintainers to improve test generation tools, give recommendations for developers using these tools, and serve as a foundation for future research in test flakiness or test generation.

### ACM Reference Format:

Martin Gruber, Muhammad Firhard Roslan, Owain Parry, Fabian Scharnböck, Phil McMinn, and Gordon Fraser. 2024. Do Automatic Test Generation Tools Generate Flaky Tests?. In *2024 IEEE/ACM 46th International Conference on Software Engineering (ICSE '24)*, April 14–20, 2024, Lisbon, Portugal. ACM, New York, NY, USA, 12 pages. <https://doi.org/10.1145/3597503.3608138>

## 1 INTRODUCTION

A *flaky test* is a test case that produces inconsistent results, meaning that the same test can pass or fail for no apparent reason, even when the system being tested has not changed [51]. They are a major problem for software developers because they limit the efficiency of testing, complicate continuous integration, and reduce productivity [19, 39, 48]. The negative effects of flaky tests are ubiquitous, experienced by large companies such as Google, Microsoft, and Facebook, as well as the developers of smaller open-source projects [19, 36, 45, 47]. Indeed, recent surveys found that a majority of developers observe flaky tests on at least a monthly basis [29, 52]. As well as being a burden on developers, flaky tests are also a persistent problem in research, limiting the deployment of several state-of-the-art techniques for test selection and prioritization [45, 54, 68].

Increasing research interest in the area of flaky tests has produced a range of empirical studies regarding the causes, origins, and impacts of developer-written flaky tests [21, 37, 44, 64]. However, far less attention has been paid to flaky tests produced by automatic test generation tools [53, 60]. This research gap is problematic for

# Catastrophic interference (forgetting)

- Novel APIs and languages emerge
  - Knowledge crucial for providing up-to-date suggestions
- Need for continuous learning
- Risk of catastrophic forgetting
  - The model forgets the learning from previous data

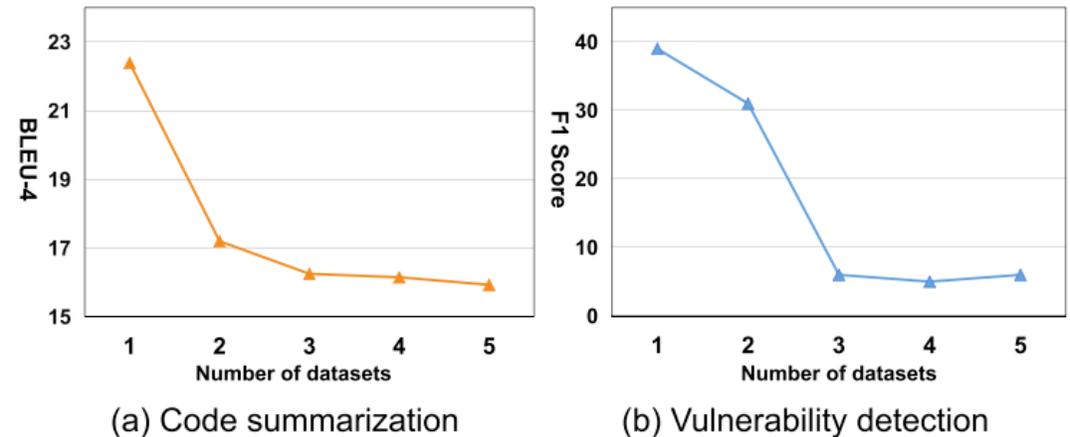


Fig. 1: The performance degradation on the first dataset after fine-tuning CodeBERT on new datasets. The datasets of code summarization and vulnerability detection are from CodeSearchNet and Big-Vul respectively. For more details refer to Section IV-A.

Source: Gao et al. (2023) “Keeping Pace with Ever-Increasing Data: Towards Continual Learning of Code Intelligence Models”

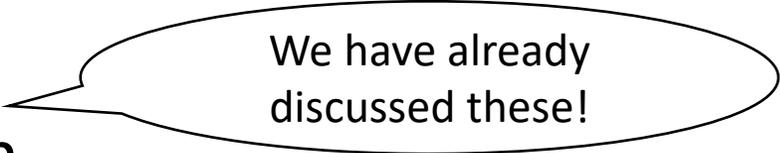
# Limited size of the input

- The input size of the model is limited
  - *gpt-3.5-turbo-16k* is limited to 16k tokens
  - Remember: one word is generally more than one token
- How to handle large codebases or even complete systems?



# Criteria to select a tool

- Time/difficulty to train/run
- Cost/infrastructure needed to run
- License fees
- Performance metrics



We have already  
discussed these!

# Time/difficulty to train/run

- GPT-2
  - Trained in approximately 7 days on 256 TPU cores
- Scarcity of labelled datasets

# License fees

The image shows a comparison of two Copilot license options. On the left is 'Copilot for Individuals' priced at \$10 per month or \$100 per year. It lists features like editor integration, natural language prompts, multi-line suggestions, faster test generation, and blocking public code. A 'Start a free trial' button is provided. On the right is 'Copilot for Business', marked as 'new', priced at \$19 per user per month. It includes all individual features plus organization-wide management, privacy, and proxy support. 'Buy Now' and 'Contact Sales' buttons are provided.

License Type	Price	Key Features	Action
Copilot for Individuals	\$10 per month / \$100 per year	Plugs right into your editor, Turns natural language prompts into code, Offers multi-line function suggestions, Speeds up test generation, Blocks suggestions matching public code	Start a free trial >
Copilot for Business (new)	\$19 per user per month	Everything included in Copilot for Individuals, plus... Simple license management, Organization-wide policy management, Industry-leading privacy, Corporate proxy support	Buy Now > / Contact Sales >

# Performance metrics

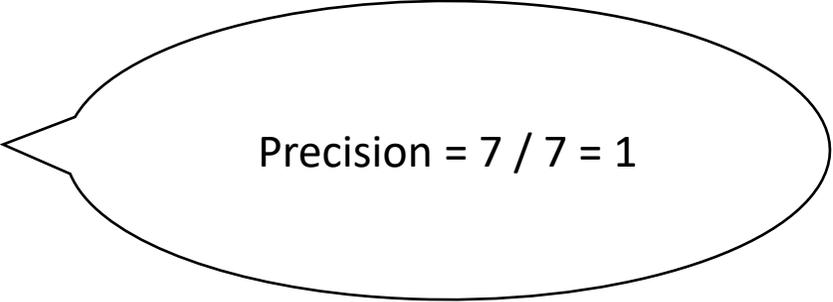
- Precision, recall, F1 and other performance metrics also apply
- Different models can lead to different results

# Performance metrics for generation

- BLEU
- ROUGE
- METEOR

# BLEU – an intuition

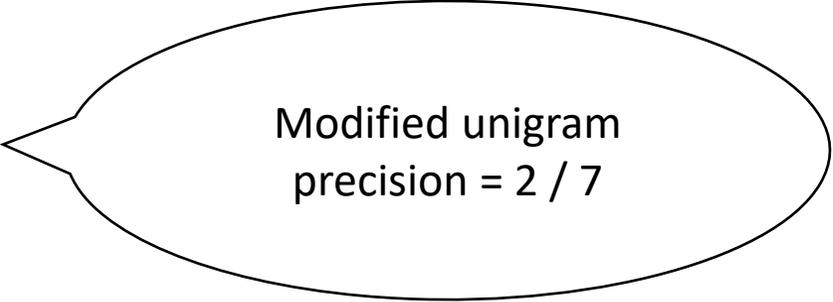
- Precision: number of candidate words that occur in any possible reference divided by the total number of words in the candidate
- Problem: generation of improbable but high-precision outputs
- Example:
  - Candidate: the the the the the the the
  - Reference 1: The cat is on the mat.
  - Reference 2: There is a cat on the mat.



Precision =  $7 / 7 = 1$

# Modified unigram precision

- Counts the maximum number of times a word occurs in any single reference
- Clips the total count of each candidate word by its maximum reference count
- Adds these clipped counts up
- Divides by the total (unclipped) number of candidate words
- Example:
  - Candidate: the the the the the the the
  - Reference 1: The cat is on the mat.
  - Reference 2: There is a cat on the mat.



Modified unigram  
precision = 2 / 7

# Modified n-gram precision

- All candidate n-gram counts, and corresponding maximum reference counts are collected
- The candidate counts are clipped by their corresponding reference maximum value, summed, and divided by the total number of candidate n-grams

# BLEU as a family of metrics

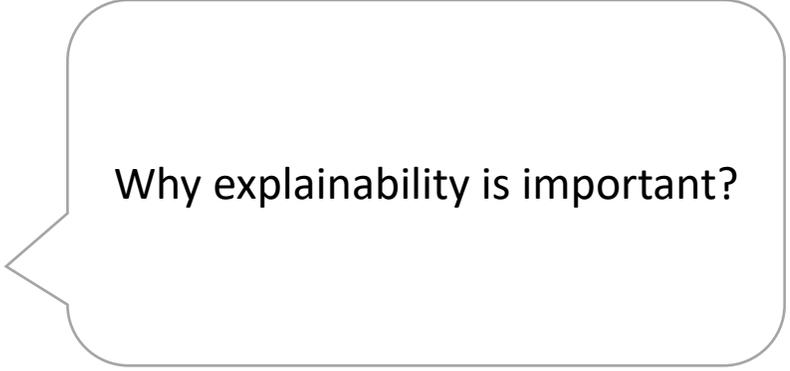
- BLEU is calculated as the weighted geometric mean of all the modified n-grams precisions
  - The weights are numbers from 0 to 1 which sum to 1
- A brevity penalty multiplies the average to avoid long sentences
- The most typical BLEU consider 1 to 4-gram giving  $\frac{1}{4}$  weight for each

# Lack of Explainability

- State-of-the-art ML-models tend to be highly complex and black-box
- Less explainable solutions
- Reduce trust of AI4SE solutions by industry practitioners

# Explainability goals

- Justification
- Improvement
- Understanding the model
- Fairness
- Transparency



Why explainability is important?

# Explainability goals

- Justification
  - Justifying the model's decision increases its credibility
  - Answers why an instance gives a specific output

# Explainability goals

- Improvement
  - Understanding why the model made wrong decisions can help to improve the model

# Explainability goals

- Understanding the model
  - A general term that supports the other goals

# Explainability goals

- Fairness
  - Prevent biases and discriminations
- Potential issue:
  - Imbalanced or biased training data

# Explainability goals

- Transparency
  - When the model makes a decision, the whole process can be comprehended by a human
- Full transparency is usually impossible to fulfill
  - Some partial levels are used
  - For example, when smaller components of the model can be separately explained or explainable

# Related question: can these models really think?

## GSM-Symbolic: Understanding the Limitations of Mathematical Reasoning in Large Language Models

Iman Mirzadeh<sup>†</sup>   Keivan Alizadeh   Hooman Shahrokhi\*  
Oncel Tuzel   Samy Bengio   Mehrdad Farajtabar<sup>†</sup>

Apple

### Abstract

Recent advancements in Large Language Models (LLMs) have sparked interest in their formal reasoning capabilities, particularly in mathematics. The GSM8K benchmark is widely used to assess the mathematical reasoning of models on grade-school-level questions. While the performance of LLMs on GSM8K has significantly improved in recent years, it remains unclear whether their mathematical reasoning capabilities have genuinely advanced, raising questions about the reliability of the reported metrics. To address these concerns, we conduct a large-scale study on several state-of-the-art open and closed models. To overcome the limitations of existing evaluations, we introduce GSM-Symbolic, an improved benchmark created from symbolic templates that allow for the generation of a diverse set of questions. GSM-Symbolic enables more controllable evaluations, providing key insights and more reliable metrics for measuring the reasoning capabilities of models. Our findings reveal that LLMs exhibit noticeable variance when responding to different instantiations of the same question. Specifically, the performance of all models declines when only the numerical values in the question are altered in the GSM-Symbolic

## Different Levels of GSM-Symbolic Difficulty

**GSM-Symbolic-M1:** To make a call from a phone booth, you must pay \$0.6 for each minute of your call. ~~After 10 minutes, that price drops to \$0.5 per minute.~~ How much would a 60-minute call cost?

**GSM-Symbolic:** To make a call from a phone booth, you must pay \$0.6 for each minute of your call. After 10 minutes, that price drops to \$0.5 per minute. How much would a 60-minute call cost?

**GSM-Symbolic-P1:** To make a call from a hotel room phone, you must pay \$0.6 for each minute of your call. After 10 minutes, that price drops to \$0.5 per minute. **After 25 minutes from the start of the call, the price drops even more to \$0.3 per minute.** How much would a 60-minute call cost?

**GSM-Symbolic-P2:** To make a call from a hotel room phone, you must pay \$0.6 for each minute of your call. After 10 minutes, the price drops to \$0.5 per minute. **After 25 minutes from the start of the call, the price drops even more to \$0.3 per minute. If your total bill is more than \$10, you get a 25% discount.** How much would a 60-minute call cost?

Figure 5: Modifying the difficulty level of GSM-Symbolic by modifying the number of clauses.

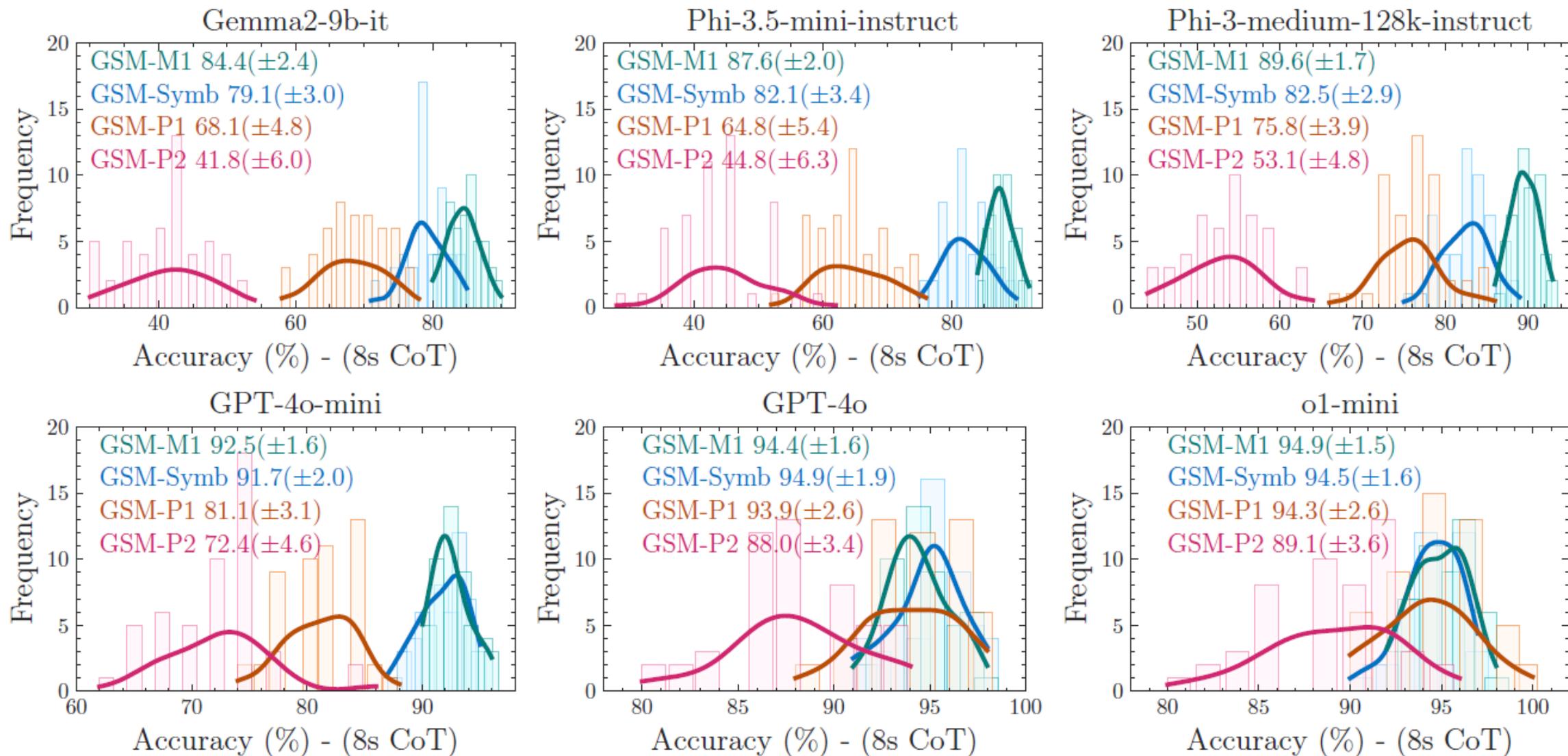


Figure 6: The impact of increasing the number of clauses on performance: As the difficulty increases from **GSM-M1** → **GSM-Symb** → **GSM-P1** → **GSM-P2**, the distribution of performance shifts to the left (i.e., accuracy decreases), and the variance increases.

## 5 Conclusion

In this work, we have investigated the reasoning capabilities of large language models (LLMs) and the limitations of current evaluations on GSM8K. We introduced GSM-Symbolic, a novel benchmark with multiple variants designed to provide deeper insights into the mathematical reasoning abilities of LLMs. Our extensive study reveals significant performance variability across different instantiations of the same question, challenging the reliability of current GSM8K results that rely on single-point accuracy metrics.

We found that while LLMs exhibit some robustness to changes in proper names, they are more sensitive to variations in numerical values. We have also observed the performance of LLMs deteriorating as question complexity increases.

The introduction of GSM-NoOp exposes a critical flaw in LLMs' ability to genuinely understand mathematical concepts and discern relevant information for problem-solving. Adding seemingly relevant but ultimately inconsequential information to the logical reasoning of the problem led to substantial performance drops of up to 65% across all state-of-the-art models. Importantly, we demonstrate that LLMs struggle even when provided with multiple examples of the same question or examples containing similar irrelevant information. This suggests deeper issues in their reasoning processes that cannot be easily mitigated through few-shot learning or fine-tuning.

# We should be careful!

- A paper in *Nature!*
- Not ascribe human behavior to these models

## Perspective

# Role play with large language models

<https://doi.org/10.1038/s41586-023-06647-8>

Murray Shanahan<sup>1,2</sup>✉, Kyle McDonell<sup>3</sup>✉ & Laria Reynolds<sup>3</sup>✉

Received: 10 July 2023

Accepted: 14 September 2023

Published online: 8 November 2023

 Check for updates

As dialogue agents become increasingly human-like in their performance, we develop effective ways to describe their behaviour in high-level terms without falling into the trap of anthropomorphism. Here we foreground the concept of role play. Casting dialogue-agent behaviour in terms of role play allows us to draw on folk psychological terms, without ascribing human characteristics to language models that they in fact lack. Two important cases of dialogue-agent behaviour are addressed this way, namely, (apparent) deception and (apparent) self-awareness.

Large language models (LLMs) have numerous use cases, and can be prompted to exhibit a wide variety of behaviours, including dialogue. This can produce a compelling sense of being in the presence of a human-like interlocutor. However, LLM-based dialogue agents are, in multiple respects, very different from human beings. A human's language skills are an extension of the cognitive capacities they develop

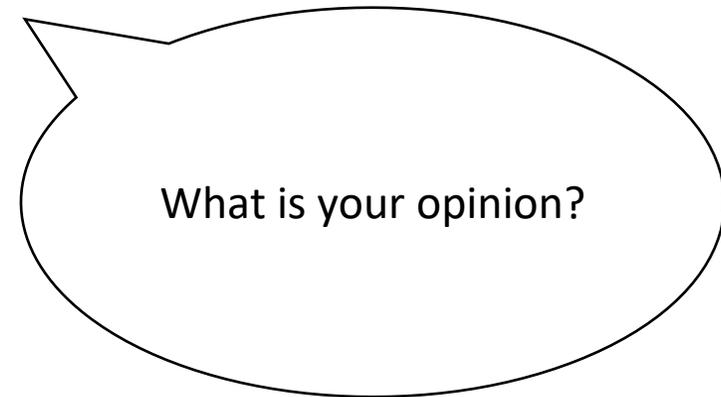
agents without falling into the conceptual trap of applying folk psychological concepts to LLMs in the literal sense in which we apply them to humans.

## LLM basics

Crudely put, the function of an LLM is to answer questions.

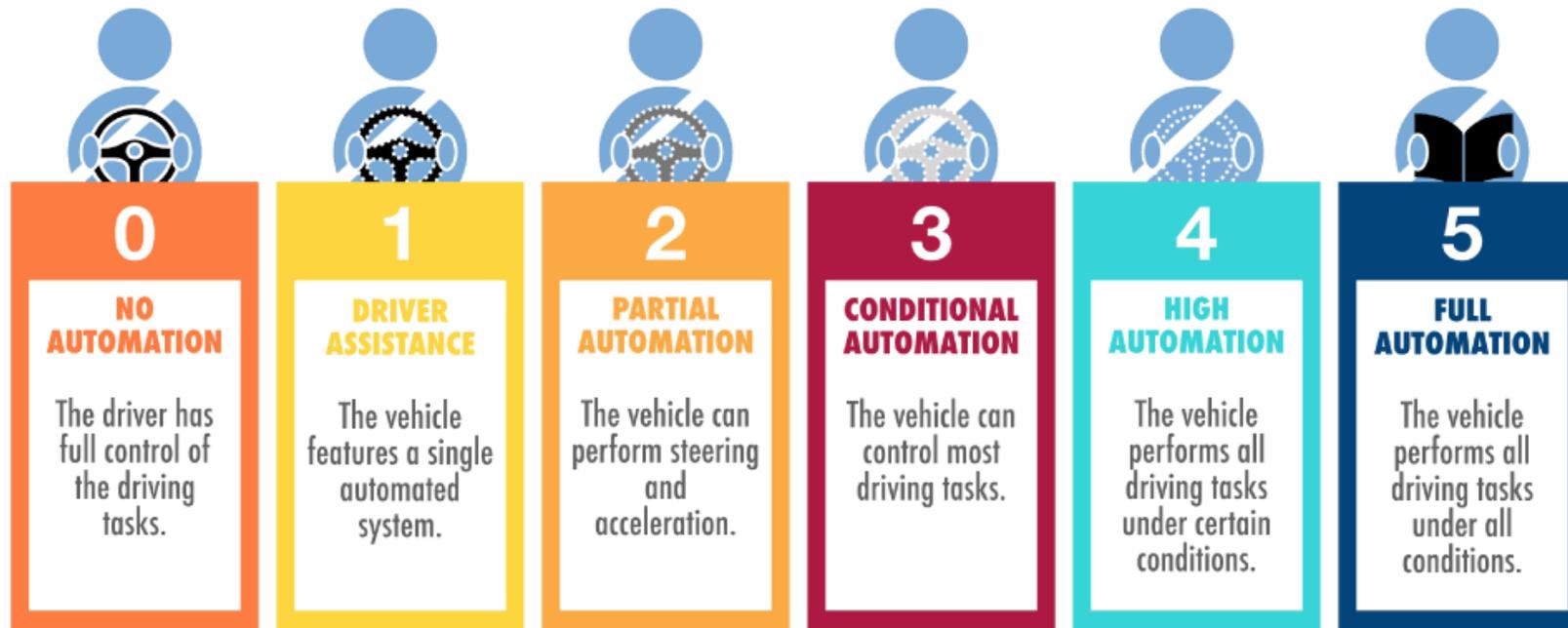
# Conclusion

- What will be the future?
- Will AI replace software developers?



# Comparison: Driving automation levels

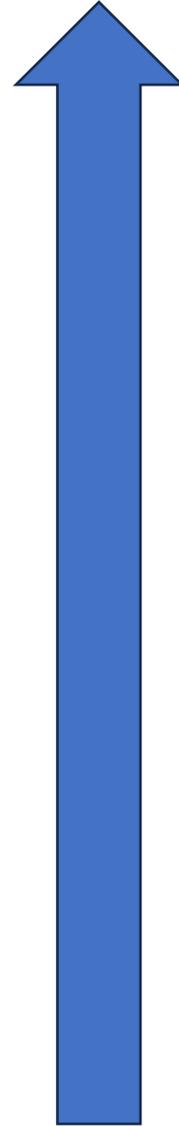
## LEVELS OF AUTONOMOUS DRIVING



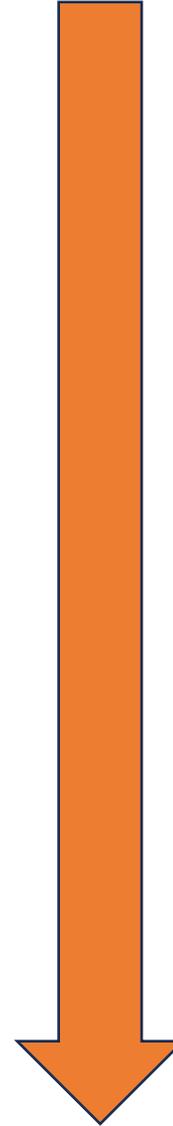
# DAnTE

Degree of  
Automation  
Taxonomy for software  
Engineering

Increasing  
tools'  
capabilities



Increasing  
developers'  
effort



**Level 5**  
**Full generator**

**Level 4**  
**Global generator**

**Level 3**  
**Local generator**

**Level 2**  
**Suggester**

**Level 1**  
**Informer**

**Level 0**  
**No automation**

**DAnTE: A Taxonomy for the Automation  
Degree of Software Engineering Tasks**

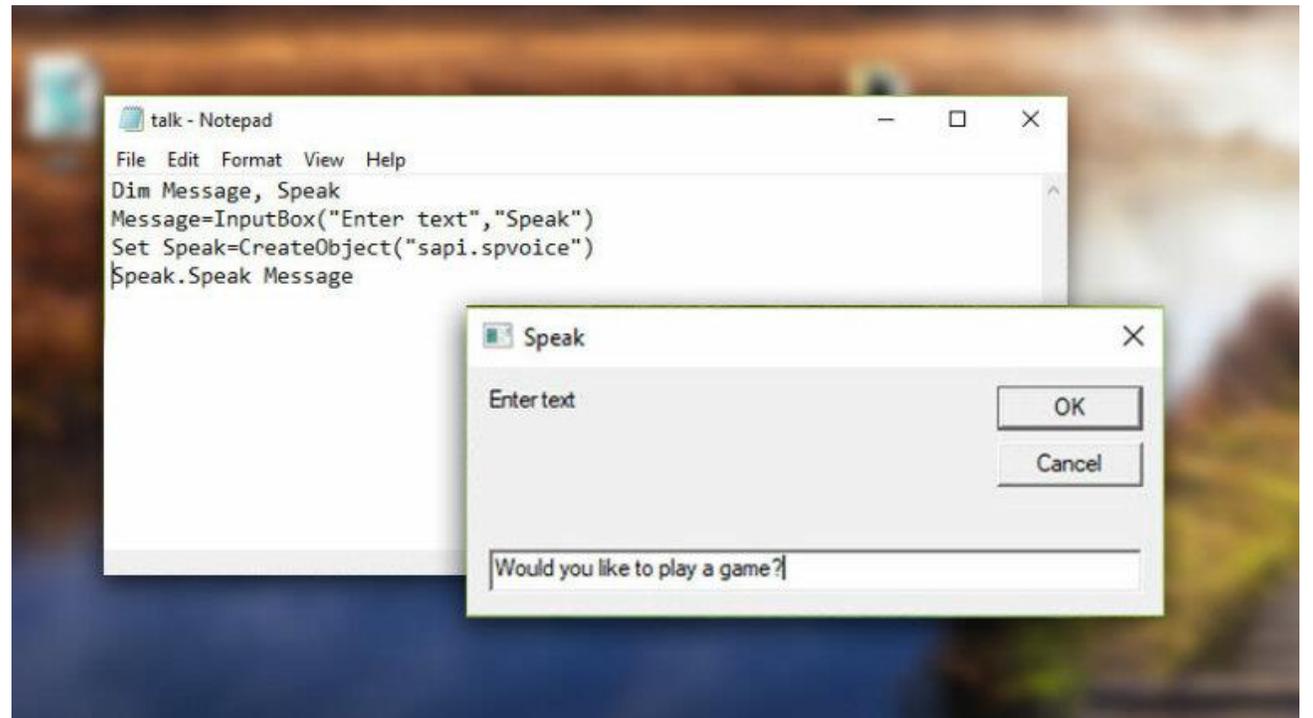
Jorge Melegati  and Eduardo Guerra 



Abstract Software engineering researchers and practitioners aim to reduce the amount of time and effort required for software development. In this paper, we propose a taxonomy of software engineering tasks, DAnTE, a Degree of Automation Taxonomy for Software Engineering Tasks, consisting of several levels of automation. The taxonomy is designed to help researchers and practitioners understand the current state of software engineering automation and to guide the design of tools and processes that can improve software engineering productivity.

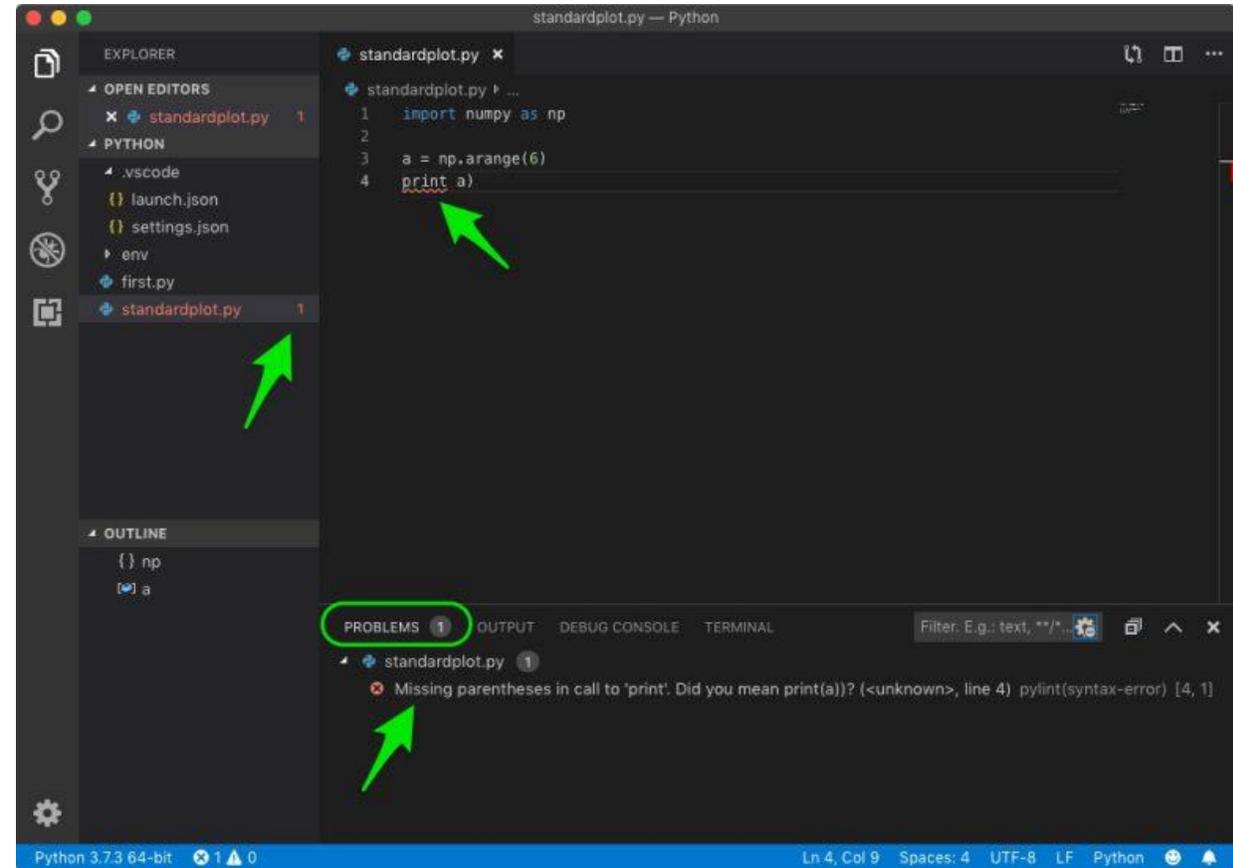
# Level 0 – No automation

- Total lack of supporting tools
- Developers are responsible for all



# Level 1 – Informer

- Tools provide information but are not able to suggest solutions
- Developers are responsible for all



# Level 2 – Suggester

- Tools can provide suggestions for modifying the implementation
- Developers perform the tasks but can accept suggestions from the tool

```
private set_spwi(s : string) is only {
  case s {
    "1'bz":{
      spwi_rw.put_mvl_string("1'b1");
      spwi_tx.put_mvl_string("1'bz");
      spwi_rx.p
    };
    "1'b0":{
      spwi_rw.
      spwi_tx.
    };
    "1'b1":{
      spwi_rw.
      spwi_tx.
    };
    default : {
      error("A
    };
  };
};
```

✪ pack\_options() : string  
✪ pass\_by\_pointer() : bool  
✪ post\_generate()  
✪ pre\_generate()  
✪ print\_line(struct\_descriptor : struct\_descriptor)  
✪ put\_mvl(mvl)  
✪ put\_mvl\_list(list of mvl)  
✪ put\_mvl\_string(string)  
✪ put\_mvl\_to\_bit\_slice(int, int, list of mvl)

put\_mvl\_string  
PREDEFINED:  
[any\_simple\_port method]  
Put an mvl value on a port of a non-mvl type when a value is represented as a string!

Press 'Ctrl+Space' to show Template proposals

# Level 3 – Local generator

- Tools can generate solutions at a restricted level such as functions, methods, or classes
- At certain levels, developers can describe the needs and the tools produce one or more solutions that can be accepted or not

```
5 type Task struct {  
6     Id int  
7     Title string  
8     Priority int  
9 }  
10 func createTables(db *sql.DB) {  
11     db.Exec("CREATE TABLE tasks (id INTEGER PRIMARY KEY, title TEXT, priority INTEGER)")  
12 }  
13 func selectTaskByPriority(db *sql.DB, priority int)  
14
```

# Level 4 – Global generator

- Tools can provide complete solutions, but they should be checked by developers
- Developers provide descriptions of the solution and check the proposed solutions.

# Level 5 – Full generator

- Tools can reliably create full solutions given descriptions
- Developers just describe what is expected

# The current landscape

- Level 2 tools are consolidated and the de-facto standard

```
package algorithm;

import java.io.IOException;
import java.util.*;

public class Anagrams {

    //O(NlogN)
    public static Boolean isAnagramViaSort(String s1, String s2){
        if (s1.length() != s2.length())
            return false;

        LinkedBlo

        Arrays.sort(arr);
        Arrays.sort(arr2);
    }
}
```

LinkedBlo

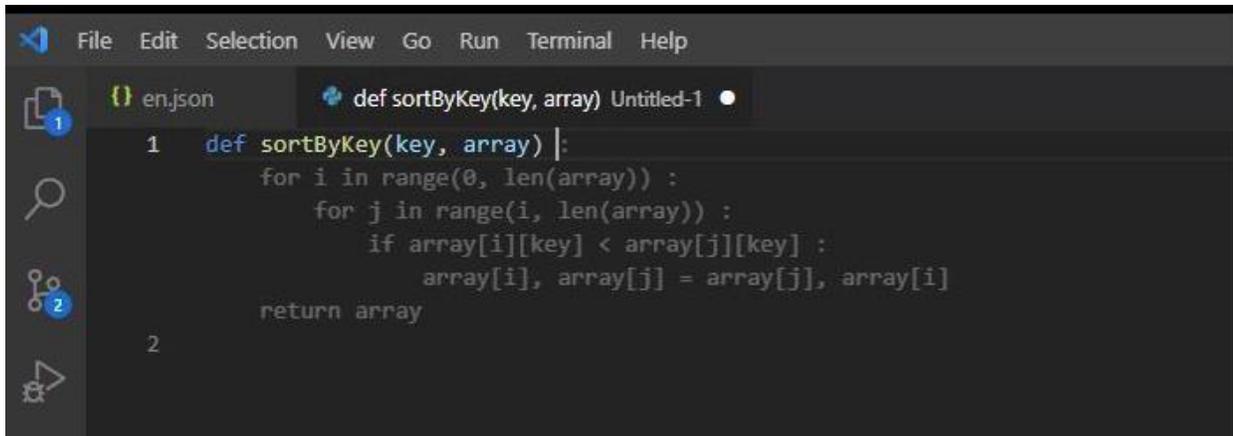
- LinkedBlockingDeque <E> (java.util.concurrent)
- LinkedBlockingQueue <E> (java.util.concurrent)

Ctrl+Down and Ctrl+Up will move caret down and up in the editor >>

```
JS app.js x
24
25 app.use('/', index);
26 app.use('/users', users);
27 app.us
28 // catch 404 and forward to error handler
29 app.use(function(req, res, next) {
30     var err = new Error('Not Found');
31     err.status = 404;
32     next(err);
33
34     Extract to function in global scope
35 // error handler
36 app.use(function(err, req, res, next) {
```

# The current landscape

- Level 3 have started and are getting traction



```
File Edit Selection View Go Run Terminal Help
en.json def sortByKey(key, array) Untitled-1
1 def sortByKey(key, array):
  for i in range(0, len(array)):
    for j in range(i, len(array)):
      if array[i][key] < array[j][key]:
        array[i], array[j] = array[j], array[i]
  return array
2
```

## 92% of programmers are using AI tools, says GitHub developer survey

AI isn't programming's future, it's its present.



Written by **Steven Vaughan-Nichols**, Senior Contributing Editor

on June 14, 2023

Reviewed by **Min Shin**

# The current landscape

- Level 4 tools started being proposed as we saw in this course...
- ... and they will improve

Will we ever reach Level 5?

“In short, automatic programming always has been a euphemism for programming with a higher-level language than was presently available to the programmer.”

David Parnas, “Software Aspects of Strategic Defense Systems”, 1985