Argus: Academic Integrity in the Era of Generative AI

David Racovan
Purdue University
West Lafayette, Indiana, USA
dracovan@purdue.edu

Ajay Rawat
Purdue University
West Lafayette, Indiana, USA
rawat10@purdue.edu

Christopher K. May
Purdue University
West Lafayette, Indiana, USA
may5@purdue.edu

Jeffrey A. Turkstra
Purdue University
West Lafayette, Indiana, USA
jeff@cs.purdue.edu

Abstract

The rapid proliferation of large language models (LLMs) in the context of education has introduced significant challenges in enforcement of academic integrity, especially in programming courses. We present Argus, an automated detection system for LLM-assisted student work in undergraduate C programming assignments. Argus integrates behavioral and stylistic indicators to create a holistic picture of the student's progress through an assignment and surfaces anomalies that point to potential misuse of LLM assistance. We quantify and analyze data over six years of Spring semester offerings in a large-enrollment CS2 course at Purdue University using Argus, finding that 45% of enrolled students exhibited patterns consistent with LLM-assisted code development in Spring 2026. To contextualize these results, we analyze the relationship between flagged LLM use and student performance on written, in-person proctored examinations, and find a significant negative correlation. We also explore the problem of mitigating false positives, recognizing that erroneous accusations of academic integrity carry significant consequences for students and instructors alike, particularly in the context of large enrollment courses. We argue that any automated detection system must be accompanied by a structured process for human review. We discuss the consequences for future course design, changing academic policy as these tools become more ubiquitous, and the pedagogical implications of LLM-based tools in computer science education.

CCS Concepts

• **Applied computing** → **Education**; • **Social and professional topics** → **Student assessment**; • **Information systems** → **Near-duplicate and plagiarism detection**.

Keywords

Academic integrity, AI-generated code detection, Large language models, Programming education, Software tools

1 Introduction

Large language models (LLMs) like ChatGPT, Claude, and Gemini [4, 6, 10] have recently seen substantial progress and growth in their capabilities (see Figure 1), which has introduced new challenges in computer science education.

Figure 1: Timeline of LLM releases

As these tools have become increasingly accessible, students can now generate solutions to programming assignments and debug code without needing to learn the underlying concepts themselves [20].

This paper presents Argus, a static analysis system for classifying potentially LLM-assisted code generation in an undergraduate introductory C programming course. We report on six Spring semester offerings of the course: 2020, and 2022–2026. All six were taught by the same instructor, and homework assignments were of a similar format. Argus was developed and utilized during the Spring 2026 semester, but our analysis extends to anonymized prior student data.

2 Related Works

Classic approaches to plagiarism detection in student code include MOSS, which analyzes similarities in source code between students [15]. While effective at detecting instances where students reference each other's solutions during the programming process, LLM-generated code may share no surface-level resemblance to other submissions in the class [25, 28]. This renders traditional, similarity-based approaches largely ineffective for detecting academic dishonesty.

Recent work has explored alternative approaches to detect the use of LLMs for code generation. Some have attempted to use

general text-generation models like GPTZero [1], DetectGPT [21], or Sapling [13] to classify code as student-written or LLM-generated. Pan et al. [23] show that existing detectors perform poorly in this task, noting accuracies near 0.5. Similarly, Argotty and Manrique [18] find that existing detectors tend to prioritize either recall or accuracy, meaning that detectors with a high detection rate of LLM-generated code also have a significant false positive rate. This raises concerns when utilized in an educational format, where accusations of academic misconduct can carry large consequences for students. Additionally, they have shown that minor obfuscation of LLM-generated code tends to lead to significant declines in the rate with which detectors classify LLM-generated code correctly.

Others have created models that are specialized for detecting AI-generated code, rather than being generalized on text. A model created by Xu and Sheng [29] uses code's perplexity and burstiness to determine a score, improving performance over a GPTZero baseline, though this research was limited by testing on a single code generation model, OpenAI's `text-davinci-003` [2]. Oedingen et al. [22] show similar successes but with language constraints (Python) and restricted evaluation models. Ramachandra et al. [25] focused on training a model within the context of a programming course where assignments do not change on a term-by-term basis, and achieved an F1 score of 0.98-0.99; this is also limited by the portability of the tool to other institutions, along with being trained on specific LLMs (Gemini and OpenAI models).

A separate method that has been proposed is the detection of style anomalies in code, which are defined by the researchers as "coding styles that depart from a class's style," like untaught language features or nonstandard spacing. Denzler et al. [17] found that a significant portion (44%) of code written by ChatGPT triggered these checks, while 26% of submissions in their most recent course offering had high style anomaly counts. Our study is most similar to this work as it focuses on similar aspects of student code, but we correlate these results to independent measures of student performance to attempt to validate whether detections have pedagogical significance.

3 Methodology & Tool Design

In our C programming course, assignments are completed on a university-managed server to which students have SSH access. Each of the 13 assignments present in each offering requires students to implement C programs that are compiled and tested against a pre-built, hidden test suite. The suite is specific to each assignment; students are told what the test cases evaluate and receive granular feedback, but do not have access to the test source. Invoking `make` against the student's code compiles their code against the test suite, while also automatically committing their code to a per-student Git remote hosted on the same server, using an implementation described in [27], similar to [26]. Students are informed of this behavior both in class and via the `Makefile` output, though it requires no explicit action on their part. Intermediate commits between submissions are retained and form the basis of `Argus`'s behavioral analysis.

`Argus` processes each student's repository by analyzing the full commit history. For each commit, a set of static heuristic indicators is evaluated against the source code. These fall into two distinct categories:

(1) Advanced C topics not taught in the course curriculum, and
(2) stylistic patterns inconsistent with course conventions.

`Argus` primarily targets C idioms and language features that do not provide value in the context of an assignment, like the use of advanced library functions before they are introduced, unnecessary dynamic memory allocation, or unneeded function modifiers. For example, the tool evaluates the presence of `inline` function declarations; while common in production codebases to optimize function call overhead (and therefore frequently produced by LLMs trained on such repositories), these optimizations are unnecessary for the scope of the course programming assignments, and are not taught formally. `Argus` also targets highly irregular comments, like those that mention LLM model names, which appear when students erroneously copy more content than intended.

Each indicator is weighted according to how much it appears in the student's code, along with a constant that indicates the level with which course staff has observed the indicator appearing in LLM-generated code. This was developed by testing frontier coding and general-purpose LLMs against previous assignments, and observing the generated output. These are added to create an aggregate heuristic score (H-score) which indicates the confidence with which a student's working history is determined to be LLM-assisted.

`Argus` also employs a dynamic normalization routine, which calculates the 10th-percentile class occurrence for each indicator k, subtracting this baseline from individual student counts. The penalty weight w_k of an indicator is also scaled by the inverse of its class prevalence (ρ_k). Through this mechanism, if an LLM-favored idiom is used as part of an assignment, its diagnostic weight automatically attenuates to zero.

The temporal structure of the student's work across the assignment is also analyzed. The elapsed time between a student's earliest commit and their final submission is compared against a per-assignment threshold scaled to expected assignment complexity based on the average number of lines of code written by students for that assignment. A student who submits work within a very short window relative to the assignment's scope raises suspicion, especially when their span falls well outside of the class average.

Burst detection identifies individual commits where a large number of novel lines appear in a short interval. Specifically, commits where at least 30 new lines are introduced at a rate of 15 or more lines per minute are classified as such; this was empirically chosen based on observed student and staff behavior, and to minimize false positives. These bursts are consistent with pasting externally generated code rather than typing incrementally. Both span and burst detections are added to the H-score as components.

The resulting scores are surfaced through a web interface used by course staff. Students are ranked by H-score, and staff can inspect each flagged indicator in the context of the surrounding source code. All flagged cases undergo manual review before any academic integrity action is taken, by policy. We implement features to ensure that course staff can perform these checks quickly, especially in offerings with large enrollment. These include shortcuts to move through large amounts of code quickly, temporal charts to show code in the context of the amount of time it took to write, and

charts that show how the H-score changed during the lifetime of the assignment. In many cases, a high H-score during the course of the assignment followed by a sharp decrease immediately prior to the deadline can serve as an additional factor for course staff to consider, as it usually indicates a student attempting to remove parts of their code that they may consider suspicious.

At institutions where similar infrastructure is not present, the tool may still function on the basis of static heuristic indicators. If development environments that provide insight into the student's working history with their code are available, this will provide the most accurate data; we have found that intermediate steps often reveal more indicators than a student's final submission.

3.1 Limitations and False Positives

Any automated system operating within the context of academic integrity must be designed with the risk of false positives as a central concern. Accusations of academic dishonesty invoke significant anxiety for students, and false convictions can have detrimental future academic and career effects. In the context of Argus, it is designed such that a high H-score is never treated as evidence of misconduct. Its primary purpose is to expedite the academic integrity triage process, especially in courses with large enrollment where individual, manual review of every student's code and assignment history is infeasible.

Several benign behaviors can produce elevated scores; a student who has prior programming experience may use language patterns not taught or taught late in the course, especially if they have previously been enrolled in the course. A student who drafts code in an external editor may also produce burst indicators if they paste sections of their code between rapid commits. To mitigate this, staff take each finding in the context of surrounding code, and observe the student's full working history.

3.2 Analysis

To validate Argus as a reliable diagnostic instrument that other practitioners can confidently adopt, we compare its generated H-scores with objective measures of student performance and learning. Examinations are closed-note, written, and proctored, and the use of any external assistance is prohibited by course policy. This helps to serve as an independent baseline of student performance, where students are assessed on their ability to complete similar programs to those completed in prior assignments. Three in-person, closed-book exams are administered each semester. The two midterm exams involve writing substantial amounts of code drawing from concepts covered on the homework assignments while final exams are typically composed of shorter, simpler questions. H-scores are averaged across all 13 homework assignments per student, and exam scores are expressed as a percentage of total points.

For the purposes of this research, Argus was executed against data from prior offerings of the course in the same way in which it was used in the Spring 2026 offering. All offerings presented in this research used similar syllabi under the same professor, with similarly-written examinations. This produced a dataset of H-scores for each student in each semester. A manual review process did not take place, as the data was not used to make individual decisions on academic integrity.

4 Results

4.1 Elevated H-scores

Across all six offerings of the course, Argus processed 3,893 distinct student-semester pairs, with a total of 42,695 individual homework submissions. Score distributions were heavily right-skewed in all semesters, as shown in Figure 2.

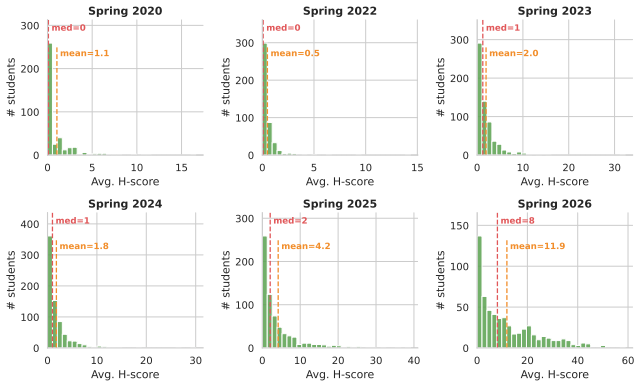

Figure 2: Distribution of Avg. H-scores, Per Semester

The near-zero mean scores in Spring 2020 (mean = 1.1) and Spring 2022 (mean = 0.5) validate Argus's calibration; frontier LLM-based coding assistants were not widely available to the public during these offerings, so Argus accordingly found little to flag. The modest increase in Spring 2023 (mean = 2.0) and Spring 2024 (mean = 1.8) coincides with the initial public release and growing adoption of general-purpose LLMs, though aggregate scores remained low. This suggests that LLM-assisted development had not yet become widespread among students in the course.

A more pronounced shift appears in Spring 2025 (mean = 4.2), and Spring 2026 (mean = 11.9). This suggests a large change in student behavior towards the use of LLM-based assistance, rather than a change driven by a small number of outliers. Maximum observed H-scores also increased substantially, reaching 81 in Spring 2026 compared to a maximum of 40–45 in earlier offerings. These trends align with the rapid spread of coding assistants with capabilities equal to or beyond the complexity of course assignments, as charted in Figure 1. This is consistent with the hypothesis that Argus's results in earlier years demonstrate a low false-positive rate when evaluating historical, pre-LLM submissions as human-authored.

4.2 H-scores and Exam Performance

To assess whether the identification of LLM use by Argus is associated with reduced mastery of course material, this analysis compared average homework H-scores against performance on the examinations mentioned in Subsection 3.2 on a per-student basis. In Spring 2020, a project was used in lieu of exams beyond Midterm 1 due to COVID-19, so the analysis for that offering is limited. All other semesters were analyzed on scores from two midterms and a final exam.

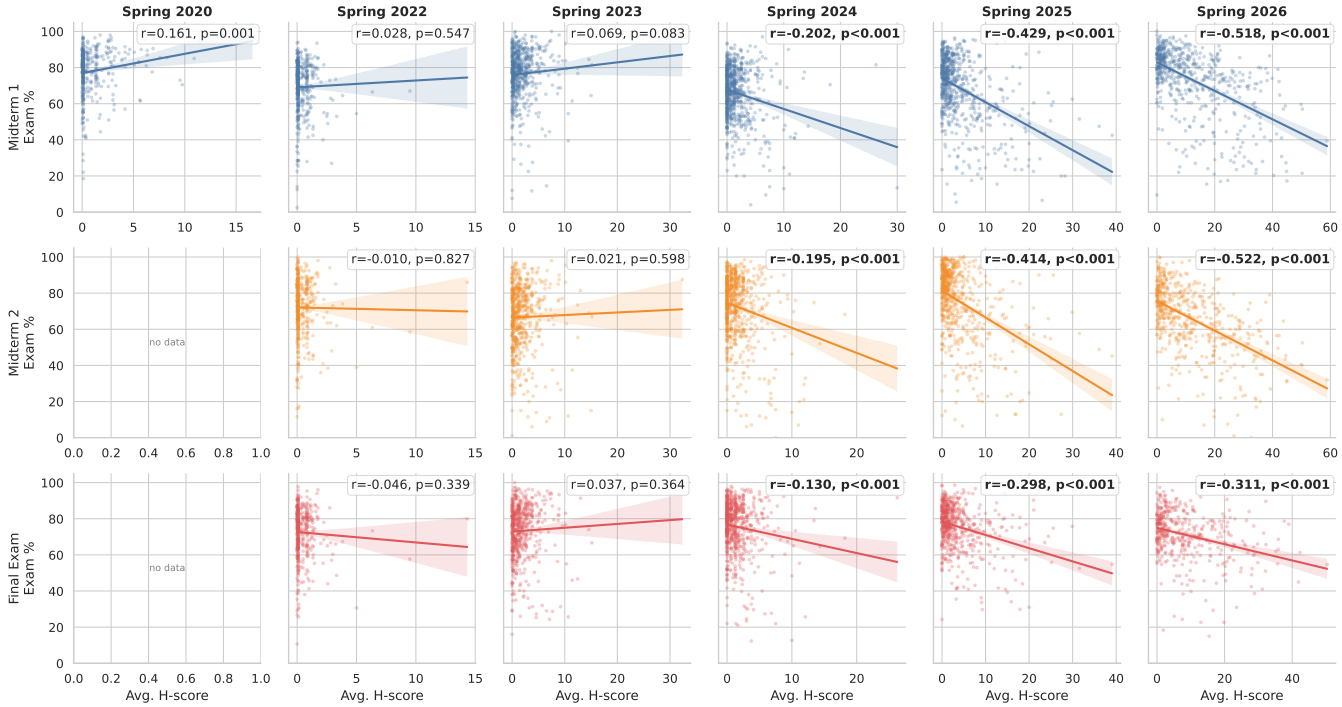

Figure 3: H-score vs. Exam Score, Per Semester; bands visualize a 95% confidence interval of the regression fit.

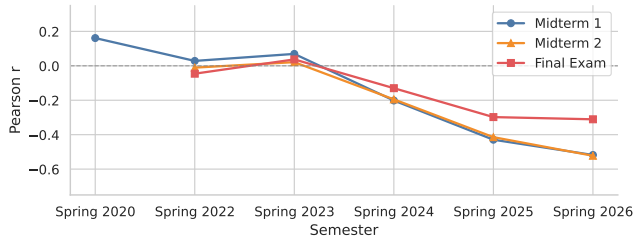

Figure 4: Correlation in H-score vs. Exam Score Over Time

When analyzed on a per-semester basis, the relationship between H-score and exam performance changes markedly over time, as shown in Figures 3 and 4. In Spring 2020, the correlation was weakly *positive* ($r = 0.161$), and in Spring 2022 and 2023 it was near-zero. This is consistent with the low mean H-scores in those years, since the heuristic has little variance to correlate against. Beginning in Spring 2024, a negative relationship begins to emerge ($r = -0.244$), and strengthens in Spring 2025 ($r = -0.450$) and Spring 2026 ($r = -0.537$). In Spring 2026, the per-exam breakdown has correlations of $r = -0.518$ for Midterm 1 and $r = -0.522$ for Midterm 2, with a somewhat weaker relationship on the Final Exam ($r = -0.311$). This is consistent with final exams requiring less coding overall.

The relationship between H-score and exam performance is also evident when students are grouped into scoring regions (Figure 5). Overall, students with an average H-score in the 0–4 range ($n = 2780$) achieved a mean exam percentage of 73.6%, and performance declined steadily as H-scores increased: 68.7% for scores in the 5–9 range, 65.8% for 10–14, 60.0% for 15–19, and 55.9% for 25–29, falling to 42.5% for scores above 50. The decline is most pronounced in later semesters, where there is sufficient score variance to populate the higher groups. Similar findings are shown by Pierce & Zilles [24] and Chen et al. [16], who find that plagiarism is negatively correlated with final grades in their courses.

4.3 Withdrawal and Course Completion

We examined whether students who withdrew from the course after Midterm 1 (students who attended Midterm 1 but neither Midterm 2 nor the Final Exam) exhibited higher H-scores than students who completed the course. This criterion was chosen to avoid drawing conclusions about students who withdrew early in the semester, before a significant portion of the course was completed. Spring 2020 is excluded, due to the availability of only a single exam.

Across the five remaining semesters, 96 students (approximately 3.1% of the population) met the withdrawal criteria. Their mean H-score was 9.9, compared to 3.9 among the 2,974 students who completed the course. When evaluated per-semester, this disparity widened notably over time; these effects were the smallest in Spring 2022 and Spring 2023, but in Spring 2026, withdrawn students averaged a H-score of 19.5 against 11.4 for continuing students, detailed in Figure 6. A two-sample Kolmogorov-Smirnov test indicates that the two groups are drawn from significantly different distributions ($D = 0.362$, $p < 0.001$). The association between elevated H-scores and course withdrawals is consistent with a scenario in which students who rely heavily on external assistance accumulate less understanding of the material over the course of the semester and subsequently struggle to apply it in a proctored context.

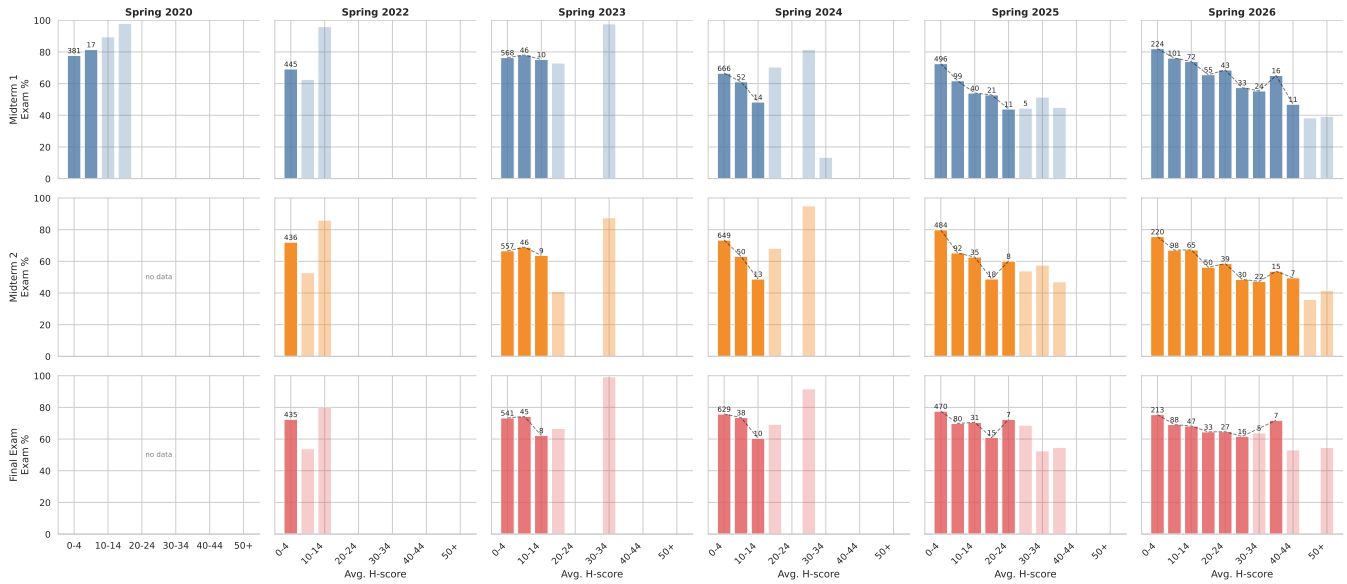

Figure 5: H-score vs. Exam Score; lighter boxes indicate groups with $n \leq 5$.

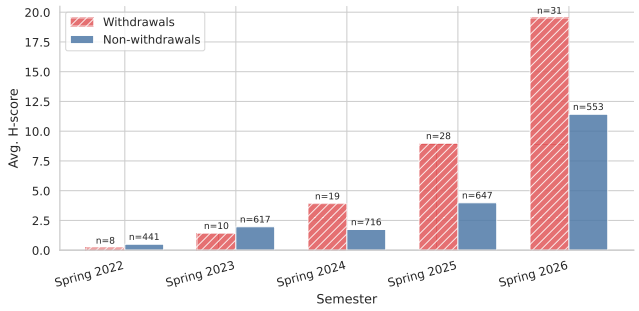

Figure 6: Mean H-score vs. Withdrawals by Semester

4.4 Case Study: Spring 2026

The results presented rely on Argus's H-scores as the sole proxy for LLM-assisted development. While this enables longitudinal comparison across all six offerings, human review remains necessary to draw conclusions for an individual student. In Spring 2026, we implemented a structured manual review process, in which course staff examined students with high H-scores as identified by Argus. Students were ranked on H-score both cumulatively and per-assignment and reviewed top-to-bottom. Course staff made a binary determination of whether the evidence was sufficient to pursue the case, per course policy. Three separate members of course staff evaluated students independently, then met to reach consensus. Extreme benefit of the doubt was given throughout to avoid false positives; students were only determined to have used LLM-assisted tools if the evidence was remarkably strong.

To characterize the practical cost of review, we tracked the time it required. Reviewers examined the top 300 students by H-score, plus any student in the top 150 for an individual assignment. Three course staff spent approximately 8 hours each, with a median review time of roughly 1.5 minutes per student: clearly suspicious commits were often identified in under a minute, while borderline cases required 5 minutes or more of examining the full working history. The temporal charts and per-indicator navigation were essential to this throughput; in informal comparison, reviewing a raw repository without the tool took upwards of 10 minutes per student.

Among the top 150 students by H-score, we noted 4 instances where there was insufficient evidence to escalate their cases. In these, the indicators had a benign explanation in context: bursts traced to code and comments copied from the assignment description itself, or untaught language features were used coherently throughout, consistent with prior experience.

Of the 584 students who completed the first midterm exam in the Spring 2026 offering, 267 (45.7%) were flagged following manual review, and 317 were not. The prevalence of flagged cases in this offering is consistent with the substantially increased H-scores observed in Spring 2026 relative to other semesters. Flagged students performed notably worse on all three examinations than their non-flagged peers (Figure 7). Averaged across all three exams, flagged students achieved a mean of 61.9% (median 65.9%), compared to 73.5% (median 75.7%) for non-flagged students. The gap was largest on Midterm 2, where flagged students averaged 59.0% versus 72.2% for non-flagged students.

Among flagged students, performance declined as the number of flagged assignments increased, also suggesting a relationship between amount of LLM-based assistance and exam outcomes. Students with a single flagged assignment averaged 69.4%, close to the non-flagged mean of 73.5%. Performance then declined progressively; students with 2 flagged assignments averaged 65.0%, students with 5 averaged 59.0%, students with 6 averaged 53.5%, and those with 8 or more averaged 54.4%.

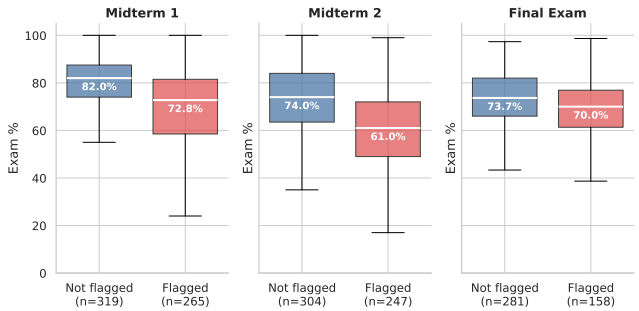

Figure 7: Exam Score, by Flagged vs. Not Flagged (Spring 2026)

We also utilized MOSS in parallel with `Argus` for the Spring 2026 offering, but detected cases were significantly decreased compared to prior offerings, with an average of about 2-3 suspicious student-to-student plagiarism cases per assignment. Since LLM-assisted submissions often do not bear similarities that tools like MOSS depend on, dishonest submissions become progressively less visible to similarity-based tools.

The large performance gap between flagged and non-flagged students on in-person, proctored examinations supports the interpretation that flagged students had not developed the independent mastery of course material that the homework assignments were designed to build. The gap widening as the number of assignments flagged increases further suggests that the amount of LLM reliance, not merely its presence, is predictive of reduced outcomes.

5 Discussion & Reflections

5.1 Automated Detection and Human Review

The scale of apparent LLM adoption in Spring 2026 surfaces a tension any institution deploying detection tooling will face: the volume of cases can quickly exceed what course staff can review, while the consequences of acting on an automated score alone are too serious to accept. Ideally, humans would review all code from every student, but this is infeasible at scale; Spring 2026 alone produced 381,748 individual commits across 7,769 submissions. `Argus` was designed to help resolve this tension rather than decide student outcomes. By ranking students on H-score and surfacing the specific indicators behind it, staff can move through a large case queue efficiently. We accept a non-zero false negative rate in doing so, but by surfacing the students harming their learning the most, action can be taken early to correct these behaviors.

The failure mode of any similar detection tool used in the context of academic integrity is asymmetric; a missed case of LLM use affects only that student's own learning, but a false accusation can have lasting consequences for a student's academic record and wellbeing. We argue that the appropriate standard for any detection system is not whether it can identify misconduct autonomously, but whether it can make the human review process fast and structured enough to be feasible for large-enrollment courses.

5.2 The Scale of LLM Adoption

The most striking finding of this research is the speed with which the presented correlation increased over time. In Spring 2020 and Spring 2022, `Argus` flagged very little, and the relationship between its scores and exam outcomes were near-zero. Within four years, 45% of the students in the same course, taught by the same instructor, with structurally similar assignments, exhibited patterns consistent with LLM-assisted development. This suggests that the question facing CS2 courses is no longer whether students will have access to and use capable coding assistants, but how course design and policy should respond to the assumption that they do.

5.3 Changing Academic Policy

The findings also raise questions about how academic integrity policy should evolve as LLM-based tools become more capable and more ubiquitous. The definition of unauthorized assistance continues to become more nebulous; there is a pedagogical difference between a student using an LLM to debug a single function versus generating an entire submission, but this distinction may not be reflected in common institutional policies. However, any new guidelines must balance this nuance with the practical requirement that policies remain straightforward to understand and enforce. As detection tools improve, institutions will need policies that are both adaptable to new technologies and highly unambiguous in practice.

6 Conclusion

We have presented `Argus`, a static analysis system for detecting patterns consistent with LLM-assisted code development in an undergraduate C programming course, applied across six semester offerings from Spring 2020 through Spring 2026. H-scores were near-zero in the pre-LLM era and rose substantially as LLM-based coding assistants became publicly available, validating that the tool reflects real changes in student behavior. Average H-score was negatively correlated with performance on in-person, closed-note examinations, with correlations strengthening in more recent offerings. The pedagogical implication is that students who rely heavily on LLM assistance do not develop the independent programming ability the course is designed to produce.

We emphasize that `Argus` is a triage tool, not designed to render automated determinations of academic dishonesty; institutions deploying automated detection should invest equally in the review process around it. Given the pace of LLM capability growth, the patterns observed in Spring 2026 will likely grow more prevalent. We hope `Argus` and the methodologies described here offer a useful foundation for institutions responding to this shift, and that our longitudinal analysis provides a strong baseline against which future detection methods can be measured.

References

[1] [n. d.]. https://gptzero.me

[2] [n. d.]. https://developers.openai.com/api/docs/guides/completions

[3] 2020. OpenAI API. https://openai.com/index/openai-api/

[4] 2022. Introducing ChatGPT. https://openai.com/index/chatgpt/

[5] 2023. GPT-4. https://openai.com/index/gpt-4-research/

[6] 2023. Introducing Gemini: our largest and most capable AI model. https://blog.google/innovation-and-ai/technology/ai/google-gemini-ai/

[7] 2024. Hello GPT-4o. https://openai.com/index/hello-gpt-4o/

[8] 2024. Introducing Claude 3.5 Sonnet. https://www.anthropic.com/news/claude-3-5-sonnet

[9] 2024. Introducing OpenAI o1. https://openai.com/o1/

[10] 2024. Introducing the next generation of Claude. https://www.anthropic.com/news/claude-3-family

[11] 2024. Our next-generation model: Gemini 1.5. https://blog.google/innovation-and-ai/products/google-gemini-next-generation-model-february-2024/

[12] 2025. Claude 3.7 Sonnet and Claude Code. https://www.anthropic.com/news/claude-3-7-sonnet

[13] 2026. https://sapling.ai/ai-content-detector

[14] 2026. Claude Opus 4.6. https://www.anthropic.com/news/claude-opus-4-6

[15] K.W. Bowyer and L.O. Hall. 1999. Experience using "MOSS" to detect cheating on programming assignments. In *FIE'99 Frontiers in Education. 29th Annual Frontiers in Education Conference. Designing the Future of Science and Engineering Education. Conference Proceedings (IEEE Cat. No.99CH37011*, Vol. 3. 13B3/18–13B3/22 vol.3. doi:10.1109/FIE.1999.840376 ISSN: 0190-5848.

[16] Binglin Chen, Colleen M. Lewis, Matthew West, and Craig Zilles. 2024. Plagiarism in the Age of Generative AI: Cheating Method Change and Learning Loss in an Intro to CS Course. In *Proceedings of the Eleventh ACM Conference on Learning @ Scale (L@S '24)*. Association for Computing Machinery, New York, NY, USA, 75–85. doi:10.1145/3657604.3662046

[17] Benjamin Denzler, Frank Vahid, and Ashley Pang. 2024. Style Anomalies Can Suggest Cheating in CS1 Programs. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 2 (SIGCSE 2024)*. Association for Computing Machinery, New York, NY, USA, 1624–1625. doi:10.1145/3626253.3635519

[18] Juan Esteban Cuellar Argotty and Ruben Manrique. 2026. AI-Generated Code Detection: An Examination of Current Tools in Education. In *Generative Systems and Intelligent Tutoring Systems*, Sabine Graf and Angelos Markos (Eds.). Springer Nature Switzerland, Cham, 192–201. doi:10.1007/978-3-031-98281-1_15

[19] Nat Friedman. 2021. Introducing GitHub Copilot: your AI pair programmer. https://github.blog/news-insights/product-news/introducing-github-copilot-ai-pair-programmer/

[20] Juyong Jiang, Fan Wang, Jiasi Shen, Sungju Kim, and Sunghun Kim. 2024. A Survey on Large Language Models for Code Generation. doi:10.1145/3747588 DOI: 10.1145/3747588.

[21] Eric Mitchell, Yoonho Lee, Alexander Khazatsky, Christopher D. Manning, and Chelsea Finn. 2023. DetectGPT: Zero-Shot Machine-Generated Text Detection using Probability Curvature. In *Proceedings of the 40th International Conference on Machine Learning*. PMLR, 24950–24962. https://proceedings.mlr.press/v202/mitchell23a.html

[22] Marc Oedingen, Raphael C. Engelhardt, Robin Denz, Maximilian Hammer, and Wolfgang Konen. 2024. ChatGPT Code Detection: Techniques for Uncovering the Source of Code. *AI* 5, 3 (Sept. 2024), 1066–1094. doi:10.3390/ai5030053

[23] Wei Hung Pan, Ming Jie Chok, Jonathan Leong Shan Wong, Yung Xin Shin, Yeong Shian Poon, Zhou Yang, Chun Yong Chong, David Lo, and Mei Kuan Lim. 2024. Assessing AI Detectors in Identifying AI-Generated Code: Implications for Education. In *Proceedings of the 46th International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET '24)*. Association for Computing Machinery, New York, NY, USA, 1–11. doi:10.1145/3639474.3640068

[24] Jonathan Pierce and Craig Zilles. 2017. Investigating Student Plagiarism Patterns and Correlations to Grades. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education (SIGCSE '17)*. Association for Computing Machinery, New York, NY, USA, 471–476. doi:10.1145/3017680.3017797

[25] Aryan Ramachandra, Suhani Chaudhary, Justin Tran, Riti Desai, Ashley Pang, and Mariam Salloum. 2026. Detecting AI-Generated Code in Introductory Programming Courses. In *Proceedings of the 57th ACM Technical Symposium on Computer Science Education V.1 (SIGCSE TS 2026, Vol. 1)*. Association for Computing Machinery, New York, NY, USA, 894–900. doi:10.1145/3770762.3772522

[26] Karen L. Reid and Gregory V. Wilson. 2005. Learning by doing: introducing version control as a way to manage student assignments. In *Proceedings of the 36th SIGCSE technical symposium on Computer science education (SIGCSE '05)*. Association for Computing Machinery, New York, NY, USA, 272–276. doi:10.1145/1047344.1047441

[27] Gustavo Rodriguez-Rivera, Jeffrey Turkstra, Jordan Buckmaster, Killian LeClainche, Shawn Montgomery, William Reed, Ryan Sullivan, and Jarett Lee. 2022. Tracking Large Class Projects in Real-Time Using Fine-Grained Source Control. In *Proceedings of the 53rd ACM Technical Symposium on Computer Science Education*. ACM, Providence RI USA, 565–570. doi:10.1145/3478431.3499389

[28] Zachary Taylor, Cy Blair, Ethan Glenn, and Thomas Ryan Devine. 2023. Plagiarism in Entry-Level Computer Science Courses Using ChatGPT. In *2023 Congress in Computer Science, Computer Engineering, & Applied Computing (CSCE)*. 1135–1139. doi:10.1109/CSCE60160.2023.00189

[29] Zhenyu Xu and Victor S. Sheng. 2024. Detecting AI-Generated Code Assignments Using Perplexity of Large Language Models. *Proceedings of the AAAI Conference on Artificial Intelligence* 38, 21 (March 2024), 23155–23162. doi:10.1609/aaai.v38i21.30361