Let’s Do It Ourselves: Ensuring Academic Integrity in the Age of ChatGPT and Beyond

Rina Azoulay ¹, Tirzta Hirst ¹, and Shulamit Reches ²

¹Affiliation not available
²Jerusalem College of Technology

October 31, 2023

Abstract

This paper addresses the emerging challenge presented by large language models (LLMs) such as ChatGPT that are able to generate solutions to tasks traditionally used to enhance student’s analytical and programming skills, particularly in programming education. This widespread availability of AI-generated solutions risks undermining the learning process and skill acquisition by enabling students to use AI generated solutions instead of practicing themselves. Addressing this challenge, our paper outlines a holistic strategy that combines educational initiatives, state-of-the-art plagiarism detection mechanisms, and an innovative steganography-based technique for watermarking AI-produced code. This multifaceted approach aims to provide evaluators with the tools to distinguish between a code generated by ChatGPT and a code genuinely created by students. With the collective efforts of educators, course administrators, and partnerships with AI developers, we believe it is feasible to uphold the integrity of programming education in this age of code-producing LLMs.
Let’s Do It Ourselves: Ensuring Academic Integrity in the Age of ChatGPT and Beyond

Rina Azoulay, Tirza Hirst and Shulamit Reches
Jerusalem College of Technology, Jerusalem, Israel
September 25, 2023

Abstract

This paper addresses the emerging challenge presented by large language models (LLMs) such as ChatGPT that are able to generate solutions to tasks traditionally used to enhance student’s analytical and programming skills, particularly in programming education. This widespread availability of AI-generated solutions risks undermining the learning process and skill acquisition by enabling students to use AI generated solutions instead of practicing themselves. Addressing this challenge, our paper outlines a holistic strategy that combines educational initiatives, state-of-the-art plagiarism detection mechanisms, and an innovative steganography-based technique for watermarking AI-produced code. This multifaceted approach aims to provide evaluators with the tools to distinguish between a code generated by ChatGPT and a code genuinely created by students. With the collective efforts of educators, course administrators, and partnerships with AI developers, we believe it is feasible to uphold the integrity of programming education in this age of code-producing LLMs.

1 Introduction

Large language models (LLMs), such as ChatGPT, can be a valuable resource for programmers, especially those looking for assistance, inspiration, or seeking clarification on programming concepts. They are trained on a vast amount of programming-related text, which enables them to understand and produce code in various programming languages. They can assist with tasks such as identifying syntax errors, suggesting code corrections, or explaining language-specific features. As a result, LLMs are capable of engaging in high-order thinking tasks and producing human-like text, raising concerns about their potential role in facilitating academic dishonesty [52].

One of the significant challenges that has emerged as a result of LLMs is their ability to provide solutions for programming tasks that students traditionally tackle to sharpen their analytical and writing skills. In particular, ChatGPT and other tools can assist in the software development process, from code generation
to optimization [5]. This causes an issue in the field of programming education, where students are expected to work on programming tasks to enhance their analytical and programming skills, and there is a considerable temptation for students to utilize these ready-made solutions instead of engaging in self-practice and skill development. Depending on AI-generated solutions could hinder the student’s learning process and obstruct the crucial development of their skills.

The challenge of ensuring integrity in assignments given the current AI abilities occupies the best teachers and lecturers around the world. In this paper, we propose a comprehensive set of strategies, designed to ensure the responsible and effective utilization of AI while preserving the integrity of educational objectives.

Our contribution is threefold: First, from an educational standpoint, we aim to instill both ethical and pragmatic incentives in students, promoting self-reliance and the value of original work. Second, from a technical perspective, we advocate the use of sophisticated plagiarism detection techniques, including human based detection and automated tools, which empower educators and teaching assistants to effectively discern original work from AI-generated submissions. Lastly, we propose a steganography-based approach to subtly watermark AI-generated code, enabling a clear identification of its artificial origin. This method will ensure accountability by visibly attributing any AI-created content to its source. By applying these perspectives, we aspire to construct a balanced and sustainable ecosystem of AI use within the educational sphere.

The remainder of this paper is structured as follows: Section 2 provides a literature review on academic integrity in the context of AI tools. Educational and ethical guidelines are discussed in Section 3. Section 4 elaborates on technical methods for detecting AI-generated code, and Section 5 outlines methods for AI signaling within the AI code. The paper concludes with Section 6, which provides conclusions and recommendations for future research directions.

2 Related Work

We proceed by offering a detailed overview of various strategies and recommendations aimed at upholding academic integrity, particularly in light of the capability of LLMs to produce intricate content. When ChatGPT was unveiled as the inaugural LLM for public use, it spurred widespread discourse both among the general public and specifically within educational institutions. The dialogue often centered on its appropriate utilization and its broader societal impact.

The integration of ChatGPT into student education brings with it a multitude of opportunities and challenges. Neumann et al. [31] focuses on articles addressing the impact of ChatGPT on higher education, specifically in the areas of software engineering and scientific writing. The paper recommends utilizing plagiarism checkers and AI detection tools, or manually examining the texts for ChatGPT fingerprints. Additionally, it suggests implementing an oral examination or requiring documentation of the examination process. Similarly, the
review [27] on the impact of ChatGPT’s on education, reveals the capabilities of it across subjects (strong in economics, moderate in programming, weak in math), and suggests that schools adopt assessment methods, update policies, train instructors, and educate students to effectively integrate ChatGPT.

Excessive reliance on AI may have lasting impacts and potentially undermine the professional development of upcoming generations. The study referenced in [30] discusses concerns about the next generation relying solely on AI to complete tasks without putting in effort, while emphasizing the importance of educating them about the limitations and potential biases of AI and how to evaluate the information it provides. The paper’s main conclusion is that artificial intelligence applications like ChatGPT function as tools to support human work rather than replacing it entirely. While they can assist in task completion and enhance the quality of writing, they cannot completely replace human expertise in writing and critical thinking.

As highlighted in our introduction, the impact on computer science education holds unique significance. This topic is further explored by Quersi [42], who delves into the integration of ChatGPT in undergraduate computer science curricula, specifically focusing on foundational programming concepts. An experiment was organized with two distinct groups of students, divided into teams, tasked with solving programming assignments: one group had access only to textbooks and notes with no internet, while the other group was equipped with ChatGPT. As it turned out, the teams using ChatGPT achieved higher average grades compared to the group that did not use it across all programming tasks. Nonetheless, they also spent more time grappling with the most complex problems. Finally, advantages and disadvantages of ChatGPT in teaching computer science are discussed, and various strategies are suggested to avoid misuse of ChatGPT by programming students, including the use of automated tools to detect plagiarism.

However, detecting AI-generated content remains a formidable challenge. While numerous tools have been developed, their efficacy is yet to be fully realized. In a study conducted by Khalil et al. [24], the aim was to explore the originality of content generated by ChatGPT. To accomplish this, the researchers employed two popular plagiarism detection tools to assess the originality of 50 essays produced by ChatGPT on various topics. The findings of the study manifest that ChatGPT possesses a significant potential to generate sophisticated and intricate text outputs, largely eluding detection by plagiarism-checking software.

To address complexities involved in identifying and mitigating academic dishonesty, Cotton et al. [12] propose the establishment of policies and procedures, provision of training and support, and utilization of diverse methods to detect and prevent cheating. In addition, they suggest that teachers may consider mandating a written declaration from students asserting the originality of their work. However, such a declaration may not be genuine in actual situations. The question is what can be done to encourage students to submit independent work.

Considering foreign language studies, Perkins [36] highlights the use of LLMs
by foreign Language (EFL) learners, including potential assistance in digital writing and beyond, and delves into the concerns regarding academic integrity associated with students’ use of these tools. He suggests that the use of LLMs should not be considered plagiarism, provided that students clearly disclose their use of the technology in their submissions. Moreover, there are legitimate uses of these tools in student education. He concludes that the determination of whether a specific use of LLMs by students is deemed academic misconduct should be based on the academic integrity policies of the institution. These policies must be updated to reflect how these tools will be utilized in upcoming educational settings.

We proceed by describing some studies suggesting how to incorporate the appropriate use of LLMs in education while handling its challenges and limitation. Kasneci et al. [23] caution against over-reliance, emphasizing the importance of recognizing LLM limitations and promoting teacher training. The authors advocate the use of LLMs as supplementary tools alongside other educational resources, fostering student creativity through independent projects, and integrating critical thinking into curricula. They also stress the significance of human oversight in reviewing LLM outputs and the necessity of a strategy focused on critical thinking and fact-checking for effective LLM integration.

Kumar et al. [26] detail how LLM technology targets education and propose recommendations that educators may adopt to ensure academic integrity in a world with pervasive LLM tools. They emphasize the importance of fostering a genuine desire to learn and develop deep research skills among students to counteract the effects of LLM generators. They recommend to prioritize conferencing with students about their writing to foster collaboration and skill development, although this approach is challenging in large size classes.

We end this section with a description of two studies related to the impact of LLMs on higher education. Tlili et al. [53] conducted a qualitative study to explore the impact of ChatGPT on education, structured in three phases. In the first phase, a social network analysis of tweets showed that the majority of public sentiment on social media was positive towards ChatGPT, with positive sentiments were expressed nearly twice as much as negative ones. The second phase involved interviews with 19 stakeholders who blogged about their experiences with ChatGPT. The analysis revealed that many viewed ChatGPT as transformative for education, but there were concerns about its potential to hinder innovation and critical thinking. While many found ChatGPT’s responses satisfactory, some noted occasional errors and outdated information. Ethical concerns included potential plagiarism, misinformation risks, and privacy issues. The final phase presented user experiences from ten educational scenarios with ChatGPT, highlighting challenges such as cheating, truthfulness, and potential manipulation.

Sullivan et al. [51] analyzed 100 news articles to study ChatGPT’s influence on higher education across four countries (US, UK, Australia and NZ), published between 2020 and February 2023. Key themes included academic integrity concerns, potential AI misuse, and debates on cheating. In response, some universities reintroduced supervised exams, while others focused on as-
signments demanding critical thinking. Institutional policies varied, with some banning ChatGPT and others permitting its use under conditions, often citing its inevitable workplace integration. Many articles also provided strategies to combat plagiarism and promote student originality.

The numerous educational benefits of ChatGPT include personalizing learning experiences and enhancing employability as AI reshapes industries. It aids non-traditional and non-native English-speaking students, serves as a quasi-translator, and provides tools for those with disabilities, mitigating associated stigmas. However, concerns about ChatGPT encompass potential disinformation, copyright violations, privacy breaches, and data security. While some worry about students losing critical thinking abilities, others champion the integration of AI into teaching and assignments.

Finally, the authors warn that portraying ChatGPT mainly as a cheating tool rather than a learning aid can shape public opinions about university education, influence academic reactions, and affect student perspectives on the appropriate use of this tool. Students exposed to articles about cheating with ChatGPT might be more inclined to cheat themselves. Research shows that the perception of frequent cheating opportunities increases the actual incidence of cheating among students.

Our study emphasizes achieving integrity within computer science education and outlines active strategies to enhance self-work on programming exercises. In particular, we detail practical implementations and suggestions to enhance integrity, focusing on education methods, as well as plagiarism detecting methods, used by course teams with or without the cooperation of AI developers.

3 Educational Guidelines to Encourage Self-Work

Plagiarism is the act of copying or closely imitating the work of another person or source without proper attribution or permission. It involves presenting someone else’s work as one’s own, without giving credit to the original author or source [18]. The challenge of maintaining honesty and integrity in academic writing, especially in regard to student assignments, has been a longstanding concern. Its prominence has notably grown with the rise of the Internet [9]. However, in the age of ChatGPT and other LLMs, this issue has evolved and taken on new facets.

In order to effectively combat the challenges posed by AI-generated code, especially concerning plagiarism, it is essential to begin with a foundation in ethical education. This phase involves defining the ethics code of honest AI use, fostering trust between educators and students, encouraging students to honestly submit their work. The purpose of this is twofold - to maintain integrity and to enable students to acquire the necessary experience for their future professional tasks.

This principle is especially pertinent in the field of computer science, which is the main focus of this article, wherein exercises provide developers with the fundamental skills needed for their profession.
In an interesting dialog about plagiarism, conducted by Prof. Frye with ChatGPT3, shortly after it was published [18] the chatbot explained the disadvantages of plagiarism. Firstly, the student themselves may be harmed if they are caught. In addition, such usage violates the ethical standards of academic writing and can have negative effects on the student’s academic career and future opportunities. In addition, the copyrights of the original authors or sources of the text are violated. Finally, the educational institution, its credibility and its reputation may be harmed if students use an AI text generator to produce academic writing. The chatbot “concludes” that using an AI text generator to produce academic writing can be seen as a form of academic dishonesty.

3.1 The boundary question: what is recommended and what is not allowed

To address the matter of honesty in the completion of exercises from an educational and ethical perspective, we must initially delineate what constitutes fair use of AI and what unequivocally qualifies as plagiarism. Thus, we begin by clarifying the nuanced distinctions between the recommended uses of AI tools in academic settings and those practices that are strictly prohibited, aiming to clarify the importance of defined boundaries for ethical and appropriate utilization.

Leveraging AI tools can be useful in both academia and industry. Hence, it is critical for students to become proficient in their use. In particular, there may be cases where the use of AI tools is allowed, and even encouraged, in programming assignments, for example, in large projects where some AI generated sections can be integrated. In such cases, it is important to emphasize what use is possible and what use is not legitimate, and even ask the student to describe the uses he made of AI tools during his work.

A widely advocated strategy, as highlighted by Qureshi in 2023 [42], is to assign tasks where students are encouraged to employ AI tools. This methodology not only preps them for a competitive job market where AI integration is increasingly prevalent but also offers educators an insight into how well students can exploit the capabilities of these advanced technologies. Andres [4] promotes the idea of treating AI use as a type of tool, the use of which should be practiced and regulated. He also recommends refining the definition of each assignment to allow or forbid the use of these tools, as determined by the course team.

However, it is important to emphasize that students should cultivate basic programming skills, even in the age of AI, for several reasons. Firstly, in environments with stringent safety requirements, the use of AI tools is often prohibited. As a result, graduates relying solely on AI tools, without genuine programming experience, may find themselves ill-equipped. Furthermore, despite AI’s capacity to generate code snippets, a professional approach is essential to craft precise prompts, comprehend the generated solutions, critique them, and seamlessly integrate these snippets into a broader project. Consequently, fundamental programming skills remain indispensable, even with the advent of advanced AI tools.
This significant issue with this approach is that it might lead to programming course graduates who can only use AI tools and who actually do not know how to program. This situation could limit them from being able to audit, monitor, repair, and refurbish the products of AI tools, as well as tackle unique problems that the AI cannot solve because it has not been trained on similar problems. This scenario can be compared to programmers coding in C programming language and then auto-converting it to Assembly. While this showcases a distinct skill set, it does not reflect proficiency in Assembly programming. Therefore, when our objective is to instruct students in programming, the foundation should be on hands-on coding, rather than relying on prompts for an AI-based code generator.

It is imperative to clearly delineate what qualifies as acceptable use of AI and what is deemed as plagiarism. For instance, leveraging ChatGPT to brainstorm solutions for a glitch or to interpret an error message is perfectly valid. It is akin to seeking guidance from a peer or a course assistant. However, relying on ChatGPT to complete the exercise in its entirety is tantamount to plagiarism.

It is crucial to recognize that the boundary separating permissible and impermissible AI uses can be subtle, and its position can vary based on different applications. A general guideline is: any task where seeking help from a friend, private tutor, or teaching assistant does not equate to plagiarism is permissible. Conversely, if a task, when done by another person, would signify a lack of individual effort, seeking ChatGPT’s assistance for the same would breach the stipulation of independent work. This demarcation, as highlighted, should be clearly outlined by the course’s administrative team in advance.

Figure 1 presents categorization that illustrates the most prevalent uses of AI tools, the permissible uses, and those that are strictly off-limits.

Given the fact that plagiarism caused by using AI tools may be ambiguous, it is important to define a clear ethics code of honest AI use. Rudolph et
al. [44] state the critical importance of digital literacy education and the integration of AI tools into the curriculum, and advise universities to update their academic integrity policies to include clear guidelines on AI tool usage, emphasizing proper use, update academic integrity policies, and encourage, support and share research on the effects of AI tools on learning and teaching.

3.2 Encouraging Academic Integrity

An interest statement was addressed by [13], "Plagiarism is best addressed pedagogically. Only through education can students begin to change and improve how they write". In this section, we concentrate on the educational means to encourage students to work independently. Students can be educated about the ethical implications of passing off AI-generated work as their own. This could involve discussions about academic integrity, the importance of developing one’s own skills, and the potential consequences of cheating.

One aspect is honesty and ethics. Students with high personal morals would prefer to receive a slightly lower grade, and not cheat the public. Furthermore, unjustified high grades will ultimately hurt them and their friends because once the companies in the workplace recognize that the grades are unjustified they will not take their degree seriously.

Park reviews [35] the efforts of North American institutions to enhance academic integrity in the digital age. The review explores various dimensions, such as the definition of plagiarism, student perspectives, its prevalence, underlying causes, challenges posed by the digital environment, and strategies to reinforce academic integrity. In particular, Park examined the honor codes adopted by many institutions to foster values like truth, accountability, and responsibility. A prime example is the Center for Academic Integrity (CAI) at Duke University, which places greater emphasis on imparting academic values rather than merely identifying undesirable behaviors. While debates continue about whether or not integrity can be taught, evidence indicates that institutions with honor codes witness enhanced academic honesty and fewer cases of self-reported cheating. For honor codes to be effective, their goals must be clearly communicated, augmented by discussions between faculty and students, and supported by the institution’s steadfast commitment to academic integrity. Additionally, honor codes are most effective when students play a significant role in their implementation. However, systems entirely managed by students or those relying on students to report dishonesty rarely work effectively.

Malik et al. [28] investigated an online university in Pakistan to determine the reasons students plagiarize. They identified several factors: a deficiency in plagiarism education, ambiguous guidelines for citing sources, poorly trained educators, inadequate penalties for plagiarists, students’ poor time management, fear of failure, lack of confidence, and a prevalent culture that accepts copying. They proposed a tri-fold solution, based on awareness, support and prevention: Awareness: Train educators and educate students about plagiarism, its implications, university responses, and the correct ways to cite sources. Support: Assist students with their challenges and foster an environment where they can
discuss their difficulties, thereby decreasing their likelihood to resort to copying.

**Prevention:** Universities should enforce clear anti-plagiarism regulations and ensure consistent penalties. If there are visible, strict repercussions for plagiarism, it might deter both students and educators. A common motivation for students to plagiarize is the belief that they will not be detected or that penalties are lenient.

The educational efforts to reduce plagiarism are considered in the study by Perkins et al. [37]. They introduce the AEM program, which aims to enhance students’ English proficiency, educate them about expected academic conduct, and discourage contract cheating. They illustrate how this program assists in lowering overall plagiarism levels. Moreover, they show how 'fingerprints' of a writing style can be used to cross-check submissions for plagiarism.

Uzun and Kilis [54] extended the Theory of Planned Behavior (TPB) [3] to understand the factors leading to plagiarism, replacing the traditional concept of perceived behavioral control with a measure of Information and Communication Technologies (ICT) literacy. Using a survey of 588 university students and subsequent analysis via partial least squares structural equation modeling, the findings showed that attitude, information literacy, moral obligation, and past behavior were significant predictors of the intent to plagiarize. However, subjective norms, Internet literacy, and computer literacy did not play a significant role in predicting this intention. Their findings indicate that teaching ethical morality and literacy may assist in reducing dishonest behaviors.

In their study, Elshafei and Jahangir [16] explored the reasons for plagiarism among Jazan University students. The key reasons identified include pressure on students and emerging innovators to perform, leading them to academic dishonesty; the potential influence of new technologies, particularly the Internet, in increasing plagiarism; difficulties in time management; and social pressures. In addition, ambiguity in plagiarism policies, exacerbated by the absence of specific punitive measures in current regulations, can lead to academic misconduct. The study suggests employing a mix of proper education and technological interventions to address plagiarism and underscores the need for heightened vigilance on this matter in transnational higher education.

After reviewing studies that outline the objectives of plagiarism and the methods academic institutions use to address it, we then present a logical model that explains the motivation for genuine academic preparation.

### 3.3 A Utilitarian Method to Promote Independent Work

Beyond the educational and ethical considerations, the practical implications of dishonesty, especially regarding the long-term advantages of independent work, cannot be ignored. A student who realizes that assignments are designed for their personal growth and skill development will be more inclined to work independently. It is essential that they understand that, in the end, their true abilities will be evident in the professional arena. Without genuine practice, making mistakes, and learning from them, one cannot truly become an expert. Thus, relying on AI tools for assignment preparation deprives students of fun-
damental knowledge and the chance to apply what they have learned. Without these experiences, they will not find a position that matches the potential they could have achieved through independent work.

Combining these motives together with the danger of being caught given the existence of sufficient ways to identify the use of the AI tool (as defined later in Section 4 and 5), then there will be sufficient motivation among the absolute majority of students to work on their assignments independently. The following three motivations together, namely the moral side, the long-term professional benefit, and the fear of being caught, can help motivate the rational student to submit exercises that he has completed on his own.

Let’s define the following parameters:

- $P^{CE}$: The probability of being caught if independent work is enforced by the institute.
- $P^{CN}$: The probability of being caught if independent work is not supposed to be checked and enforced. Clearly, $P^{CN} < P^{CE}$, because if efforts are invested on enforcing independent work, the chance of getting caught increases.
- $\delta$: The discounted value of future benefits. The discount factor $\delta \in [0, 1]$ determines how much importance is given to future rewards compared to immediate ones in decision-making processes. A higher $\delta$ means future rewards are valued similarly to the current ones, while a lower $\delta$ emphasizes immediate rewards.
- $\text{Cost}_{SelfWork}$: The cost (in time and efforts, a negative value) of independent work on exercise.
- $\text{Cost}_{Manip}$: The cost (in time and efforts, a negative value) of performing some manipulations to alter AI generated exercises. Clearly, $\text{Cost}_{Manip} < \text{Cost}_{SelfWork}$, otherwise, no student will prefer to take an AI generated code and alter it, if independent work is easier.
- $P^{CE}_{Manip}$: The probability of being caught when AI code was manipulated, if independent work is enforced by the institute.
- $P^{CN}_{Manip}$: The probability of being caught when AI code was manipulated, if independent work is not enforced by the institute.
- $\text{Penalty}$: The penalty the student receives if plagiarism is discovered.
- $\text{ExpVal}$: The future value of professional experience and ethical behavior.

Given the above parameters, the utility of a student to be honest is defined as follows:

$$\text{Util}_{student}(honest) = -\text{Cost}_{SelfWork} + \delta \cdot \text{ExpVal}$$
On the other hand, the utility of a student that submits an AI product, depends on the probability of being caught. This depends on the case whether the institute enforces independent work and invests resources to detect plagiarism.

$$\text{Util}_{\text{student}}(\text{submitAI}) = \begin{cases} -P^{CE} \cdot \text{Penalty} & \text{if independent work is enforced} \\ -P^{CN} \cdot \text{Penalty} & \text{otherwise} \end{cases}$$

where the probability of being caught is higher if enforcement is performed: $P^{CE} > P^{CN}$.

A third option is to use AI tools but manipulate their results. This causes the student to spend time and effort on manipulation, but the probability of being caught still exists.

$$\text{Util}_{\text{student}}(\text{manipulate}) = \begin{cases} -\text{Cost}_{\text{Manip}} - P^{CE}_{\text{Manip}} \cdot \text{Penalty} & \text{if independent work is enforced} \\ -\text{Cost}_{\text{Manip}} - P^{CN}_{\text{Manip}} \cdot \text{Penalty} & \text{otherwise} \end{cases}$$

Figure 2 represents a decision tree where the institute should decide whether to enforce independent work on programming exercises, and then the student has to decide whether to perform his exercises independently, or rely on AI tools.

Given the above formulas, we can compute the expected utility of the student for each option, and compute the calculations of the conditions for honest independent work on the programming exercises.

In particular, the student is motivated to submit exercises completed independently if:

$$\text{Util}_{\text{student}}(\text{honest}) > \text{Util}_{\text{student}}(\text{submitAI})$$

and

$$\text{Util}_{\text{student}}(\text{honest}) > \text{Util}_{\text{student}}(\text{manipulate})$$
Given the definitions above, this means that, if independent work is enforced, the student is motivated to perform his/her exercises independently, if:

\[-\text{Cost}^{\text{SelfWork}} + \delta \cdot \text{ExpVal} > \max(-P^{CE}, -\text{Cost}^{\text{Manip}} - P^{CE}_{\text{Manip}}) \cdot \text{Penalty}\]

if independent work is enforced, and

\[-\text{Cost}^{\text{SelfWork}} + \delta \cdot \text{ExpVal} > \max(-P^{CN}, -\text{Cost}^{\text{Manip}} - P^{CN}_{\text{Manip}}) \cdot \text{Penalty}\]

otherwise.

Therefore, as the likelihood of being caught increases, students’ motivation to submit their own work also rises. However, if completing work independently is more challenging, entails higher costs, and if altering AI-generated work is easier and harder to detect, stronger efforts and more severe penalties are necessary to motivate students to personally complete their exercises.

In summary, the combination of effective educational motivation and a campaign to promote the benefits of independent work is crucial. When an institution upholds high standards for independent work, it increases the perceived risk of using AI-assisted solutions and boosts the rewards of individual effort. This deters students from relying on AI and encourages them to hone their skills. The next sections detail technical methods for detecting and preventing plagiarism to ensure academic integrity and the proper training of future software developers.

4 Educational Staff-Based Solutions for the Plagiarism Challenge

In this section, we address the challenge of developing appropriate programming exercises and code assignments for undergraduate students, and evaluating them. This challenge has become prominent in the age of AI tools, given the widespread and free access to such tools that can effortlessly generate code solutions for students. Consequently, many educators and professors are wrestling with ensuring that programming students truly possess authentic programming skills and a deep grasp of the basics. How can we deter students from merely depending on AI tools to breezily finish their assignments? How can we ascertain that the work was genuinely executed, at least in part, by the student, and not wholly produced by AI? Here, in-depth cogitation of the educational team on how to deal with this challenge is required. In this section, we aim to present two approaches to overcome this challenge. One approach involves the development of learning and practice exercises that will encourage students to work independently, and the second explores methods to detect AI-generated, non-original code.
4.1 Practical Methods to Encourage Original Work.

In this section, we would like to propose practical suggestions for exercises and challenges that will provide practical incentives to students and compel them to work independently. We focus on suggestions of various kinds, including those concerning the formulation of the exercise, monitoring the solving process, and assessment ideas that can be applied to check independent work.

**Unique Assignment Design:** In order to reduce the option to rely on AI content in programming, is to assign a task on an extremely unique and uncommon topic with highly specific requirements. In such instances, since ChatGPT relies on information available on the internet, which is limited regarding this particular topic, it may provide the wrong solution, or alternatively it may provide one uniform response. Consequently, multiple responses that exhibit a high level of uniformity, strongly suggests a high probability of copied content.

**Multimedia Integration:** Another approach, suggested in several studies [19, 32, 44, 52] is to incorporate multimedia inside the exercise definition, like images and voice. This can make it more difficult for students to cheat and for ChatGPT to generate accurate responses.

**Logical Questions:** One approach is to leverage the relative weakness of AI tools in understanding written code logically, including the code they generate. As highlighted in previous studies ([10, 19, 21, 29, 58]), AI often struggles with accurately responding to such questions and can exhibit errors or logical contradictions. By integrating logical questions that emphasize analysis over mere techniques, educators can evaluate students’ understanding of their submitted code. Given AI’s inconsistent proficiency in this area, accurate responses might indicate that students have genuinely understood and worked on their assignments, even if they have made some use of AI tools.

The following examples can be added to the programming assignment, in order to assess the students’ comprehension:

1. Describe the flow of execution or control flow within the code.
2. Identify any potential errors or bugs in the code and propose solutions.
3. Explain the algorithm or logic behind a particular section of the code.
4. Discuss the efficiency or complexity of the code and suggest possible optimizations.
5. Provide conclusions based on the programming code.
6. Describe any challenges encountered during the coding process, specifically highlight functions or concepts that posed difficulties, and explain the reasons behind those challenges.
7. Modify or extend the code to achieve a specific outcome and explain the changes made.
8. Compare the given code with alternative approaches and explain the advantages or disadvantages.
By incorporating such comprehension-based questions into programming assignments, you can effectively assess the students’ understanding of the code and identify any potential instances of plagiarism or undue reliance on artificial intelligence-generated code.

**Emphasizing the Holistic Learning and Programming Process**

To increase student’s motivation and involvement, teachers can emphasize the work process, rather than solely focusing on the final outcome. In order to reach this goal, using dedicated editors that trace and monitor the students’ activity may be useful to ensure integrity. In addition, students may be required to provide comprehensive documentation of their programming work process, encompassing each stage from inception to completion. The following strategies can be employed to fulfill this requirement:

- Ask the students to program using a dedicated tracing and monitoring editor, to ensure an independent work process.
- Ask the students to highlight any errors encountered, subsequent modifications made to rectify them, the initial algorithm, the initial code functions written, and more.
- Request a video clip in which the students articulate and describe their work process [50].
- Ask for screenshots that capture the entire programming process, accompanied by written explanations and descriptions.
- Instruct the students to capture a video of their entire work process and submit it along with the exercise. (Note: Implementing this requirement could face substantial obstacles due to privacy concerns).

The mere knowledge that students will be required to present their complete work process significantly enhances the likelihood of independent effort, discouraging them from simply obtaining pre-made work from an artificial intelligence programmer. Even if they do acquire a completed work, the expectation to showcase their own work process will compel them to strive for a deeper understanding in order to replicate such a process.

**Strategic Approach**

Teachers can incorporate challenging elements into the programming task, which might be too advanced for students. The aim is to determine whether students can tackle a task that appears beyond their capabilities. If students provide answers that significantly exceed their expected skill level, this will suggest that they might have copied it or sought assistance from sources such as AI tools.

**Oral Presentation**

Incorporating oral presentations or defenses is an effective method to combat plagiarism, as cited by [31, 50, 52]. Requiring students to verbally explain their code not only ensures their genuine understanding [50] but also helps educators assess their comprehension of programming concepts. As suggested by [20], students should be prepared to defend their ideas and address questions or modifications proposed by instructors in real-time.
oral examinations with code submissions is favored by educators. Examinations can be administered to all students, a random selection, or those arousing suspicion. Potential questions during the oral defense may include:

1. Discuss the role of specific variables or parameters in the code.
2. Explore the impact of altering a parameter on the outcome.
3. Elaborate on the implications of a recursive call from one function to another.
4. Describe the significance of a particular function and the consequences of modifying it.
5. Predict the program’s output after a parameter’s value or type is changed.

**Hand-Written Exams** An alternative approach with a similar outcome is to periodically gather students for in-person assessments of their programming exercises, specifically when they lack internet access [52]. As part of these assignments, students will be required to write code without the ability to run it.

To summarize, this section presents various practical strategies educators can use to motivate students to produce original work and deter reliance on AI tools for programming assignments. These include designing unique assignments, incorporating multimedia elements, emphasizing logical questions, focusing on the entire learning and programming process, requiring students to give oral presentations, and conducting handwritten exams. These methods aim to ensure students genuinely understand their work and discourage plagiarism.

4.2 Plagiarism detection methods

From the above, we understand that the key to ensuring honesty in exercises is the students’ awareness that any copied work is likely to be detected. In the subsequent subsection, we shift our focus to the identification of non-original code, especially code produced by AI systems, whether or not it has undergone slight modifications. We will distinguish between manual detection techniques, where an examiner identifies suspect patterns, and automatic detection methods utilizing state of the art advanced tools.

4.2.1 Manual Detection of AI-Generated Code

Determining whether a code or programming essay came from an AI tool might be challenging, as the language model’s outputs do not contain identifiable markers indicating their source. However, there are several methods that can be used to assess if part of the code or programming project might have been generated by a language model like ChatGPT:
Completely Standardized Code Patterns: Language models often generate a code that follows certain patterns, which may differ from a human-written code. Specifically, its output aligns precisely with the standard stylistic guidelines of the language.

For instance, in Python, ChatGPT’s output adheres to the standards outlined in the official style guide for Python code [55]. Specifically, certain coding style markers can indicate ChatGPT’s output in Python code.

- **Identifiers naming conventions**: short and lowercase function, variables, modules and packages names, underscore for clarity; CapWords convention for classes; uppercase with underscores for constants.

- **Indentation**: Four spaces per indentation level; when continuing a line, and an extra level of indentation to distinguish the continued line from the next code block.

- **Specific spaces format**: one space before and one space after arithmetic, assignment and comparison operators; one space after commas; no spaces immediately inside parentheses, brackets, or braces; and no space before a comma, semicolon, or colon.

- **Blank lines**: Two blank lines to separate top-level functions and classes, one blank line to separate methods within a class, one blank line in functions to indicate logical sections.

- **Specific type of comments**: block comments, inline comments, and docstrings: with predefined guidelines for writing and formatting, as well as recommendations on which type to use for different situations.

Even though the adoption of the official style is strongly suggested, developers often veer away from it occasionally. Hence, full adherence to the suggested style might raise suspicion. Consequently, a practical method to detect AI-crafted output from ChatGPT involves inspecting structural patterns in the text, like unchanged formatting or repetitive function names. See Figure 3 for an example of the code styling in a ChatGPT generated code, where the uniform styling according to the standard is noticeable.

It’s worth noting that automated refactoring tools are available to standardize existing code; PyCharm, for example, offers this functionality. If the goal is to identify code that has been automatically generated by examining uniform coding style, educators should request students to avoid these tools when submitting assignments for evaluation.

**Knowledge of Student’s Proficiency**: Become familiar with the typical programming proficiency of the student or the class. If the submitted code is vastly different in style, quality, or complexity compared to their previous work or the class average, it may warrant further investigation.

These methods can only provide indicators or raise suspicions, but they cannot definitively confirm the origin of the code as AI-generated code. In order to conclusively determine this, additional evidence or insights from the student.
themselves may be required. Nonetheless, these techniques can provide a high probability of detecting potential plagiarism. Consequently, these techniques serve as tools to help determine which students should be tested orally, or asked additional questions to check their comprehension. This process is similar to the random selection of individuals to be checked at airports, where the randomness of the distribution increases the likelihood of selecting individuals with suspicious parameters.

4.2.2 AI-Generated Code Detection Software

We proceed by describing commonly used automated tools available for educators that can detect non-original code, whether it results from copying from another student or is generated by AI. Over the past few years, several automated tools have been developed for automated exercise submission and grading [34]. Specifically, automated tools often have capabilities to detect similarity in the submitted exercises [2, 40]. Such detection tools designed for programming assignments can be utilized to compare the submitted code with existing code.
repositories, public code bases, or other students’ submissions.

Unlike classical plagiarism detection tools, which identify cheating by locating overly similar assignments, detecting AI-generated output is trickier. If students pose the same question to an AI, they might receive different answers. Thus, even if they provide nearly identical prompts to the AI, their responses might not align. This suggests that traditional methods and software designed to detect copied work may not be effective.

Numerous methods exist to identify whether a text has been generated by a language model like ChatGPT. One strategy involves employing machine learning methodologies to develop a model capable of discerning the distinct writing style and patterns characteristic of the language model. This entails scrutinizing elements like text structure, grammar usage, and vocabulary selection. An alternate tactic compares the text with many examples that the language model has already made. This helps reveal similarities and connections. Also, the detection tool can search for specific clues in the text, like certain words or patterns that may indicate it was produced by the language model. These methods can also be used to locate code that has been generated by AI tools in a manual manner, as described in Section 4.2.1.

We evaluated 9 free code-origin detection tools, each described in detail in [6]. For each tool, we ran 10 pairs of Python programming codes, where each pair solved a specific code challenge, with two solutions, the first one generated by ChatGPT, and the second solved by a human. Each of the 20 examples was then executed on each of the AI detecting tools. Table 1 describes the tools examined in this study. Note that the "True AI" column refers to the number of true positives, which indicates the number of instances where the system correctly identified the code as AI-generated code. Conversely, the "True human" column represents True Negatives, signifying instances where the system accurately recognized the code as human-generated (and not AI-generated). The "Error" column provides the number of cases where the software crashed, sometimes due to a long input code. Note that some of the software returned percentages for their classifications. In such instances, any category (either human or AI) with a percentage greater than 50

<table>
<thead>
<tr>
<th>Tool Name</th>
<th>True AI</th>
<th>True Human</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEO.ai¹</td>
<td>2/10</td>
<td>9/10</td>
<td></td>
</tr>
<tr>
<td>Copyleaks²</td>
<td>10/10</td>
<td>5/10</td>
<td>-</td>
</tr>
<tr>
<td>GPTZero³</td>
<td>6/10</td>
<td>4/10</td>
<td>-</td>
</tr>
<tr>
<td>GPT-2 Output Detector⁴</td>
<td>5/10</td>
<td>2/10</td>
<td>-</td>
</tr>
<tr>
<td>Corrector⁵,</td>
<td>1/10</td>
<td>9/10</td>
<td>6/20</td>
</tr>
<tr>
<td>Content at Scale⁶</td>
<td>0/10</td>
<td>10/10</td>
<td>-</td>
</tr>
<tr>
<td>Roberta OpenAI Detector - Huggingface⁷</td>
<td>2/10</td>
<td>3/10</td>
<td>9/20</td>
</tr>
<tr>
<td>ChatGPT Detector - Huggingface⁸</td>
<td>0/10</td>
<td>6/10</td>
<td>6/20</td>
</tr>
<tr>
<td>Writer⁹</td>
<td>0/10</td>
<td>9/10</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 1: Sample Table with URL and Classification Results
As you can see, the results are not satisfactory. For Copyleaks, which was able to identify all the AI cases but incorrectly identified 50% of the human code cases, the rest of the software achieved accuracy that does not exceed random classification. In fact, it is important to note that these programs are designed for human text recognition or AI text, and some of them are configured for chatGPT2 output recognition and do not claim to achieve accuracy for chatGPT3.5 or higher AI. In any case, it is possible in relatively simple ways to create software that will detect whether there is suspicion of AI output, through the tests we recommended in Section 4.2.1.

Note that there are also additional tools available for detecting AI-written content, such as Turnitin\textsuperscript{10}, PoemOfQuotes\textsuperscript{11}, Packback\textsuperscript{12}, Codequiry\textsuperscript{13} and AICheatCheck\textsuperscript{14}. We did not examine these tools in the current study because they required registration to use them with multiple examples.

5 Solution Methods Based on Cooperation with AI Developers

To address the concerns of plagiarism underscored in this article, we advocate a joint approach involving educators, assignment creators, and the AI company. One effective method to distinguish between ChatGPT-generated code snippets and those crafted by students is by embedding distinct watermarks into the chatbot’s output. These watermarks would be easily identifiable by teachers and instructors, enabling swift detection of plagiarism. Specifically, we suggest leveraging steganography to flag such instances of academic dishonesty.

A major point of deliberation is what might motivate the developers of the AI tool to cooperate in identifying code fragments issued by their system. Nevertheless, we project that there will be legal boundaries and obligations for AI companies to meet specific criteria. With the global momentum towards crafting regulations to ensure the ethical and safe application of AI, it is reasonable to assume that such rules will emerge in this field. In particular, these regulations could include offering special licenses to lecturers and teachers, allowing them to work with the AI tool and request the application of the steganography-based techniques.

In the subsequent parts of this section, we discuss relevant studies on text-based steganography, proceed with various ways to incorporate these methods into AI-generated code, and present communication protocols to achieve teacher-bot collaboration, resulting in the necessary encoded outputs.

\textsuperscript{10}\url{https://www.turnitin.com/solutions/ai-writing}
\textsuperscript{11}\url{https://www.poemofquotes.com/tools/chatgpt-content-detector.php}
\textsuperscript{12}\url{https://www.packback.co/news/packback-detects-ai-generated-text-chatgpt/}
\textsuperscript{13}\url{https://codequiry.com/}
\textsuperscript{14}\url{https://www.aicheatcheck.com/}
5.1 Text Steganography

We begin with an overview of text steganography, followed by suggestions on how it can be employed to identify AI-generated content, thereby preventing various forms of plagiarism.

Text steganography refers to methods of using text as a means to conceal information [49]. In our context, we need this type of steganography, since code is represented as text, without any way to change letter sizes, fonts, etc. We can only change the characters themselves. Text steganography is notably more challenging than other forms of steganography, primarily due to the minimal redundant data available in a text file as opposed to multimedia files such as images or audio.

Numerous studies, such as [1,7,14,15,22,39,43,48,49], focus on text steganography and encryption methods. Krishnan et al. [25] provide a comprehensive review and classification of text steganography techniques, along with a comparison of existing approaches.

Additionally, some studies suggest using advanced algorithms and machine learning methods for the text steganography task [8,17,45,46,56,57]. In general, the approach to text steganography relies on utilizing the unique properties of text files. These unique properties provide opportunities to hide information. For instance, one could subtly alter the text document’s structure (in our case, the program code and comments) to incorporate concealed information while ensuring the changes are subtle enough not to arouse suspicion or significantly alter the output.

Additional strategies could involve the use of typographical errors, the positioning of spaces, or even the application of invisible characters. Another method could be the crafting of sentences that hold dual meanings. Here, the literal interpretation maintains the appearance of a standard document, while dedicated software will be able to reveal the information encoded in the text.

5.2 Utilizing Steganographic Methods for FingerPrinted Code

Our focus is on developing a strategy that allows ChatGPT to insert unique markers or “digital fingerprints” into its generated code snippets. Steganographic methods that manipulate white space to conceal messages [7,38] appear to be suitable for this application. However, techniques like line-shift and word-shift [22,43], which necessitate altering the spatial layout of text, are not feasible here. We will now examine multiple techniques that ChatGPT can utilize to insert hidden messages into its output code:

- **White Spaces Encoding**: Using a combination of "space" and "tab" characters within blank lines can facilitate binary encoding [38]. This approach allows both data and control messages to be encoded by adjusting the distribution of extra spaces and tabs. For instance, one might designate "space" to represent 0 and "tab" to represent 1. This binary
representation can convey pertinent information about the text, such as its
date or the subject matter. The encoded information can then be strate-
gically placed within the text, such as at the ends of lines or within empty
lines separating code sections. Figure 6 demonstrates how a "GPT" fin-
gerprint can be embedded within the generated code using only invisible
tabs and spaces, which can be viewed by selecting the entire code within
the Colab framework. Figure 5 depicts a code generated by ChatGPT [33]
that aims to produce such a tabs-and-spaces string when given input text
to encrypt. Though students can detect and remove these whites pace
strings, they can be dispersed throughout the code, making them hard to
identify and eliminate.

• Steganography using different types of comments For a language
that includes two or more comment environments, for example, ”//” or
”/* */” in C++, or ”#” and extended string in Python, a coding can be
performed by choosing different types of comments and using them as a
signaling method, in a way determined by the teacher and the AI engine.

• Unique identifiers for variables and functions: Various techniques
can be employed to craft distinctive names for identifiers, such as vari-
ables, functions, classes, etc., enabling the detection of the code’s origin.
Some examples include: using a specific and rare name, for some aux-
iliary variable, utilizing a mix of uppercase and lowercase letters at the
start of an identifier to embed specific information; alternating between
rounded or angular letters in English [15]; using identifiers with a certain
common denominator, for example, with a specific sum of their ASCII
values, or with a specific modulo value, when dividing by a certain k, that
will be chosen in advance. Another possibility is selecting identifier names
such that a hash function applied to the name yields a particular value or
possesses a certain characteristic (like being even, prime, etc.).
It is also viable to use the names of variables that have a certain common
denominator, for example that the sum of the ASCII values of all the
characters in the names of these variables will be 0 modulo k, for a certain
k that the lecturer will choose in advance.

• Formatting in different and special forms In every piece of code gen-
erated, there exists a distinct structural pattern, such as indentation style,
spacing between functions, and the presence or absence of spaces before
and after operators. While Section 4 outlines how these characteristics can
be used to recognize the software’s code output format, it is also feasible
to embed information by manipulating these formatting nuances.

• Insert lines of code and comments with certain characteristics
In addition to the aforementioned methods, one can encrypt a text that
details the exercise’s origin and embed this encrypted version within the
code comments, or implant a specific message inside the original comment,
with certain specific typos [48], etc. While it may look like a linguistic error at first glance, it would indeed represent a concealed message. Broadly speaking, various text steganography techniques – like synonyms, linguistic mistakes, or words that have different spellings in British and American English – can be integrated into the comment sections of the generated code.

- **Clear and Visible markings** Beyond the steganography-based solutions discussed, we can also embed clear and unequivocal markers indicating the code’s origin and purpose. This could take the form of a header comment, along with internal comments, specifying that the subsequent code was generated by the AI tool. Deleting these identifiers would demand considerable editing on the part of the student, potentially deterring plagiarism. An example of detecting work submitted with such a comment is illustrated in Figure 4.

In general, it is recommended that the encryption be in local ranges (and not spread over the entire document), so that if the student changes something in the output program, not immediately all the encryption will be compromised. When the encryptions are local and in different places along the code, it will require thorough and even Sisyphean work on behalf of the student to identify and delete the various watermarks. Apparently, dedicated software can be created to perform this editing, and this is also an issue that the regulator will have to handle in an appropriate manner.
5.3 Enhancing Cooperation Between Teaching Staff and AI Companies

We proceed by describing how steganography methods can be applied and recognized by the education staff. In general, in this study, we suggest three options: the first, as discussed in Section 4, involves signs that are independently produced by an AI tool, and may be recognized by any teacher or student, with or without using AI detection software. The second option, includes steganography applied independently by the AI tool, and reports in a dedicated way to a certain teacher, given cooperation between a teacher and the system. Finally, the third option, enables the teacher to ask the AI tool to produce specific encoded messages when it is given some specific prompts or it is asked to provide a specific generated code.

How can the cooperation between the exercise providers and the AI company be carried out?

First, we assume that lecturers and teachers will be able to obtain a special license, that will enable them to cooperate on this issue with the AI company. Before giving an assignment to the students, the lecturer will inform the AI company that he is giving a certain assignment, and ask it to implant a secret
message inside each code it issues at the request of someone who requests a code for this assignment, within a certain time frame, for example - within the coming week. According to the second approach, the AI company itself will report to the lecturer the message it encrypted in every assignment it received on a certain topic. If there are special licenses, the lecturer can ask the AI company to report to him and even send him any piece of code that he issued at the request of a student on a specific topic, within a certain time frame and even from a certain geographical area. All this within the framework of the law, subsequent to the enactment of the appropriate laws in this field.

According to the third approach, the lecturer will be able to guide the AI company, which technique to use, and what messages to implant.

The following algorithm describes a communication process for using such a method of steganography with cooperation between the teacher and the AI generator, to detect contents it generated:

1. The instructor creates a distinctive assignment with specific instructions that can be easily identified. For instance, the task might require printing the string "BLABLA" when a particular condition is satisfied, or evaluating a unique formula using a specified number.

2. The lecturer reports to the AI company about the exercise details and how it can be recognized. The exercise details may include some unique contents, such as unique request (for example, printing the "BLABLA" string), the exercise date interval, and geographical area.

3. In the second approach, the AI engine determines the coding method to employ. Meanwhile, in the third approach, the lecturer designates the specific steganographic techniques that will be implemented. At this stage, choices are made regarding the steganographic method, the text targeted for encoding, an associated encoding algorithm, a predetermined cipher for encryption, etc.

4. When an appropriate prompt, to perform this unique task, is detected by the AI engine (according to the second or third approach), it will produce the steganographic method determined earlier within the generated code output. For example, when the AI tool is asked to print the string "BLABLA" given the above particular condition.

5. Then, the lecturer will be able to use dedicated software to identify code snippets that students have copied from the AI tool, using the steganographic signs implanted by the AI engine.

It is important to acknowledge that requesting the AI tool to identify users who pose a specific question could raise privacy concerns. Therefore, we propose a solution that does not necessitate such reporting. In summary, this section suggests a collaborative approach between educators and AI developers to tackle plagiarism issues exacerbated by AI tools such as ChatGPT. By leveraging the techniques of text steganography, the proposal is to embed unique "fingerprints"
within the code generated by the AI tool, thus allowing educators to identify AI-generated submissions. This system can be enforced by legal frameworks, ensuring that AI developers are motivated to participate. The ultimate aim is to uphold academic integrity amidst the challenges posed by rapidly advancing technology.

6 Conclusions

In this age of fast-growing AI technology, we are presented with both exciting opportunities and significant concerns. These AI tools offer fantastic educational advantages, such as interactive learning, tailored study modules, and a broad knowledge base, particularly in the realms of computer science and programming assistance. However, they also pose challenges, such as students opting for academic shortcuts or depending too much on automated solutions rather than honing their skills. This paper delves into the influence of the existence of automated code generating tools on higher education programming courses, with emphasis on the tendency of students to over-rely on AI for immediate answers, especially in their early programming journey.

We propose a comprehensive strategy to tackle these issues that encompasses innovative teaching techniques, practical examples, and assignments that promote genuine student engagement. A crucial aspect of our approach involves collaborating with AI developers. We suggest incorporating unique encrypted markers in the AI’s responses, enabling educators to detect if students have relied on AI assistance in their tasks.

Moving forward, our intention is to test the encrypted marker concept in actual classroom settings. We also would like to examine the long-term implications of students using ChatGPT. Gathering feedback from students about these novel teaching approaches would be valuable. Another area of interest is to foster a collaborative environment between academia and tech industries, possibly by establishing standardized guidelines or collaboration platforms. Moreover, as AI continues to evolve, our strategies will need periodic revisions. Given that AI’s influence is not limited to programming, it is essential to assess its role across various academic disciplines, to ensure a holistic educational framework suitable for the AI-driven future.

[Recommended Tools and Strategies to Handle the Academic Integrity Challenge]

As mentioned in this paper, numerous studies have explored the profound effects of LLM tools, especially ChatGPT, on students’ learning methods and processes. These studies offer a range of recommendations to tackle issues of academic integrity and reduce plagiarism in computer science programming tasks. The subsequent list presents the most prevalent recommendations, derived from both academic literature and insights from computer science educators. It is important to emphasize that this study specifically concentrates on computer science education.
The first set of recommendations is centered around educational efforts:

- **Defining Clear rules**: Clear guidelines will need to be established for respective university staff and students as to how ChatGPT can be used in ethically appropriate ways [51].

- **Clear expectations**: Educators need to explicitly convey to the students that ChatGPT is not an appropriate tool for learning programming and should be avoided for such purposes. [42]

- **Explanatory effort**: Students should be motivated to practice programming independently. [42]

- **Educational effort**: Emphasize that those who copy are the real losers. (teachers’ suggestions)

- **Personal Relationships**: Cultivating a close rapport between lecturers and students can enhance student motivation to genuinely engage in the assignments. (teachers’ suggestions)

- **Personalized reflection**: Adopting more personalized reflective tasks contextualized to the subject content. [51]

- **Provide recommended use**: Emphasizing positive and recommended use of ChatGPT instead of focusing on misuse: It should be noted that portraying ChatGPT primarily as a tool for cheating, may lead more students to misuse it after being exposed to such narratives. [51]

The second batch of strategies revolves around revising exercises to meet evolving technological challenges:

- **Challenging assignments**: set up exercises that challenge students’ understanding and application of programming concepts [42]; design projects that are not easily replicable by AI tools. (teachers’ suggestions); revise assessments that were perceived as difficult for ChatGPT to emulate (e.g., podcasts, oral presentations, laboratory activities, group work, handwritten work, participation grades, viva voce examinations, and very specific assignment prompts) [51];

- **Creativity engagement**: Engagement of students in creative and independent projects that allow them to develop their own ideas and solutions [23].

- **Alternative educational frameworks**: Alternative methods, like the flipped classroom, Universal Design for Learning (UDL) and adaptive learning, can decrease students’ motivation to cheat by offering flexible and diverse methods of expression and pacing. [26]

- **Integrate AI Tools**: Allow the integration of AI tools in assignments while posing expansive questions. (teachers’ suggestions)
• **Review LLM solutions** incorporating human expertise and teachers to review, validate and explain the LLM solution. [23]

• **Embrace Project-Based Learning:** Promote collaborative learning through team projects. (teachers’ suggestions)

Additionally, adjusting grading methodologies is recommended to lessen the allure of plagiarism:

• **Interviews:** These can be effective in identifying plagiarism, but they demand considerable time and effort from the instructor. [42]

• **Multi-modal assessments:** Combine written tasks with oral questions, including defenses of the exercises (teachers’ suggestions).

• **Student demonstrations:** Randomly select students to elucidate their solutions in front of the class. (teachers’ suggestions)

• **Homework integration:** Incorporate homework questions into final examinations (teachers’ suggestions).

• **Grading structure:** Restrict the contribution of homework to the final grade to a range of 5%-10%. (teachers’ suggestions)

Lastly, leveraging technical tools can be used to monitor students’ activities and to enhance academic integrity:

• **Student conferencing:** Conferencing with students throughout the semester may ensure their investment. This can be achieved virtually through shared documents, facilitating continuous asynchronous discussions using the ‘comments’ feature. [26]

• **Activity monitoring:** Require students to use online platforms that offer code tracing for the problem solving process. (teachers’ suggestions)

• **Monitor LLM use:** similarly, if the use of LLM is allowed or recommended, the course staff should monitor and evaluate the use of LLMs in the classroom to ensure that they are being used effectively and are not negatively impacting student learning [23].

• **Plagiarism detection tools:** Employ specialized tools, i.e., Moss [47], JPlag [41], and Codequiry [11] to detect potential plagiarism. [42]

• **Automated AI fingerprints:** [31] AI tools might be mandated to embed their fingerprints within the code they generate. This would allow educators to pinpoint the origin of the submitted exercise. To the best of our knowledge, this method has not been previously introduced in the literature for programming submissions.

Every strategy and tool we have discussed plays a vital role in upholding the integrity of computer science assignments in the age of LLMs.
References


