Educational Dimension, 2025, Vol. **12**, pp. 1–13 https://doi.org/10.55056/ed.801

Using artificial intelligence tools for automating the assessment of future computer science teachers' work

Oleksandr M. Spazhev, Oksana V. Klochko

Vinnytsia Mykhailo Kotsiubynskyi State Pedagogical University, 32 Ostrozhskogo Str., Vinnytsia, 21100, Ukraine

Abstract. This paper examines the problem of automated testing of modified programming tasks for future computer science teachers. It is recommended that GitHub Copilot be used to generate tests based on the code. This approach makes it possible to solve the following tasks: reducing the time and effort required for manual checking of programming tasks completed by students; promoting better assimilation of the material of the relevant disciplines by students; promoting the development and improvement of students' skills in algorithmisation and programming; compliance by students with academic integrity; effective use of GitHub Copilot to generate baseline tests to test modified programming assignments completed by students; ensuring the flexibility and scalability of the approach to the development of various training courses in programming; development of students' software product testing skills. In the process of research, we found the following disadvantages of using the GitHub Copilot system for generating basic tests: GitHub Copilot does not always generate perfect code or tests; for complex tasks, GitHub Copilot may require additional correction of the generated code. Therefore, it is important to check and refine the generated tests carefully, if necessary. Therefore, at the moment, we recommend using GitHub Copilot as a template generator for writing tests. The proposed approach is a promising solution for facilitating the verification of modified programming tasks and increasing the effectiveness of the education process of future informatics teachers. The conducted research opens up new prospects for effective improvement of the verification of modified tasks performed by students and the generation of tests for verification. In particular, the integration of the proposed system based on GitHub Copilot with learning management systems (LMS) and automated task verification systems. Another area of research could be exploring the possibilities of using other tools for generating tests instead of GitHub Copilot or combining them in order to obtain better results.

Keywords: automated work checking, modified tasks, programming, GitHub Copilot, future computer science teachers, educational process, testing

1. Introduction

World institutions are rapidly transforming under the influence of the development and implementation of artificial intelligence technologies. The use of these technologies in the educational process and their promising opportunities open new horizons in the field of education as well. First of all, it concerns the effective improvement of the learning process, the implementation of an individual approach, adapted learning, accessible learning, interactive learning, etc. [4, 13].

Due to the growing popularity of artificial intelligence (AI), the number of studies on the use of AI in various fields is also increasing. As a relevant and modern approach, the directions of using AI in education are actively explored by domestic and foreign scientists: as a modern trend of e-learning [19, 24], in the application of software applications of AI in education [9, 18], in educational and scientific activities [1, 22],

0009-0004-5456-1783 (O. M. Spazhev); 0000-0002-6505-9455 (O. V. Klochko)
o.spazhev@gmail.com (O. M. Spazhev); klochkoob@gmail.com (O. V. Klochko)

from the point of view of cognitive activity in the Human-AI interaction system [9], from the point of view of sustainable development and development of skills of the 21st century [18], in the study of educational analytics [14, 15], in increasing the effectiveness of the learning process and ensuring a student-centred approach [16], and other directions.

Vakaliuk [29] analysed the most popular automated systems in Ukraine for checking tasks of computer science students in programming. The main capabilities of cloud-based systems for automated verification of students' programming work Algotester [2], Eolymp [8], Topcoder [25], USACO [28], etc. were clarified [29]. Algotester is the first modern platform in Ukraine for automatic testing and conducting sports programming competitions with freely available educational materials [2]. It offers more than 200 tasks, the possibility of creating competitions, determining the overall rating, automated solution verification, and a task queue [2]. Created in 2007 to prepare for programming competitions, the Eolymp website has evolved into a platform offering assignments, learning materials, and automated problem-solving tools for teachers and students [8]. Topcoder is a web platform for programming competitions covering a wide range of tasks, from algorithmic programming to software design development, testing problem solutions, and error detection [25]. USACO is a web platform for online learning programming, where only the best students can be invited to compete [28]. However, Vakaliuk [29] notes that some platforms do not provide users with the opportunity to freely register and train, focusing only on holding international competitions.

One of the important directions of automating the verification of student works is the detection of borrowings in the students' program code. This problem was studied by Kharchenko, Didkowsky and Serdiuk [12]. The authors found out that the existing text-checking systems for plagiarism (in particular, MOSS, Codequiry, Unicheck, CCFinderX) are adapted for detecting borrowings in the text and are less effective for checking software codes [12]. In addition, among the shortcomings of these programs, the researchers identified the need for network downloads, the need for additional settings, paid access, and shortcomings in the presentation of results that the teacher cannot monitor [12]. Taking this into account, the authors developed their own system for identifying borrowings in student code in the IntelliJ IDEA development environment based on REST API, JSON, OAuth2 and JWT technologies, choosing Java 11 as the main development language. They also used the Gradle assembly system, the Spring Boot 2 framework. The system uses the Wagner-Fisher algorithm based on the Levenshtein distance for code comparison and similarity determination [12].

Antonov [3] analysed the methods of automated checking of students' works, which improves the objectivity of evaluation and saves time: The use of input/output flow redirection saves time when checking console applications by creating a text file with data for testing. Unit testing is carried out using libraries and frameworks to create tests that implement automatic code verification.

However, most studies are limited to checking whether a certain tool can solve the problems that are usually put to the student of education [30, 31]. At the same time, additional research is needed on the possibility of using AI tools to help the teacher in checking these tasks.

Thus, in Ukraine, there is a need to improve the quality of training of IT specialists [17]. Automated systems for checking tasks can increase the effectiveness of training for future computer science teachers and other specialists [6, 7]. They make it possible to effectively assess the practical skills of students in solving problems in programming and algorithmisation [23].

Popular platforms such as Topcoder [25], HackerRank [11], CodeChef [5], and others offer a wide selection of tasks and automated verification of solutions. However, they

contain a fixed set of tasks related to a specific topic and do not have enough flexibility in this.

Usually, there is a problem in the system for the solution, and it is necessary to implement the classic algorithm without modifications. At this stage, the problem of academic integrity may arise [27]. There are many implementations of the classical algorithm on the Internet, but it is usually difficult to determine whether a student completed the task independently. This can lead to the assimilation of the material at a low level [26].

If we consider more complex tasks, for example, we must determine whether to use this algorithm and perform certain modifications. This can be a difficult task for students who have only had time to master the classical algorithm or have mastered the material at a low level.

To ensure academic integrity and student assimilation of educational material at a higher level, it is often necessary to offer students modified tasks to complete. This will allow the acquirers to avoid copying ready-made solutions and stimulate the assimilation of educational material at a higher level and a better understanding of algorithmisation problems and programming concepts. However, manually checking modified assignments can be time-consuming and resource-intensive for teachers.

Therefore, this paper considers the possibilities of using AI systems based on deep learning to process and generate data for automated verification of students' work. Although scientists around the world are actively looking for effective solutions to these problems, there are still many unresolved issues in this area, specifically using GitHub Copilot for automated validation of modified student assignments.

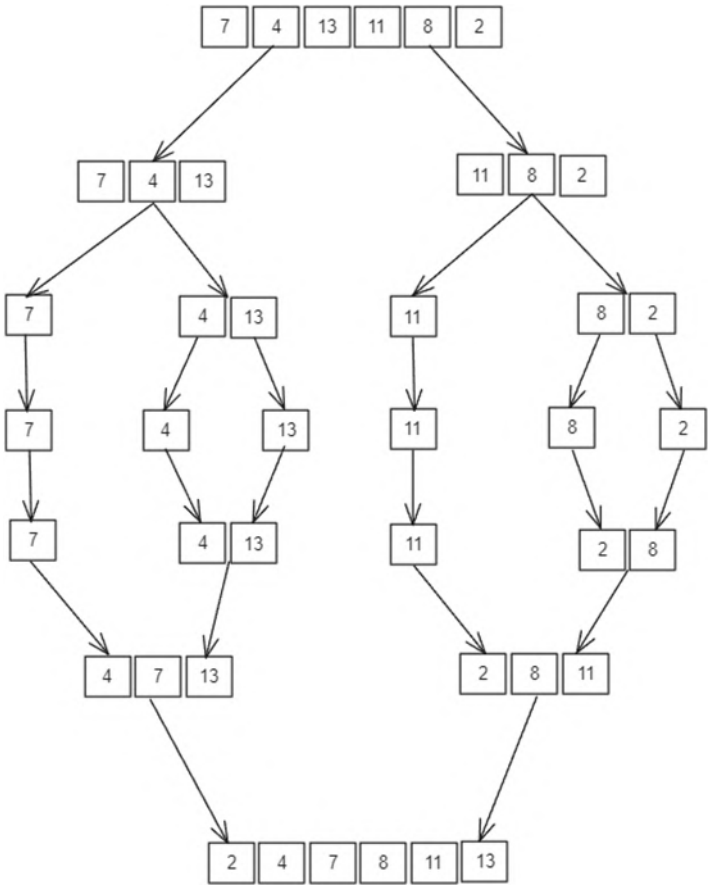

Figure 1: The working principle of the merge sort algorithm.

The purpose of the article. Research and develop methods of using artificial intelligence tools to automate the process of checking the results of the work of future computer science teachers.

2. Research results

2.1. The problem and proposals for its solution

The main problem is that checking modified tasks (tasks in which the content, parameters, etc., were purposefully changed) requires considerable time and resources from teachers. Unlike standard tasks, which automated systems can quickly check, modified tasks require careful manual checking of each solution. This can become an obstacle to the effective use of modified tasks in the educational process.

To solve this problem, it is suggested to use GitHub Copilot [10, 21], a powerful tool for code generation. One of the main functionalities of this is the generation of tests for working code. GitHub Copilot is a support system that uses deep learning to generate code and tests based on context and user input. The Copilot functionality responsible for generating tests (unit tests) can be used to create automated tests for modified tasks.

2.2. An example with the implementation of the merge sort algorithm

Let us consider an example of applying the proposed approach to the task of implementing the merge sort algorithm [20, chap. 5.2.4] in Python. Merge sort is an

```python
def merge_sort(arr):
    if len(arr) <= 1:
        return arr

    mid = len(arr) // 2
    left_half = arr[:mid]
    right_half = arr[mid:]

    left_half = merge_sort(left_half)
    right_half = merge_sort(right_half)

    return merge(left_half, right_half)

def merge(left, right):
    result = []
    i = j = 0

    while i < len(left) and j < len(right):
        if left[i] < right[j]:
            result.append(left[i])
            i += 1
        else:
            result.append(right[j])
            j += 1

    result.extend(left[i:])
    result.extend(right[j:])

    return result
```

Figure 2: Implementation of the merge sort algorithm.

efficient sorting algorithm based on the divide and conquer principle. It splits the array into two parts, recursively sorts them, and then merges them into the sorted array. The principle of operation is shown in figure 1.

One can implement the standard version of the merge sort algorithm and check its correctness on various online platforms, such as Topcoder, HackerRank, CodeChef and others. Additionally, many unmodified implementations of this algorithm are available online.

However, to better assimilate the material and ensure academic integrity, students can be asked to implement a modified version of the merge sort algorithm. For example, instead of the standard merge function of two sorted subarrays, students can be asked to implement a modified version that takes into account additional restrictions or conditions.

Online platforms likely will not be able to verify a modified version of the algorithm, making the verification process more difficult for educators. This is where GitHub Copilot can come in handy.

First, you need to have the correct code for the basic implementation of the merge sort algorithm. Let us take the code presented in figure 2 as a correct implementation of the task of the basic implementation of the merge sort algorithm.

The diagram of the hypothesis testing process is shown in figure 3.

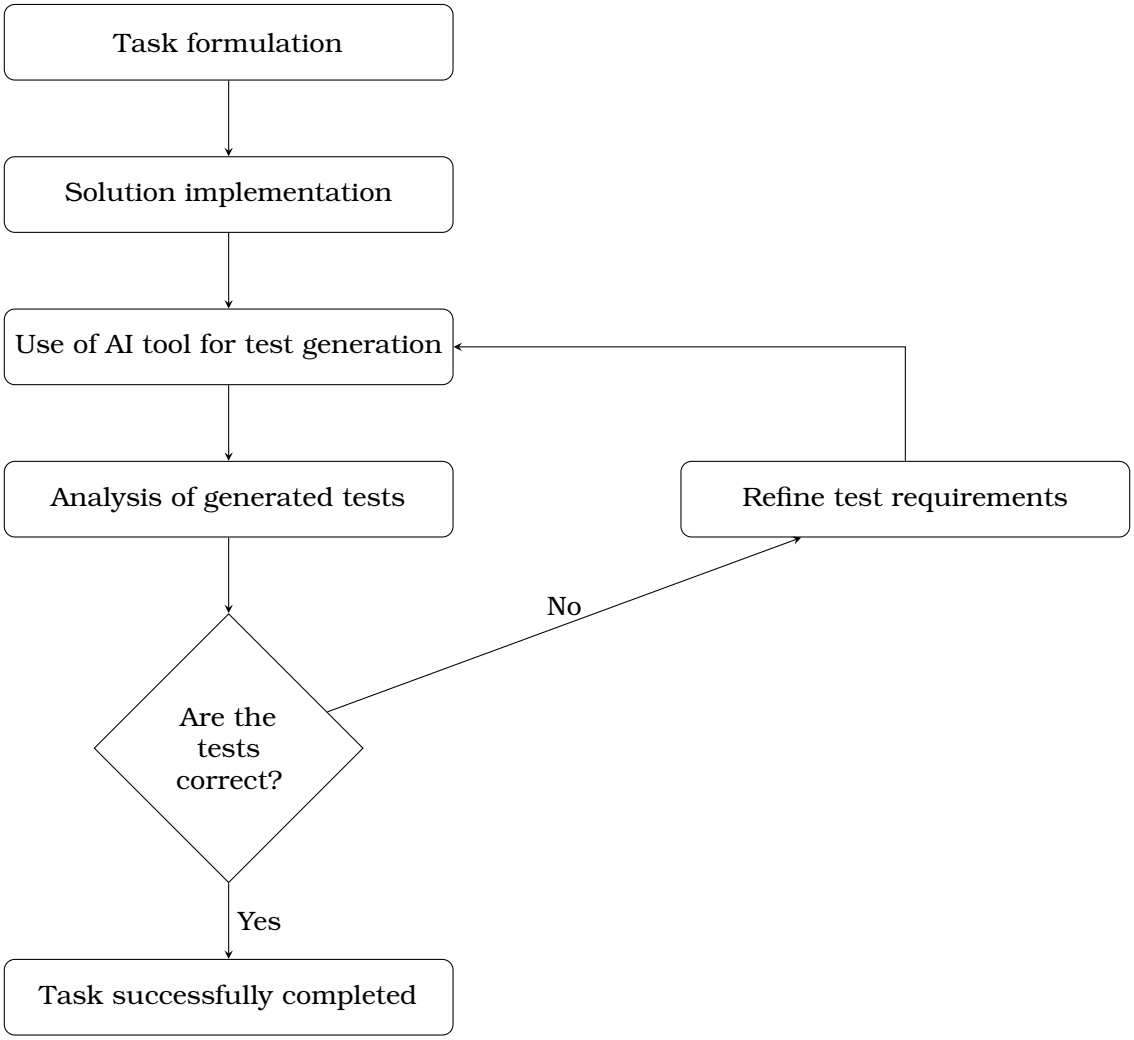

Figure 3: Scheme of the hypothesis testing process.

To generate tests, we will use the Visual Studio Code (VS Code) environment. Let us install the GitHub Copilot extension. Highlight the code of the merge function and click the right mouse button to select Copilot from the drop-down list. Next, select the Generate Tests tab as shown in figure 4.

After clicking on Generate Tests, the dialogue shown in figure 5 will open, and test generation will start automatically. The result will be displayed as shown in figure 5. Also, when you click the Create button, the file test_(the name of the file in which the test generation was called) will be automatically created.

Since the student will be asked to change the basic logic of the algorithm implementation, it is suggested that unit tests be generated, namely for the function.

The generated tests look as shown in figure 6.

The first significant problem with this code is that the merge function is not in the generated file. So, the code will not even run. In order to run it, you need to add at

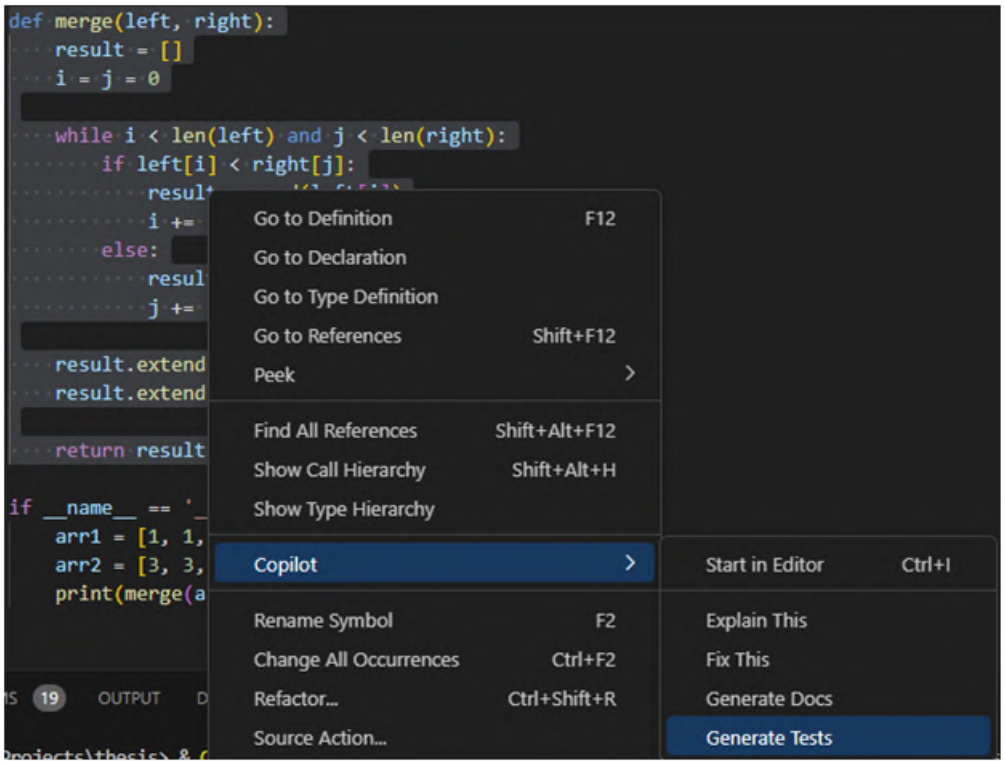

Figure 4: Opening the test generation dialog box.

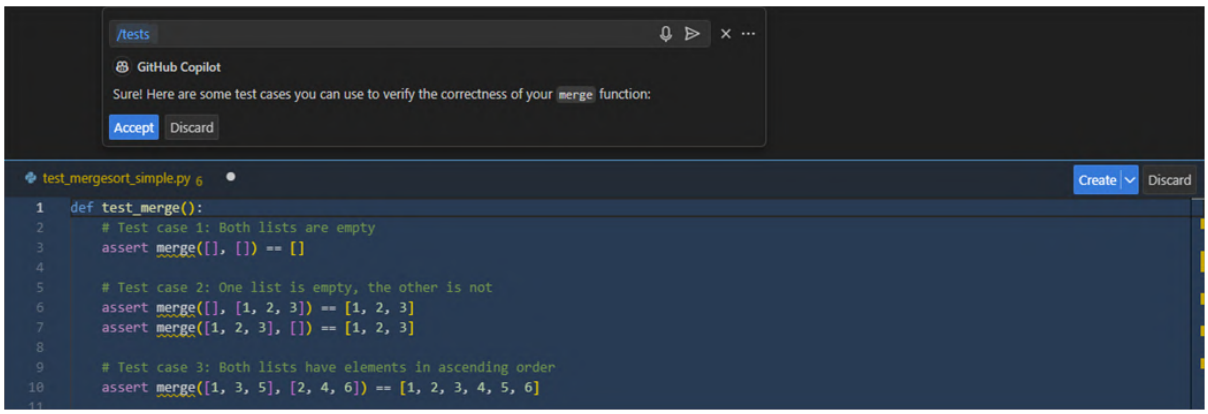

Figure 5: Test generation.

```python
def test_merge():
    # Test case 1: Both lists are empty
    assert merge([], []) == []

    # Test case 2: One list is empty, the other is not
    assert merge([], [1, 2, 3]) == [1, 2, 3]
    assert merge([1, 2, 3], []) == [1, 2, 3]

    # Test case 3: Both lists have elements in ascending order
    assert merge([1, 3, 5], [2, 4, 6]) == [1, 2, 3, 4, 5, 6]

    # Test case 4: Both lists have elements in descending order
    assert merge([5, 3, 1], [6, 4, 2]) == [6, 5, 4, 3, 2, 1]

    # Test case 5: Both lists have elements in random order
    assert merge([1, 4, 6], [2, 3, 5]) == [1, 2, 3, 4, 5, 6]

    print("All test cases passed!")

test_merge()
```

Figure 6: Generated tests.

the beginning:

```
from mergesort_simple import merge
```

where `mergesort_simple` should be replaced by the appropriate name of the module in which the merge function is located.

After correcting this flaw, the code was able to run.

However, another problem was discovered. Namely, ♯ Test Case 4. Github Copilot generated a test according to which, at the end of execution, the arrays will be sorted in descending order. However, in the given implementation, it is impossible. The expected result is $[5,\ 3,\ 1,\ 6,\ 4,\ 2]$.

After removing the test code related to Test Case 4, we will have working tests that modularly check the necessary part of the code, covering it 100%.

2.3. Modified merge sort algorithm

Next, we modify the merge function according to the given task. For example, you need to sort the elements in ascending order so that there are no identical elements. All identical elements must be replaced by a tuple(float, int), where the first value is the element itself and the second is the number of such elements. This modification of the problem is quite non-standard and introduces mixed data types in the data list.

An example of a task.

Input:

1 10 3 5 5

Output:

1 3 $(5, 2)$ 10

In the code provided at the beginning of figure 2, you need to replace the merge function with the code presented below:

```python
def get_value(value: Union[tuple,float]) -> float:
    if isinstance(value, tuple):
        return value[0]
```

```
    return value

def join_to_tuple(element1: Union[tuple,float],
    element2: Union[tuple,float]) -> tuple:
    total_number = 0
    if isinstance(element1, tuple):
        total_number += element1[1]
    else:
        total_number += 1
    if isinstance(element2, tuple):
        total_number += element2[1]
    else:
        total_number += 1
    return (get_value(element1), total_number)

def merge(left, right) -> List[Union[tuple,float]]:
    pre_result = []
    i = j = 0

    while i < len(left) and j < len(right):
        left_value = get_value(left[i])
        right_value = get_value(right[j])

        if left_value < right_value:
            pre_result.append(left[i])
            i += 1
        elif left_value > right_value:
            pre_result.append(right[j])
            j += 1
        else:
            pre_result.append(join_to_tuple(left[i], right[j]))
            i += 1
            j += 1


    pre_result.extend(left[i:])
    pre_result.extend(right[j:])

    result = []
    if pre_result:
        result = [pre_result[0]]
    for i in range(1, len(pre_result)):
        if get_value(pre_result[i]) == get_value(result[-1]):
            result[-1] = join_to_tuple(result[-1], pre_result[i])
        else:
            result.append(pre_result[i])

    return result
```

Now, let us use GitHub Copilot to generate tests for a modified version of the merge sort algorithm, namely the merge function. To do this, we will repeat the same steps as in the first part of the study.

The generated tests are presented below:

```
from mergesort_hardest import merge

def test_merge():
    # Test case 1: Both lists are empty
    assert merge([], []) == []
```

```
6
7       # Test case 2: One list is empty, the other is not
8       assert merge([], [1, 2, 3]) == [1, 2, 3]
9       assert merge([1, 2, 3], []) == [1, 2, 3]
10
11      # Test case 3: Both lists have elements in ascending order
12      assert merge([1, 3, 5], [2, 4, 6]) == [1, 2, 3, 4, 5, 6]
13
14      # Test case 4: Both lists have elements in descending order
15      assert merge([5, 3, 1], [6, 4, 2]) == [1, 2, 3, 4, 5, 6]
16
17      # Test case 5: Both lists have elements in random order
18      assert merge([1, 4, 6], [2, 3, 5]) == [1, 2, 3, 4, 5, 6]
19
20      # Test case 6: Both lists have elements with equal values
21      assert merge([1, 1, 1], [1, 1, 1]) == [(1, 1), (1, 1), (1, 1)]
22
23      print("All test cases passed!")
24
25  test_merge()
```

In this case, the code runs without the need for additional changes.

However, there is again a problem with the 4th test. Although now Github Copilot expects a different result, the program on this input will work exactly as in the previous case. The expected result will still be $[5,\ 3,\ 1,\ 6,\ 4,\ 2]$.

In addition, GitHub Copilot generated only the 1st test to verify the significant changes that were made to the code. However, many more tests are needed to cover all cases. In addition to quantity, there is also a problem with quality. After all, the expected result does not correspond at all to the result set by the requirements. Namely $[(1,\ 6)]$.

Maybe this is an unfortunate coincidence, and by adding more information to the request, GitHub Copilot will do a better job. After making a number of additional adjustments and adding context, Github Copilot was still unable to generate a single test that would verify the correctness of the modified program when the same elements were present.

This may be related to bias in AI-generated content. Namely, language models generate answers based on the data provided for training. Given the specificity of this task, the AI agent could have had difficulty generating an answer.

2.4. Advantages of the proposed approach

The proposed approach using GitHub Copilot to generate tests based on modified tasks has several key advantages:

- *Simplification of the process of checking modified tasks*. Instead of manually testing each solution submitted by students for review, instructors can rely on automated tests generated by GitHub Copilot. This greatly saves the time and effort needed to check student solutions to modified problems.
- *Promotion of better assimilation by students of the material offered to them for study*. By providing various modified tasks to students, teachers reduce the likelihood of students simply copying ready-made solutions to tasks, thereby motivating students to a deeper understanding of algorithms and programming concepts and assimilation of educational material at a higher level. This contributes to the improvement of the quality of the educational process.
- *Adherence to the principles and rules of academic integrity defined by law*. This approach to the development of modified tasks and the automated verification

of solutions provided by students to these tasks minimises the possibility of them copying ready-made solutions. Motivates students to perform tasks independently and qualitatively. In this way, confidence in the results of students' educational activities and their creative achievements increases, which contributes to the students' deeper awareness of the values of academic integrity and its observance.

- *Effective use of GitHub Copilot*. Although Copilot has some limitations and does not always generate perfect code or tests, it is a powerful tool that can greatly facilitate the process of creating tests. Instead of spending time writing tests by hand, educators can rely on Copilot to generate baseline tests that can then be refined and adapted as needed.
- *Flexibility and scalability*. The proposed approach can be applied not only to the tasks of implementing sorting algorithms but also to other types of programming and algorithmisation tasks. It can be easily adapted and scaled for different relevant courses and disciplines.
- *Facilitating the development of testing skills*. In addition to the practice of implementing modified algorithms, students can also learn about the test generation process and understand the importance of the testing process in software development. This contributes to the development of important skills necessary for future professional activity.

3. Conclusions and prospects for further research

This work considered the problem of testing modified programming tasks for future computer science teachers and proposed a solution using GitHub Copilot for test generation. During the research, an example of applying this approach to implementing a modified version of the merge sort algorithm in Python was implemented.

The proposed approach has several advantages, including simplifying the review process, ensuring better student learning and academic integrity, effective use of GitHub Copilot, flexibility and scalability in the development of programming tasks, and promoting the development of testing skills, which will improve efficiency and the objectivity of assessing students' knowledge and skills, as well as reducing the burden on teachers for developing and checking solutions to modified tasks.

However, it should be noted that while GitHub Copilot is a powerful tool, it has some limitations and does not always generate working code. With tasks of greater complexity, when using GitHub Copilot, problems may arise that it is not able to solve on its own. Therefore, it is important to check and refine the generated tests carefully, if necessary. Also, for now, GitHub Copilot should be seen as a text generator. For example, templates for writing tests that will need to be corrected according to the task. Despite these disadvantages of using GitHub Copilot in the process of ensuring the completion of the above tasks, it will allow automating and scaling the verification of student tasks.

In the future, the possibilities of integrating the proposed approach with existing learning management systems (LMS) and automated assignment verification systems can be explored. It may also be interesting to explore using alternative test generation tools or combining them with GitHub Copilot for better results.

In general, the proposed approach is a promising solution for facilitating the verification of modified programming tasks and improving the effectiveness of the educational process not only for future computer science teachers but also for students from various specialities in the field of information technology.

Funding: This research received no external funding.

Data availability statement: No new data were created or analysed during this study. Data sharing is not applicable.

Conflicts of interest: The authors declare no conflict of interest.

Declaration on generative AI: The authors have not employed any generative AI tools.

References

[1] Albusac, J., Castro-Schez, J., González, C. and Vallejo, D., 2018. Model for detecting academic failure automatically and early on. *INTED2018 Proceedings*. IATED, 12th International Technology, Education and Development Conference, pp.8388–8397. Available from: https://doi.org/10.21125/inted.2018.2033.

[2] algotester.com, 2025. Algotester. Available from: https://algotester.com/en.

[3] Antonov, Y.S., 2022. Avtomatyzatsii perevirky prohramnoho kodu na C# studentamy ta vykladachamy pid chas vyvchennia dystsyplin prohramuvannia/tekhnolohii prohramuvannia. Available from: https://tinyurl.com/2btsytpt.

[4] Burov, O.Y., Lytvynova, S.H., Semerikov, S.O. and Yechkalo, Y.V., 2023. ICT for disaster-resilient education and training. In: O.Y. Burov, S.H. Lytvynova, S.O. Semerikov and Y.V. Yechkalo, eds. *Proceedings of the VII International Workshop on Professional Retraining and Life-Long Learning using ICT: Person-oriented Approach (3L-Person 2022), Virtual Event, Kryvyi Rih, Ukraine, October 25, 2022*. CEUR-WS.org, *CEUR Workshop Proceedings*, vol. 3482, pp.1–25. Available from: https://ceur-ws.org/Vol-3482/paper000.pdf.

[5] CodeChef, 2025. Learn and Practice Coding with Problems. Available from: https://www.codechef.com/.

[6] Divasón, J., Martínez-de-Pisón, F.J., Romero, A. and Sáenz-de-Cabezón, E., 2023. Artificial Intelligence Models for Assessing the Evaluation Process of Complex Student Projects. *IEEE Trans. Learn. Technol.*, 16(5), p.694–707. Available from: https://doi.org/10.1109/TLT.2023.3246589.

[7] Dunder, N., Lundborg, S., Wong, J. and Viberg, O., 2024. Kattis vs ChatGPT: Assessment and Evaluation of Programming Tasks in the Age of Artificial Intelligence. *Proceedings of the 14th Learning Analytics and Knowledge Conference*. New York, NY, USA: Association for Computing Machinery, LAK '24, p.821–827. Available from: https://doi.org/10.1145/3636555.3636882.

[8] Eolymp, 2025. Available from: https://eolymp.com/en.

[9] Fedorets, V.M., Klochko, O.V., Tverdokhlib, I.A. and Sharyhin, O.A., 2024. Cognitive aspects of interaction in the "Human — Artificial Intelligence" system. *Journal of Physics: Conference Series*, 2871(1), p.012023. Available from: https://doi.org/10.1088/1742-6596/2871/1/012023.

[10] GitHub, Inc., 2025. GitHub Copilot · Your AI pair programmer. Available from: https://github.com/features/copilot.

[11] HackerRank, 2025. Online Coding Tests and Technical Interviews. Available from: https://www.hackerrank.com/.

[12] Kharchenko, V., Didkowsky, R. and Serdiuk, O., 2021. Development the server-side part of the system of checking software code for the presence of plagiarism. *Cherkasy University Bulletin: Applied Mathematics. Informatics*, (1), pp.68–69. Available from: https://doi.org/10.31651/2076-5886-2021-1-68-84.

[13] Kiv, A.E., Semerikov, S.O., Striuk, A.M., Osadchyi, V.V., Vakaliuk, T.A., Nechypurenko, P.P., Bondarenko, O.V., Mintii, I.S. and Malchenko, S.L., 2023. XV International Conference on Mathematics, Science and Technology Education. *Journal of Physics: Conference Series*, 2611(1), p.011001. Available from: https://doi.org/10.1088/1742-6596/2611/1/011001.

[14] Klochko, O. and Fedorets, V., 2019. An empirical comparison of machine learning

clustering methods in the study of Internet addiction among students majoring in Computer Sciences. In: A.E. Kiv, S.O. Semerikov, V.N. Soloviev and A.M. Striuk, eds. *Proceedings of the 2nd Student Workshop on Computer Science & Software Engineering (CS&SE@SW 2019), Kryvyi Rih, Ukraine, November 29, 2019*. CEUR-WS.org, *CEUR Workshop Proceedings*, vol. 2546, pp.58–75. Available from: https://ceur-ws.org/Vol-2546/paper03.pdf.

[15] Klochko, O., Fedorets, V., Klochko, V. and Kormer, M., 2022. The Use of Ensemble Classification and Clustering Methods of Machine Learning in the Study of Internet Addiction of Students. *Proceedings of the 1st Symposium on Advances in Educational Technology - Volume 1: AET*. SciTePress, pp.241–260. Available from: https://doi.org/10.5220/0010923500003364.

[16] Klochko, O., Fedorets, V., Tkachenko, S. and Maliar, O., 2020. The Use of Digital Technologies for Flipped Learning Implementation. In: O. Sokolov, G. Zholtkevych, V. Yakovyna, Y. Tarasich, V. Kharchenko, V. Kobets, O. Burov, S. Semerikov and H. Kravtsov, eds. *Proceedings of the 16th International Conference on ICT in Education, Research and Industrial Applications. Integration, Harmonization and Knowledge Transfer. Volume II: Workshops, Kharkiv, Ukraine, October 06-10, 2020*. CEUR-WS.org, *CEUR Workshop Proceedings*, vol. 2732, pp.1233–1248. Available from: https://ceur-ws.org/Vol-2732/20201233.pdf.

[17] Klochko, O. and Sharyhin, O., 2023. Professional training experience of computer science students in European Universities. *Ukrainian polonistics*, 21(1), p.694–707. Available from: https://doi.org/10.35433/2220-4555.21.2023.ped-3.

[18] Klochko, O.V., 2024. Development of critical thinking of future teachers of computerscience and mathematics using artificial intelligence tools [Rozvytok krytychnoho myslennia maibutnikh vchyteliv informatyky ta matematyky z vykorystanniam zasobiv shtuchnoho intelektu]. *Modern Information Technologies and Innovation Methodologies of Education in Professional Training Methodology Theory Experience Problems*, 1(72), p.14–26. Available from: https://doi.org/10.31652/2412-1142-2024-72-14-26.

[19] Klochko, O.V., Fedorets, V.M., Klochko, V.I. and Klochko, K.A., 2023. Anthropologically oriented strategies of interaction in the Human-Computer system. *Journal of Physics: Conference Series*, 2611(1), p.012018. Available from: https://doi.org/10.1088/1742-6596/2611/1/012018.

[20] Knuth, D.E., 1998. *The Art of Computer Programming*, vol. 3. Sorting and Searching. 2nd ed. Addison Wesley Longman. Available from: https://seriouscomputerist.atariverse.com/media/pdf/book/Art%20of%20Computer%20Programming%20-%20Volume%203%20(Sorting%20&%20Searching).pdf.

[21] Moradi Dakhel, A., Majdinasab, V., Nikanjam, A., Khomh, F., Desmarais, M.C. and Jiang, Z.M.J., 2023. GitHub Copilot AI pair programmer: Asset or Liability? *Journal of Systems and Software*, 203, p.111734. Available from: https://doi.org/10.1016/j.jss.2023.111734.

[22] Osadchyi, V.V., Pinchuk, O.P. and Vakaliuk, T.A., 2023. From the digital transformation strategy to the productive integration of technologies in education and training: Report 2023. In: T.A. Vakaliuk, V.V. Osadchyi and O.P. Pinchuk, eds. *Proceedings of the 2nd Workshop on Digital Transformation of Education (DigiTransfEd 2023) co-located with 18th International Conference on ICT in Education, Research and Industrial Applications (ICTERI 2023), Ivano-Frankivsk, Ukraine, September 18-22, 2023*. CEUR-WS.org, *CEUR Workshop Proceedings*, vol. 3553, pp.1–8. Available from: https://ceur-ws.org/Vol-3553/paper00.pdf.

[23] Paiva, J.C., Leal, J.P. and Figueira, A., 2022. Automated Assessment in Computer Science Education: A State-of-the-Art Review. *ACM Trans. Comput. Educ.*, 22(3),

pp.1–40. Available from: https://doi.org/10.1145/3513140.

[24] Semerikov, S.O., Vakaliuk, T.A., Mintii, I.S., Hamaniuk, V.A., Soloviev, V.N., Bondarenko, O.V., Nechypurenko, P.P., Shokaliuk, S.V., Moiseienko, N.V. and Shepiliev, D.S., 2022. Immersive E-Learning Resources: Design Methods. *Digital humanities workshop*. New York, NY, USA: Association for Computing Machinery, DHW 2021, p.37–47. Available from: https://doi.org/10.1145/3526242.3526264.

[25] Topcoder, 2024. Home. Available from: https://www.topcoder.com/.

[26] Turnitin LLC, 2022. Emerging trends in academic integrity: A free guide from Turnitin. Available from: https://uatpia.elearn.net.au/pluginfile.php/71044/mod_resource/content/2/TII_AI_EmergingTrends_eBook_UK_0222.pdf.

[27] Tymokhina, V., Yurchenko, V. and Nalyvaiko, O., 2024. Akademichna dobrochesnist vs shtuchnyi intelekt: etychna dyskusiia v osvitnomu prostori. In: Y.D. Boichuk et al., eds. *Tsyfrova transformatsiia osvity ta nauky : materialy II Vseukrainskykh naukovo-praktychnoi konferentsii, 14-15 berez. 2024*. Kharkiv, pp.202–207. Available from: https://dspace.hnpu.edu.ua/server/api/core/bitstreams/ef7f9271-4458-4426-97bc-fe308e498325/content#page=203.

[28] USACO, 2024. Available from: https://usaco.org/.

[29] Vakaliuk, T.A., 2019. *Theoretical and methodical principles of the cloud-based learning environment design and use in the training of bachelors in computer science*. The dissertation for a Doctor of Pedagogical Sciences degree, specialty 13.00.10 – "Information and Communication Technologies in Education" (011 – Educational, pedagogical science). Zhytomyr Ivan Franko State University, Zhytomyr; Institute of Information Technologies and Learning Tools of NAPS of Ukraine, Kyiv. Available from: https://lib.iitta.gov.ua/id/eprint/715709/1/dis_15.pdf.

[30] Wermelinger, M., 2023. Using GitHub Copilot to Solve Simple Programming Problems. *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1*. New York, NY, USA: Association for Computing Machinery, SIGCSE 2023, p.172–178. Available from: https://doi.org/10.1145/3545945.3569830.

[31] Yetistiren, B., Ozsoy, I. and Tuzun, E., 2022. Assessing the quality of GitHub copilot's code generation. *Proceedings of the 18th International Conference on Predictive Models and Data Analytics in Software Engineering*. New York, NY, USA: Association for Computing Machinery, PROMISE 2022, p.62–71. Available from: https://doi.org/10.1145/3558489.3559072.