WIP: ARTful Insights From A Pilot Study on GPT-based Automatic Code Reviews in Undergraduate Computer Science Programs

Aaron S. Crandall
Department of Computer Science
Gonzaga University
crandall@gonzaga.edu

Bryan J. Fischer
Department of Computer Science
Gonzaga University
fischerb@gonzaga.edu

Johannah L. Crandall
Marsal Family School of Education
University of Michigan
jlcrand@umich.edu

***Abstract*—This work in progress research paper describes a pilot study using a Large Language Model (LLM) Generative Pre-Trained Transformer-based (GPT) system that generates industry-style code reviews for student feedback on software development projects in Computer Science 2nd, 3rd, and 4th+ semester classes (CS2, CS3, CS4+) at an ABET accredited baccalaureate institution. Code reviews are a valuable, but work-intensive, component of the software engineering process and provide important training to undergraduate students in the form of mentor-peer knowledge transfer. Participants in this study engaged in iterative experiential learning using the Automatic Review Tool (ART), an artificial intelligence tool to support software engineering as an Automatic Static Analysis Tool in the Continuous Integration pipeline alongside software testing harnesses and code style checkers. This pilot study was based on earlier results from a full computer science second semester (CS2) class (n = 74) to develop an ART-generated code review intervention pilot study with a small group of students in CS2 / 3 and CS4. The project underway uses an experiential learning and iterative feedback process to answer research questions including "Does ART provide accurate and actionable code reviews for students" and "Which levels of students are best prepared to receive and use ART-based code reviews?" During this pilot study, the project used a mixed methods research approach with a series of surveys, code review interventions, and numerical analysis of the code reviews' accuracy. Results showed a reasonable degree of code review accuracy by ART and the students learned code review skills from interaction with the ART-based reviews they received. Ongoing work includes increasing the scale of data collection, using this work to refine and focus the ART-based reviews onto the categories of feedback that students find the most valuable, and building out a more modular tool for wider release in the academic community.**

***Index Terms*—Adaptive computer learning, Computer science, Qualitative, Mixed methods research, Code Reviews, Software Engineering Education**

I. Introduction

The introduction of Chat Generative Pre-Trained Transformer (GPT) systems has raised serious questions about the role of software developers and code creation processes, both in industry and in the classroom. The greatest focus on GPT's role in software development has been on its use for code synthesis. The field of computer science education needs to explore ways to integrate GPT and other Artificial Intelligence-based tools into pedagogical practices [1], just as the software development industry is seeking strategies to leverage GPT-style tools into industry practice.

Modern Code Review (MCR) is an industry practice that lends itself well to automation with GPT. Participating in a code review process is strongly aligned with professional success, reputation, and lifelong learning required to stay up-to-date in the software world [2]. Ideally, it is important for students to be exposed to the MCR process early and frequently, which has been explored in educational contexts often through peer review strategies [3] and this work contributes to the ongoing effort to study the possibilities for didactic adaptations of this professional practice. However, since code review is typically done by other programmers, it can be time-consuming to implement at scale for a class.

This builds on the work of [4] which introduced an Automatic Review Tool (ART) that provides GPT-derived code review output to undergraduate programming learners. The ART code review process uses iterative feedback aligning with established theories of experiential learning [5] [6] [7] in which learners engage in a cycle of experimentation, conceptualization, and reflection well-scaffolded by learner-specific feedback. The extension of the previous work was to perform a pilot exploration with students of CS2 / 3 and CS4+. The goal was to evaluate both its code review suggestion accuracy and to seek evidence of how the two different groups would view and make use of the ART-style output.

The primary thesis question being explored is: "Is there a notable difference in the benefits seen by CS2/3 vs. CS4+ students from being exposed to ART-based code reviews?" Additionally, a pilot evaluation of the accuracy and efficacy of ART-style reviews was collected to determine whether it should continue to be evaluated as a successful tool to incorporate into computer science curricula.

II. Related Works

While the definition of code quality and Automatic Static Analysis Tools are rapidly evolving [8], the field is quickly adapting to new innovations in AI in software engineering

applications. Examples of this include Lewowski et al. and detecting code smells [9] with AI tools and Yaozong et al's work on improving code conformance over classic Automatic Static Analysis Tools (ASAT) like Cppcheck [10] by incorporating new AI-based lexical analysis. The introduction of large language models and GPT models will only hasten this revolution in automatic analysis.

Teaching code review skills in the classroom, or to junior developers in a mentor-mentee relationship, is a well studied area. To distribute the work of performing code reviews, they are often done through peer review approaches [3], [11], [12]. While students value instructor feedback in their projects [13], the cost per student means that there are opportunities to help in automating both the grading and feedback generation in the classroom [14], [15]. The state of the art in these tools have focused on some components of the code review process with notable success, but have not yet established a comprehensive, fast, in-context, and well evaluated system to provide automated code reviews.

Effective code reviews address at least style, structure, architecture, execution errors, and testing issues [16]. Adapting the MCR professional practices to pedagogically effective approaches requires significant adaptation [17].

Paper Contributions: This work in progress paper presents evidence that the ART GPT-based approach from [4] to generating code reviews for students shows promise in CS2/3/4+ courses. ART demonstrates accurate code feedback suggestions and students perceive value from the interactions with ART.

III. Methods

This pre-post intervention study was designed to identify CS students' level of understanding and level of practical experience with code review before and after exposure to an ART-style code review. Thus, the design considers both dimensions of the Bloom's Extended Computer Science Matrix Taxonomy [18] when seeking to characterize change or growth in code review skill. The study was performed with a small group ($n = 5$) of second year (CS 2/3) and third year (CS4+) students. These volunteers were asked to bring a copy of an assignment done in a prior course centered around Java-based object-oriented software development to be reviewed by peers and with ART. The activities outlined were conducted on a weekend in one of their campus classroom spaces, and participants were not informed in advance the AI-generated nature of the code review intervention.

The study had seven phases which are summarized in Table I. These included gathering student background information, asking them to perform code reviews, reading an ART-derived review on their own code, and then performing another code review with writing reflections about their learning from the intervention.

IV. Results

Five participating students performed a total of ten peer code reviews, received five ART-generated reviews, and filled out 68 questions each about their technical background, code reviewing knowledge, and perceptions of a professional code review. These results were aggregated and reviewed independently by two of the authors for themes and notable differences between the CS2/3 and CS4+ groups.

A. Phase 1: Pre-ART Review Survey Results

Student themes and results of the survey:

1) When asked "how important do you feel code reviews are for group projects" the group averaged a 6.0 out of 7 on a Likert scale where 1 is not at all important.
2) Students said that in their experience, most programming assignments had few, if any, notes giving feedback from Teaching Assistants (TAs) or faculty.
3) Students also noted that TA feedback mostly centers around what they did wrong, whether tests pass, and general correctness, not about style, structure, or bugs.

B. Phase 2: First Round Peer Review (Review A) Notes

Results addressed variable declarations, class and variable naming, using Java files and classes properly, initialization and memory management, class responsibilities, and other areas. The most notable difference was that the CS2/3 students wrote longer, more verbose paragraph style reviews while the CS4+ students wrote more direct, bullet point style reviews.

C. Phase 3: Post-First Round Peer Review Survey Results

From the post-Review A survey, the participants' results noted a few key things:

1) For the CS2/3 students this was their first code review, while all CS4+ students had some experience.
2) When questioned about what the participants focused on while writing their review they listed:
   a) Features and overall structure rather than details.
   b) One participant did not know what to focus on.
   c) Several talked about focusing on classes, variable visibility, and structure.
   d) Several included documentation and code clarity.
3) Overall, they gave a 3.6 out of 5 when saying the other student wrote better code than the reviewer did.

D. Phase 4: ART Review Student Markup Results

Student markups on the ART-generated reviews were counted and reviewed by the authors for accuracy and conformance to the instructions provided to the students in categorizing the review's statements. The participants received a total of 58 pages of reviews to read through and mark up during a 45 minute reading session. In total, the students marked up 173 total suggestions from ART to consider and categorize. The breakdown of the results into the four categories are shown in Table II. Of these, almost 50% were marked as something they understood and could use to improve their code, with another 35% as understandable, but not something they would use when refactoring their work.

TABLE I
ART CODE REVIEW PILOT TEST PHASES.

Phase	Name	Description
1	Code review experience and background survey	This survey included questions about their programming skills, prior code review experience, and any professional experience they might have.
2	Peer reviewing another student's code (Review A)	In this phase, each student was randomly assigned another participant's code to read and write a review on. They had about 45 minutes to perform this task. The review was written on a GitHub pull request to simulate a situation where they were assigned the task in a project.
3	Questionnaire about their Review A and what they focused on	This questionnaire focused on the student reflecting about what they chose to address while performing Review A. They were asked if it was the first peer review they had done, how confident they were in their work, and an open ended question about what they chose to write about in their review.
4	Received an ART-generated review on their code to read, marked up the code review suggestions into various categories, and reflected on the feedback	Every student's code was used to make an ART-Generated code review. The students were given the review for their own code on paper and asked to read through it. While reading they were to mark statements that they felt were specific suggestions that fell into one of four categories: "Understood, Can Use" Suggestions that they understood and could use to improve their code. "Understood, Won't Use" Suggestions that they understood, but would choose not to use to improve their code. "Do Not Understand" Suggestions that they felt were specific, but did not understand what was either being stated or meant by ART's suggestion. "Incorrect/Wrong" Suggestions that they felt were just incorrect or wrong. During this stage, the ART-Generated code review was referred to as a "professional code review" done by a "professional code reviewer" as to obfuscate the origin of the review.
5	Questionnaire about the ART-generated review on their code, including reflections	After reading the ART-Generated code review on their work, participants were given a questionnaire to gauge their thoughts and perspectives about the code review. The questions were built on the earlier work done in [4] to help align these results with that earlier work. The questions included topics of the review's tone and whether it was written in a friendly manner. The questions also asked the students about how well ART addressed code style, structure, and design, the review's overall length, and any comments for their "professional reviewer."
6	Performed a second peer review on another student's code (Review B)	After receiving the intervention of having their own code reviewed by ART, the students were asked to perform a second peer code review, Review B. They were randomly assigned a new student's project to review and wrote their review in a GitHub pull request, just as they did in Review A.
7	A final set of questions about the skills and ideas they learned for reviewing code	The final phase questionnaire asked the students to answer questions about what they chose to focus on in the second review, what they thought about in Review B that they did not in Review A, if there were any categories of defects they started looking for, and whether they felt this session would improve their code reviewing skills and how much. This helped compare and contrast their approaches between Review A and Review B.

TABLE II
STUDENT MARKUP COUNTS FOR ART-GENERATED REVIEWS

Understood Can Use	Understood Won't Use	Do Not Understand	Incorrect/Wrong
85	61	25	2

TABLE III
CAN USE VS. WON'T USE BY STUDENT GROUPS

	Understood Can Use	Understood Won't Use	Do Not Understand
CS2/3 Participants	70.3%	19.8%	9.1%
CS4+ Participants	26.3%	52.6%	19.7%

Where the split between the two student groups came was mostly in the rate of the Can Use vs. Won't Use categories. This result is shown in Table III, where there is a clear inversion of these two categories between the upper and lower division of students. The reasons for this difference can only be hypothesized about so far, and should be further studied in line with the main thesis question for this pilot study.

There is also a notable difference between the groups in the Do Not Understand category as well. The CS4+ group marked 19.7% of ART review suggestions as Do Not Understand while the CS2/3 participants only marked 9.1%. Upon further inspection, this was attributed to the CS4+ students using more advanced programming strategies which caused ART to respond with more advanced terminology and design pattern discussion that are encountered in later courses taken by the students. The ART suggestions marked as Did Not Understand were all reviewed by the authors and were found to be accurate and correct ones, just beyond the current knowledge of the students. The authors also feel that this is evidence that ART style code reviews will help different student levels in different ways and necessitates further study.

E. Phase 5: Post-ART Review Questionnaire Results

The students' results for both lower and upper groups mostly aligned with the results found in the earlier work done on a purely CS2 student cohort in [4]. The students found the reviewer to have supportive tone, topical knowledge, and the reviewer reviewed the code and not the student. Among the questions that address specific categories of code reviewing, there were several where the CS2/3 and CS4+ groups notably differed, as shown in Table IV. These values are the averages of the two groups and are the top three areas that the groups differed the most. Overall, the CS4+ students were more

critical of the ART-Generated review. Based on the qualitative feedback, the students felt the ART-Generated reviews were helpful, accurate, and that they learned approaches to successful reviewing.

TABLE IV
RATE OF REVIEW DEFECTS BY CS2/3 VS CS4+ GROUPS

Please rate the review your code received in terms of:	**CS2/3**	**CS4+**
Visual representation defects	4.0	2.7
Documentation defects	5.0	4.0
Structural defects	4.5	3.6

F. Phase 6: Peer Code Review B Notes

The Review B reviews were notably longer and more specific than the Round A reviews. The students uniformly used a bullet point style with shorter, more specific statements. They now covered the same range of topics, as well as included additional areas of variable member access and documentation that were not as prevalent in the Review A set. The notable improvements to the review content came from the CS2/3 students whose style became more direct, vocal, and performed a deeper analysis of the code they were reviewing.

G. Phase 7: Final Post-Review B and Wrap Up Questions

When asked about specific areas that they started looking for in Review B, the results were mixed. The students primarily noted starting to look for structural defects (class design, data structure design, file names) and visual representation defects (readability, indentation, whitespace use).

When asked "What did you make sure to do in the second round (B) of reviewing that you didn't think about or know to do in the first round (A) of reviewing?" here were some of main answers and themes:

- "Look at the amount of comments and quality of comments"
- "I made sure to give more general recommendations, mostly on structural/clarity. I thought those comments were really helpful on my reviews."
- "I tried to find more things such as design and logic to comment on."

In the final wrap up questions, the students were directly asked "What were some key things that you feel you learned by reading and marking up the ART code review by an experienced reviewer?" included these results:

- "I got a way better understanding of what exactly code review is and how to do it properly. "
- "I also learned that I should know when and where to change the style in which I name my variables to make the code more readable."
- "I mostly found clarification about how to word issues."
- "I've learned that the knowledge of others is very helpful when trying to understand optimization and practices that may have slipped my mind or don't understand that well."

The last question was "Do you feel this training session improved your skills in code reviewing?" on a Likert scale of 1 to 7. The CS2/3 students rated this an average of 5 while the CS4+ students rated it an average of 3.3. This aligns with our original thesis idea that CS2/3 students would benefit in more notable ways from an ART-Generated code review feedback on their work.

V. DISCUSSION

The primary purpose of this work was to pilot the ART technologies and to guide future studies in where automatic code review tools can fit into an educational framework. After reviewing the foundational work done in [4], it demonstrated that CS2 students would likely benefit from ART-generated code reviews. This work started to establish whether the same held true for CS4+ students, as well as started to identify a baseline of how accurate/good the reviews would be for students. Based on the results shown in Table II, the accuracy of ART in its reviews is likely good enough to be evaluated with more student cohorts without being overly misleading or incorrect. It also indicates that CS2/3 and CS4+ students are able to make use of the recommendations made by ART.

The shear between the CS2/3 and CS4+ students shown in Table III provides some evidence that this work's original question has some merit. When identifying themes among CS2/3 students about changes between Review A and Review B, it was noted how much their code reviewing had improved, especially that they gained confidence in what they were allowed to speak about during the review. The students increased their ability to understand and analyze code as well as use those skills to write much more direct feedback, to classify and identify main categories of code reviews to look for, and then generate reviews based on that analysis.

VI. CONCLUSION & FUTURE WORK

Applied Artificial Intelligence approaches, such as ART as an Automatic Static Analysis Tool, demonstrate significant opportunities to provide insightful, accurate, and timely feedback (minutes instead of days) to software development learners. ART generates code reviews on the student's work rather than directly generating the code itself. This system allows the student to continue to grow their understanding of how to create code and move the project forward themselves. With CS2/3 students reporting around 70% of the specific code review suggestions being useful to improve their code, and a rate of around 1% for erroneous suggestions, this warrants further study of using ART to generate code reviews in classroom settings. ART also showed success at helping students gain code reviewing skills. This pilot study demonstrated that the ART approach is worth exploring for eventual integration into practice for CS2/3 course settings.

Future Work: To move this work in progress pilot forward, a streamlined, asynchronous, and refined study needs to be performed. Students should be able to sign up as participants, proceed through the series of reviews and feedbacks at their own pace to allow for a much larger scale of participation to help provide more evidence for or against the efficacy of ART-style code reviews as a pedagogically sound tool.

References

[1] B. Puryear and G. Sprint, "Github copilot in the classroom: Learning to code with AI assistance," *J. Comput. Sci. Coll.*, vol. 38, no. 1, pp. 37–47, Nov. 2022, ISSN: 1937-4771.

[2] A. Alami, M. L. Cohn, and A. Wasowski, "Why does code review work for open source software communities?" In *International Conference on Software Engineering*, ser. ICSE, IEEE, 2019, pp. 1073–1083. DOI: 10.1109/ICSE.2019.00111.

[3] S. Krusche, M. Berisha, and B. Bruegge, "Teaching code review management using branch based workflows," in *Proceedings of the 38th International Conference on Software Engineering Companion*, 2016, pp. 384–393. DOI: 10.1145/2889160.2889191.

[4] A. S. Crandall, G. Sprint, and B. Fischer, "Generative pre-trained transformer (GPT) models as a code review feedback tool in computer science programs," *The Journal of Computing Sciences in Colleges*, vol. 39, no. 1, pp. 38–47, Oct. 2023, ISSN: 1937-4771.

[5] T. H. Morris, "Experiential learning–a systematic review and revision of kolb's model," *Interactive learning environments*, vol. 28, no. 8, pp. 1064–1077, 2020. DOI: 10.1080/10494820.2019.1570279.

[6] D. A. Kolb, *Experiential learning: Experience as the source of learning and development*, 2nd ed. FT press, 2014, ISBN: 978-0133892406.

[7] J. W. Gentry, "What is experiential learning," *Guide to business gaming and experiential learning*, vol. 9, no. 1, pp. 20–32, 1990.

[8] C. Vassallo, S. Panichella, F. Palomba, S. Proksch, A. Zaidman, and H. C. Gall, "Context is king: The developer perspective on the usage of static analysis tools," in *International Conference on Software Analysis, Evolution and Reengineering*, ser. SANER, IEEE, 2018, pp. 38–49. DOI: 10.1109/SANER.2018.8330195.

[9] T. Lewowski and L. Madeyski, "Code smells detection using artificial intelligence techniques: A business-driven systematic review," *Developments in Information & Knowledge Management for Business Applications: Volume 3*, pp. 285–319, 2022. DOI: 10.1007/978-3-030-77916-0_12.

[10] X. Yaozong, S. Xuebin, Z. Shuhua, Z. Qiujun, and J. Weinan, "Static analysis method of C code based on model checking and defect pattern matching," in *IEEE 5th International Conference on Power, Intelligent Computing and Systems (ICPICS)*, IEEE, 2023, pp. 567–573. DOI: 10.1109/ICPICS58376.2023.10235566.

[11] T. D. Indriasari, A. Luxton-Reilly, and P. Denny, "A review of peer code review in higher education," *ACM Transactions on Computing Education*, vol. 20, no. 3, pp. 1–25, Sep. 2020. DOI: 10.1145/3403935.

[12] X. Song, S. C. Goldstein, and M. Sakr, "Using peer code review as an educational tool," in *Conference on Innovation and Technology in Computer Science Education*, 2020, pp. 173–179. DOI: 10.1145/3341525.3387370.

[13] M. R. Weaver, "Do students value feedback? Student perceptions of tutors' written responses," *Assessment & Evaluation in Higher Education*, vol. 31, no. 3, pp. 379–394, 2006. DOI: 10.1080/02602930500353061.

[14] M. E. Califf and N. Dunne, "Feedback in context: Using a code review tool for program grading," in *ACM Technical Symposium on Computer Science Education*, vol. 1, 2022, pp. 92–97. DOI: 10 . 1145 / 3478431 . 3499402.

[15] A. K. Turzo, "Towards improving code review effectiveness through task automation," in *37th IEEE/ACM International Conference on Automated Software Engineering*, 2022, pp. 1–5. DOI: doi.org/10.1145/3551349.3559565.

[16] C. Sadowski, E. Söderberg, L. Church, M. Sipko, and A. Bacchelli, "Modern code review: A case study at Google," in *Iinternational Conference on Software Engineering: Software Engineering in Practice*, 2018, pp. 181–190. DOI: 10.1145/3183519.3183525.

[17] O. Hazzan, Y. Dubinsky, and O. Meerbaum-Salant, "Didactic transposition in computer science education," *ACM Inroads*, vol. 1, no. 4, pp. 33–37, 2010. DOI: 10.1145/1869746.1869759.

[18] U. Fuller, C. G. Johnson, T. Ahoniemi, *et al.*, "Developing a computer science-specific learning taxonomy," *ACM SIGCSE Bulletin*, vol. 39, no. 4, pp. 152–170, 2007. DOI: 10.1145/1345375.1345438.