

The Journal of Computing Sciences in Colleges

**Papers of the 37th Annual CCSC
South Central Conference**

April 24th, 2026
McNeese State University
Lake Charles, Louisiana

Abbas Attarwala, Editor
California State University, Chico

Tina Johnson, Regional Editor
Midwestern State University

Volume 41, Number 7

April 2026

The Journal of Computing Sciences in Colleges (ISSN 1937-4771 print, 1937-4763 digital) is published at least six times per year and constitutes the refereed papers of regional conferences sponsored by the Consortium for Computing Sciences in Colleges.

Copyright ©2026 by the Consortium for Computing Sciences in Colleges. Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the CCSC copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Consortium for Computing Sciences in Colleges. To copy otherwise, or to republish, requires a fee and/or specific permission.

Table of Contents

The Consortium for Computing Sciences in Colleges Board of Directors	5
CCSC National Partners	7
Welcome to the 2026 CCSC South Central Conference	8
Regional Committees — 2026 CCSC South Central Region	10
Keynote Speaker: Doug Pickle, President Global Data Systems — Keynote	11
<i>Mr. Doug Pickle, President, Global Data Systems</i>	
Evaluating LLMs for SQL Instruction: A Comparative Study of Five Commercial Models	13
<i>Shristi Thapaliya, University of Central Oklahoma; Gang Qian, University of Central Oklahoma; Fei Zuo, University of Central Oklahoma</i>	
Beyond Binary Detection: Evaluating Transformer and Hybrid Architectures for Multi-Class Classification of AI-Generated Phishing	23
<i>John W. Burris, Southeastern Louisiana University; Sushovan Adhikari, Southeastern Louisiana University</i>	
Infusing AI Literacy Across Disciplines Using a ChatGPT-Based POGIL Module	35
<i>Rebecca A. Zulli, Cynosure Consulting; Adrienne A. Smith, Cynosure Consulting; Sambit Bhattacharya, Fayetteville State University; Bogdan Czejdo, Fayetteville State University; Xiaochen Hu, Fayetteville State University; Khalid Lodhi, Fayetteville State University; Zahra Shekarkhar, Fayetteville State University</i>	
From Dorm Room to Deployment: A Faculty-Mentored, Student-Led Case Study in Early Undergraduate Computing Education	48
<i>Augustus Scarlato, Stetson University; Ryan Ouardawi, Columbia University; Oybeck Ablakulov, Vanderbilt University</i>	
When Discomfort Teaches: Using Cognitive Dissonance to Develop Adversarial Thinking in AI-Driven Computing Courses	60
<i>Christian Servín, El Paso Community College; Nadia V. Karichev, El</i>	

Paso Community College; Ivan Alonso, El Paso Community College

Introducing Network Hardening and Vulnerability Detection using CUDA Parallel Graph Algorithms in Computer Network Courses 70

Quoc-Nam Tran, Southeastern Louisiana University

LLM-Assisted PDF-to-LaTeX Conversion for Academic Document Management 81

Brandon Dean, Ball State University; Adithya Kulkarni, Ball State University

GLIDE: Developing a Usability-Driven Programming Environment for Novices 91

Mika Morgan, Midwestern State University; Stephanie Ludi, University of North Texas; Tina Johnson, Midwestern State University

Not Quite Dead Yet: Strategies to Mitigate the Detrimental Effects of LLMs on CSE 100

Michael P. Rogers, University of Wisconsin Oshkosh; Christopher L. Groves, University of Wisconsin Oshkosh; Hannah M. Hillberg, University of Wisconsin Oshkosh; Aziz Fellah, Northwest Missouri State University

How Students Use AI in Project Development 111

Catherine Stringfellow, Midwestern State University; N. (Thor) Lang, Midwestern State University; R. (Cooper) Ricketts, Midwestern State University; Jose Marquez, Midwestern State University; Mekhi Prospero, Midwestern State University

Reviewers — 2026 CCSC South Central Conference 121

The Consortium for Computing Sciences in Colleges

Board of Directors

Following is a listing of the contact information for the members of the Board of Directors and the Officers of the Consortium for Computing Sciences in Colleges (along with the years of expiration of their terms), as well as members serving CCSC:

Bryan Dixon, President (2026), bcdixon@csuchico.edu, Computer Science Department, California State University Chico, Chico, CA 95929.

Shereen Khoja, Vice President/President-Elect (2026), shereen@pacificu.edu, Computer Science, Pacific University, Forest Grove, OR 97116.

Abbas Attarwala, Publications Chair (2027), aattarwala@csuchico.edu, Department of Computer Science, California State University Chico, Chico, CA 95929.

Ed Lindoo, Treasurer (2026), elindoo@regis.edu, Anderson College of Business and Computing, Regis University, Denver, CO 80221.

Haiyan Cheng, Membership Secretary (2028), hcheng@willamette.edu, Department of Computer Science, Willamette University, Salem, OR 97301.

Judy Mullins, Central Plains Representative (2026), mullinsj@umkc.edu, University of Missouri-Kansas City, Kansas City, MO (retired).

Michael Flinn, Eastern Representative (2026), mflinn@frostburg.edu, Department of Computer Science & Information Technologies, Frostburg State University, Frostburg, MD 21532.

Gabe Ferrer, Midsouth Representative (2028), ferrer@hendrix.edu, Department

of Computer Science, Hendrix College, Conway, AR 72032.

David Largent, Midwest Representative (2026), dllargent@bsu.edu, Department of Computer Science, Ball State University, Muncie, IN 47306.

Michael Gousie, Northeastern Representative (2028), gousie_mike@wheatoncollege.edu, Computer Science Department, Wheaton College, Norton, MA 02766.

Ben Tribelhorn, Northwestern Representative (2027), tribelhb@up.edu, School of Engineering, University of Portland, Portland, OR 97203.

Mohamed Lotfy, Rocky Mountain Representative (2028), MohamedL@uvu.edu, Information Systems & Technology Department, College of Engineering & Technology, Utah Valley University, Orem, UT 84058.

Mika Morgan, South Central Representative (2027), mika.morgan@msutexas.edu, Department of Computer Science, Midwestern State University, Wichita Falls, TX 76308.

Karen Works, Southeastern Representative (2027), keworks@pc.fsu.edu, Department of Computer Science, Florida State University Panama City, Panama City, FL 32405.

Michael Shindler, Southwestern Representative (2026), mikes@uci.edu, Computer Science Department, UC Irvine, Irvine, CA 92697.

Serving the CCSC: These members are serving in positions as indicated:

Bin Peng, Associate Editor, bin.peng@park.edu, Computer Science and Information Systems, Park University, Parkville, MO 64152.

Lucas Cordova, Associate Treasurer, lpcordova@willamette.edu, Department of Computer Science, Willamette University, Salem, OR 97301.

George Dimitoglou, Comptroller, dimitoglou@hood.edu, Department of Computer Science, Hood College, Frederick, MD 21701.

Megan Thomas, Membership System

Administrator, mthomas@cs.csustan.edu, Department of Computer Science, California State University Stanislaus, Turlock, CA 95382.

Karina Assiter, National Partners Chair, KarinaAssiter@landmark.edu, Landmark College, Putney, VT 05346.

Ed Lindoo, UPE Liaison, elindoo@regis.edu, Anderson College of Business and Computing, Regis University, Denver, CO 80221.

Deborah Hwang, Webmaster, hwangdjh@acm.org.

2026 CCSC Midsouth Sponsors/Partners

The Consortium is very happy to have the following as National Partners and Sponsors. If you have the opportunity, please thank them for their support of computing in teaching institutions. As National Partners and Sponsors they are invited to participate in our regional conferences. Visit with their representatives there.

Sponsors/Partners

ACM2y & ACM CCECC

Blossoms

College Board

Eduicator

MAP-CS

Rephactor

Chairs' Welcome

Welcome to the 37th Annual Conference of the CCSC: South Central Region! We hope you have a wonderful time at the conference and that you enjoy the beautiful college campus at McNeese State University in Lake Charles, Louisiana.

The Steering Committee are the people who have made this conference happen, managing all the logistics, arranging the food, reserving the halls, making arrangements for the paper submissions to be peer reviewed, inviting the guest speaker, and for all the wonderful things you see at the conference as you walk around.

I would like to thank Dr. Tina Johnson (Professional Papers Chair), Dr. Jennifer Lavergne (Reviewer Chair), and Professor Anne Marie Eubanks (Registrar) for their work in making this conference happen. They have worked with authors, presenters and reviewers, and have done a fantastic job of managing the whole process. Our professional papers are peer-reviewed by the double-blind peer review process.

I would also like to thank the rest of the members of the CCSC-SC Steering Committee for all their hard work. For any person who is attending CCSC-SC for the first time, please check out www.ccsc.org. Clicking on the "Regions" link on the "CCSC Regions" section takes one to: ccsc.org/regions. Clicking on the Website for the South Central Region then takes us to: ccsc.org/southcentral.

The Registrar for this conference is Professor Anne Marie Eubanks, and the National Board Representative is Dr. Mika Morgan, and they have done a great job in managing all the financials and the registration process. I am thankful to the Posters Chair Professor Shyam Karrah for always doing a fabulous job of managing student posters and faculty posters. Dr. Eduardo Colmenares-Diaz has been serving as our Publicity Chair, and all those of us who have received the CFPs from him over these years know what a phenomenal job he is doing. If you would like for Dr. Mika Morgan (CCSC:SC's National Board Representative) to carry your ideas to CCSC National, please meet her during the conference and pass on your ideas to her. Special thanks go to Dr. Christian Servin (Tutorials Chair), Dr. Michael Kart (for Managing ConfOps), Dr. Michael Hostetler (Treasurer), and our At-Large members Dr. Steele Russell and Dr. Michael Rogers.

The Steering Committee members deserve all the accolades and thanks we can give, and some more, for doing such a great job in maintaining our web pages. Much of the information in this conference program you are holding can also be found on the web site for CCSC-SC: <http://www.ccsc.org/southcentral/>. There are some very important links from that page – please be sure to explore it thoroughly. Thank you to the Steering Committee for putting all this

together.

My first time at a CCSC-SC conference was the 19th annual CCSC-SC conference at Texas A&M University in Corpus Christi, TX in April 2008. It was an unbelievably fun learning experience, with well over a hundred attendees. I have been going to CCSC-SC ever since except for the COVID-19 years when some conferences went online. Thanks to all of you who have made new members feel welcome! Please continue to invite your friends and fellow researchers and teachers to attend the CCSC-SC conferences in the coming years.

I've learned some very important lessons from attending these CCSC-SC conferences. Among other things, it proved to me that in this day and age of online tools and online learning, there is nothing that can compete with face-to-face learning and attending meaningful outstanding conferences such as the CCSC-South Central Conference.

I am very thankful to Mr. Juste-Marie Fagla for helping to design this wonderful brochure. Thanks to Mr. Caden Fontenot, Mr. Payton Todd and our student volunteers for their help with all tasks related to the conference.

Dr. Jennifer Lavergne is a valued colleague in the CS department, and her work has been hugely impactful for the successful conduct of this conference. She helped with getting students registered, getting papers reviewed, and in managing various logistics aspects. Dr. Vasan is the department head of Engineering and Computer Science at McNeese State University. From day one, he has been supporting all arrangements. He has been with us every step of the way with his ideas, support and encouragement. A huge thank you to Dr. Vasan. Special thanks go to Ms. Callie Rolfe, our Academic and Operations Coordinator who has played a vital role in helping us make all the arrangements for the conference.

We are always looking for ways and means to improve CCSC-South Central. Please make sure you meet the members of the Steering Committee, and give your inputs and suggestions. We also have a feedback form and a suggestion box. We would love to hear your ideas, and we look forward to the chance to implement any ideas that will make the CCSC-SC conference even better next year.

Thank you, and Welcome to Lake Charles.

Dr. Vipin Menon
menon@mcneese.edu
Chair, 2026 CCSC South Central Conference

2026 CCSC South Central Conference Steering Committee

- Vipin Menon, Conference Chair .. McNeese State University in Lake Charles, Louisiana
- Eduardo Colmenares-Diaz, Conference Publicity Chair Midwestern State University in Wichita Falls, Texas
- Vipin Menon, Moderator Chair .. McNeese State University in Lake Charles, Louisiana
- Mika Morgan, National Board Representative Midwestern State University in Wichita Falls, Texas
- Shyam Karrah, Posters ChairThe University of Texas at Dallas in Richardson, Texas
- Tina Johnson, Papers Chair ...Midwestern State University in Wichita Falls, Texas
- Anne Marie Eubanks, RegistrarStephen F. Austin State University in Nacogdoches, Texas
- Jennifer Lavergne, Reviewer Chair McNeese State University in Lake Charles, Louisiana
- Christian Servin, Panels, Tutorials, Workshops, Nifty Assignments Chair . El Paso Community College in El Paso, Texas
- Vacant Position, Webmaster

Keynote Speaker: Doug Pickle, President Global Data Systems*

Keynote

Mr. Doug Pickle
President, Global Data Systems



Abstract

Mr. Doug Pickle is President of Global Data Systems, where he leads the company's shift from managing IT to helping organizations make better decisions with technology.

*Copyright is held by the author/owner.

Under his leadership, GDS has evolved from a traditional managed service provider into a Managed Intelligence Provider. The focus moved from tickets and uptime to insight, execution, and measurable business outcomes. Technology is no longer the product. Discipline, clarity, and applied intelligence are.

Before joining GDS, Doug served as Senior Director of National Sales for Optimum Business, leading a national team delivering telecommunications and connectivity solutions to mid market, enterprise, and wholesale customers. He attended McNeese State University with a focus on Electrical Engineering and has spent more than 30 years in the technology industry.

Beyond his corporate role, Doug serves on nonprofit boards focused on leadership development, education, and technology advancement across Louisiana. His guiding belief is consistent across business and community work: progress comes from clear direction, accountability, and sustained investment in people.

Evaluating LLMs for SQL Instruction: A Comparative Study of Five Commercial Models*

Shristi Thapaliya, Gang Qian, Fei Zuo

Department of Computer Science

University of Central Oklahoma, Edmond, OK 73034

{sthapaliya1, fzuo, gqian}@uco.edu

Abstract

Large Language Models (LLMs) have demonstrated notable SQL coding proficiency. However, their performance in educational contexts remains underexplored. This paper presents a systematic evaluation of five prominent commercial LLMs: ChatGPT, Claude, Gemini, Copilot, and Grok. We introduce an assessment framework aligned with undergraduate database curricula, comprising a representative database schema, six query categories, a zero-shot, multi-turn prompting strategy, and a scoring methodology that evaluates both correctness and efficiency. Our empirical results indicate that while the evaluated models exhibit broadly comparable performance, accuracy significantly degrades as query complexity increases. Furthermore, qualitative error analysis reveals a "Precision Paradox": effective prompting for complex queries requires foundational SQL knowledge, suggesting that LLMs currently function as sophisticated assistants for experts rather than fully autonomous solutions for novices. These findings highlight the continuing importance of SQL education in the generative AI era.

1 Introduction

Since their inception, large language models (LLMs) have been widely used for automated coding tasks, including the generation of SQL query statements

*Copyright ©2026 by the Consortium for Computing Sciences in Colleges. Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the CCSC copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Consortium for Computing Sciences in Colleges. To copy otherwise, or to republish, requires a fee and/or specific permission.

from natural language prompts [1]. Although these models have substantially reduced development time and improved accessibility to databases, concerns remain regarding their accuracy and reliability in SQL generation, as erroneous SQL statements can produce incorrect results [2] and introduce potential security risks [3]. Furthermore, the rapid advancement of LLM capabilities necessitates continuous evaluation [4], as limitations observed in earlier models may be mitigated or resolved in more recent ones.

From an educational perspective, the growing use of LLMs presents new challenges for database instruction. As students increasingly use these tools to generate SQL statements, instructors must understand the extent to which current models can accurately handle queries of varying complexity. Accordingly, teaching strategies must evolve to ensure that students develop strong foundational SQL skills in the age of generative AI. Initial efforts in this direction have begun to emerge [5].

The study presented in this paper aims to provide a systematic evaluation of the capability of major commercial LLMs, including ChatGPT, Claude, Gemini, Copilot, and Grok, for SQL query statement generation with different complexity. Unlike prior work that primarily emphasizes benchmark performance for industry applications, this research adopts an assessment framework aligned with how SQL is taught and evaluated in undergraduate database courses. The framework includes a “university” database schema with populated tables, six query categories with varying levels of complexity, a zero-shot, multi-turn prompting strategy, and a scoring scheme based on correctness and efficiency.

The main contributions of the study are as follows:

- We provide an up-to-date evaluation of leading commercial LLMs with respect to their capability in generating SQL query statements.
- We propose an assessment framework aligned with SQL teaching and evaluation practices in typical undergraduate database courses.
- We present observations and insights that can inform database instructors in developing appropriate approaches to the use of LLMs in SQL learning.

The remainder of this paper is organized as follows. Section 2 reviews recent studies on the evaluation of LLMs for SQL query generation. Section 3 describes the proposed assessment framework. Section 4 presents the experimental results, along with key observations and insights. Finally, Section 5 concludes the paper with future work.

2 Related Work

Text-to-SQL research translates natural language into executable SQL, bridging semantic parsing, natural language understanding, and database systems [6],

[7]. This survey focuses specifically on prior studies that apply LLMs to Text-to-SQL, and examines them across three dimensions: query categorization, prompting strategies, and evaluation methodologies.

2.1 Categorization of SQL Queries

Categorization is essential for understanding LLM performance across varying levels of complexity. Several studies utilize query categories from established benchmarks; for instance, Liu et al. [1] employed the Spider benchmark [8], which defines four levels (easy, medium, hard, and extra hard) based on the number of components such as joins, aggregations, and nested subqueries. Similarly, the BIRD benchmark [9] utilizes three difficulty levels (simple, moderate, and challenging), which factor in structural complexity alongside question comprehension, schema-linking, and the requirement for external knowledge.

Other research evaluates LLMs through alternative or implicit categorization strategies. For example, Chadha et al. [10] utilized the MIMICSQL dataset [11], where query complexity is implicitly reflected through real-world question-SQL pairs derived from multi-table clinical schemas. More explicit methods include Ma et al.’s categorization based on structural and numeric features like common table expressions [4], or Rosca et al.’s grouping by syntactic features and schema familiarity [12].

Our study adopted a similar methodology that recognized structural complexity, ranging from single-table to multi-table queries, while isolating specific types like division and Top-X operations to align with the types of SQL queries typically taught in undergraduate database courses.

2.2 Prompting Strategy

Prompting generally involves a database schema and a natural language query. Common strategies include: (1) *Zero-Shot*, containing no SQL examples (only the query) [1]; (2) *Few-Shot*, providing query-SQL pairs to guide generation [13], [14]; and (3) *Single- vs. Multi-Turn* interactions, where the schema is provided either with every query or once per session [1]. A self-reflection prompting technique, evaluated in [14], allows an LLM to review and critique its own responses, thereby reducing errors. We utilized a zero-shot, multi-turn strategy to mimic the most likely usage pattern by undergraduate students.

2.3 Evaluation Methodology

Most evaluations rely on execution accuracy on a DBMS [1], [10], [15], often supplemented by exact-match and logical-form metrics [1], [10] or qualitative

error analysis to identify failure patterns like incorrect joins or column selections [1], [4]. Unlike benchmark-driven scoring, we employed a manual quantitative methodology mirroring undergraduate grading.

Overall, extant research consistently indicates that while LLMs show strong potential, performance correlates negatively with query complexity and schema scale [4], [15].

3 Assessment Framework

In this section, we describe the assessment framework used in our study. As introduced in Section 1, this framework was designed to simulate the academic setting of a typical undergraduate database course.

3.1 Database Schema

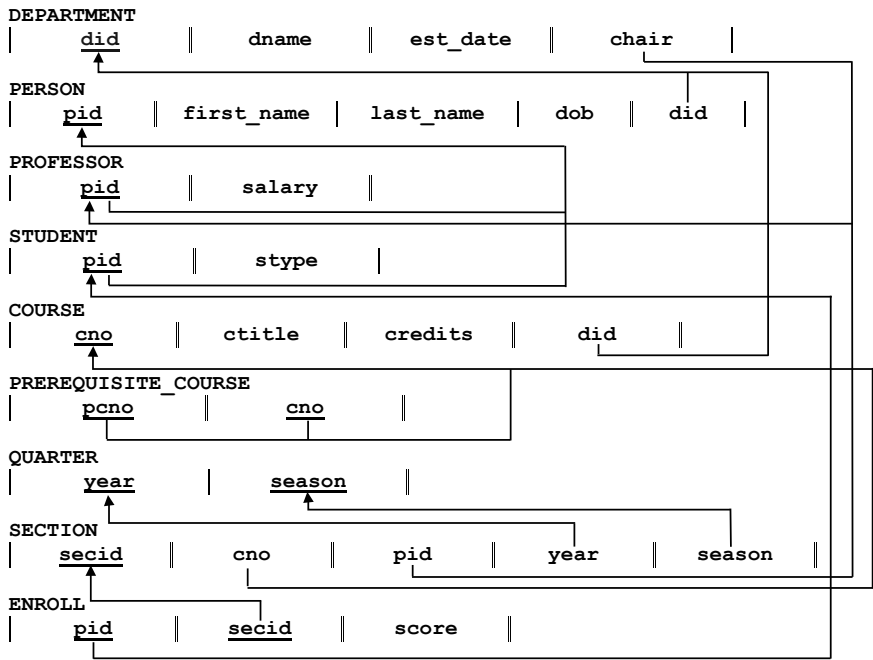


Figure 1: Database Schema Used in the Study

The database schema was created around an imaginary university information management system, with a level of complexity typical of schemas used

to teach SQL in undergraduate courses. It contains multiple tables to support the different categories of queries evaluated in this study. Figure 1 illustrates the schema, where primary keys are underlined and arrows represent foreign key relationships. The schema was implemented in an Oracle DBMS, with all tables populated for experimental evaluation.

3.2 Query Categorization

Query categories were defined based on both their structural complexity, ranging from single-table to multi-table queries, and specific query types such as the division operation and Top-X queries. Table 1 provides a summary of the categories used in this study.

Ten queries were manually created in plain English for each of the six categories. Examples of the queries can be found in Table 1. A standard answer (SELECT statement) was manually written for each query. The standard answers were executed in the Oracle DBMS, and the results were recorded.

Query Category	Description	Query Example
Single-Table	Simplest query type that involves only one table	Retrieve <i>ctitle</i> and <i>credits</i> of the course whose <i>cno</i> is CS480.
Multi-Table	More than one table is needed for data retrieval.	Retrieve <i>ctitle</i> of the courses that have prerequisite CS380.
Multi-Unit	Multiple SELECT blocks, such as nested queries and queries with set operations like INTERSECT or MINUS	Retrieve <i>cno</i> and <i>ctitle</i> of the courses with 3 credits that are offered in both Spring 2025 and Fall 2025.
Division	The division operation is needed in the query.	Retrieve <i>first_name</i> and <i>last_name</i> of the professors who have taught all courses with did 'PHY'.
Aggregate Functions	Aggregate functions are needed, such as COUNT, SUM, and AVERAGE, with GROUP BY and HAVING.	Retrieve <i>dname</i> , <i>ctitle</i> , <i>secid</i> , and <i>cno</i> of the sections that have fewer than 2 students enrolled for Fall 2024.
Top-X	Queries that return the top or bottom x results	Retrieve <i>pid</i> of the top 2 professors by the number of sections taught.

Table 1: Query Categories Used in the Study

3.3 Prompting Strategy

A zero-shot, multi-turn prompting strategy was adopted in the study, reflecting the way in which undergraduate students are most likely to use LLMs for SQL generation. The database schema was provided at the beginning of each session

to the LLMs in the form of CREATE TABLE statements. Then, the plain English queries were used as the prompts.

Column names were explicitly specified in the prompts according to the schema to minimize ambiguity. In addition, efforts were made to provide relevant conditions and requirements in the prompts to reduce false assumptions by the models.

3.4 Scoring Approach

The study used a quantitative evaluation approach aligned with typical grading practices for SQL assignments in undergraduate database courses. LLM-generated SQL statements were evaluated based on a scoring system in which the maximum score is 100%. The tuples retrieved by each generated SQL statement were compared with the results provided by the corresponding standard answer. In addition, a manual evaluation of each generated statement was performed.

For minor issues, such as incorrect column ordering, inclusion of extra columns, or inefficiency, 10 points were deducted per occurrence. Statements that produced incorrect query results were assigned a score of zero. Similarly, statements containing syntax errors or non-standard SQL constructs (e.g., ROWNUM) were also scored as zero to ensure consistency with standard SQL practices emphasized in undergraduate instruction.

3.4.1 Example of an inefficient query statement

The following example belongs to the Top-X category. The prompt was “Retrieve *did* of the top 3 departments in terms of professor counts with ties, i.e., you can retrieve more than three departments if there are top departments with equal numbers of professor counts.” The SQL generated by Claude was considered inefficient because it unnecessarily included the Department table, and the second left outer join should have been an inner join.

```
SELECT did, professor_count
FROM (
  SELECT d.did, COUNT(p.pid) AS professor_count,
         DENSE_RANK() OVER (ORDER BY COUNT(p.pid) DESC) AS rank
  FROM Department d
       LEFT JOIN Person per ON d.did = per.did
       LEFT JOIN Professor p ON per.pid = p.pid
  GROUP BY d.did )
WHERE rank <= 3
ORDER BY professor_count DESC, did;
```

4 Empirical Study

4.1 Environment Settings

The study was conducted from October 2025 to January 2026, a period during which several models received updates. We evaluated ChatGPT (5.0, 5.1), Claude Sonnet (4.5), Grok 4, Gemini (2.5 and 3 Flash), and five web builds of Copilot ranging from 20251014.50.2 to 20251216.44.1. All SQL execution was performed on Oracle 19c Enterprise Edition.

4.2 Results and Statistical Analysis

As discussed in Section 3.4, a score was assigned to each SQL statement generated for the 10 queries within each of the six query categories for every evaluated LLM. Category-level averages were then calculated from these scores, followed by overall averages across categories for each model. The results are summarized in Table 2.

LLM	Single-Table	Multi-Table	Multi-Unit	Division	Aggregate Functions	Top-X	Average
ChatGPT	97	98	86	80	79	96	89.33
Claude	100	100	100	80	90	76	91.00
Gemini	100	100	98	90	100	88	94.33
Copilot	99	99	88	70	79	99	89.00
Grok	100	100	98	70	97	98	93.83
Average	99.20	99.40	94.00	76.00	89.00	91.40	91.50

Table 2: Average Scores per Category per LLM

Performance results by model (pooled across query categories) show Gemini leading (94.33) over Grok (93.83), Claude (91.00), ChatGPT (89.33), and Copilot (89.00). Despite a high grand mean (91.50), reflecting high proficiency across models, a one-way ANOVA found no statistically significant effect of model identity on performance ($F(4, 25) = 0.33, p = 0.85$); pairwise t-tests confirmed this parity (all $p > 0.10$). These findings indicate that while mean variations exist, the evaluated LLMs achieve statistically comparable performance in SQL generation.

Results by category (pooled across models) revealed near-ceiling accuracy for Single-Table (99.2%) and Multi-Table (99.4%) queries, contrasted by significantly lower performance in Division (76.0%). Aggregate Functions and Top-X showed intermediate proficiency. ANOVA confirmed a significant effect of query category ($F(5, 24) = 6.24, p < 0.001$), suggesting that query complexity, rather than model identity, is the primary driver of performance variation.

4.3 Discussion and Insights

Among the incorrect query statements generated, some contained syntax errors. In addition, a few Division queries could not be correctly answered by any of the evaluated LLMs, even after prompt refinement. However, we observed that incorrect SQL statements were more likely to occur when prompts had the following issues: (1) ambiguity, (2) missing conditions, or (3) lack of common-sense knowledge.

A simple example of ambiguity is a Top-X query with the original prompt: “Retrieve *did* of the top 3 departments in terms of professor counts.” This prompt was ambiguous because it did not specify how ties should be handled.

A more complex example involves a Division query: “Retrieve *pid* of the professors who have taught all graduate students in his/her department.” Several LLMs generated SQL statements that returned substantially more professors than the standard answer. Closer inspection revealed that their approach treated professors in departments with zero graduate students as valid results.

An example illustrating the lack of common-sense knowledge was the Top-X query: “Retrieve *pid*, *first_name*, *last_name*, and *GPA* of the top 2 students in terms of GPA in the department whose *did* is PHY.” Most LLMs calculated the average scores instead of the GPAs.

Our observations of recurring error patterns, as well as the interaction between prompting and SQL generation, led to the following insights:

Insight 1. The Precision Paradox in Natural Language Prompting:

There is a clear mismatch between the inherent flexibility and ambiguity of a natural language (e.g., English) and the level of precision required for coding-oriented prompts, even when such prompts appear to be straightforward plain language statements. While adding more conditions to a prompt can reduce ambiguity and improve query quality, it remains difficult to construct prompts that fully capture all constraints without unintended interpretations, even for experienced SQL practitioners. In some cases, ambiguities only become apparent after examining the executed query results and analyzing the generated SQL statements. Therefore, it is unlikely at present that individuals without prior SQL knowledge can consistently formulate correct prompts for complex query types, despite the broader goal of Text-to-SQL research. Effective prompting for complex queries appears to require at least a foundational understanding of SQL, with deeper knowledge becoming necessary as query complexity increases.

Insight 2. Implications for SQL Education: Building on the first insight, SQL education remains essential in the era of LLMs. For database instruction, unrestricted reliance on LLMs for solving SQL assignments may hinder

the development of fundamental skills. Instructors may therefore need to carefully consider when and how LLM usage is appropriate, particularly in graded coursework and assessments.

5 Conclusion and Future work

In this paper, we presented a comparative evaluation of five prominent commercial LLMs to assess their capability in generating SQL query statements. To address the needs of database educators in an era of rapid AI evolution, we introduced an assessment framework suitable for an empirical study aligned with typical undergraduate database instruction and grading practices. The framework includes a database schema, six query categories with varying complexities, a prompting strategy, and an approach to quantitative evaluation.

The experimental results indicate that, while the evaluated LLMs performed comparably, their performance degrades as query complexity increases, a finding consistent with the existing literature. These results highlight a critical "precision paradox": effective prompting for complex relational tasks requires a foundational mastery of SQL, suggesting that LLMs currently serve better as sophisticated assistants for experts than as complete solutions for novices. Consequently, we argue that SQL education remains indispensable, and instructors must strategically integrate AI tools to supplement, rather than replace, the development of core logic and coding skills.

Future work may include a deeper examination of how advanced prompting techniques influence the quality of generated SQL statements. Furthermore, the study could be extended to the SQL Data Definition Language (DDL) for the generation of database objects such as tables, views, and stored procedures.

References

- [1] A. Liu, X. Hu, L. Wen, and P. S. Yu, "A comprehensive evaluation of ChatGPT's zero-shot Text-to-SQL capability," *arXiv preprint*, 2023. arXiv: 2303.13547.
- [2] R. Roberson, G. Kaki, and A. Trivedi, "Analyzing the effectiveness of large language models on Text-to-SQL synthesis," *arXiv preprint*, 2024. arXiv: 2401.12379.
- [3] E. Kim and S. Lee, "SQL injection in LLM-generated queries: Systematic analysis of detection gaps and security risks," *IEEE Access*, vol. 14, pp. 12 797–12 815, 2026.
- [4] L. Ma, K. Pu, and Y. Zhu, "Evaluating LLMs for Text-to-SQL generation with complex SQL workload," *arXiv preprint*, 2024. arXiv: 2407.19517.

- [5] K. Prakash, S. Rao, R. Hamza, J. Lukich, V. Chaudhari, and A. Nandi, “Integrating LLMs into database systems education,” in *Proc. 3rd Int. Workshop on Data Systems Education (DataEd ’24)*, 2024, pp. 33–39.
- [6] N. Deng, Y. Chen, and Y. Zhang, “Recent advances in Text-to-SQL: A survey of what we have and what we expect,” in *Proc. 29th Int. Conf. Comput Linguistics (COLING)*, 2022, pp. 2166–2187.
- [7] B. Qin et al., “A survey on Text-to-SQL parsing: Concepts, methods, and future directions,” *IEEE Trans. Knowl Data Eng.*, vol. 36, no. 5, pp. 1891–1912, May 2024.
- [8] T. Yu et al., “Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and Text-to-SQL task,” in *Proc. Conf. Empirical Methods in Nat. Lang. Process.(EMNLP)*, 2018, pp. 3911–3921.
- [9] J. Li et al., “Can LLM already serve as a database interface? a big bench for large-scale database grounded text-to-SQLs,” *arXiv preprint*, 2023. arXiv: 2305.03111.
- [10] I. K. Chadha, A. Gupta, S. Sarkar, M. Tomer, and T. Rathee, “Performance evaluation of open-source LLMs for Text-to-SQL conversion in healthcare data,” in *Proc. Int. Conf. Pervasive Comput Technologies (ICPCT)*, 2025, pp. 606–609.
- [11] P. Wang, T. Shi, and C. K. Reddy, “Text-to-SQL generation for question answering on electronic medical records,” in *Proc. Web Conf. 2020*, 2020, pp. 350–361.
- [12] C. Rosca and A. Stancu, “Quality assessment of GPT-3.5 and Gemini 1.0 Pro for SQL syntax,” *Comput Standards & Interfaces*, vol. 95, p. 104 041, 2026, Article No. 104041.
- [13] Z. Gu et al., “Few-shot Text-to-SQL translation using structure and content prompt learning,” *Proc. ACM Manag. Data*, vol. 1, no. 2, pp. 1–28, 2023, Article No. 147.
- [14] B. Han, A. Mathrani, and T. Susnjak, “Evaluating prompting strategies and large language models in systematic literature review screening: Relevance and task-stage classification,” *arXiv preprint*, 2025. arXiv: 2510.16091.
- [15] M. Ganti, L. Orr, and S. Wu, “Evaluating Text-to-SQL model failures on real-world data,” in *Proc. IEEE 40th Int. Conf. Data Eng. (ICDE)*, 2024, pp. 1–1.

Beyond Binary Detection: Evaluating Transformer and Hybrid Architectures for Multi-Class Classification of AI-Generated Phishing*

John W. Burris* and Sushovan Adhikari*
of Computer Science
Southeastern Louisiana University
Hammond, LA 70402
jburris@southeastern.edu
of Computer Science
Southeastern Louisiana University
Hammond, LA 70402
sushovan.adhikar@gmail.com

Abstract

As artificial intelligence (AI) has advanced rapidly and integrated into a wide range of domains, generative AI-particularly Large Language Models (LLMs) has seen a significant surge in adoption [13]. While these tools offer immense utility for generating novel text content, they present a dual-use dilemma. Despite incorporated safety mechanisms, LLMs remain highly susceptible to exploitation for generating malicious content, such as sophisticated phishing emails. Because AI-generated phishing emails closely mimic human writing, they are exceptionally difficult to detect. Furthermore, by combining LLMs with automation, attackers can execute polymorphic attacks where email structures and semantics constantly shift to evade conventional security filters. This paper addresses these vulnerabilities by evaluating the effectiveness of a hybrid

*Copyright ©2026 by the Consortium for Computing Sciences in Colleges. Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the CCSC copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Consortium for Computing Sciences in Colleges. To copy otherwise, or to republish, requires a fee and/or specific permission.

Long Short-Term Memory and Convolutional Neural Network (LSTM-CNN) architecture alongside a fine-tuned Bidirectional Encoder Representations from Transformers (BERT) model. In Phase I, models were trained for binary classification (human phishing versus human legitimate), achieving impressive accuracy rates of 98% and 99%, respectively. However, testing these seemingly robust binary models against an unseen dataset of AI-generated phishing emails revealed that both models classified at least 10% of the AI-generated malicious emails as legitimate. Subsequent error analysis indicated that AI-generated phishing communications systematically exhibit fewer urgency keywords and more complex vocabulary (higher Flesch-Kincaid Grade levels) than traditional human-crafted phishing. To mitigate this security gap, Phase II of this research adapted these models to a three-way classification framework. Through strategic freezing and unfreezing of model layers, we found that the BERT architecture demonstrated superior adaptability, achieving 99.48% accuracy on the multi-class task, whereas the LSTM-CNN model severely struggled, peaking at only 48.61%. This paper provides critical insights into the structural characteristics of AI-generated malicious content and establishes benchmarks for multi-class deep learning defenses.

1 Introduction

Artificial Intelligence (AI) and Machine Learning (ML) are continuously reshaping the digital landscape. Among recent advancements, Generative AI has garnered unprecedented attention [13]. Modern chatbots powered by Large Language Models (LLMs) such as ChatGPT, Gemini, and Claude are now widely accessible to the public, enabling users to generate highly articulate text based on simple prompts [7]. While developers have integrated safety guardrails to prevent misuse, these models remain vulnerable to exploitation by malicious actors seeking to generate phishing emails. Traditional phishing campaigns often rely on mass-distributed templates that contain discernible linguistic errors, predictable structures, and common urgency-inducing keywords. In contrast, AI-generated phishing emails are characterized by enhanced sophistication, context-awareness, and a seamless emulation of authentic human communication. When combined with automated deployment pipelines, LLMs empower attackers to execute polymorphic phishing attacks, where the text of each malicious email is dynamically altered to continually bypass static signatures and heuristic-based security software.

To effectively mitigate the growing threat of AI-generated phishing, cybersecurity frameworks must transition from binary classifiers to multi-class detection systems. It is crucial to develop models capable of distinctly categorizing communications into three classes: human-generated phishing emails,

legitimate human emails, and AI-generated phishing emails.

1.1 Motivation

Using easily accessible LLMs, bad actors can instantly generate malicious emails that systematically bypass machine learning models trained exclusively on human-crafted phishing data. This research addresses this critical vulnerability by introducing a three-way classification system utilizing BERT (large-uncased) and LSTM-CNN architectures. Additionally, this study provides an in-depth characterization of the distinct properties of AI phishing emails that enable them to evade binary detection models, representing a rapidly growing security concern.

1.2 Research Aims and Objectives

AI-generated text exhibits highly complex semantic patterns that make it challenging to discern from authentic human content. Transformer architectures, such as BERT, are particularly well-suited for this challenge because of their ability to capture intricate bidirectional semantic relationships within the language. By applying transfer learning to pre-trained networks, we can effectively compensate for the current scarcity of large-scale AI-generated phishing datasets.

2 Background and Literature Review

2.1 Phishing Attack Vectors

The term "phishing," coined in 1996, describes a form of cyber fraud where attackers use deceptive telecommunications to harvest sensitive personal data, such as usernames and passwords, akin to using a "fishing hook". While phishing can be executed via SMS, VoIP, eFax, and Wi-Fi, email remains the most prevalent medium because it enables attackers to distribute fraudulent messages globally at virtually zero cost while masking their geographic origins. Several specialized types of phishing exist within the email domain:

- **Spoofing:** Attackers disguise their email address to make it appear as though the message originates from a trusted entity, such as a bank or a known colleague. This technique exploits fundamental vulnerabilities in the Simple Mail Transfer Protocol (SMTP), which historically lacks robust built-in authentication.
- **Spear-Phishing:** Unlike generic mass-produced spam, spear-phishing is highly targeted. Attackers perform extensive reconnaissance using plat-

forms like LinkedIn to gather personal data, allowing them to craft highly relevant and persuasive emails. Studies have shown that spear-phishing accounted for over 65% of malware distribution cases in recent years.

- **Whaling:** A subset of spear-phishing directed specifically at high-profile individuals, such as senior executives and government officials, with the intent of authorizing massive financial wire transfers or exposing classified intellectual property.

Despite extensive research and multi-layered defense strategies involving server-side filters, machine learning classifiers, and Domain-based Message Authentication (DMARC), a completely foolproof prevention method remains elusive.

2.2 Machine Learning and Deep Learning

Machine Learning is a specialized branch of AI that utilizes mathematical models to enable computers to automatically learn patterns from data without explicit programmatic instructions [9]. The origins of ML trace back to 1959, though modern applications have been catalyzed by the exponential growth in computational power and data availability [11]. ML paradigms are generally divided into three categories:

1. **Supervised Learning:** Algorithms are trained on input datasets with predefined, labeled outputs. In phishing detection, algorithms like Logistic Regression and Support Vector Machines (SVM) have historically shown strong performance [1].
2. **Unsupervised Learning:** Algorithms analyze unlabeled datasets to discover hidden structural patterns through clustering (e.g., K-means, PCA), anomaly detection, or dimensionality reduction.
3. **Reinforcement Learning (RL):** An agent learns to make sequential decisions by interacting with an environment, receiving rewards for correct actions and penalties for incorrect ones [8].

2.3 Natural Language Processing (NLP)

NLP bridges linguistics and artificial intelligence to enable computers to interpret and generate human language. Early systems from the 1940s relied on manually crafted dictionaries and rigid rules for basic machine translation [10]. The field has evolved drastically; modern NLP is dominated by deep learning models that excel at machine translation, sentiment analysis, and generation. The seminal paper "Attention is All You Need" introduced the Transformer

architecture, which fundamentally shifted NLP away from recurrent models. The Transformer utilizes an encoder-decoder structure composed of stacked self-attention layers. Prominent architectures derived from this include Generative Pretrained Transformer (GPT) models for text generation and BERT (Bidirectional Encoder Representations from Transformers) for classification and understanding [4].

2.4 Literature Review

Prior research has demonstrated the difficulty of coercing commercial LLMs into generating malicious content. Studies found that while major platforms like ChatGPT and Claude have stringent filters, some platforms like DeepAI complied more easily with requests to generate phishing templates. In evaluating models against AI-generated phishing, prior studies showed that algorithms like Naive Bayes and LSTM could achieve high accuracy (up to 99.3%) when properly trained. Notably, AI-generated emails were found to possess significant stylistic deviations from human text, such as higher pronoun and verb frequencies, and longer average word lengths [5]. Regarding the BERT architecture, multiple studies have validated its efficacy in email security. Research comparing Convolutional Neural Networks (CNNs), Long Short-Term Memory (LSTM) networks, Recurrent Neural Networks (RNNs), and BERT consistently placed BERT at the top echelon of accuracy [2, 12, 6].

3 Dataset Selection and Data Analysis

3.1 Data Sources

The human-crafted datasets (phishing and legitimate) were sourced from a widely recognized curated repository that aggregates emails from seven distinct sources, totaling 203,176 email instances [3]. To represent the novel threat landscape, the AI-generated phishing dataset comprising 865 emails created via the DeepAI platform was utilized [5].

3.2 Phase I Dataset Engineering

For the Phase I binary classification task, 50,000 emails were randomly sampled for both human phishing and human legitimate categories from the curated repository, producing a balanced training corpus of 100,000 email instances.

3.3 Phase II Dataset Engineering

For the Phase II three-way classification task, it was imperative that the human and AI datasets were structurally comparable. The average word count for the

DeepAI generated emails was identified as 75.13 words. To standardize the data, human phishing and human legitimate emails falling within 1.5 standard deviations of this average word count were strategically filtered. DeepAI emails were categorized using regular expressions targeting common phishing themes, such as financial fraud (identifiable by terms like 'bank', 'credit', and 'money'). This systematic filtering produced a highly balanced and rigorously selected Phase II dataset consisting of 711 AI-generated phishing emails, 711 human phishing emails, and 711 human legitimate emails. Because initial filtering left a deficit of credential-harvesting themes in the human phishing dataset, the corpus was dynamically updated by extracting additional finance-themed human phishing emails to achieve thematic parity with the DeepAI dataset. The Phase II fine-tuning datasets were held entirely separate from the Phase I training corpus to ensure no data leakage between the binary and multi-class classification stages.

3.4 Data Analysis and Statistical Significance

The Mann-Whitney U test is a non-parametric statistical method uniquely suited for analyzing datasets with skewed, non-normal distributions, which is typical of textual features like sentence counts and dependency depths. To strictly control the false positive rate across multiple feature comparisons, the Benjamini-Hochberg False Discovery Rate (FDR) correction was applied, establishing an alpha threshold of 0.05 for statistical significance. The statistical analysis confirmed that AI-generated phishing emails differ significantly from their human counterparts. Notably, AI-generated emails exhibited higher Flesch-Kincaid Grade levels, implying a more sophisticated and complex vocabulary usage, while simultaneously registering lower Flesch Reading Ease scores.

4 Methodology

4.1 Phase I: Binary Classification Architecture

4.1.1 LSTM-CNN Hybrid Model

A hybrid network was constructed to leverage the spatial pattern recognition of Convolutional Neural Networks (CNN) and the sequential dependency mapping of Long Short-Term Memory (LSTM) networks. LSTMs are a specialized variant of recurrent neural networks designed to combat the vanishing gradient problem, allowing the model to remember long-term context within a paragraph .

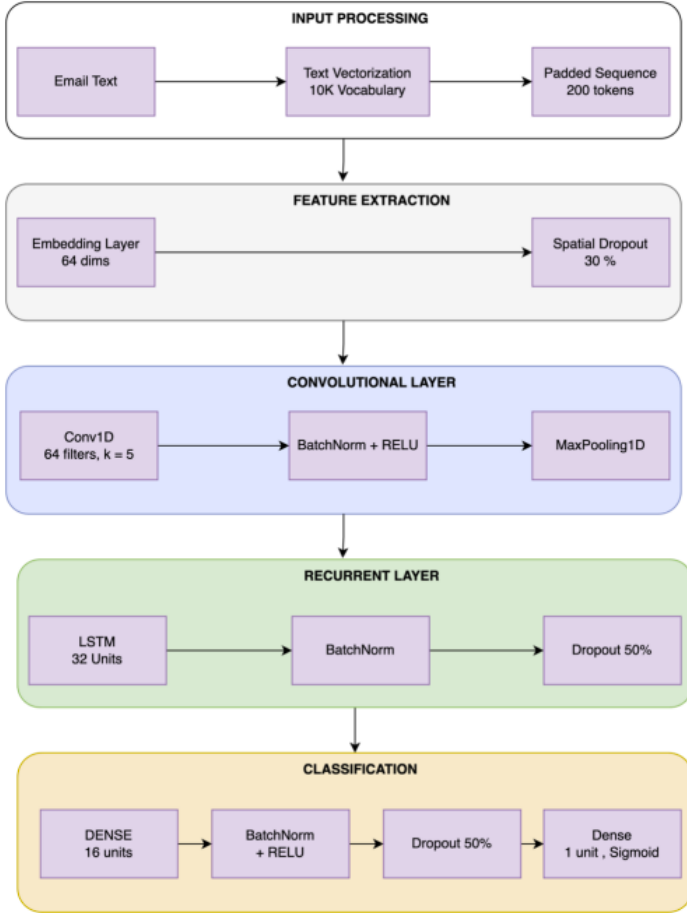


Figure 1: Phase I - LSTM-CNN Model

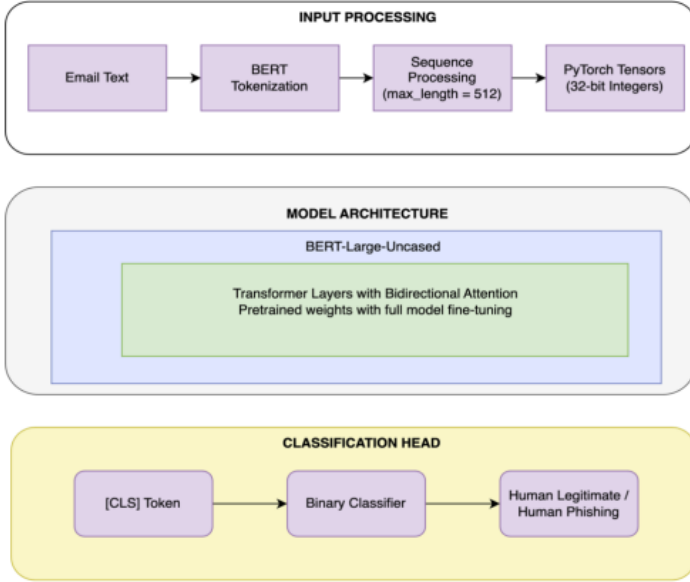


Figure 2: Phase I - BERT Model

The raw email text was aggressively vectorized into numerical sequences bounded by a fixed vocabulary size of 10,000 words and strictly padded or truncated to a standardized sequence length of 200 tokens. An Index Validator layer was implemented to guarantee that no token indices exceeded the defined vocabulary boundaries. The input sequences were fed into a dense embedding layer regulated by L2 regularization. To mitigate overfitting and reduce interdependencies between feature channels, a spatial dropout layer was strategically applied. During training, data augmentation was introduced by randomly masking 10% of the input tokens to artificially induce noise and enhance the model's robustness against adversarial text variations.

4.1.2 BERT Large Uncased

Simultaneously, the BERT Large Uncased model was fine-tuned for the identical binary classification task.

BERT's massive pre-trained parameter space provides it with a profound, context-aware understanding of the English language. A classification head was appended to the token output of the final transformer layer, and the model was optimized to distinguish between the two human categories.

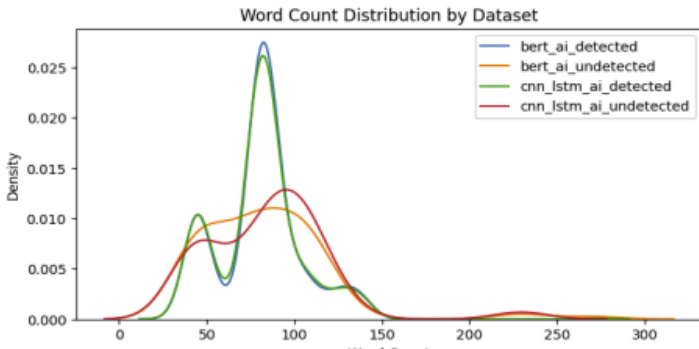


Figure 3: Error Analysis - Word Count Distribution

4.2 Phase II: Three-Way Classification and Fine-Tuning

Phase II expanded the challenge by integrating the AI-generated phishing dataset, requiring the models to perform three-way classification. Adapting a pre-trained model to a completely new class geometry requires careful hyper-parameter tuning through the gradual freezing and unfreezing of deep neural layers.

5 Results and Discussion

5.1 Phase I: Baseline Performance and Error Analysis

Both the hybrid LSTM-CNN and the BERT Large Uncased models demonstrated exceptional performance on the baseline binary classification task. The LSTM-CNN model achieved an accuracy of 98%, while the BERT model reached an accuracy of 99% when differentiating between human-crafted phishing and legitimate emails. However, the core research question centered on whether these highly accurate models could generalize to detect zero-day, AI-generated threats. When the trained baseline models were exposed to the unseen dataset of 711 AI-generated phishing emails, the results were concerning. Consequently, both models misclassified at least 10% of the highly sophisticated AI-generated phishing emails as completely legitimate, representing a massive security vulnerability in real-world deployment scenarios. An exhaustive error analysis was conducted specifically on the subset of AI-generated emails that successfully bypassed the filters.

The analysis revealed that the undetected AI emails systematically lacked the high-pressure "urgency" vocabulary that heavily weights binary classifiers

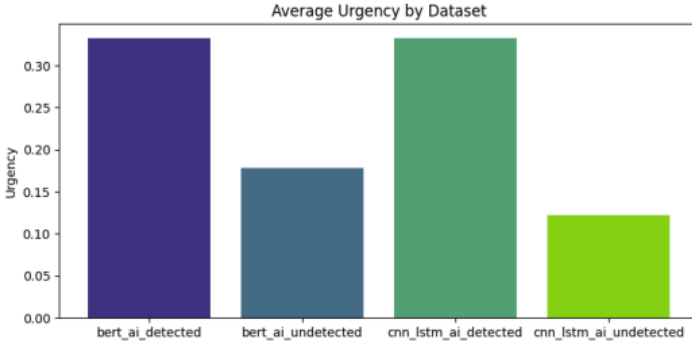


Figure 4: Error Analysis - Average Urgency

toward a "phishing" prediction.

5.2 Phase II: Multi-Class Adaptation Performance

5.2.1 LSTM-CNN Multi-Class Failure

The hybrid LSTM-CNN model exhibited catastrophic failure when forced to expand its latent space to accommodate three classes. The highest multi-class accuracy achieved by the LSTM-CNN architecture was an inadequate 48.61%.

5.2.2 BERT's Superior Adaptability

In profound contrast, the BERT model demonstrated remarkable plasticity and learning capacity. The optimal configuration was discovered through strategic layer freezing: by freezing the initial 16 layers (layers 0-15) to preserve foundational language representations and unfreezing the final 8 layers (layers 16-23) for fine-tuning, the BERT model achieved a phenomenal three-way classification accuracy of 99.48%.

6 Conclusion

As the accessibility of Generative AI and Large Language Models accelerates, the barrier to entry for executing highly sophisticated, polymorphic phishing campaigns continues to plummet. This research demonstrates a clear and present danger: machine learning defense systems trained exclusively on human-generated data are highly susceptible to AI-generated attacks. Our Phase I binary evaluation showed that while traditional LSTM-CNN and BERT

models can easily catch human-crafted phishing with up to 99% accuracy, they allow over 10% of AI-generated phishing to pass undetected into user inboxes. The error analysis confirms that AI-generated phishing circumvents traditional heuristic detection by employing advanced vocabulary and minimizing overt urgency, thereby mimicking legitimate professional correspondence. By selectively unfreezing the deeper layers of a pre-trained BERT Large Uncased model, we achieved an extraordinary 99.48% accuracy across human legitimate, human phishing, and AI-generated phishing categories. This study establishes that deep Transformer models, properly fine-tuned through layer-specific freezing, offer a highly resilient and adaptable defense mechanism capable of securing digital communication channels against the next generation of artificially intelligent cyber threats.

References

- [1] S. Alkatheeri et al. "Classification of Phishing Webpages using Supervised Machine Learning Algorithms". In: *2024 15th Annual Undergraduate Research Conference on Applied Computing (URC)*. Dubai, 2024, pp. 1–6.
- [2] S. Atawneh and H. Aljehani. "Phishing Email Detection Model Using Deep Learning". In: *Electronics* 12.20 (2023), p. 4261. DOI: 10.3390/electronics12204261.
- [3] A. I. Champa, M. F. Rabbi, and M. F. Zibran. "Curated Datasets and Feature Analysis for Phishing Email Detection with Machine Learning". In: *2024 IEEE 3rd International Conference on Computing and Machine Intelligence (ICMI)*. Mt Pleasant, MI, USA, 2024, pp. 1–7. DOI: 10.1109/ICMI60790.2024.10585821.
- [4] J. Devlin et al. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Minneapolis, MN, USA, 2019, pp. 4171–4186.
- [5] C.S. Eze and L. Shamir. "Analysis and Prevention of AI-Based Phishing Email Attacks". In: *Electronics* 13.10 (2024), p. 1839. DOI: 10.3390/electronics13101839.
- [6] B. Gogoi and T. Ahmed. "Phishing and Fraudulent Email Detection through Transfer Learning using pretrained transformer models". In: *IEEE 19th India Council International Conference (INDICON)*. Kochi, India, 2022, pp. 1–6.

- [7] R. Gozalo-Brizuela and E. C. Garrido-Merchán. “ChatGPT is not all you need. A state of the art review of large generative AI models”. In: *arXiv preprint arXiv:2301.04655* (2023), pp. 1–22.
- [8] S. Gu et al. “A Review of Safe Reinforcement Learning: Methods, Theories, and Applications”. In: *IEEE Trans. Pattern Anal. Mach. Intell.* 46.12 (2024), pp. 11216–11235.
- [9] C. Janiesch, P. Zschech, and K. Heinrich. “Machine learning and deep learning”. In: *Electronic Markets* 31.3 (2021), pp. 685–695.
- [10] D. Khurana et al. “Natural language processing: state of the art, current trends and challenges”. In: *Multimedia Tools and Applications* 81.1 (2022), pp. 3714–3720.
- [11] B. Mahesh. “Machine Learning Algorithms - A Review”. In: *International Journal of Science and Research (IJSR)* 9.1 (2020).
- [12] D. O. Otieno, A. Siami Namin, and K. S. Jones. “The Application of the BERT Transformer Model for Phishing Email Classification”. In: *IEEE 47th Annual Computers, Software, and Applications Conference (COMPSAC)*. Torino, Italy, 2023, pp. 1303–1310. doi: 10.1109/COMPSAC57700.2023.00198.
- [13] N. H. Patil, S. H. Patel, and S. D. Lawand. “Research Paper On Artificial Intelligence And It’s Applications”. In: *Journal Of Advanced Zoology* 44.S8 (2023), pp. 229–238.

Infusing AI Literacy Across Disciplines Using a ChatGPT-Based POGIL Module*

Rebecca A. Zulli¹, Adrienne A. Smith¹,
Sambit Bhattacharya², Bogdan Czejdo²,
Xiaochen Hu³, Khalid Lodhi⁴ and Zahra Shekarkhar³

¹Cynosure Consulting, USA

{adrienne.ann.smith, rebecca.anne.zulli}@gmail.com

²Department of Mathematics and Computer Science

³Department of Criminal Justice

⁴Department of Biology and Forensic Science

Fayetteville State University

Fayetteville, NC 28301, USA

{sbhattac, bczejdo, xhu, klodhi, zshekark}@uncfsu.edu

Abstract

Artificial Intelligence (AI) is integrating into daily communication, entertainment, and business sectors. As AI continues to evolve, the necessity for a comprehensive understanding of its capabilities, limitations, and ethical considerations has never been more critical. This study evaluates a two-session, ChatGPT-based Process-Oriented Guided Inquiry Learning (POGIL®) module designed to introduce undergraduate students to core AI concepts, including applications, processes, limitations, and trustworthiness. The intervention was implemented in three disciplinary contexts—computer science (CS), forensic science (FS), and criminal justice (CJ)—and assessed using a pre-post Foundational Knowledge of AI instrument. Statistically significant differences across time were found in each of the three areas of foundational knowledge of AI (knowledge of AI, process of AI, and impact of AI) for both computer

*Copyright ©2026 by the Consortium for Computing Sciences in Colleges. Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the CCSC copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Consortium for Computing Sciences in Colleges. To copy otherwise, or to republish, requires a fee and/or specific permission.

science and non-computer science students. The findings from this study underscore the substantial benefits of a singular educational module in promoting AI literacy across various disciplines.

Keywords: AI literacy, Generative AI, ChatGPT, Undergraduate computer science education, Process-Oriented Guided Inquiry Learning (POGIL).

1 Introduction

Artificial intelligence (AI) is transforming nearly every domain of modern life, including communication, healthcare, finance, environmental sciences, criminal justice, and education. Applications ranging from generative language models to autonomous vehicles have moved AI from a specialized technical field into an embedded component of everyday decision-making. As AI systems become more accessible through intuitive interfaces and publicly available tools, individuals across disciplines are no longer passive observers of AI development but active users and evaluators of AI-generated outputs.

In higher education, this rapid expansion presents both opportunity and responsibility. The increasing presence of AI tools such as ChatGPT in academic environments has altered how students write, research, and problem-solve [12], [9]. While these systems offer potential to enhance learning and productivity, they also introduce risks related to misinformation, bias, overreliance, and ethical misuse. As AI continues to shape professional and civic life, institutions must ensure that students develop a foundational understanding of how AI systems function, where their limitations lie, and how their outputs should be critically evaluated. Although computing programs have traditionally addressed AI from a technical perspective, the democratization of AI tools necessitates broader AI literacy efforts that extend beyond computer science (CS). Students in fields such as forensic science (FS), criminal justice (CJ), business, healthcare, and education will increasingly encounter AI-driven systems in their professional contexts. Consequently, AI literacy must be understood as an interdisciplinary imperative.

1.1 Importance of the Proposed Work

Scholarly and policy conversations increasingly emphasize the need to integrate AI education across educational levels. Casal-Otero et al. (2023) [2], in a systematic review published in the *International Journal of STEM Education*, argue that AI literacy is essential for preparing citizens to navigate social and economic systems shaped by AI technologies. Touretzky et al. (2019) [13], through the AI for K–12 initiative, propose foundational “Big Ideas” intended to guide national AI curriculum development. Similarly, Wing’s (2006) [16]

concept of computational thinking highlights the cross-disciplinary relevance of algorithmic reasoning and data-informed problem solving.

At the postsecondary level, the rapid advancement of generative AI intensifies these concerns. Tools such as ChatGPT enable the automated generation of essays, analyses, and responses across nearly any topic. While such tools may support learning when used responsibly, they also complicate assessment practices and challenge traditional notions of authorship and originality. More importantly, their outputs may appear authoritative despite being incomplete, biased, or contextually inaccurate.

These developments underscore the need for instructional models that not only expose students to AI systems but also cultivate critical evaluation skills. Faculty must help students understand how AI models are trained, how probabilistic outputs are generated, and why issues such as bias, lack of contextual awareness, and susceptibility to adversarial manipulation arise. In this context, AI literacy becomes not simply a matter of technical familiarity but a matter of informed and ethical engagement.

However, integrating AI literacy into non-CS curricula presents practical challenges. Courses are already densely structured, and instructors may lack the technical background to design AI-focused content independently. There is therefore a need for structured, scalable, and adaptable instructional models that computing faculty and interdisciplinary collaborators can implement without displacing existing course objectives.

1.2 Defining Foundational Areas of AI Knowledge

Efforts to conceptualize AI literacy have produced several complementary frameworks. Long and Magerko (2020) [8] synthesize interdisciplinary research to define core AI competencies, emphasizing understanding of training data, algorithms, limitations, and human-AI interaction. Faruqe et al. (2021) [4] propose a competency model aligned with workforce preparation, advocating structured progression from consumer-level awareness to creator-level proficiency. Allen and Kendeou (2023) [1] introduce the ED-AI Lit framework, which highlights knowledge, evaluation, contextualization, autonomy, collaboration, and ethics as essential components.

These models collectively suggest that AI literacy encompasses conceptual knowledge, process awareness, and ethical evaluation. Building on this literature, the present study organizes foundational AI knowledge into three domains:

1. Knowledge of AI – awareness of AI concepts, applications, and personal intersections with AI systems.
2. Process of AI – understanding of how AI systems function, including

data-driven learning, probabilistic output generation, and algorithmic limitations.

3. Impact of AI – awareness of societal implications, ethical considerations, and discipline-specific applications.

This structure provides a coherent framework for both instructional design and empirical assessment. By operationalizing foundational AI knowledge across these domains, the study aims to measure changes in students’ conceptual understanding following a targeted instructional intervention.

1.3 Pedagogical Approaches Aligned to Developing Foundational AI Knowledge

Although AI literacy frameworks are expanding, there remains a gap between conceptual models and classroom implementation strategies. Much of the existing literature focuses on programmatic initiatives, workshops, or K–12 interventions [7], [10], with fewer empirically evaluated studies examining short, scalable modules suitable for integration into existing undergraduate courses.

Process-Oriented Guided Inquiry Learning (POGIL®) offers a promising approach. POGIL® emphasizes collaborative learning through structured activities in which students construct knowledge by responding to sequenced questions. Research in computing education indicates that POGIL® can enhance conceptual understanding and increase course success rates [6], [5], [14].

Guided inquiry is particularly well suited to AI literacy development because it promotes active evaluation rather than passive reception. When applied to generative AI tools, guided inquiry activities can prompt students to interrogate outputs, identify inconsistencies, and examine underlying assumptions. Such engagement aligns closely with the goal of fostering critical, informed AI users.

1.4 Testing a Research-Based Best Practice Approach

The present study employs a design-based research framework [11], [15] to develop and evaluate a two-session ChatGPT-based POGIL® module. The intervention was designed to introduce foundational AI concepts while foregrounding issues of trustworthiness, bias, contextual misunderstanding, and limitations of machine learning systems. The module was implemented in three undergraduate contexts:

- A computer science (CS) course
- A forensic science (FS) course

Session #1			Session #2		
Lecture Discus- sions and Video	POGIL® Discus- sion and Practice	Homework and Video	Session 1 Review	Team- based POGIL® Activity	Team- based Submis- sion and Report- ing
40 minutes	10 minutes	2 hours	10 minutes	35 minutes	5 minutes

Table 1: The Intervention Plan. Session #1 is Lecture & Class Discussion and Session #2 is Team Project

- A criminal justice (CJ) course

This cross-disciplinary implementation allowed examination of how students with differing levels of technical background responded to structured AI exposure.

1.5 Module Description and Intervention Design

The intervention was intentionally limited to two consecutive class meetings to ensure feasibility and replicability.

Session 1 combined interactive lecture, discussion, and multimedia examples. Students explored AI applications including AI-generated images and self-driving cars, discussed distinctions between machine learning algorithms and rule-based systems, and examined the role of data in shaping model performance. The session also introduced generative AI and principles of AI trustworthiness, including transparency, accountability, fairness, and reliability.

Session 2 consisted of a team-based POGIL® activity centered on a discipline-relevant theme: “Laws for Collecting DNA Samples from People Who Are Arrested.” Students used ChatGPT to investigate the topic, compared AI-generated responses to expectations for academic research, and critically analyzed issues of bias, accuracy, and contextual adequacy.

Throughout the module, generative AI was positioned not as a productivity tool but as an object of inquiry. The instructional design sought to balance conceptual understanding with evaluative skills, supporting the development of foundational AI literacy across disciplinary contexts. See Table 1 for more details on the session timelines.

1.6 Research Questions

This study examined the impact of a ChatGPT-based POGIL® module on students' foundational AI knowledge across three undergraduate populations: computer science (CS), forensic science (FS), and criminal justice (CJ) students.

The following research questions guided the study:

RQ1: To what extent does exposure to the ChatGPT POGIL® module improve students' knowledge of AI?

RQ2: To what extent does exposure to the module improve students' understanding of the process of AI?

RQ3: To what extent does exposure to the module improve students' awareness of the impacts and applications of AI?

2 Methods

A within-subjects pre-post design was used to evaluate the impact of the ChatGPT POGIL® module. The intervention was implemented in three undergraduate courses representing different disciplinary contexts: a CS course (high AI alignment), an FS course (STEM-related), and a CJ course (humanities-oriented).

Students completed a Foundational Knowledge of AI instrument immediately before and after participating in the two-session module (contact author for instrument). Pre- and post-scores were compared within each discipline using paired-samples t-tests to determine statistically significant changes in AI knowledge.

2.1 Measures

To assess learning outcomes, items were aligned to the Foundational Knowledge of AI framework, organized into three domains (See Figure 1):

1. Knowledge of AI
 - Self-assessment of AI knowledge
 - Awareness of personal intersections with AI
2. Process of AI
 - Awareness of how AI systems function
3. Impact of AI

- Awareness of societal impacts
- Awareness of discipline-specific applications (FS or CJ)

Survey items used Likert-scale responses. Composite scores were calculated for each construct by summing relevant items and normalizing by total possible points. All composites demonstrated acceptable reliability

(Cronbach's $\alpha > .80$)

Paired-samples t-tests were conducted to evaluate pre-post differences within each disciplinary group.

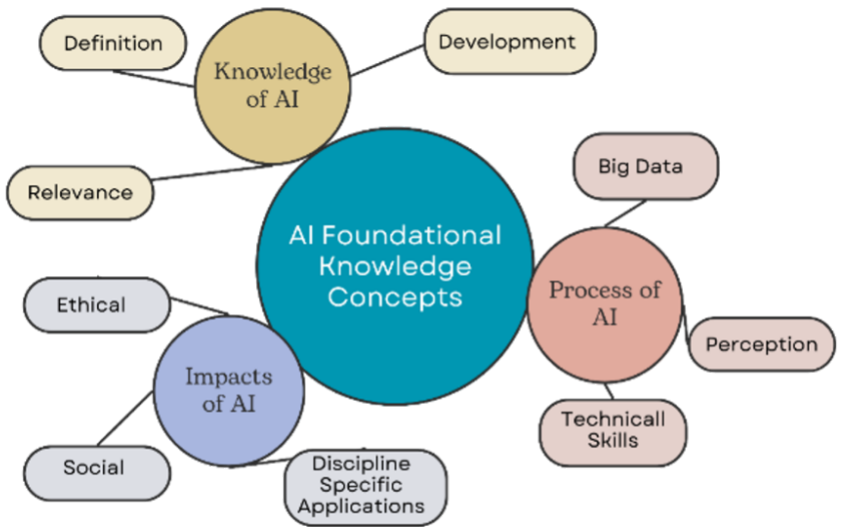


Figure 1: Foundational Knowledge of AI Framework.

3 Results

Composite scores for all constructs demonstrated acceptable internal reliability

(Cronbach's $\alpha > .80$)

Paired-samples t-tests were conducted separately for CS, FS, and CJ students to evaluate pre-post changes following exposure to the ChatGPT POGIL® module.

3.1 Computer Science (CS)

There were 16 students in the CS course with pre- and post-ChaptGPT module data. Eighty percent of the students identified as male, and the 20 percent identified as female. Students were asked about their racial identity. Multiple responses were permitted. Half of the students were Black, 44 percent were White, 13 percent were Hispanic or Latino(a), and six percent were Asian or Asian American. All of the students were high school juniors who planned to major in CS and had already taken a CS course. Three quarters had taken a course where they had learned something about AI and nearly all had hoped to take courses in AI in the future.

Two statistically significant improvements were observed:

- Self-assessed AI knowledge increased from $M = 3.44$ ($SD = 0.68$) to $M = 4.02$ ($SD = 0.86$), $t(15) = -3.3$, $p < .05$, with a large effect size ($d = 0.82$).
- Awareness of AI applications within criminal justice increased from $M = 3.42$ ($SD = 0.65$) to $M = 3.85$ ($SD = 0.75$), $t(15) = -2.3$, $p < .05$, with a medium effect size ($d = 0.59$).

No statistically significant changes were found for awareness of AI processes, general impacts, or applications within forensic science.

3.2 Forensic Science (FS)

There were 20 students in the FS course, across two sections (20 in the face-to-face section and 23 in the online section), with pre- and post-ChaptGPT module data. Seventy percent of the students were female, 20 percent were male, and 10 percent identified as non-binary. Students were asked about their racial identity. Multiple responses were permitted. Sixty-five percent of the students were Black, 25 percent were White, and 25 percent were Hispanic or Latino(a). All students were college seniors, some in their fifth year, who were majoring in FS.

Forty percent are first generation college students. A little under half had taken a course in CS, but only 20 percent have taken a course where they had learned something about AI. Before the course, 60 percent of students hoped to take courses in AI in the future.

Although mean scores increased descriptively for self-assessed AI knowledge ($M = 2.51$ to 2.84), no statistically significant pre-post differences were observed across any construct.

3.3 Criminal Justice (CJ)

There were 44 students in the CJ course, across two sections (20 in the face-to-face section and 23 in the online section), with pre- and post-ChaptGPT module data. Sixty-six percent of the students were female, 27 percent were male, two percent were non-binary, and five percent preferred not to answer. Students were asked about their racial identity. Multiple responses were permitted. Sixty-eight percent of the students were Black, nine percent were White, 14 percent were Hispanic or Latino(a), seven percent were American Indian or Alaskan Native, two percent were another race, and seven percent preferred not to answer. Students were spread across class levels (see Table 2).

Nearly all of the students had declared a major in CJ. Over a quarter are first generation college students. A little over half had taken a course in CS, but only 11 percent have taken a course where they had learned something about AI. Before the course, 63 percent of students hoped to take courses in AI in the future.

Four statistically significant improvements were found:

- Self-assessed AI knowledge increased from $M = 2.52$ ($SD = 0.75$) to $M = 3.01$ ($SD = 0.99$), $t(42) = -5.2$, $p < .05$, large effect ($d = 0.80$).
- Awareness of the AI process increased from $M = 2.10$ ($SD = 0.95$) to $M = 2.65$ ($SD = 1.12$), $t(41) = -4.8$, $p < .05$, medium effect ($d = 0.70$).
- Awareness of AI impacts increased from $M = 2.90$ ($SD = 1.05$) to $M = 3.18$ ($SD = 1.08$), $t(42) = -2.2$, $p < .05$, small-to-medium effect ($d = 0.30$).
- Awareness of AI applications within CJ increased from $M = 2.65$ ($SD = 1.15$) to $M = 3.06$ ($SD = 1.02$), $t(42) = -2.4$, $p < .05$, medium effect ($d = 0.40$).

3.4 Summary of Findings

Significant gains were observed in foundational AI knowledge for students in CS and CJ courses, with effect sizes ranging from medium to large. Improvements were strongest in self-assessed AI knowledge and discipline-specific AI application awareness. No statistically significant changes were detected among FS students. See Table 3.

These findings suggest that a short, structured ChatGPT-based POGIL® intervention can meaningfully increase AI literacy, particularly in computing-aligned and humanities-oriented contexts.

Class Level	Percent of Participants
High School Junior	2
High School Senior	0
College Freshman	0
College Sophomore	9
College Junior	32
College Senior	5
Did not answer	2

Table 2: Table 2. Class Level Distribution of Participating Students in CJ Class

4 Discussion

This study examined the impact of a short, structured ChatGPT-based POGIL® module on foundational AI knowledge across CS, forensic science (FS), and criminal justice (CJ) courses. Results indicate that even a two-session intervention can produce meaningful gains in AI literacy, particularly in computing-aligned and humanities-oriented contexts.

Significant improvements were observed among CS and CJ students in self-assessed AI knowledge, with additional gains for CJ students in understanding AI processes, impacts, and discipline-specific applications. Effect sizes ranged from medium to large, suggesting that structured, guided interaction with generative AI can substantially improve students’ perceived understanding of AI concepts. While FS students showed descriptive gains, no statistically significant changes were detected in that group.

For computing educators, these findings have several implications.

First, the results reinforce the role of CS faculty in leading AI literacy efforts across campus. The module demonstrates that computing-informed instructional design—grounded in guided inquiry and explicit discussion of AI limitations and trustworthiness—can be successfully infused into non-CS courses without requiring extensive curricular overhaul. The intervention was intentionally constrained to two class sessions, suggesting a scalable and adaptable model for institutions seeking to integrate AI exposure broadly.

Second, the differential effects across disciplines highlight the importance of contextualization. Gains were strongest when AI applications were directly aligned with students’ disciplinary interests (e.g., AI in criminal justice). This suggests that AI literacy initiatives are likely to be most effective when computing faculty collaborate with domain instructors to tailor examples and inquiry activities to authentic disciplinary problems.

Third, the integration of generative AI within a structured pedagogical

Discipline	Construct	Pre M (SD)	Post M (SD)	t(df)	Effect Size (d)	Sig.
CS (n=16)	Self-assessment of AI knowledge	3.44 (0.68)	4.02 (0.86)	-3.3 (15)	0.82	*
	Awareness of AI process	3.34 (0.68)	3.56 (0.86)	ns	—	
	Awareness of AI impacts	4.29 (0.59)	4.27 (0.71)	ns	—	
	AI applications – FS	3.34 (0.58)	3.40 (0.70)	ns	—	
	AI applications – CJ	3.42 (0.65)	3.85 (0.75)	-2.3 (15)	0.59	*
FS (n=20)	Self-assessment of AI knowledge	2.51 (0.82)	2.84 (0.87)	ns	—	
	Awareness of AI process	2.08 (1.02)	2.11 (0.97)	ns	—	
	Awareness of AI impacts	3.27 (1.13)	3.24 (1.05)	ns	—	
	AI applications – FS	3.11 (0.67)	3.06 (0.83)	ns	—	
CJ (n=44)	Self-assessment of AI knowledge	2.52 (0.75)	3.01 (0.99)	-5.2 (42)	0.80	*
	Awareness of AI process	2.10 (0.95)	2.65 (1.12)	-4.8 (41)	0.70	*
	Awareness of AI impacts	2.90 (1.05)	3.18 (1.08)	-2.2 (42)	0.30	*
	AI applications – CJ	2.65 (1.15)	3.06 (1.02)	-2.4 (42)	0.40	*

Table 3: Table 3. Pre-Post Changes in Foundational AI Knowledge by Discipline. *Note.* $p < .05$. *ns* = not statistically significant. *Effect sizes reported as Cohen's d [3]. Reliability for al* exceeded $\alpha = .80$.

framework (POGIL®) appears to mitigate the risk of superficial tool use. Rather than positioning ChatGPT as a productivity shortcut, the module emphasized evaluation, trustworthiness, and limitations of AI systems. This approach supports the development of informed and critical AI users—an increasingly important goal for computing education.

While the study is limited by sample size and reliance on self-reported measures, the consistent improvements in CS and CJ courses provide preliminary evidence that guided, discipline-aware AI modules can meaningfully enhance foundational AI knowledge. For computing educators, this work offers a replicable model for cross-disciplinary AI integration that is feasible, scalable, and aligned with emerging expectations for AI literacy in undergraduate education.

References

- [1] L. K. Allen and P. Kendeou. “ED-AI Lit: An interdisciplinary framework for AI literacy in education”. In: *Policy Insights from the Behavioral and Brain Sciences* (2023).
- [2] L. Casal-Otero et al. “AI literacy in K-12: A systematic literature review”. In: *International Journal of STEM Education* 10.1 (2023), p. 29.
- [3] J. Cohen. *Statistical power analysis for the behavioral sciences*. Lawrence Erlbaum Associates, 1988.
- [4] F. Faruqe, R. Watkins, and L. Medsker. “Competency model approach to AI literacy: Research-based path from initial framework to model”. In: *Advances in Artificial Intelligence and Machine Learning* 2 (2021), pp. 580–587.
- [5] H. H. Hu et al. “Results from a survey of faculty adoption of process-oriented guided inquiry learning (POGIL) in computer science”. In: *Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education*. 2016, pp. 186–191.
- [6] C. Kussmaul, H. H. Hu, and M. Lang. “Guiding students to discover CS concepts and develop process skills using POGIL”. In: *Proceedings of the 45th ACM Technical Symposium on Computer Science Education*. 2014, p. 745.
- [7] I. A. Lee et al. “Developing middle school students’ AI literacy”. In: *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education*. 2021.
- [8] D. Long and B. Magerko. “What is AI literacy? Competencies and design considerations”. In: *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*. 2020. DOI: 10.1145/3313831.3376727.

- [9] S. Marche. *The college essay is dead*. <https://www.theatlantic.com/technology/archive/2022/12/chatgpt-ai-writing-college-student-essays/672371/>. The Atlantic, December 10, 2022. 2022.
- [10] D. Ng et al. “Using digital story writing as a pedagogy to develop AI literacy among primary students”. In: *Computers & Education: Artificial Intelligence* 3 (2022), p. 100054.
- [11] E. Oh and T. C. Reeves. “The implications of the differences between design research and instructional systems design for educational technology researchers and practitioners”. In: *Educational Media International* 47.4 (2010), pp. 263–275.
- [12] OpenAI. *ChatGPT: Optimizing language models for dialogue*. <https://openai.com/blog/chatgpt/>. November 30, 2022. 2022.
- [13] D. Touretzky et al. “Envisioning AI for K-12: What should every child know about AI?” In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 33. 2019, pp. 9795–9799.
- [14] L. E. Walker and A.-R. M. Warfa. “Process-oriented guided inquiry learning (POGIL®) marginally affects student achievement measures but substantially increases the odds of passing a course”. In: *PLoS ONE* 12 (2017).
- [15] F. Wang and M. J. Hannafin. “Design-based research and technology-enhanced learning environments”. In: *Educational Technology Research and Development* 53.4 (2005), pp. 5–23.
- [16] J. M. Wing. “Computational thinking”. In: *Communications of the ACM* 49.3 (2006), pp. 33–35.

From Dorm Room to Deployment: A Faculty-Mentored, Student-Led Case Study in Early Undergraduate Computing Education*

Augustus Scarlato¹, Ryan Ouardaui², Oybeck Ablakulov³

¹Department of Computer Science

Stetson University

DeLand, FL, USA

`ascarlato@stetson.edu`

² Columbia University

New York, NY USA

`rao2148@columbia.edu`

³ Vanderbilt University

Nashville, TN, USA

`oybek.ablakulov@vanderbilt.edu`

Abstract

This experience report presents a multi-semester case study of a mobile application designed and developed by two first-year computer science students to address the challenge of making meaningful friendships in new environments. Motivated by personal experiences as international students navigating social isolation and digital fatigue, the students proposed an “anti-social media” concept centered on privacy-first interest-based matching. Over the course of Fall 2024 through Summer 2025, and guided by weekly formal faculty mentorship, the students built a React Native application with Supabase backend services and Mapbox integration, eventually forming a legal entity to pursue public release.

*Copyright ©2026 by the Consortium for Computing Sciences in Colleges. Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the CCSC copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Consortium for Computing Sciences in Colleges. To copy otherwise, or to republish, requires a fee and/or specific permission.

This paper documents the technical and educational journey of this student-led initiative guided by formal faculty mentorship, highlighting key development milestones, project pivots, and pedagogical insights. We describe the structured mentoring model used to support the project, challenges encountered in sustaining momentum outside the classroom, and the learning outcomes that emerged through authentic, self-directed software development. Finally, we reflect on how this model can inform broader practices in computing education, especially for supporting entrepreneurial learning and real-world impact among undergraduates.

1 Introduction

1.1 Real World Need

Developing mobile applications has become a crucial need in addressing real-world educational needs, specifically within Computer Science degrees. The integration of mobile app development into educational curricula is rooted deeply in many computer science programs across the world. However, this curriculum is often in 300 to 400 level courses when students have a deeper programming foundation. As a result, students spend much of their time learning two to three new programming languages before they ever start their first group project. The importance of early group work with mobile applications in higher education cannot be overstated, as they provide a dynamic platform for delivering instructions and enhancing student learning outcomes [1].

1.2 Project Origin

Students often come into colleges and universities, bringing unique perspectives and experiences that enrich the learning environment and foster collaboration. However, it is extremely rare that first-year students have a strong enough base in software development in order to embark on full mobile app development and run through software development lifecycles. In Fall 2024, two Computer Science freshmen seeking a deeper challenge during their introductory course approached their instructor for an additional learning opportunity.

1.3 Context

A project rooted in solving a key problem for international students sparked the ideation of an anti-social media mobile application allowing like-minded individuals to connect in person. This need arose when two international students met at orientation and felt overwhelmed in a new country, a new state, and a new school. Essentially, entering a whole new world physically and psychologically. Starting in the Fall of 2024 and continuing formally with weekly men-

torship sessions into the summer of 2025, two international computer science students with faculty mentorship created an RC (Release Candidate) mobile application, which is set to launch on the Apple mobile platform.

1.4 Paper Goals

This paper shares the journey of a multi-semester, student-led, faculty-mentored initiative that challenged conventional expectations about the capabilities of first-year students when supported with intentional mentorship, real-world problems, and autonomy. We share the project’s technical and educational evolution, including the structure of mentorship, development sprints, challenges encountered, and successes achieved. The key purpose of this paper is to provide an adaptable model that educators and institutions can use to empower undergraduates to create a real-world impact from day one. We invite you to review, add, remove, and adapt this model for your own.

2 Related Work

2.1 App Landscape

There are a small number of mobile applications geared to enable students to be more engaged when in new geographic areas; however, they focus on broad network expansion rather than connecting like-minded individuals in person [2]. Similar applications, such as Yubo, Wizz, and Bumble BFF, focus on allowing individuals to expand their friendship networks. These apps often come with privacy issues and allow individuals to connect regardless of shared interests.

2.2 Computing Education Parallels

There is a strong body of work supporting project-based learning and early exposure to real-world software development in computer science education [3]. Prior research highlights how hands-on, applied projects increase student motivation, engagement, and retention. Integrating consumer technologies, such as mobile applications, into educational activities has been found to boost student enjoyment and reduce attrition [4]. Research shows that mobile platforms can effectively support student learning in computing courses. For example, the SIGCSE TS 2023 experience report Integrating Accessibility in a Mobile App Development Course observed that teaching accessibility concepts alongside mobile development improved student technical skills and inclusive design awareness [5]. This case study extends that model by combining agile development practices with student-driven innovation, which is rarely seen at the first-year level. Other works support the idea of allowing high school students

exposure to mobile app development to inspire a career in tech [6]. Some reports show how engaging high schoolers with Smart and Connected Technologies can lead to awareness and skills in technology [7]. This study focuses on the first-year college students who have recently exited the high school environment.

2.3 Contribution Relevance

This project contributes to ongoing efforts within undergraduate computing education to provide earlier, more authentic experiences in software development. While upper-level capstone courses often support project-based learning, fewer models exist for first-year students to lead mentor-supported, real-world projects outside of formal coursework. By documenting a multi-semester, student-driven initiative that integrates agile practices, privacy-first design, and entrepreneurial thinking, this case study offers a replicable example for institutions seeking to expand early-stage engagement.

3 Project Design

3.1 Team Structure

The project team comprised two first-year undergraduate students from diverse international backgrounds with previous coding experience in the C programming language. In addition, one faculty member with experience in mobile app development and entrepreneurship served as the mentor, guiding the students through the technical and business aspects of the project. Formal weekly sessions occurred throughout the entire project.

3.2 Timeline

Beginning in the Fall of 2024, the team engaged in whiteboarding sessions focused on ideation, feature planning, wire framing, and UI design using Figma. In Spring 2025, development accelerated with the kickoff of weekly scrum meetings, burn-down charts, and full-stack implementation using React Native, Supabase, and Google Maps integration. Work continued into the Summer of 2025 with an emphasis on business planning, beta launch preparation, and the formal establishment of a Limited Liability Company.

The project evolved across three terms, following a structured, milestone-driven development process summarized in Table 1.

Table 1: Development Process Timeline

Term	Focus Areas	Key Activities
Fall 2024	Ideation & UI Planning	- Whiteboarding, feature planning, and wireframes using Figma - Selection of tech stack (React Native, Supabase, Mapbox)
Spring 2025	Full-Stack Development & Agile Sprints	- Weekly Scrum meetings and sprint planning - Frontend/backend implementation - Integration of Mapbox API and OpenAI matching
Summer 2025	Business Strategy & Deployment Preparation	- Formation of LLC - Final testing and polish - Beta launch prep and App Store submission - Branding and outreach work

3.3 Technical Decisions

Prioritization around user privacy and control was one of the first key decision-making processes that allow users to fully customize their profiles, including usernames, profile pictures, and bios. Empowering the end user with the ability to share what they choose was a key guiding principle. Personally identifiable information, such as phone numbers or email addresses, is only accessible at the point of a mutual friendship establishment within the app. Personal posts are similarly restricted to being visible only to friends, reinforcing a private social environment.

Friendship matching is at the core of the mobile application and uses a live Open AI API integration to pair users based on selected interest tags and regional proximity, both with customizable weighting to reflect user preferences. All personal information is restricted from entering the OpenAI API. First contact interactions occur in anonymized chatrooms where neither usernames nor profile pictures are visible, preserving user privacy until both users agree to connect. All confidential information, including messages, profile pictures, email addresses, location, and password information, is securely encrypted and stored using Supabase.

3.4 Mentorship Format

Weekly formal check-ins, milestone goals, and product sprints enabled the team to make the best of the mentor/student dynamic. The structure allowed students to follow a process, fail fast, and take calculated risks aligned with the best practices. This mentorship was hands-on and provided students with a safe space to ask questions, discuss roadblocks, and iterate on solutions. The professor would ask guiding questions, offer advice, and engage resources to help students overcome challenges. This was a vital step to ensuring the students remained engaged in their learning journey, as they were already engaged in a challenging first-year academic environment.

3.5 Assessment Methods

In order to maintain continuous feedback, weekly check-ins consisted of demo checkpoints, GitHub code reviews, and progress logs. Professor-led code reviews enabled code restructuring for best practices and future scalability. Using a common agile methodology, the team adopted a burn-down chart methodology to track progress against goals. The assessment was ongoing, allowing for adjustments to be made as the project evolved, ensuring that the final product met the goals of the project and the needs of future end users.

4 Reflection

4.1 Successes

Within the application, technical success can be interpreted based on the app components that worked as intended from the start. One of these successes is the matching algorithm, which takes the gathered user interest from an engaging onboarding process and then matches it with other users. The use of artificial intelligence, although limited, allows for a more advanced matchmaking due to user biographies being compared and their meaning being compared to each other for similarities. Other successes include the privacy-oriented chat rooms, which automatically generate a random username for every new person that the user messages after matching, while only sharing personal information after mutual consent.

From a learning perspective, student success was achieved in multiple ways, with the most tangible evidence demonstrated in the app itself, which continued to operate as a Release Candidate (RC) for the Apple App Store at the conclusion of the Summer 2025 development period. Students were able to create something from scratch, solve a problem, and develop a tangible working product. Informal feedback was gathered through weekly mentoring sessions, where students regularly reflected on their progress, discussed technical decisions, and identified areas for improvement. In addition, other CS students and faculty heard of the project's progress and requested office hours and meetings to discuss how this approach could be incorporated into curricula beginning in Fall 2025.

4.2 Challenges

The most significant challenge that we experienced during the creation of the app was deciding which parts of the app to cut from the final version. One of the larger cuts was the initially planned trip-making process that could be used to plan trips manually or with the help of AI. Even though it was developed

fully, it was scrapped due to the high costs of both the AI token consumption and the data needed per location. A major reason was also due to how it did not fully relate to the main concept of the app, which was to make friends, and for a simple and effective user experience. Therefore, it was decided not to include it in the release candidate stage.

Table 2 summarizes key development milestones, highlighting where progress aligned with original plans and where adjustments were made due to technical or time constraints.

Table 2: Key Milestones and Development Progress

Milestone	Planned Timeline	Actual Outcome	Key Insights
Matching Engine (AI)	Week 6	Completed Week 7	Required deeper tuning and prompt engineering for match quality
Anonymous Chatrooms	Week 9	Completed on schedule	Early win; reinforced privacy-by-design strategy
Notification Features	Week 12	Delayed (post-launch)	Supabase limitations; deferred to post-beta roadmap
UI Polish & Testing	Week 13	Completed Week 14	Extra iteration improved usability
Trip Planner Feature	Week 11	Built but cut	Dropped due to AI cost, scope misalignment, and unclear user value

As shown in Table 2, the team navigated a range of technical challenges and design trade-offs. While some features were delivered as planned, others required adjustments, rescoping, or difficult cut decisions to maintain a coherent user experience.

Some technical difficulties included notification issues due to fundamental server-side aspects not being available in Supabase. This resulted in some elements of notifications, like replying through the notifications, being held off until after launch. Additionally, APIs like Google Maps were switched in favour of Mapbox due to the high costs of the API calls. Other more straining challenges were dealing with both TypeScript and JavaScript in the main parts of the application, which required additional effort when transferring data between processes.

Throughout the project, students ran into roadblocks with API integration and agile practices that had to be overcome to meet their milestone goals. Without the direct mentorship and consistent formal check-ins, these foreign concepts, normally taught in 300 and 400 level classes, challenged the students to stay on task in a structured way. Many times, they simply wanted to have the freedom to develop and change items at will. The often-repeated term “trust the process” was stated by the professor, gently guiding them back on

track.

“Around Week 10, students began self-managing sprint goals with little prompting – a major sign of maturity and buy-in to the process.” – Faculty Mentor

5 Pedagogical and Institutional Takeaways

5.1 Formal Mentorship Model

This project confirms how a structured, credit-free mentorship model supports student-led innovation [8]. Weekly one-hour session with a faculty mentor coupled with consistent deliverables such as problem solving, planning, and progress updates. These check-ins created accountability for the students while allowing full ownership of the project.

Unlike the course curriculum, the mentorship was framed as a collaborative partnership. The mentor fulfilled the role of coach rather than manager/scrumb master, offering technical feedback and progress without truly directing the work. With no formal grading, this model continuously engaged students across multiple semesters and helped navigate the same challenges they would face in an upper-level software development course [8].

Faculty looking to replicate this approach may consider adapting it to an honors project, students with a dual computer science major and business minor, or the start of a research apprenticeship.

5.2 Learning Outcomes

5.2.1 Technical fluency

An important item learned was not to attempt to build the entire application as a single program at once. Although the students did this from the start due to convenience instead of planning their workflow, it was one critical element in the examination of the app, helping it speed up the process significantly. Even if it meant they had challenges integrating each individual part.

Another key item learned was to stick with one language for the entirety of the program unless it is required for a unique feature. Due to starting off with both TypeScript and JavaScript, it required more time commitments in the long run to make sure all data was corrected when transferring to different file types. Although this should have been foreseeable, due to how each language was better for different parts of the application, they started using both languages. Retrospectively, they should have and could have aligned to one and only one language.

5.2.2 Communication skills

As working professionals, the linchpin of success often can be pointed to communication, which was a key factor in the success of this project. From the start, both student team members focused on continuous communication and transparency to ensure nothing was hidden from one another, which built trust even when they had only met one month prior.

By treating the work as a collaborative effort rather than a competitive one, they avoided key conflicts that are often seen in industry software development. This reinforced an important lesson: presenting a bold but focused vision of transparency early on helped foster trust within the team.

5.2.3 Product thinking

During the ideation process, the team realized the market for friend-making apps was well saturated with platforms relying on profile-based matching and personal data sharing. To differentiate, the focus aligned to privacy and interest-based connections, designed via the matching algorithm for prioritization, shares interests rather than surface-level profiles.

Their goal was to create an authentic, user-controlled experience that challenged the scrolling-driven algorithmic nature of traditional social media applications.

5.3 Institutional Lessons

While the genesis of this project emerged from a unique combination of student initiative and prior experience, it does highlight the potential value of introducing a group-based project and real-world software development earlier in the CS curriculum. Early exposure to the verbiage, processes (such as Agile), and tools can help ensure students build an early confidence muscle before upper-level coursework.

"Empowering us to lead out on our passion project and get top-level mentoring gave us a strong sense of confidence, knowing we can do this in the real world." – Student participant

5.4 Actionable Advice

It is important to launch structured, interest-driven side projects with students. As many students seek education, they ultimately align with the end goal of landing a real-world job post-graduation. The more circumstances that we can align them to, the better, while also fulfilling academic requirements.

Institutions should build bridges between Computer Science education and Business entrepreneurship, as this alignment in many colleges and universities

is lacking. This collaboration could take the form of joint courses or structured feedback loops, with business students acting as beta testers, particularly international students who may share the transitional experiences that inspired this project.

Before outlining opportunities for replication and extension, we briefly clarify the scope and intended contribution of this work. This work is presented as an experience report rather than a controlled empirical study. The primary contribution lies in documenting a replicable, multi-semester mentorship structure that supported early-stage undergraduate students in navigating authentic software development challenges. Evidence for the outcomes described is artifact-based and longitudinal, including the developed mobile application, source code, deployment milestones, and reflective observations gathered through sustained faculty mentorship. Rather than aim for statistical generalizability, this case study emphasizes transferability by outlining the process, constraints, and design decisions that can be adapted by educators seeking to support a student-led innovation beyond formal coursework.

6 Replicability and Future Work

6.1 Curriculum Pilot

This course concept could be adapted into an immersive, project-based learning experience that encourages creative thinking beyond the traditional classroom setting. Small teams of entry-level students could collaboratively build a simplified prototype of the mobile application. Students would first outline user privacy requirements, discuss data handling, and encryption. In addition, they would design mock user profiles, pull templated anonymized chatroom code, and align to predefined AI matching flows guided by faculty mentors.

Teams could split into front-end and backend logic for focused planning prior to a full software development course. Throughout this process, the instructor would facilitate discussions on privacy, best practices, high-level weekly scrum meetings, and encryption standards to be used. The course would be graded on documentation, team contribution, and a working build of the pilot application.

The pilot could align with two computer science seniors (from a CS honors program or CS club) and three first-year students, thus a team of five, creating opportunities for cross-pollination, peer mentoring, and collaborative learning.

6.2 Future Work

The following replication and future work are described as they were anticipated at the conclusion of the Summer 2025 development period.

Future work includes completing Apple Store deployment and expanding the application to the Android marketplace via the Google Play Store, as well as exploring potential collaboration with an internal school of business for testing and structured feedback loops. In addition, our university had expressed support for aligning the Fall 2025 new student orientation with testing of the mobile application.

As envisioned, during student onboarding a 5 x 7 postcard with a brief description, application images, and a QR code would be distributed out for app download and profile creation. This feedback loop was intended to support server load testing, bug reporting, feature enhancement, and post-interaction surveys. The approach simulates a paid beta testing environment commonly used in industry, while remaining zero cost and highly contextualized to a campus community. Because the app targets students in transition, the model offers a repeatable, scalable feedback loop that can be incorporated in future new student orientation cycles for ongoing product refinement.

References

- [1] D. Truong. “How to Design a Mobile Application to Enhance Teaching and Learning?” In: *International Journal of Emerging Technologies in Learning (iJET)* 9.3 (2014), p. 4. DOI: 10.3991/ijet.v9i3.3507.
- [2] A. Hinze et al. “A Study of Mobile App Use for Teaching and Research in Higher Education”. In: *Technology, Knowledge and Learning* 28.3 (2022), pp. 1271–1290. DOI: 10.1007/s10758-022-09599-6.
- [3] M. Prince and R. Felder. “A Review of Research on Problem-Based Learning”. In: *Journal of Engineering Education* 95.2 (2006), pp. 123–138. DOI: 10.1002/j.2168-9830.2006.tb00884.x.
- [4] V. Cateté, K. Wassell, and T. Barnes. “Use and Development of Entertainment Technologies in After-School STEM Programs”. In: *Proceedings of the 45th ACM Technical Symposium on Computer Science Education (SIGCSE)*. 2014, pp. 163–168. DOI: 10.1145/2538862.2538952.
- [5] J. S. Bhatia et al. “Integrating Accessibility in a Mobile App Development Course”. In: *Proceedings of the 54th ACM Technical Symposium on Computer Science Education (SIGCSE 2023)*. 2023, pp. 910–916. DOI: 10.1145/3545945.3569797.
- [6] J. E. Burge et al. “Girls on the Go: Engaging Middle School Girls in Computer Science Through Mobile Application Development”. In: *Proceedings of the 44th ACM Technical Symposium on Computer Science Education (SIGCSE)*. 2013, pp. 615–620. DOI: 10.1145/2445196.2445376.

- [7] M. W. Rutherford et al. *Promoting Rural Entrepreneurship through Technology: A Case Study using Productivity Enhancing Technology Experience Kits (PETE-Kits)*. 2023. DOI: 10.48550/arxiv.2311.04327. arXiv: 2311.04327.
- [8] D. Lopatto. “Undergraduate Research Experiences Support Science Career Decisions and Active Learning”. In: *CBE Life Sciences Education* 6.4 (2007), pp. 297–306. DOI: 10.1187/cbe.07-06-0039.

When Discomfort Teaches: Using Cognitive Dissonance to Develop Adversarial Thinking in AI-Driven Computing Courses*

Christian Servín, Nadia V. Karichev, and Ivan Alonso

¹ El Paso Community College,
El Paso, TX, USA

¹{cservin1/nmerzlya/ialonso4}@epcc.edu

Abstract

The growing use of artificial intelligence (AI) tools and technologies in computing education offers powerful support for code generation, explanation, and problem solving, but it also introduces new pedagogical challenges. One such challenge is cognitive dissonance, the discomfort students experience when AI-generated artifacts appear correct despite limited understanding of their underlying logic or assumptions. This work argues that cognitive dissonance should not be treated as a barrier to learning, but rather as a pedagogical signal that can be leveraged to cultivate adversarial thinking in computing education. This work presents an instructional framework that positions adversarial thinking as a professional disposition, encouraging students to question AI outputs, examine hidden assumptions, and evaluate failure modes, security risks, and ethical implications. Through AI-supported programming and analysis activities, students are guided to transform uncertainty and discomfort into systematic verification and critical reasoning. This approach reframes AI from a convenience tool into a catalyst for deeper learning, accountability, and robust problem solving, preparing students to engage responsibly with AI-enabled computing systems.

*Copyright ©2026 by the Consortium for Computing Sciences in Colleges. Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the CCSC copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Consortium for Computing Sciences in Colleges. To copy otherwise, or to republish, requires a fee and/or specific permission.

1 Introduction

Artificial intelligence (AI) tools are increasingly embedded in computing education, supporting code generation, explanation, debugging, and problem solving, particularly in introductory courses (e.g., [1, 2, 8, 19]). These tools can lower barriers to entry and provide timely support for novice learners. However, their growing use also introduces a pedagogical tension: students may successfully produce correct and sophisticated solutions without fully understanding the underlying logic, assumptions, or design decisions embedded in those solutions.

This tension frequently manifests as *cognitive dissonance* [6, 7], a state of discomfort that can arise when students value rigor and correctness yet rely on AI systems whose reasoning is opaque and difficult to inspect. Learners may trust an AI-generated artifact while simultaneously feeling uncertain about their own comprehension or accountability for the outcome.

Rather than treating this discomfort as a learning failure, we argue that cognitive dissonance can serve as a productive instructional signal. When surfaced intentionally, it reveals gaps between perceived and actual understanding and highlights misplaced trust or unexamined assumptions. Building on this insight, we present an instructional approach that integrates *adversarial thinking* into AI-supported computing education. Adversarial thinking encourages students to question assumptions, explore failure modes, consider edge cases or misuse, and critically evaluate AI-generated artifacts rather than accepting them at face value (e.g., [15]). This work is informed by a grant-funded initiative at a U.S. two-year community college where adversarial thinking has been infused into CS0, CS1, and CS2.

2 Background and Motivation

Computing education has long emphasized the alignment between effort, authorship, and understanding. Traditional pedagogies assume that students who design algorithms and write code develop mental models of program behavior. Generative AI disrupts this alignment by enabling students to produce plausible and often correct artifacts through prompts, sometimes without explicit engagement in the reasoning required to justify them.

In practice, this shift introduces recurring forms of cognitive dissonance: confidence without comprehension, responsibility without agency, and plausibility without verification. While prior work has examined abstraction, automation, and tool use, the cognitive tension created by AI-mediated workflows remains underexplored in early undergraduate contexts. At the same time, adversarial thinking has emerged as a *critical professional disposition*

in cybersecurity, secure software development, and responsible AI (see [14]). It emphasizes anticipating misuse, questioning assumptions, and considering worst-case scenarios—skills that align naturally with moments of dissonance. This paper builds on the premise that cognitive dissonance is not a barrier to be minimized, but a signal that can be intentionally structured to support adversarial thinking in computing education.

Related Work

Recent computing education research documents the growing impact of generative AI in introductory programming [13]. These studies show that AI systems can generate correct and complete solutions from minimal prompts, raising concerns about overreliance and reduced engagement with underlying logic when artifacts are accepted without critical evaluation. Subsequent work emphasizes pairing AI use with verification and validation practices, as students may submit functional code without understanding correctness, security, or robustness [5]. Other analyses caution that fluent and plausible AI outputs can obscure misconceptions [9]. Complementary studies frame interaction with AI systems as a skill that must be explicitly taught, including prompt-based programming activities [4]. Collectively, this literature motivates instructional approaches that surface cognitive dissonance and leverage it to promote verification habits and responsible AI use.

Two-Year Programs and Workforce Needs

Community colleges play a critical role in the computing education ecosystem by providing transfer pathways and workforce-oriented preparation aligned with employer needs [3, 16]. These institutions often respond quickly to regional labor demand through applied curricula and industry-aligned credentials. This role is increasingly important as agencies and industry shift toward skills-based hiring models in cybersecurity and AI-enabled roles, placing greater emphasis on demonstrated competencies than on degree pedigree [11, 17, 18]. As a result, two-year programs serve as key contributors to a job-ready computing workforce.

Centers of Academic Excellence in Cyber Defense

The Centers of Academic Excellence in Cyber Defense increasingly emphasize interdisciplinary competencies that connect computing with broader domain knowledge. This trend is especially visible at two-year institutions that must align programs with evolving workforce demands. Related efforts at four-year

institutions often pursue similar goals via experiential learning pathways (internships, research, capstones) that prepare students for professional practice in AI- and cybersecurity-enabled environments.

Alignment with CS2023

This approach aligns with the *ACM/IEEE-CS/AAAI CS2023 Curricula guidelines* [10], particularly their emphasis on cross-cutting competencies beyond syntax and algorithms.

Critical Thinking and Problem Solving CS2023 emphasizes analyzing solutions, evaluating trade-offs, and reasoning about correctness. Adversarial thinking operationalizes these goals by prompting students to question AI-generated artifacts and test robustness.

Ethics, Responsibility, and Professional Practice The guidelines emphasize accountability and ethical responsibility. By making authorship and responsibility explicit in AI-assisted work, students are encouraged to reflect on professional obligations when using AI tools.

Security and Robustness as Cross-Cutting Concerns CS2023 treats security and robustness as themes that should appear throughout the curriculum. Adversarial thinking supports this goal by embedding assumption checking, misuse analysis, and failure exploration in introductory coursework.

Human–Computer Interaction and AI Literacy Understanding limitations, affordances, and risks of AI systems aligns with CS2023’s call for AI literacy. Students learn not only how to use AI tools, but when and why to distrust them.

3 Teaching Context: A Two-Year Community College Perspective

This work is informed by instructional practices at a U.S. two-year community college serving a diverse, predominantly first-generation student population (e.g., [16]). As part of a grant-funded initiative focused on infusing adversarial thinking, AI-supported activities were introduced into CS0, CS1, and CS2.

CS0 (Introductory Computing). AI tools are used primarily for sense-making rather than solution production. Students use AI to generate explanations, worked examples, and starter code for simple problems. Instruction

emphasizes identifying assumptions in problem statements, questioning AI explanations, and comparing expected versus observed behavior. Cognitive dissonance is surfaced through reflection prompts such as: *What do you trust about this output, and why? What assumptions might be hidden? What do you need to verify?*

CS1 (Introductory Programming). AI-generated code becomes the object of adversarial analysis. Students use AI to generate initial solutions and then evaluate them for correctness, robustness, and style. Coursework emphasizes edge-case testing, identifying logical errors, and examining potential security or misuse scenarios. Students revise AI-generated code to address weaknesses, with learning outcomes centered on critique, validation, and improvement rather than generation alone.

CS2 (Data Structures and Algorithms). Adversarial thinking is extended to data abstractions, algorithmic trade-offs, and larger program structures. Students analyze AI-generated implementations of data structures and algorithms, evaluating correctness, efficiency, and suitability under constraints. Cognitive dissonance often arises when AI-generated solutions appear correct but exhibit poor performance, scalability issues, or design flaws. Students resolve this tension through prompts focused on complexity, constraint testing, and abstraction boundaries.

Across the three courses, adversarial thinking is framed as a professional habit rather than a security-only skill. Early exposure in a two-year program prepares students for transfer pathways, workforce roles, and responsible engagement with AI-enabled computing systems.

4 Cognitive Dissonance and the Use of AI in Computing

The integration of AI into computing education introduces cognitive dissonance [12, 6] when students value understanding and authorship yet rely on AI-generated artifacts with opaque reasoning. In AI-enabled introductory courses, this tension becomes systemic as generative AI enables students to produce correct solutions through prompts without fully engaging with underlying assumptions. Students may submit working code they did not author and do not fully understand, leading to confidence without comprehension and reliance on plausible but unverified outputs.

Rather than treating this discomfort as failure, we argue it should be leveraged as a *learning signal*. When paired with adversarial thinking—questioning assumptions, exploring failure modes, and testing robustness—cognitive dissonance becomes a catalyst for deeper learning, ethical judgment, and responsible

participation in AI-enabled computing practice. Table 1 provides illustrative examples.

Dissonant Thought	Adversarial Question
“This works, but I don’t get it.”	What assumptions is the AI making?
“The AI says this is secure.”	How could this fail or be exploited?
“Everyone is using AI this way.”	What risks are being normalized?
“The answer sounds confident.”	What evidence supports this?

Table 1: Examples of cognitive dissonance and corresponding adversarial questions in AI-supported computing education.

5 Instructional Framework

We propose an instructional framework that treats *cognitive dissonance* as a pedagogical signal and *adversarial thinking* as the mechanism by which students resolve that dissonance in AI-supported computing education. AI tools are positioned as objects of inquiry rather than authoritative sources, and learning outcomes emphasize validation, critique, and justification over solution generation.

Framework Principles

AI as an object of inquiry. AI-generated artifacts are treated as provisional and inspectable rather than authoritative; students are expected to analyze and interrogate outputs.

Dissonance as signal. Uncertainty or discomfort is surfaced intentionally and framed as evidence of hidden assumptions or gaps in understanding.

Adversarial thinking as practice. Students engage in systematic questioning, including edge cases, misuse scenarios, failure modes, and security/ethical implications.

Verification over generation. Assessment prioritizes testing, empirical evaluation, and revision; credit is awarded for evidence-based critique and improvement.

Progressive sophistication. Adversarial reasoning is scaffolded across CS0–CS2: from sense-making and assumptions (CS0), to correctness and robustness (CS1), to efficiency and abstraction (CS2).

Instructional Cycle

Instruction follows an iterative cycle (Figure 1):

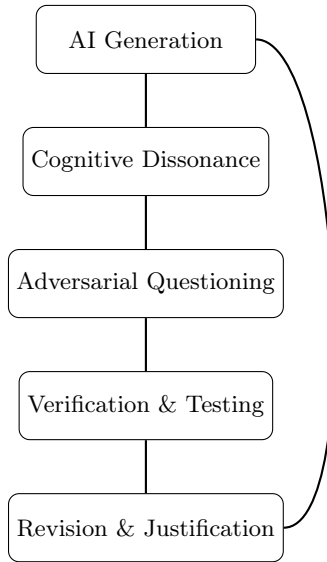


Figure 1: Iterative instructional framework leveraging cognitive dissonance to drive adversarial thinking in AI-supported computing education. Students generate, critique, verify, and revise AI-generated artifacts across CS0–CS2.

AI-supported construction: generate an explanation or initial solution appropriate to course level.

Dissonance surfacing: identify uncertainty about correctness, authorship, or understanding.

Adversarial questioning: examine assumptions, edge cases, misuse, and trade-offs.

Verification and stress testing: validate via tests, evaluation, or complexity analysis.

Revision and justification: revise artifacts and justify changes with evidence.

Course-Level Mapping

CS0: AI supports conceptual explanations and examples; students practice assumption checking and trust calibration.

CS1: AI-generated code is tested and revised; students practice robustness reasoning, edge-case testing, and accountability.

CS2: AI-generated algorithms are evaluated for complexity, scalability, and design; students practice evidence-based judgment under constraints.

6 Summary and Future Work

AI tools can reduce barriers for novice learners, but they also create a tension when students produce correct solutions without understanding the underlying logic, assumptions, or design decisions. We argue that this tension often manifests as *cognitive dissonance* and should be leveraged as a productive instructional signal rather than treated as a learning failure. We present an approach that integrates *adversarial thinking* into AI-supported computing education by guiding students to question assumptions, examine failure modes, consider misuse and edge cases, and validate AI-generated artifacts through evidence-based critique and revision. Grounded in a grant-funded initiative at a U.S. two-year community college, the approach has been infused across CS0–CS2 to promote deeper learning, responsible AI use, and professional habits aligned with modern computing practice.

Future work will evaluate this approach using mixed methods, combining analysis of student artifacts (tests, revisions, justifications), reflection responses, and performance measures across CS0–CS2. We also plan to develop instruments to characterize cognitive dissonance in AI-supported work and to assess growth in adversarial thinking as a professional disposition. Additional studies will examine longitudinal impacts on transfer readiness and workforce-relevant competencies, including verification practices, robustness reasoning, and AI literacy aligned with CS2023. Finally, we will develop instructor-facing supports (rubrics, prompt libraries, and professional development) to enable sustainable adoption in two-year programs.

References

- [1] Dennis J Bouvier et al. “GenAI Integration in Upper-Level Computing Courses”. In: *Proceedings of the 30th ACM Conference on Innovation and Technology in Computer Science Education V. 2*. ITiCSE 2025. Nijmegen, Netherlands: Association for Computing Machinery, 2025, pp. 691–692. ISBN: 9798400715693. DOI: 10.1145/3724389.3731276. URL: <https://doi.org/10.1145/3724389.3731276>.
- [2] Fernando Brito et al. “Enhancing Cybersecurity Education with Artificial Intelligence Content”. In: *Proceedings of the 56th ACM Technical Symposium on Computer Science Education V. 1*. SIGCSETS 2025. Pittsburgh, PA, USA: Association for Computing Machinery, 2025, pp. 158–164. ISBN: 9798400705311. DOI: 10.1145/3641554.3701958. URL: <https://doi.org/10.1145/3641554.3701958>.

- [3] Jill Denner, Paul Tymann, and Huihui Wang. “Community College Pathways”. In: *Proceedings of the 2023 CISE EWF PI Meeting*. Georgia Tech Conference Center, June 2023.
- [4] Paul Denny et al. “Using Prompt Problems to Teach Learners How to Prompt AI Code Generators”. In: *Proceedings of an ACM Computing Education venue (Promptly / Prompt Problems paper)*. Prompt-based programming exercises for AI code generators. 2023.
- [5] Amanda Fernandez and Kimberly A. Cornell. “CS1 with a Side of AI: Teaching Software Verification for Secure Code in the Era of Generative AI”. In: *Proceedings of the 55th ACM Technical Symposium on Computer Science Education (SIGCSE TS '24)*. 2024. DOI: 10.1145/3626252.3630817.
- [6] Leon Festinger. *A Theory of Cognitive Dissonance*. Stanford, CA: Stanford University Press, 1957.
- [7] Eddie Harmon-Jones. “A Cognitive Dissonance Theory Perspective on the Role of Emotion in the Maintenance and Change of Beliefs and Attitudes”. In: *Emotions and Beliefs: How Feelings Influence Thoughts*. Cambridge: Cambridge University Press, 2000, pp. 185–211.
- [8] Irene Hou et al. “The Evolving Usage of GenAI by Computing Students”. In: *Proceedings of the 56th ACM Technical Symposium on Computer Science Education V. 2*. SIGCSETS 2025. Pittsburgh, PA, USA: Association for Computing Machinery, 2025, pp. 1481–1482. ISBN: 9798400705328. DOI: 10.1145/3641555.3705266. URL: <https://doi.org/10.1145/3641555.3705266>.
- [9] Ishika Joshi et al. “ChatGPT in the Classroom: An Analysis of Its Strengths and Weaknesses for Solving Undergraduate Computer Science Questions”. In: *Proceedings of the 55th ACM Technical Symposium on Computer Science Education (SIGCSE TS '24)*. 2024. DOI: 10.1145/3626252.3630803.
- [10] Amruth N. Kumar et al. *Computer Science Curricula 2023*. New York, NY, USA: Association for Computing Machinery, 2024. ISBN: 9798400710339.
- [11] Nextgov Staff. “White House Looks to Eliminate College Degree Requirements for Cyber Jobs with Federal Contractors”. In: *Nextgov* (2024). URL: <https://www.nextgov.com/cybersecurity/2024/01/white-house-looks-eliminate-college-degree-requirements-cyber-jobs-federal-contractors/393329/>.
- [12] Donald A. Norman. *The Design of Everyday Things*. Revised and Expanded. New York, NY: Basic Books, 2013.

- [13] James Prather et al. “The Robots Are Here: Navigating the Generative AI Revolution in Computing Education”. In: *Proceedings of the 2023 ACM Conference on Innovation and Technology in Computer Science Education – Working Group Reports (ITiCSE-WGR ’23)*. Working Group Report. 2023.
- [14] Christian Servin. “Adversarial Thinking as a Professional Disposition for Computing Education”. In: *Proceedings of the 26th Annual ACM Conference on Cybersecurity and Information Technology Education (SIGCITE 2025)*. Sacramento, CA, USA: Association for Computing Machinery, Nov. 2025. DOI: 10.1145/3769694.3771130. URL: <https://doi.org/10.1145/3769694.3771130>.
- [15] Christian Servin and Nadia Karichev. “Adversarial Computing: Integrating Resilience and Security Into Foundational CS Education”. In: *Proceedings of the 26th Annual ACM Conference on Cybersecurity and Information Technology Education (SIGCITE 2025)*. Sacramento, CA, USA: Association for Computing Machinery, Nov. 2025. DOI: 10.1145/3769694.3771125. URL: <https://doi.org/10.1145/3769694.3771125>.
- [16] Christian Servin et al. “Curricular and Pedagogical Considerations in Computer Science Education: The Role of Community Colleges for the Next Decade”. In: *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1*. SIGCSE 2024. Portland, OR, USA: Association for Computing Machinery, 2024, pp. 1196–1201. ISBN: 9798400704239. DOI: 10.1145/3626252.3630819. URL: <https://doi.org/10.1145/3626252.3630819>.
- [17] U.S. Office of Personnel Management. *General Schedule Qualification Standards*. Many GS-4 positions require completion of two years of college or an associate degree. 2023. URL: <https://www.opm.gov/policy-data-oversight/classification-qualifications/general-schedule-qualification-standards/>.
- [18] USAJOBS. *Education Level Definitions*. USAJOBS defines an Associate Degree as a 2-year undergraduate degree at an accredited institution. 2023. URL: <https://www.usajobs.gov/Help/how-to/account/profile/education/>.
- [19] J.D. Zamfirescu-Pereira et al. “61A Bot Report: AI Assistants in CS1 Save Students Homework Time and Reduce Demands on Staff. (Now What?)” In: *Proceedings of the 56th ACM Technical Symposium on Computer Science Education V. 1*. SIGCSETS 2025. Pittsburgh, PA, USA: Association for Computing Machinery, 2025, pp. 1309–1315. ISBN: 979840070531 DOI: 10.1145/3641554.3701864. URL: <https://doi.org/10.1145/3641554.3701864>.

Introducing Network Hardening and Vulnerability Detection using CUDA Parallel Graph Algorithms in Computer Network Courses

Quoc-Nam Tran*

Southeastern Louisiana University, USA

Quoc-Nam.Tran@selu.edu

Abstract

Network courses have become popular in many graduate and undergraduate computer science curriculum. In these courses we usually give a comprehensive coverage of networks and communication concepts, algorithms, methodologies, management issues, and tools, which are illustrated through simple examples for an easy understanding and mastering of these materials. However, these courses often fail to provide students with opportunities to develop and evaluate applications in real-world environments. We fill these gaps by creating new materials in Network Hardening and Vulnerability Detection research which would open opportunities to develop and evaluate applications in real-world environments for the senior-level and graduate students in computer science. A network vulnerability detection framework which we designed for many-core GPUs makes use of CUDA as its parallel programming platform. The proposed approach enables continuous scalability for large graph processing beyond GPU memory capacity while supporting multiple GPUs for efficient routing operations across extensive networks.

In this paper, we show how new parallel computing techniques can be developed and used to improve the Network Hardening and Vulnerability Detection research problems. The outcomes of this research are used as new materials for students in our network courses.

*Copyright ©2026 by the Consortium for Computing Sciences in Colleges. Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the CCSC copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Consortium for Computing Sciences in Colleges. To copy otherwise, or to republish, requires a fee and/or specific permission.

Keywords: CUDA, Multi-core Parallel Algorithms, Graph Algorithms, Network Hardening, Vulnerability Detection.

1 Introduction

Network courses have been offered in many graduate and undergraduate computer science curriculum. However, in many undergraduate curriculum, especially at small schools, these courses usually present a comprehensive coverage of network concepts, algorithms, methodologies, management issues, and tools, which are illustrated through simple examples for an easy understanding and mastering of these materials, but often fail to provide students with opportunities to develop and evaluate applications in real-world environments. As a result, students are not as prepared as they should be for graduate study or careers in industry. We propose to fill the aforementioned gaps by developing new network algorithms for Network Hardening and Vulnerability Detection research which open the opportunities to develop and evaluate applications in real-world environments for the senior-level and graduate students in computer science. Specifically, we show how to develop new network algorithms for finding the shortest-path closure of a graph and use these new approach to improve the Network Hardening and Vulnerability Detection problems,

Traditional vulnerability detection methods together with network analysis face difficulties when dealing with current large and complex data environments. Modern networks with their numerous interconnected nodes and shifting routing patterns need faster computational techniques that can handle large-scale analysis. Our research develops an efficient parallel computing system which utilizes in-memory computing principles along with CUDA-enabled many-core GPUs to detect and analyze network vulnerabilities.

Network vulnerability detection requires identification of security weaknesses through which attackers can exploit unpatched software and poor protocol configuration and insecure routing mechanisms. Real-time analysis of large graphs faces performance challenges because conventional detection methods depend on sequential CPU processing and disk I/O-intensive frameworks. The combination of in-memory computing with massive parallel GPU architectures establishes a compelling framework for network vulnerability algorithms which require high speed and scalability.

In this paper, we present a parallel algorithm based on a sequential $O(\log n \cdot n^3)$ algorithm in Lawler (1976). By exploiting the global scope of the GPU's global memory and optimizing the use of the GPU's shared memory, we are able to achieve a large performance benefit with a speed-up of 12,000x on a desktop computer in comparison with a single core program in contrast to a speedup of 5-6x found in the literature. Our algorithm overcomes many challenges which

specifically arose from Floyd-Warshall based algorithms (Katz and Kider 2008; Bondhugula et al. 2006; Han, Franchetti, et al. 2006; Han and Kang 2005; Harish and Narayanan n.d.; Subramanian et al. 1993; Venkataraman et al. 2003; Prihozy 2023; Tardivo et al. 2023; Xia et al. 2022).

The paper is organized as follows: In Section 2, we present our Skeleton of Connectivity for Network Optimization Through Shortest-Path-Preserving Subgraphs. In Section 3, we present our Shortest-Path Preservation and how to Illuminate Network Vulnerabilities. In Section 4, we present the design, analysis and experimental results of our new parallel algorithms for finding the Shortest-Path-Preserving Subgraphs on a large graph.

2 The Skeleton of Connectivity: Network Optimization Through Shortest-Path-Preserving Subgraphs

In an era defined by sprawling digital infrastructures, colossal social networks, and complex logistical systems, the networks that underpin our world are becoming increasingly intricate. This complexity, while often a source of robustness, presents significant challenges for analysis, computation, and operational efficiency. A powerful yet elegant strategy to confront this challenge lies in a fundamental concept from graph theory: the extraction of a minimal subgraph that preserves all-pairs shortest path distances. This technique, which distills a network down to its essential connective skeleton, offers a profound method for optimization and simplification without sacrificing critical information about the system's core functionality.

The core premise is intuitive yet mathematically rigorous. Given a graph G (composed of nodes and edges), the goal is to find a subgraph G' that contains the same set of nodes but a significantly reduced set of edges. The crucial constraint is that for any two nodes, the shortest path distance between them in G' must be identical to their shortest path distance in the original graph G . This is not merely about maintaining connectivity; it is about preserving the exact "cost," be it in time, distance, or latency, of the most efficient routes between all points in the network.

The theoretical foundation for constructing such a subgraph often rests on the concept of Shortest-Path Trees and the related Transitive Reduction of a graph with edge weights. For each node in the network, one can compute a Shortest-Path Tree (SPT)—a minimal tree rooted at that node that contains a shortest path to every other node. The union of these SPTs for all nodes is a subgraph that, by definition, preserves all-pairs shortest paths. However, this union is not necessarily minimal. The truly minimal subgraph, sometimes called the shortest-path-preserving kernel, is a further refinement where an edge (u, v) is only retained if it is the unique shortest path between u and

v, or if it is necessary to provide a shortest path for another pair of nodes. Algorithms to compute this often involve checking, for each edge, whether its removal would force a detour, thereby increasing the shortest path distance between its endpoints.

The applications of this simplification technique are vast and transformative. In internet routing, protocols like OSPF (Open Shortest Path First) require routers to maintain a map of the network. By computing and using a shortest-path-preserving subgraph, routers can drastically reduce the amount of topological data they must store and process, leading to faster convergence and reduced overhead, while guaranteeing that packets are still routed along optimal paths. Similarly, in transportation and logistics, a company modeling its distribution network can remove redundant roads or links that do not contribute to the shortest routes between hubs. This simplifies route-planning algorithms, reduces memory footprint for simulations, and provides a clearer visual representation of the most critical arteries in the supply chain.

Furthermore, this method serves as a powerful tool for network analysis and insight. By stripping away redundant edges, the minimal subgraph reveals the true architectural backbone of the system. Analysts can immediately identify critical choke points and vital bridges—edges whose failure would necessarily increase the distance between nodes. This is invaluable for vulnerability assessments in critical infrastructure, from power grids to financial networks. The simplified graph is not just a computational convenience; it is a lens that focuses attention on the most structurally important components.

However, the pursuit of minimalism is not without its trade-offs. The primary limitation is the loss of path diversity. While the shortest distance is preserved, the number of alternative shortest paths may be reduced. In scenarios where redundancy and fault tolerance are paramount, such as in robust communication networks, removing these alternative paths obscures the system's inherent resilience. A network might have five different paths of the same minimal length; the preserving subgraph may retain only one. Consequently, while the subgraph is optimal for shortest-path distance, it may be suboptimal for metrics like network flow or survivability.

The extraction of a minimal subgraph that preserves all-pairs shortest path distances is a cornerstone technique in the optimization and simplification of complex networks. It operates on the profound principle that much of a network's complexity is redundant for the specific task of efficient routing. By meticulously pruning everything but the essential connections that define optimal distances, it yields a streamlined, computationally efficient, and analytically clear representation of the original system. While mindful of its trade-offs concerning redundancy, this approach provides an indispensable strategy for managing the sprawling connectivity of the modern world, allowing us to see

the fundamental skeleton of a network and, in doing so, to optimize its function with unparalleled clarity.

3 The Unmasked Backbone: Using Shortest-Path Preservation to Illuminate Network Vulnerabilities

In the perpetual contest between network defenders and potential adversaries, clarity is a strategic asset. Complex, meshed networks—whether digital, physical, or logistical—often obscure their own critical weaknesses beneath layers of redundant connections. A powerful analytical approach to cut through this complexity and expose systemic fragility is the construction of a minimal subgraph that preserves all-pairs shortest path distances. Far from being a mere optimization technique, this method serves as a potent diagnostic tool, revealing the hidden skeleton of a network’s connectivity and unmasking its most critical points of failure for targeted security and vulnerability analysis.

The foundational principle is one of radical simplification. By algorithmically removing every edge that is redundant for maintaining the shortest possible routes between all nodes, we distill the network down to its most essential connective tissue. The resulting subgraph is not a diminished version of the original; it is a precise map of its operational core. In this simplified landscape, the importance of each remaining edge is magnified. An edge that appears to be one of many in a dense web is revealed, in the minimal subgraph, to be an irreplaceable bridge, the sole guardian of the optimal path between critical nodes.

This process directly enables a rigorous identification of critical choke points. In the minimal subgraph, every remaining edge is, by definition, vital for preserving at least one shortest-path distance. The failure of any such edge guarantees an increase in the distance—and thus a degradation in performance—between at least one pair of nodes. For a security analyst, this provides a mathematically defensible priority list for protection. These edges represent prime targets for an attacker seeking to maximally disrupt network efficiency with minimal effort. For instance, in a nation’s internet backbone, identifying these links allows for the hardening of physical infrastructure and the implementation of heightened cyber-defenses at the most consequential locations. The analysis shifts vulnerability assessment from an intuitive, experience-based practice to a quantifiable, algorithmic one.

Furthermore, this methodology excels at modeling cascade failures and strategic attacks. Sophisticated adversaries do not always attack randomly; they follow a strategy. By analyzing the minimal subgraph, one can simulate targeted attacks. An attacker with this map would know precisely which handful of links to sever to cause the greatest possible increase in overall net-

work latency or to isolate key segments. Defenders can run these simulations proactively, asking not only "what is the most critical link?" but also "what is the second most critical link once the first has fallen?" This step-by-step deconstruction of the minimal backbone models a coordinated assault, revealing chains of vulnerability and allowing for the design of contingency plans and redundant pathways that exist outside the "optimal" shortest-path structure.

However, the very clarity of the minimal subgraph also reveals a fundamental paradox in network design: the tension between efficiency and resilience. A network optimized purely for shortest-path performance, as embodied by this minimal subgraph, is inherently fragile. Its elegance is its weakness. The process of minimization strips away the redundant paths that provide robustness. A real-world network must therefore be evaluated against its minimal form. The greater the divergence—the more redundant paths that exist—the more resilient the network is to random failures or targeted attacks. Security through obscurity is not a valid strategy, but security through redundancy is. The minimal subgraph provides the baseline against which this redundancy can be measured and justified.

The use of a minimal shortest-path-preserving subgraph transforms the abstract concept of "network vulnerability" into a concrete, mappable topology. It acts as an X-ray, revealing the structural skeleton upon which a network's optimal performance depends. By illuminating the set of absolutely critical edges, it provides an unambiguous blueprint for both fortification and offensive analysis. In the hands of defenders, it is a tool for prioritizing resources and anticipating strategic attacks; in the hands of auditors, it is a measure of a system's inherent fragility. Ultimately, this mathematical technique underscores a crucial principle in security: to defend a system effectively, one must first understand, with precision, what makes it most vulnerable. The minimal subgraph provides exactly that, profound and essential insight.

4 Parallel Graph Algorithm Design

Given a weighted, directed graph $G = (V, E)$, we denote by G^C the shortest-path closure of G , i.e. $E_{G^C}(x, y)$ is the length of the shortest path from x to y in G . A minimum shortest-path reduction or Min Graph of G is a graph G^R with minimal number of edges such that $(G^R)^C = G^C$. The problem is well studied in the literature due to its significance affecting a variety of practical problems such as network routing and verification of real-time systems Larsen et al. 1997. Its importance has led to the design of many parallel algorithms for many parallel architectures Katz and Kider 2008; Bondhugula et al. 2006; Han, Franchetti, et al. 2006; Han and Kang 2005; Harish and Narayanan n.d.; Subramanian et al. 1993; Venkataraman et al. 2003; Prihozhyy 2023; Tardivo

et al. 2023; Xia et al. 2022).

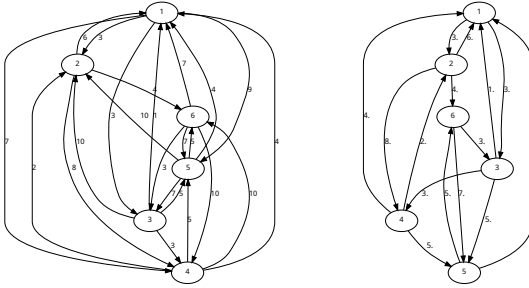


Figure 1: Graph G and its min graph

Researchers have studied network routing and optimization extensively because of its essential role in various applications. Numerous parallel algorithms exist for various parallel architectures because of the extensive work done by Katz and Kider 2008; Bondhugula et al. 2006; Han, Franchetti, et al. 2006; Han and Kang 2005; Harish and Narayanan n.d.; Subramanian et al. 1993; Venkataraman et al. 2003; Prihozhy 2023; Tardivo et al. 2023; Xia et al. 2022.

The study of recent times has centered on utilizing the massive GPU computational power which desktop systems now offer with up to 20,000 computing cores. Several parallel network routing and optimization algorithms on GPUs utilize the $O(n^3)$ Floyd-Warshall algorithm, where n denotes the number of vertices (Katz and Kider 2008; Bondhugula et al. 2006; Han, Franchetti, et al. 2006; Han and Kang 2005; Harish and Narayanan n.d.; Subramanian et al. 1993; Venkataraman et al. 2003; Prihozhy 2023; Tardivo et al. 2023; Xia et al. 2022).

However, the parallel Floyd-Warshall based algorithm slows down significantly when the size of the graph is larger than the memory available on the GPU because the algorithm needs to copy parts of the adjacency matrix n times to and from the GPU. For the same reason, the Floyd-Warshall based algorithm does not scale well on multiple GPU devices. In fact, it does not make any reasonable performance gain on multiple GPU devices in comparison with the single core CPU program.

To deal with this situation, we present a parallel algorithm based on a sequential $O(\log n \cdot n^3)$ algorithm in Lawler 1976. By exploiting the global scope of the GPU’s global memory and optimizing the use of the GPU’s

shared memory, we are able to achieve a large performance benefit with a speed-up of 12,000x on a desktop computer compared to a single-core CPU baseline—substantially outperforming the $5\times$ – $6\times$ improvements commonly reported in the literature.

The adjacency matrix shows the direct paths between nodes of the graph, which could be seen as the shortest paths with one edge. Since the shortest path with two edges between any nodes x and y will go through exactly one of the other nodes, an adjacency matrix A^2 of shortest paths with length at most two can be constructed as

$$A^2[i, j] = \min_{k \in V} (A^1[i, k] + A^1[k, j]), \text{ where } A^1 \equiv A.$$

We could construct A^3 from A^1 and A^2 by noticing that the shortest path with three edges or less would be a shortest path with two edges or less to some node followed by a shortest path of length at most one and vice versa. In the same manner, we could construct A^4 either from A^1 & A^3 or from just A^2 . Since any path with $N = |V|$ or more edges must include a cycle, A^{N-1} would be the matrix with the shortest paths through the entire graph. Also, we just need to construct $A^2, A^4 \dots A^{N'}$ where $N' \geq N - 1$ (Lawler 1976).

Algorithm 1 Shortest-Path-Preserving

Shortest-Path-Preserving(A, n)

```

 $D^0 \leftarrow A$ 
for  $k$  from 1 to  $\lceil \log_2(n-1) \rceil$ 
  for  $i$  from 1 to  $n$ 
    for  $j$  from 1 to  $n$ 
       $d_{i,j}^k \leftarrow \min_{k \in V} (d_{i,k}^{k-1} + d_{k,j}^{k-1});$ 
      if  $d_{i,j}^k \leq d_{i,j}^{k-1}$  then  $A_{i,j} \leftarrow \infty$ 
return  $D^k$ 
```

4.1 Parallel Design

Since GPU is a massively multi-threaded processor which can support several thousands concurrent threads, we will carefully design our parallel algorithm to expose substantial fine-grained parallelism and decompose the computation into independent tasks that require minimal global communication. Furthermore, besides 64K of registers available per block, 48 KB of shared memory per block, and 100KB of shared memory per SM, the GPU is equipped with 32 MB L2 cache that has very low access latency and high bandwidth.

Algorithm 2 CudaShortest-Path-Preserving

```
CudaShortest-Path-Preserving( $A, P, n$ )
  Allocate  $Ad \leftarrow A$  and  $Pd$  on the device
  for  $k$  from 1 to  $\lceil \log_2(n-1) \rceil$ 
    if  $k \% 2 \neq 0$ 
      Shortest-Path-PreservingKernel( $Ad, Pd, k$ )
    else
      Shortest-Path-PreservingKernel( $Pd, Ad, k$ )
   $k \% 2 = 0 ? P \leftarrow Pd : P \leftarrow Ad;$ 
```

Algorithm 3 CudaShortest-Path-Preserving Kernel

```
void Shortest-Path-PreservingKernel( $Ad, Bd, Pd, k$ )
   $Pvl \leftarrow \infty$ 
  Identify the bands of rows and columns to work
  For each pair of tiles  $As, Bs$  in the bands
    Identify the location  $(ty, tx)$  to work on
    Collaborative loading tiles  $As$  &  $Bs$ 
     $Pvl \leftarrow \min(As[ty, k] + Bs[k, tx], Pvl), k=1..16$ 
  if  $P[i, j] > Pvl$  then  $P[i, j] \leftarrow Pvl; A_{i,j} \leftarrow \infty;$ 
```

We take advantages of the fact that concurrent threads in a block can cooperate among themselves through synchronization and a per-block shared memory space. Since each row in A^k is combined with all columns of A^k in calculating A^{2k} and vice versa, we partition A^k into tiles such that each of the corresponding pair of tiles As and Bs will be collectively loaded into on-chip shared memory. We then process the data on the on-chip shared memory, and write the final result back out to global memory. The process is very similar to matrix multiplication except that we take the minimum value instead of the summation.

4.2 Performance & Analysis

Our algorithmic approach is derived from a sequential $O(n^3 \log n)$ approach, which theoretically performs worse than the classical Floyd-Warshall-based algorithms with $O(n^3)$ complexity. However, our parallel algorithm demonstrates significantly greater practical efficiency due to its superior parallelism and hardware-aware optimizations:

- **Logarithmic Grid Depth:** Unlike traditional algorithms that require n computation grids, our algorithm only requires $\log n$ grids, substantially reducing the number of kernel launches and global synchronizations.

- **In-Memory Computing Optimization:** The algorithm is designed to fully exploit in-memory computing by maintaining intermediate matrices entirely in GPU global memory. This eliminates the need for inter-grid memory transfers and leverages the persistent memory access locality.
- **Global Memory Coalescing:** Global memory accesses are optimized for coalescing, as threads in a warp access contiguous memory regions when loading submatrices As and Bs, thereby minimizing memory latency.
- **Shared Memory Bank Conflict Avoidance:** The algorithm prevents shared memory bank conflicts by matching memory access patterns to ensure all 16 threads in a half warp read from different memory banks when accessing on-chip shared memory.

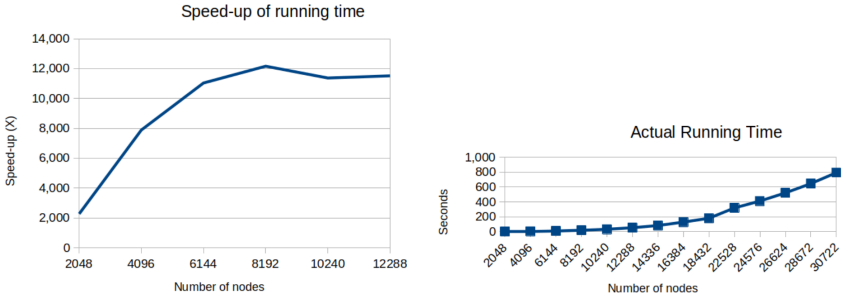


Figure 2: Speed-Up & Running Time

The experimental results demonstrate the algorithm’s scalability and performance through testing with dense graphs that have vertex counts between 2K and 30K. For instance, our algorithm computes Shortest-Path-Preserving Sub-graphs on a 14,336-vertex dense graph in just 81 seconds, whereas the serial implementation fails to complete even after 10 days. As illustrated in Figure 2, this corresponds to a speed-up of approximately 12,000 $\times$ over the CPU baseline. The performance gain is primarily attributed to (a) massive parallelism across the 216 CUDA cores per block and (b) efficient use of GPU shared memory bandwidth, which exceeds CPU memory bandwidth by an order of magnitude.

5 Conclusion and Future Work

We presented the design and analysis of CUDA parallel graph algorithms for solving the Network Hardening and Vulnerability Detection research problems.

The outcomes of this research are used as new materials for students in our network courses. We showed that a carefully designed parallel algorithm can offer large performance benefit in comparison with a single core program. These new methods and algorithms provide opportunities for students in Network courses to work on other real-world problems by applying the same ideas on other type of network routing and optimization.

References

- U. Bondhugula, A. Devulapalli, J. Fernando, P. Wyckoff, and P. Sadayappan. 2006. “Parallel FPGA - Based All-Pairs Shortest-Paths in a Directed Graph”. In: *Proceedings of the 20th IEEE International Parallel and Distributed Processing Symposium*.
- S.-C. Han, F. Franchetti, and M. Pueschel. 2006. “Program generation for the all-pairs shortest path problem”. In: *Parallel Architectures and Compilation Techniques (PACT)*.
- S.-C. Han and S.-C. Kang. 2005. *Optimizing all pairs shortest path algorithm using vector instructions*. <http://www.ece.cmu.edu/pueschel>. (2005).
- P. Harish and P. J. Narayanan. N.d. “Accelerating Large Graph Algorithms on the GPU Using CUDA”. In: *High Performance Computing (2007)* (Lecture Notes in Computer Science). Vol. 4873. Springer.
- Gary J. Katz and Joseph T. Kider. 2008. “All-Pairs Shortest-Paths for Large Graphs on the GPU”. In: *Graphics Hardware*. Ed. by David Luebke and John D. Owens.
- Kim Larsen, Fredrik Larsson, Paul Petterson, and Wang Yi. 1997. “Efficient Verification of Real-Time System: Compact Data Structure and State-Space Reduction”. In: *Proceedings of the 18th IEEE Real-Time Systems Symposium*, 14–23.
- Eugene Lawler. 1976. *Combinatorial Optimization: Networks and Matroids*. Holt, Rinehart, and Winston.
- Anatoly A Prihozhy. 2023. “Generation of shortest path search dataflow networks of actors for parallel multicore implementation”. In: *Informatics 2*. Vol. 20, 65–84.
- S. Subramanian, R. Tamassia, and S. J. Vitter. 1993. *An Efficient Parallel Algorithm for Shortest Paths in Planar Layered Digraphs*. Tech. rep., Durham, NC, USA. (1993).
- Fabio Tardivo, Agostino Dovier, Andrea Formisano, Laurent Michel, and Enrico Pontelli. 2023. “Constraint propagation on GPU: A case study for the AllDifferent constraint”. *Journal of Logic and Computation*, 33, 8, 1734–1752.
- G. Venkataraman, S. Sahni, and S. Mukhopadhyaya. 2003. “A blocked all-pairs shortest-paths algorithm”. *J. Exp. Algorithmics*, 2.
- Yang Xia, Peng Jiang, Gagan Agrawal, and Rajiv Ramnath. 2022. “Scaling and selecting GPU methods for all pairs shortest paths (APSP) computations”. In: *2022 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 190–200.

LLM-Assisted PDF-to-LaTeX Conversion for Academic Document Management*

Brandon Dean and Adithya Kulkarni

Department of Computer Science

Ball State University

Muncie, IN 47306

`brandon.dean@bsu.edu`, `adithya.kulkarni@bsu.edu`

Abstract

Academic institutions maintain large collections of scholarly documents in PDF format, which preserve visual layout but limit structural accessibility for reuse and downstream processing. While $\text{\LaTeX}$ provides a structured alternative, it is rarely available for legacy documents.

We evaluate the feasibility of using web-based large language models (LLMs) to convert academic PDFs into $\text{\LaTeX}$ under realistic archival conditions. We study four LLMs on research papers from EMNLP 2023 and 2024, covering both born-digital and scanned PDFs. Evaluation includes compilation success, compiler errors, rendered PDF comparison, and a targeted multi-step prompting case study.

Results show that LLM-assisted conversion can recover substantial document structure, but performance varies significantly across models. Stronger models achieve high compilation success, while weaker ones fail frequently. Contrary to common assumptions, scanned PDFs do not consistently reduce reliability, although reconstruction fidelity remains limited across all settings.

Overall, LLM-based PDF-to- $\text{\LaTeX}$ conversion is a practical but imperfect tool: it can reduce manual effort and enable partial recovery, but still requires human oversight. These findings provide a systematic characterization of its strengths and limitations for academic document management.

*Copyright ©2026 by the Consortium for Computing Sciences in Colleges. Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the CCSC copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Consortium for Computing Sciences in Colleges. To copy otherwise, or to republish, requires a fee and/or specific permission.

1 Introduction

Academic institutions and computing departments maintain large collections of scholarly materials in PDF format, including research papers, technical reports, theses, and archival publications. PDFs are effective for fixed-layout presentation and citation stability; however, they are fundamentally rendered artifacts rather than structured representations of knowledge. Consequently, PDFs pose persistent challenges for document management, accessibility tooling, machine-assisted analysis, and pedagogical reuse [13, 2].

A core limitation of PDFs is the lack of explicit semantic structure. While text can often be extracted from born-digital PDFs, essential elements such as document hierarchy, mathematical expressions, tables, and references are not represented in a form that is easily reusable or transformable. These challenges are amplified for scanned PDFs, where OCR artifacts distort content and disrupt logical reading order [11, 12]. Prior work on PDF parsing and structured extraction shows that no single system reliably recovers document structure across diverse layouts, particularly for technical documents with multi-column formatting and mathematics [1, 9].

Structured representations such as L^AT_EX offer a compelling alternative in computing disciplines. L^AT_EX explicitly encodes document structure and mathematical notation in a text-based form that supports version control, transformation, and downstream processing, including conversion to accessibility-oriented formats such as HTML and MathML. In contrast, XML-based pipelines and standards primarily target metadata and coarse structure, often treating mathematical content as opaque or embedding it as images [4, 6, 8].

Existing PDF-to-L^AT_EX approaches exhibit important limitations. Research systems typically focus on component-level extraction, such as tables or equations, assuming pre-segmented inputs and neglecting full-document coherence [7, 5]. Commercial conversion tools often prioritize visual fidelity over semantic reconstruction, frequently substituting complex regions or entire pages with embedded images. While such outputs may compile, they are often inaccessible, unsearchable, and unsuitable for document management or reuse.

Recent advances in large language models (LLMs) motivate renewed examination of PDF-to-L^AT_EX conversion. Modern web-based LLM systems integrate OCR, multimodal reasoning, layout heuristics, and long-context processing, enabling document-level inference that was previously impractical. Unlike traditional pipelines, LLMs can attempt to infer semantic roles and adapt to document characteristics. This raises a central question: can LLM-assisted workflows produce L^AT_EX representations that are sufficiently reliable and structurally meaningful for academic document management and accessibility-oriented use cases?

In this work, we empirically evaluate web-based LLM systems for converting academic PDFs into $\text{\LaTeX}$ as a complementary representation. We focus on research papers as a structurally demanding document class and consider both born-digital and scanned PDFs. Evaluation emphasizes realistic archival conditions in which original source files are unavailable, using compilation success, compiler error analysis, rendered PDF comparison, and qualitative inspection rather than source-level reconstruction.

Our contributions are threefold: (1) an empirical evaluation of LLM-assisted PDF-to- $\text{\LaTeX}$ conversion under PDF-only archival conditions, (2) identification of concrete failure modes and qualitative differences across models and document types, and (3) practical guidance on the appropriate role of LLM-assisted conversion in academic document management workflows.

2 Related Work

Prior work on processing academic PDFs spans document parsing, structured extraction, OCR pipelines, and neural approaches. A substantial body of research focuses on converting PDFs into structured formats such as XML for metadata extraction, indexing, and text mining [13, 2]. Systems such as GRO-BID extract bibliographic metadata and coarse document structure into XML/TEI representations [4], while standards such as JATS define schemas for scholarly archiving [6]. Although effective for metadata-centric tasks, these approaches often treat mathematical content and fine-grained academic structure as opaque or embed them as images, limiting reuse and accessibility, particularly for scanned PDFs where OCR noise propagates directly into structured outputs [11, 12].

Document layout analysis and reading order recovery have also been widely studied. Prior work has explored detecting text blocks, tables, and figures using annotated datasets and neural models [14, 10]. More recent vision-language models attempt end-to-end document parsing by jointly modeling layout and text [3]. However, these methods are typically evaluated on segmentation or detection tasks rather than on producing editable, source-level document representations suitable for document management.

Another line of research addresses component-level extraction, particularly for tables and mathematical expressions. Neural and iterative approaches demonstrate that isolated elements can be converted into $\text{\LaTeX}$ with reasonable accuracy [7, 5], but these methods assume pre-segmented inputs and do not address full-document coherence or cross-references.

Commercial PDF-to- $\text{\LaTeX}$ tools are widely used in practice but largely undocumented. Empirical usage indicates that these tools often prioritize visual fidelity over semantic reconstruction, frequently substituting complex regions

or entire pages with embedded images, yielding outputs that compile but remain unsuitable for search, accessibility tooling, or structured reuse.

Recent advances in web-based large language models integrate OCR, multimodal reasoning, and long-context processing, enabling document-level inference that was previously impractical. Despite growing interest in LLM-based document understanding, systematic evaluations of their effectiveness for full-document PDF-to-L^AT_EX conversion under realistic, PDF-only archival conditions remain limited. This work addresses that gap by evaluating LLM-assisted conversion at the level of complete academic documents, considering both born-digital and scanned inputs.

3 Experimental Setup

This section describes the dataset, models, prompting strategies, and evaluation criteria used to assess LLM-assisted PDF-to-L^AT_EX conversion under realistic archival conditions.

3.1 Dataset

The dataset consists of 50 academic research papers from the EMNLP main conferences in 2023 and 2024. The corpus includes both long and short papers, with document lengths ranging from 7 to 22 pages (average 12.82 pages). These papers exhibit multi-column layouts, mathematical expressions, tables, figures, and extensive reference sections, making them a structurally demanding class of academic PDFs.

For each paper, two PDF variants were used: the original born-digital conference PDF and a scanned version generated by printing and scanning the same document. This paired design enables controlled comparison between born-digital and scanned inputs while holding document content constant, reflecting archival scenarios in which original source files are unavailable.

3.2 Models

We evaluate four web-based large language model systems that support PDF upload and long-context document processing: Gemini3-Pro, ChatGPT-5.2, DeepSeek-V3.2, and Qwen3-Max. All experiments were conducted using publicly accessible web interfaces rather than APIs, reflecting realistic usage by institutional users.

3.3 Prompting Strategies

Single-step prompting serves as the primary evaluation setting. In this setting, each model is instructed to convert the entire PDF into a complete $\text{\LaTeX}$ source project in a single response.

Multi-step prompting is evaluated as a focused case study rather than a general benchmark. It is applied only to ChatGPT-5.2. In this setting, the model first generates a compilable $\text{\LaTeX}$ skeleton and then reconstructs document content section by section, with the goal of assessing whether task decomposition mitigates catastrophic failures.

For scanned PDFs, prompts instruct the model not to guess unclear characters and to mark unreadable content as `[ILLEGIBLE]`. To ensure consistent behavior, sessions suppress clarification questions and extraneous output. Full prompts are provided in the GitHub repository.¹

3.4 Evaluation

Evaluation focuses on practical usability rather than exact source-level reconstruction. Generated $\text{\LaTeX}$ projects are compiled using a standard compiler, and both compilation success and error counts are recorded. For outputs that compile successfully without errors, the resulting PDFs are compared against the originals using Adobe Acrobat to assess structural fidelity, including missing content, substituted images, layout disruptions, and degradation of mathematical or tabular elements. All assessments are based solely on PDF-level evidence. Outputs that compiled successfully but could not be opened in Acrobat are treated as failures.

4 Results and Discussion

This section summarizes key findings from the experimental evaluation, emphasizing qualitative behavior, counter-intuitive results, and implications for academic document management rather than restating numeric values visible in the figures.

4.1 Reliability and Failure Characteristics

Figure 1 highlights large differences in reliability across models. Qwen3-Max and DeepSeek-V3.2 achieve consistently high compilation success rates (above 80%), while ChatGPT-5.2 and Gemini3-Pro remain below 25%, indicating that performance is strongly model-dependent.

¹<https://github.com/Mrpoink/LLM-Assisted-PDF-to-LaTeX-Conversion-for-Academic-Documents-Management>

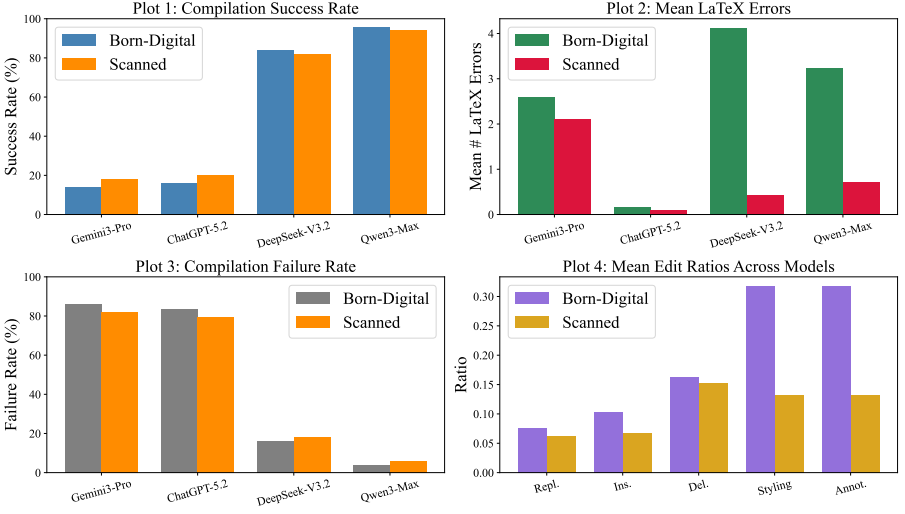


Figure 1: LLM PDF-to-L^AT_EX conversion results for born-digital and scanned datasets. Top-left: compilation success rate. Top-right: mean compiler errors. Bottom-left: compilation failure rate. Bottom-right: mean edit ratios (replacements, insertions, deletions, styling, annotations).

Interestingly, scanned PDFs do not consistently reduce performance. For weaker models, scanned inputs often yield comparable or slightly higher success rates than born-digital PDFs, suggesting that failures are not driven solely by OCR noise. Instead, structural complexity in born-digital PDFs may introduce additional challenges during reconstruction.

Error counts further clarify this behavior. DeepSeek-V3.2 and Qwen3-Max exhibit higher average compiler errors but still succeed more often, indicating greater robustness to imperfect outputs. In contrast, ChatGPT-5.2 shows very low error counts alongside high failure rates, suggesting that it fails early rather than producing recoverable outputs.

Failure rates reinforce this pattern: Gemini3-Pro and ChatGPT-5.2 fail in

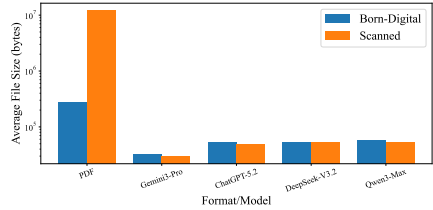


Figure 2: Average file size comparison between original PDFs and LLM-generated L^AT_EX sources for born-digital and scanned documents, shown side-by-side for each model.

over 80% of cases, whereas DeepSeek-V3.2 and Qwen3-Max remain below 20%. In practice, the ability to produce compilable outputs despite errors is more important than minimizing error counts alone.

Finally, edit ratios show that stronger models make more extensive structural changes, particularly in styling and annotations, while scanned inputs generally lead to fewer edits. This suggests a trade-off between reconstruction aggressiveness and input modality.

Overall, reliability depends not just on error frequency but on whether models can produce usable outputs under imperfect conditions. Consistent with this, Figure 2 shows that LLM-generated L^AT_EX sources remain compact across models and document types, despite large differences in conversion reliability.

4.2 Rendered Output Fidelity and Representation Efficiency

We assess reconstruction quality using Adobe Acrobat’s detected change counts. As shown in Figure 3, all models exhibit substantial divergence from the original PDFs, indicating that successful compilation does not imply high visual fidelity.

Born-digital PDFs generally incur higher change counts than scanned inputs, reflecting the difficulty of reconstructing complex layouts, equations, and formatting.

Differences across models are also evident. ChatGPT-5.2 produces the highest divergence, while DeepSeek-V3.2 and Qwen3-Max achieve lower change counts, indicating comparatively better reconstruction quality. Across all settings, deletions dominate over insertions and replacements, pointing to missing or improperly reconstructed content rather than minor textual variations.

Overall, even compilable outputs retain significant discrepancies, underscoring the gap between syntactic correctness and faithful document reconstruction.

4.3 Multi-step Prompting Case Study

We evaluate multi-step prompting for ChatGPT-5.2 across the full dataset. As shown in the results, multi-step prompting substantially improves compilation success. For born-digital documents, success increases from 16.33% under single-step prompting to 59.26%, reducing the failure rate by more than half. For scanned documents, gains are more modest, improving from 20.41% to 23.08%. These results indicate that multi-step prompting is particularly effective for structurally complex born-digital inputs, where intermediate reasoning steps help stabilize generation.

Despite these improvements in compilability, reconstruction fidelity remains limited. For born-digital documents, successful outputs still exhibit substantial divergence, with non-trivial proportions of deletions (14.26%) and styling

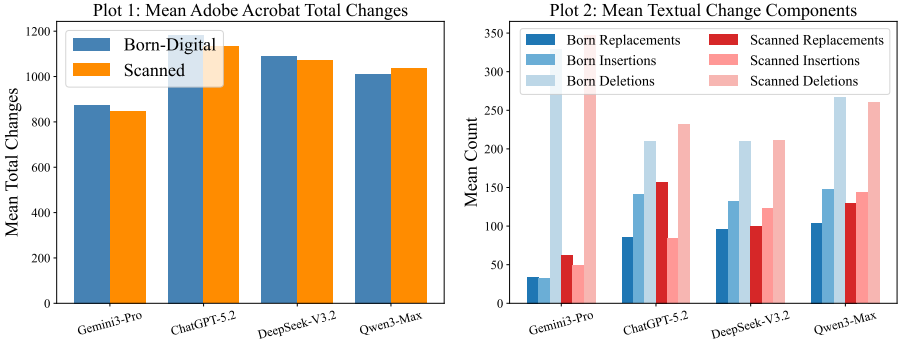


Figure 3: Adobe Acrobat-detected differences between original PDFs and LLM-generated L^AT_EX outputs. Left: mean total changes. Right: mean change components (replacements, insertions, deletions) for born-digital and scanned datasets.

or annotation changes (16.12%), indicating incomplete or altered structural recovery. This suggests that while multi-step prompting helps produce compilable code, it does not fully resolve deeper issues in layout reconstruction and semantic alignment.

Overall, multi-step prompting is best understood as a recovery mechanism that reduces catastrophic failures, especially for complex inputs, but does not guarantee high-fidelity reconstruction. Its benefits are therefore primarily in improving robustness rather than accuracy.

5 Implications and Limitations

Our results show that LLM-based PDF-to-L^AT_EX conversion is feasible, but highly dependent on the model. Stronger models are able to produce compilable outputs even with errors, while weaker ones tend to fail early. In practice, robustness matters more than minimizing error counts.

Interestingly, scanned PDFs do not consistently make the task harder. In some cases, they lead to similar or slightly better compilation outcomes, likely because their simpler visual structure is easier to process. However, this often comes at the cost of fidelity, as important structural details may be lost.

From a practical standpoint, model choice has a larger impact than document type. Systems with higher success rates and fewer failures are better suited for large-scale use. Multi-step prompting can reduce failures, especially for complex born-digital inputs, but its added cost makes it better suited as a fallback rather than a default approach.

This study has several limitations. We focus on web-based LLMs and academic papers from EMNLP, which may limit generalizability. We also do not have access to original L^AT_EX sources, preventing direct accuracy comparisons. Finally, our fidelity measure is based on PDF-level differences, which capture visual changes but not deeper semantic correctness. Despite these constraints, the overall trends are consistent and informative for real-world use.

6 Conclusion

We presented an empirical study of LLM-based PDF-to-L^AT_EX conversion under realistic, PDF-only conditions. While modern models can recover meaningful structure, performance varies widely, with stronger models achieving high compilability and weaker ones failing frequently.

We also find that scanned PDFs do not necessarily degrade performance, and may even simplify reconstruction, though often with reduced fidelity. More importantly, successful compilation does not guarantee accurate reconstruction, as substantial structural differences often remain.

Overall, LLM-based conversion is best viewed as a useful but imperfect tool. It can reduce manual effort and recover partial structure, but still requires human oversight. Continued evaluation of both robustness and fidelity will be important as these systems improve.

7 Ethical Statement

This work involves the use of LLMs to reconstruct documents that may be protected by copyright, which raises important ethical considerations. We emphasize that the generated outputs are not intended for redistribution, publication, or professional use. Instead, they are used solely for personal archival and organizational purposes. We discourage any use of these reconstructed documents that would violate copyright or intellectual property rights.

References

- [1] Narayan S Adhikari and Shradha Agarwal. “A comparative study of pdf parsing tools across diverse document categories”. In: *arXiv preprint arXiv:2410.09871* (2024).
- [2] Salvador D Atagong et al. “A review on knowledge and information extraction from PDF documents and storage approaches”. In: *Frontiers in Artificial Intelligence* 8 (2025), p. 1466092.

- [3] Xiangyang Chen et al. “Logics-parsing technical report”. In: *arXiv preprint arXiv:2509.19760* (2025).
- [4] *GROBID: GeneRation Of Bibliographic Data*. <https://grobid.readthedocs.io>. Machine learning library for extracting and restructuring raw PDF documents into structured XML/TEI. 2024.
- [5] Nan Jiang et al. “LATTE: Improving Latex Recognition for Tables and Formulae with Iterative Refinement”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 39. 4. 2025, pp. 4030–4038.
- [6] *Journal Article Tag Suite (JATS)*. <https://jats.nlm.nih.gov>. ISO standard for structured representation of scholarly articles in XML. 2024.
- [7] Pratik Kayal et al. “Tables to LaTeX: structure and content extraction from scientific tables”. In: *International Journal on Document Analysis and Recognition (IJDAR)* 26.2 (2023), pp. 121–130.
- [8] *LaTeXML: LaTeX to XML/HTML/MathML Converter*. <https://dlmf.nist.gov/LaTeXML/>. Software for converting LaTeX documents into structured XML and other machine-readable formats. 2024.
- [9] Zichao Li et al. “Readoc: A unified benchmark for realistic document structured extraction”. In: *Findings of the Association for Computational Linguistics: ACL 2025*. 2025, pp. 21889–21905.
- [10] Nikolaos Livathinos et al. “Robust pdf document conversion using recurrent neural networks”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 35. 17. 2021, pp. 15137–15145.
- [11] Mayank Singh et al. “Ocr++: a robust framework for information extraction from scholarly articles”. In: *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*. 2016, pp. 3390–3400.
- [12] Rasha Sinha et al. “Digitization of Document and Information Extraction using OCR”. In: *arXiv preprint arXiv:2506.11156* (2025).
- [13] Qintong Zhang et al. “Document parsing unveiled: Techniques, challenges, and prospects for structured information extraction”. In: *arXiv preprint arXiv:2410.21169* (2024).
- [14] Xu Zhong, Jianbin Tang, and Antonio Jimeno Yepes. “Publaynet: largest dataset ever for document layout analysis”. In: *2019 International conference on document analysis and recognition (ICDAR)*. IEEE. 2019, pp. 1015–1022.

GLIDE: Developing a Usability-Driven Programming Environment for Novices *

Mika Morgan¹, Stephanie Ludi², and Tina Johnson¹

¹Computer Science Department
Midwestern State University
Wichita Falls, TX 76308

`mika.morgan@msutexas.edu` and `tina.johnson@msutexas.edu`

²Computer Science Department
University of North Texas
Denton, TX 76205
`stephanie.ludi@unt.edu`

Abstract

Learning to code can provide many benefits. However, the field can be difficult to enter, as newcomers must memorize syntax and commands, as well as learn programming logic. To make coding more accessible to novices, we developed Glide, a hybrid IDE that combines text-based programming with many of the support features of block-based environments. Glide's design was based on usability heuristics and feedback from computer science teachers. The new environment was tested with 42 novice programmers and demonstrated improved learning outcomes.

1 Introduction

Learning to code can promote critical thinking, problem-solving, and creativity, with benefits across disciplines [8]. However, traditional text-based programming environments (TBPEs) require prior knowledge of language commands

*Copyright ©2026 by the Consortium for Computing Sciences in Colleges. Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the CCSC copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Consortium for Computing Sciences in Colleges. To copy otherwise, or to republish, requires a fee and/or specific permission.

and syntax, increasing novice’s cognitive load and reducing retention [15]. To address this challenge, researchers developed block-based programming environments (BBPEs) to simplify early learning and broaden K-12 participation in computer science [7].

Integrating the most impactful BBPE features into TBPEs as a dual-modality environment may make text-based programming more accessible and engaging for novices of all ages. This study tested that theory by developing and evaluating a dual-modality environment informed by usability heuristics to answer the research question: “Does integrating impactful features from block-based and text-based environments into a single platform improve novice programmer learning?”

2 Related Work

First, we needed to consider existing platforms and identify which BBPE features best support student learning. Prior usability evaluations over two popular BBPEs, Scratch and code.org, found that students need prominent help features, visual cues, feedback indicators, and support transitioning between block- and text-based modalities [5, 1].

Other studies solicited teacher feedback to identify critical features in programming environments- such as help-request queues, peer assessment, and assignment differentiation [9], and usability heuristics (e.g., user control, clear visual cues, context-aware errors) and support for practices like pair programming [11]. A larger survey with 100 K-12 teachers further prioritized ease-of-use and engagement features, such as web-based platforms, teacher dashboards, debugging tools, and contextual documentation [12].

Finally, we mapped the most prominent and desired features to Jakob Nielsen’s 10 usability heuristics, which was chosen as a framework because of the close relationship between these heuristics and BBPE features (such as recognition over recall for code blocks), and because of their use in foundational work on applying usability heuristics to improve IDEs for children [14]. The heuristics and their associated desired features can be seen in Table 1.

3 Methodology

3.1 Software Development

In order to answer the research question, we designed and implemented Glide, a dual-modality programming environment that integrates text-based coding with supportive features from block-based environments. Guided by existing research on IDE features that aid novice learners [11, 9, 10, 6] and Nielsen’s

Table 1: Key Usability Features to Assist Novices in Learning to Code

H1: Visibility of system status
• Variable pane
H2: Match between system and the real world
• Simple, natural language • Icons to indicate usage group
H3: User control and freedom
• Undo/redo • Version history
H4: Consistency and standards
• Code is actual Python (not pseudocode, etc.) • Code editor matches popular, professional IDEs – Layout, syntax highlighting, JetBrains Mono font
H5: Error prevention
• Inserted code snippets (avoids typos, spelling mistakes, etc.) • Auto-complete paired punctuation (parentheses, quotation marks)
H6: Recognition rather than recall
• Toolbox pane to see available Python commands • IntelliSense to prompt available options when typing in the editor
H7: Flexibility and efficiency of use
• Can insert code using toolbox or type directly into code editor • Support familiar keyboard shortcuts • Adjustable sizing for toolbox, editor, output, and variable pane
H8: Aesthetic and minimalist design
• Simple, minimalist layout • Color-coded visual cues (reserved words and code chunk scope)
H9: Help users recognize, diagnose, and recover from errors
• Clear error messaging • “Show Me Where” feature
H10: Help and documentation
• Contextual documenta
• s in toolbox)

usability heuristics [13], Glide includes a toolbox of code commands, contextual documentation, and color-coded highlighting. The full code is available open source on GitHub¹.

The final implementation uses HTML, CSS, and JavaScript, with CodeMirror 5 as the editor framework [4] and Pyodide for executing Python directly in the browser [3]. The interface uses a three-pane layout with a toolbox, central editor, and output pane, enabling students to insert snippets, write code, and view results without installing local software, which is crucial to classroom adoption [12]. CodeMirror provides syntax highlighting, auto-indentation, and line numbering, while Pyodide handles input and live execution. Glide, seen in Figure 1, is hosted using GitHub Pages².



Figure 1: Glide Screenshot Showing Hover-Over Tooltips

The toolbox was organized into collapsible categories (Basics, Conditionals, Loops, Functions, etc.) and allowed students to insert working code snippets, reducing syntax errors, typos, and the need to memorize commands- key challenges for novices [6]. Additionally, the toolbox utilized hover-over tooltips to provide contextual documentation, which was highly valued in teacher feedback [11, 12]. The editor helped students visualize code structure by highlighting everything within a conditional, loop, or function block when the heading is right-clicked. The output pane displayed results in real time and provided options for clearing or downloading work for submission.

3.2 Software Evaluation

To evaluate Glide, a comparative study was conducted in two online sections of an undergraduate introductory Python course in Fall 2025 (N = 42; 15

¹<https://github.com/mika-morgan/GLIDE>

²<https://mika-morgan.github.io/GLIDE/>

computer science majors, 27 non-majors), with IRB approval obtained prior to commencement. The course covered Python basics such as variables, user input, mathematical expressions, conditionals, loops, and functions.

Students completed three programming assignments (P1, P2, P3), four quizzes (Q1, Q2, Q3, Q4), and two exams (E1, E2). For the programming tasks, they were introduced to Glide along with two web-based platforms (Replit and Online Python) and two desktop IDEs (PyCharm and IDLE), and could freely choose and switch environments. Usage of Glide was determined automatically and not based on self-reporting. Nineteen students used Glide on the first assignment (18 novices, 1 experienced programmer), and twelve used Glide on the second and third assignments (all novices). Student performance on the nine assignments was then compared based on whether or not they used Glide to complete at least one assignment.

4 Results

To determine whether the use of Glide had an impact on student learning, we calculated the conditional mean average for three groups: current students who used Glide on at least one assignment, current students who did not use Glide, and students who never used Glide, including both current-semester students and students from three previous semesters. To help control for unrelated factors, we excluded students who had previously taken at least one college-level programming course and removed assignment grades of zero that resulted from non-participation. The results of this comparative study are shown in Table 2.

An independent-samples Welch’s t-test revealed a statistically significant difference on the functions quiz (Quiz 4), with Glide users outperforming non-users. Features in Glide that could impact function understanding include the function templates in the toolbox (including both fruitful and non-fruitful functions), and the code chunk highlighting to show function scoping. Although Glide users showed higher averages on other tasks as well (particularly the first quiz and the third programming assignment), these differences did not reach statistical significance, suggesting comparable learning outcomes across environments.

Figure 2: Glide Users v. Non-Users- Novices Only shows these results with Glide Users in blue, Glide Non-users (current semester only) in orange, and Glide Non-Users (total, including prior semesters) in green. Additional external factors could also impact student performance in the current semester compared to previous semesters, such as increased AI usage.

Table 2: Glide Users v. Non-Users Conditional Class Averages- Novices Only

	P1	Q1	P2	Q2	E1	P3	Q3	Q4	E2
Glide Users <i>n</i> = 18	97.22	94.44	99.67	92.94	76.23	100.00	93.89	88.00	72.29
Glide Non-users (current) <i>n</i> = 14	97.27	94.62	98.64	95.00	76.31	95.83	91.43	84.17	72.92
t(p) Value	0.98	0.95	0.27	0.36	0.97	0.10	0.64	0.44	0.93
Glide Non-users (total) <i>n</i> = 40	97.43	85.50	96.65	93.50	78.33	90.54	91.71	71.78	69.55
t(p) Value	0.91	0.05	0.28	0.85	0.63	0.09	0.66	0.01	0.64

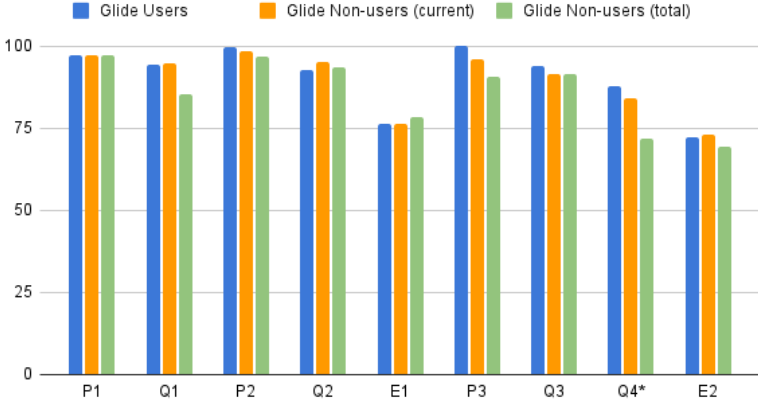


Figure 2: Glide Users v. Non-Users- Novices Only

5 Conclusion and Future Work

This study examined whether integrating impactful features from block-based and text-based environments into a single platform improves novice learning and understanding. In a pilot test of Glide in an authentic classroom setting, novice students who used Glide outperformed non-users in seven out of the nine assessments, including one quiz with statistical significance. These results demonstrate the importance of usability and learner-centered design in introductory programming tools, which supports prior findings [2, 14, 6]. By embedding features aligned with well-established HCI heuristics, such as clear error feedback, visual organization, and contextual support, educators and developers can reduce the intimidation often associated with first encounters in text-based coding.

Future work will involve a more comprehensive implementation of Glide’s planned usability features, including the variable pane and version history, which were not fully operational in this study. The next phase of testing will involve K-12 students in controlled learning settings, such as summer coding camps, to evaluate Glide’s effectiveness with its intended target population. This follow-up study will focus on measuring learning gains, engagement levels, and the ease of transition from block-based to text-based environments.

References

- [1] Kaan Dilmen, Serhat Kert, and Tuba Uğraş. “Children’s coding experiences in a block-based coding environment: a usability study on code.org”. In: *Education and Information Technologies* 28 (Feb. 2023), pp. 1–26. DOI: 10.1007/s10639-023-11625-8.
- [2] Hans Djalali, Wajdi Aljedaani, and Stephanie Ludi. “The Evolution of Software Usability in Developer Communities: An Empirical Study on Stack Overflow”. In: *Software* 4 (Oct. 2025), p. 27. DOI: 10.3390/software4040027.
- [3] Michael Droettboom. *Pyodide*. URL: <https://pyodide.org> (visited on 09/07/2025).
- [4] Marijn Haverbeke. *CodeMirror* 5. URL: <https://codemirror.net/5/> (visited on 09/07/2025).
- [5] Y. Jimenez, A. Kapoor, and C. Gardner-McCune. “Usability Challenges that Novice Programmers Experience when Using Scratch for the First Time”. In: *2018 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. Los Alamitos, CA, USA: IEEE Computer Society, Oct. 2018, pp. 327–328. DOI: 10.1109/VLHCC.2018.8506560. URL: <https://doi.ieeecomputersociety.org/10.1109/VLHCC.2018.8506560>.
- [6] Michael Kölling, Neil C. C. Brown, and Amjad Altadmri. “Frame-Based Editing: Easing the Transition from Blocks to Text-Based Programming”. In: *Proceedings of the Workshop in Primary and Secondary Computing Education*. WiPSCE ’15. London, United Kingdom: Association for Computing Machinery, 2015, pp. 29–38. ISBN: 9781450337533. DOI: 10.1145/2818314.2818331. URL: <https://doi.org/10.1145/2818314.2818331>.
- [7] Radoslava Kraveva, Velin Kravev, and Dafina Kostadinova. “A Methodology for the Analysis of Block-Based Programming Languages Appropriate for Children”. In: *Journal of Computing Science and Engineering JCSE*, Volume 13 (Mar. 2019), pp.1–10. DOI: 10.5626/JCSE.2019.13.1.1.
- [8] Sang Joon Lee, Gregory M. Francom, and Jeremiah Nuatomue. “Computer science education and K-12 students’ computational thinking: A systematic review”. In: *International Journal of Educational Research* 114 (2022), p. 102008. ISSN: 0883-0355. DOI: <https://doi.org/10.1016/j.ijer.2022.102008>. URL: <https://www.sciencedirect.com/science/article/pii/S0883035522000866>.

- [9] Ally Limke et al. “A Survey of K-12 Teacher Needs for an Online Programming Learning System”. In: *Extended Abstracts of the 2024 CHI Conference on Human Factors in Computing Systems*. CHI EA '24. Association for Computing Machinery, 2024. ISBN: 9798400703317. DOI: 10.1145/3613905.3651110. URL: <https://doi.org/10.1145/3613905.3651110>.
- [10] Ally Limke et al. “Participatory Design with Teachers for Block-Based Learning with SnapClass”. In: *2023 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. Los Alamitos, CA, USA: IEEE Computer Society, Oct. 2023, pp. 173–178. DOI: 10.1109/VL-HCC57772.2023.00028. URL: <https://doi.ieeecomputersociety.org/10.1109/VL-HCC57772.2023.00028>.
- [11] Mika Morgan et al. “Investigating the Usability of Coding Applications for Children: Insights from Teacher Interviews”. In: *2024 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. Liverpool, UK: IEEE Computer Society, Sept. 2024, pp. 47–58. DOI: 10.1109/VL/HCC60511.2024.00016.
- [12] Mika Morgan et al. “Prioritizing Usability in Programming Environments for Children”. In: CHIItaly '25. Association for Computing Machinery, 2025. ISBN: 9798400721021. DOI: 10.1145/3750069.3750075. URL: <https://doi.org/10.1145/3750069.3750075>.
- [13] Jakob Nielsen. “Enhancing the Explanatory Power of Usability Heuristics”. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '94. Boston, Massachusetts, USA: Association for Computing Machinery, 1994, pp. 152–158. ISBN: 0897916506. DOI: 10.1145/191666.191729. URL: <https://doi.org/10.1145/191666.191729>.
- [14] John F. Pane, Brad A. Myers, and Leah B. Miller. “Using HCI techniques to design a more usable programming system”. In: *Proceedings IEEE 2002 Symposia on Human Centric Computing Languages and Environments* (2002), pp. 198–206.
- [15] David Weintrop and Uri Wilensky. “Transitioning from Introductory Block-Based and Text-Based Environments to Professional Programming Languages in High School Computer Science Classrooms”. In: *Comput. Educ.* 142.C (Dec. 2019). ISSN: 0360-1315. DOI: 10.1016/j.compedu.2019.103646. URL: <https://doi.org/10.1016/j.compedu.2019.103646>.

Not Quite Dead Yet: Strategies to Mitigate the Detrimental Effects of LLMs on CSE*

Michael P. Rogers¹, Christopher L. Groves², Hannah M. Hillberg¹

¹Computer Science, ²Psychology

University of Wisconsin Oshkosh

Oshkosh, WI 54901

mprogers@mac.com, grovesc@uwosh.edu, hillbergh@uwosh.edu

Aziz Fellah

Computer Science

Northwest Missouri State University

Maryville, MO 64468

afellah@nwmissouri.edu

Abstract

This paper reports the results of a multi-university survey to assess how computer science students are currently using Large Language Models (LLMs) in their studies. We report on the frequency of usage for a variety of programming tasks and as a tutor. We inquire into their perceptions of LLMs: where it is appropriate to use them, and when it might impede their learning. Finally, we ask students to assess various techniques to dissuade them from using LLMs in situations where their use would negatively impact their education.

1 Introduction

The word "revolution" is often misused, but when applied to the effect of Large Language Models (LLMs) on Computer Science Education (CSE), it is because it is inadequate: it fails to truly capture the promise, peril, and irony of this new technology. The promise is because of its prodigious coding capability (which is why it is embraced in industry), and its ability to provide serviceable

*Copyright ©2026 by the Consortium for Computing Sciences in Colleges. Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the CCSC copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Consortium for Computing Sciences in Colleges. To copy otherwise, or to republish, requires a fee and/or specific permission.

tutelage services 24/7. The peril is because it can within seconds produce near-perfect solutions to far too many CS assignments, with no traceable evidence of their LLM-origin. The irony is because this threat to CSE is a byproduct of CSE.

We know that our students are using LLMs, but we wanted to gain a fuller understanding of:

1. how they are using them,
2. their perceptions of its promise and peril; and, most critically,
3. whether they would be amenable to strategies to guide their use of LLMs, to mitigate the perils.

To this end, we conducted a detailed survey of CS undergraduate students at two universities addressing the above objectives. This paper reports on our findings from that survey.

2 Prior Work

The positive effects of LLMs in CSE have been well documented. LLMs can engage and motivate students [9], and serve as a powerful supplementary learning tool / mentor[11]. Their use in CS1 classes correlates with a decrease in the number of homework-related questions and time to complete homework. [18]. LLMs are also a boon for CS educators, helping to create course content and assisting with grading [1, 10].

LLMs also have some downsides. Students have misgivings about the long-term effects of LLMs on their ability to learn [11]. Some rely on them so much that they cannot code without them [17]. One study found a significant negative correlation between reported usage of LLMs for out-of-class assignments and grade on an in-class, LLM-free assessment of the same topics [8].

The ability of modern LLMs to generate high-quality code may be regarded as positive or negative, depending on the context. In the workforce, it boosts programmer productivity immensely, which is why 85% of software developers use LLMs or will soon, and over 50% use it daily [16]. In academia, LLMs present a serious problem, as they can solve most typical CS1 and CS2 assignments, and the areas in which they still falter are shrinking [15, 7].

This, of course, has led inevitably to LLM-based plagiarism (aka "AI-giarism" [4]). Although the source of plagiarized code and the branding may have changed, strategies to address this are venerable:

1. Provide clear, comprehensive guidelines indicating where LLM usage is permitted, where it is not, and penalties for its misuse [2].

2. Enforce those guidelines. While there have been some promising developments [5, 13], a general, reliable way to automatically detect LLM-generated code remains elusive [3, 6].
3. Provide alternative forms of assessments, such as oral exams [6]. To reduce faculty workload, some have even suggested using LLMs to administer those exams [12].
4. Put less weight on out-of-class assessments [14].

All of these approaches are based on external measures brought to bear on the student by the instructor, and are, in a word, adversarial. We want to consider a fifth approach that instills in our students an innate predisposition to reject AI-giarism. We discuss it, and assess its potential efficacy, in Section 4, Survey Results and Discussion.

3 Survey Methodology

In the fall of 2025 we administered an IRB-approved survey to CS students at two universities, asking precise and pointed questions to quantify:

1. how much they are using LLMs in various stages of the coding process
2. which tasks they believe could be safely delegated to LLMs without degrading their educational experience
3. their amenability to techniques to regulate the use of LLMs when it would be in their best interests.

Requests to complete the anonymous survey were made in CS classes and emailed to all CS majors. Note that because "AI" is more recognizable than "LLM", we used the former in the survey instrument and will do so here when quoting question text. In the survey preamble, however, we stated that AI referred to "tools powered by large language models such as ChatGPT, Claude, and Copilot".

4 Survey Results and Discussion

We had 169 respondents (7% freshmen, 17% sophomores, 26% juniors and 50% seniors), with 40% from UniversityA, and 60% from UniversityB. Most (75%) were male, 19% female, and the remainder preferred not to say or self-described. Approximately 69% had used LLMs for at least a year.

4.1 Frequency of AI Usage

As Table 1 indicates, most respondents are using LLMs while doing coding assignments, primarily for explaining compile and runtime errors and general

Table 1: Frequency of AI Use by Task (%)

Task	Never	Rarely	Occasionally	Often	Always
Debugging code	7.1	16.7	40.5	27.4	8.3
Writing comments	48.5	22.5	16.6	7.1	5.3
Writing code ($\leq 25\%$)	22.5	36.7	28.4	10.1	2.4
Writing code (26–75%)	46.4	31.6	10.7	8.3	3.0
Writing code ($> 75\%$)	61.5	18.1	10.2	8.4	1.8
Explaining compile errors	7.7	10.1	26.6	36.1	19.5
Explaining runtime errors	10.7	8.3	27.4	33.9	19.6
Code review	19.1	17.9	27.4	19.1	16.7
Brainstorming	18.0	21.0	29.9	23.4	7.8
Explaining concepts	7.1	20.8	29.2	29.8	13.1
Refactoring code	22.8	31.7	24.0	14.4	7.2

concepts. We believe that students should attempt debugging on their own first, and resort to LLMs only when necessary, prefacing their LLM prompts with a request to "use the Socratic method". Distressingly, 20.4% are using LLMs to write over 75% of their code at least occasionally.

4.2 LLMs: Perceptions and Usage

Table 2: Student Perceptions of AI Use (%)

Statement	SD	D	N	A	SA
Enjoy coding	1.8	3.0	11.5	47.0	36.8
Enjoy coding more with AI	7.9	20.6	36.4	21.8	13.3
Satisfied without AI	4.2	3.6	13.3	28.3	50.6
AI makes me a stronger coder	12.7	20.0	34.6	23.6	9.1
AI makes me a stronger debugger	9.0	14.5	25.9	39.8	10.8
AI programs are correct	32.5	31.3	22.3	10.2	3.6
Trust AI programs over mine	40.6	31.5	13.3	10.3	4.2
AI causes less proficiency	6.1	18.8	23.0	32.1	20.0
AI availability is reassuring	3.0	9.1	13.9	49.1	24.9
AI better than standard search	3.7	4.9	15.9	40.2	35.4
AI limits serendipitous discovery	9.8	17.1	26.2	28.1	18.9

We asked several statements to gauge student perceptions of LLM usage in coding (Table 2) where SD=Strongly Disagree, D=Disagree, N=Neutral, A=Agree, SA=Strongly Agree. Since our curricula adhere closely to the ACM Curricular Guidelines, it was reassuring to see that our students really like to program: 84% agreed or strongly agreed with the statement "I enjoy writing

code". We found it equally reassuring that our students still cherish the act of problem solving: only 35% enjoyed coding more using AI, 79% said that they gained more satisfaction from programming without it. When asked specifically if AI made them stronger programmers, however, there was no consensus: the distribution was essentially uniform across all categories. Our students expressed much the same healthy skepticism of AI as professionals [16], with only 14% saying that programs written by AI can be relied upon to be correct. They also were concerned about its overall detrimental effects: 52% felt that using AI to perform a task caused them to be less proficient at it.

When asked which tasks could they safely delegate to AI without adversely impacting their proficiency, the results (Table 3) were what one would expect from a group that values problem solving. While students did have some issues with AI, they did appreciate its presence: 74% felt that they were more confident knowing that AI is available if they had difficulties (Table 2). They, like most users, found that AI did a better job than a standard search (76%), but 47% were concerned that an AI-based search would curtail those serendipitous discoveries that arise when doing a standard search.

Table 3: Perceived Appropriateness of AI Use by Task (%)

Task	Not Appropriate	Appropriate
Writing unit tests	63.9	36.1
Looking up syntax	33.7	66.3
Populating a database	38.5	61.5
Developing an algorithm	81.7	18.3
Writing code ($\leq 25\%$)	62.1	37.9
Writing code (26–75%)	84.0	16.0
Writing code ($> 75\%$)	92.9	7.1
Porting a program	57.4	42.6
Creating a user interface	80.5	19.5
Software installation	50.3	49.7
Explaining compile-time issues	18.3	81.7
Explaining runtime issues	19.5	80.5

4.3 Assessing Techniques for Restricting the Use of LLMs

For pedagogical purposes, an instructor may wish to restrict student use of LLMs when completing assignments, and myriad techniques have been suggested (see Section 2, Prior Work). We selected five, and asked students to rate their effectiveness in persuading them to turn in an LLM-free solution (Table

4). For expository purposes we will designate those who selected effective (E) or highly effective (HE) as "supportive"; those who selected ineffective (I) or highly ineffective (HI) as "skeptical"; and the remainder (N) as "undecided".

Table 4: Perceived Effectiveness of Techniques to Moderate AI Use (%)

Strategy	HI	I	N	E	HE
Reduce assignment weight	8.8	17.0	29.6	27.0	17.6
Allot more time	2.5	10.6	21.9	38.8	26.3
Remove due dates	13.1	21.3	23.8	20.0	21.9
Require reading an article	11.7	14.8	34.6	29.6	9.3
Use logic puzzles	6.9	10.0	20.6	44.4	18.1

4.3.1 Technique 1: Reduced Assignment Weight

Students were asked about the effectiveness of reducing assignment weight, so that it represented "only a very small portion of your final grade". The supportive fraction was the largest, but only 44%, and a substantial 26% were skeptical. This finding is particularly significant because while this approach has been espoused by many researchers and instructors, many of our respondents, the "target market", so to speak, were not convinced of its efficacy. Why is that? Well, as seasoned instructors know, students will argue for every point that they can, so perhaps this should not come as a surprise.

4.3.2 Technique 2: Reduced Time Pressure

Like most CS classes, ours are chock-full, so time management is always an issue. When asked if providing significantly more time than necessary to complete an assignment would curtail LLM-usage, 65% were supportive, and only 13% skeptical. This is also unsurprising, as students typically respond positively to the question, "do you need more time?". While this may be logistically infeasible to implement in practice, it is certainly worth noting.

4.3.3 Technique 3: Flexible Due Dates

When asked about a policy allowing students to turn in assignments at any time during the semester (with no firm due date), here, too, the results were mixed: 42% were supportive, but almost as many, 34%, were skeptical. These numbers, plus the technique's negative impact on instructor's workload, suggests that this technique might not be worth pursuing.

4.3.4 Technique 4: Compelling Arguments

Some students in studies cited earlier had expressed misgivings about LLMs, and numerous articles in the literature and in the media have done the same. We asked students whether "watching a video / reading an article presenting evidence of the effects of over-reliance on AI" would be effective. Most (39%) were supportive, but a significant 25% were skeptical, and 35% were neutral. We have an abiding and deeply-rooted trust in the power of education to change minds, and so we believe that this technique warrants further study.

4.3.5 Technique 5: Aha!

The techniques described above have, in a sense, been based on fear: the first three involve fear of a poor grade, and the fourth fear of intellectual degradation. This last technique, however, is different, appealing to a student's a) love of puzzles, b) delight in experiencing an "Aha!" moment, and c) sense of pride. Students were told that they would be instructed to "solve an interesting logic puzzle without AI, then solve a similar problem with AI to provide guidance", and finally "write a paragraph reflecting on which was more satisfying and beneficial". We found 63% of students to be supportive, and only 17% were skeptical, which makes this result statistically tied with the "Reduced Time Pressure" technique that generated the most support.

5 Analysis

We asked students how frequently they used AI to write $\leq 25\%$, $26 - 75\%$, and $> 75\%$ of their code (Table 1). Because these items correlated highly, ($r_s = .48$ to $.86$, $p_s < .001$), an averaged composite score was created as an overall measure of how likely they were to use LLMs to write code for assignments - which we use in the analyses throughout this section.

5.1 Technique Analyses

Techniques 4 and 5 seemed to be particularly appealing to our respondents, and a subset of the questions asked in this survey provide an opportunity for a preliminary assessment of their potential effectiveness. First, consider Technique 4: Compelling Arguments. One survey item evaluated participants' agreement with the statement "If I use AI to perform a task, then I become less proficient at that task." Of all possible compelling arguments one could provide, this item captures one of the strongest. Those who agree with this statement do not need a compelling argument: they already believe it. Yet, agreement with this item was not associated with reduced LLM use for writing code

($r = -.06$, $p = .47$), suggesting that even well-crafted, compelling arguments would be ineffective. Technique 5, however, looks more promising. Questions that captured intrinsic motivation for coding were significantly associated with reduced reliance on LLMs to write code (enjoying writing code: $r = -.15$, $p = .049$; gaining a sense of satisfaction from programming without AI: $r = -.17$, $p = .03$). So if Technique 5 can indeed lead to increased intrinsic motivation, it may prove to be a more effective approach than Technique 4, though more detailed evaluation of potential arguments presented via Technique 4 may yet find it capable of providing viable deterrence.

5.2 Motivational Analyses

We wished to understand the motivation for reliance on LLMs. We therefore conducted additional analyses to better understand the relationship between LLM usage and general Openness to Reduced Use (ORU). The ORU measure summed the ratings of the perceived effectiveness of each of the five techniques. Importantly, ORU was positively related to LLM use ($r = .20$, $p = .01$): the more students use LLMs, the more open they are to strategies to deter them from doing so. This suggests that students may find themselves compelled, rather than drawn to LLM use. Alternatively, it is possible that this relationship arises because heavy LLM users are the only ones who may need reasons to reduce use: those who rarely use LLMs do not necessarily need additional rationale to further reduce usage. However, item-by-item analyses indicate that Technique 5's correlation with LLM use was strongest by a reasonable margin ($r = .20$) and the only significant effect on its own ($p = .01$) relative to the other techniques (r s = $.03$ to $.15$, p s = $.15$ to $.65$), lending support to the notion that students would prefer intrinsically motivating, LLM-free work.

5.3 Demographic Analyses

Relationships between participant characteristics and LLM use were examined to better inform instructors and administrators regarding who are most likely to rely on these tools. Most of the characteristics measured did not relate significantly to LLM use, including gender ($r = .10$, $p = .23$), parental education ($r = .07$, $p = .39$), parental income ($r = -.12$, $p = .21$), age ($r = .002$, $p = .98$), and even GPA ($r = .13$, $p = .10$). Class, however, did significantly predict LLM use, with those in upper-level courses relying more heavily than those in lower-level courses ($r = .28$, $p < .001$). This relationship may be due to the increased difficulty of these courses, or to the more frequent incorporation of boilerplate code in assigned work. The former is concerning given this type of work is where a significant amount of student growth takes place. But the latter, relying on LLMs to generate well-understood, reusable code is valuable

and should be encouraged.

6 Future Research

We believe that Techniques 4 and 5 merit further study, because the survey merely asked students to imagine how those techniques would affect them. In a follow-up, we intend to implement both techniques. First, to convince students that LLMs do have a "dark side", we will ask the question raised in Section 4.3.4 twice: before and after being presented with (what we deem to be) compelling evidence. Second, to see if we can instill in students a compunction to eschew LLMs, we will similarly repeat the question raised in Section 4.3.5 before and after having students solve some appropriate puzzles.

7 Conclusions

There is a fundamental conflict between the overarching goals of software developers — to create — and CS educators — to teach *how* to create. LLMs exacerbate the conflict. We must respond in a deliberate but thoughtful way to this transformative new technology, but need data before formulating next steps. As a result of this survey, we now have better insight into where our students stand. They are using LLMs extensively. The majority have a healthy respect for the technology, and a mature understanding of its shortcomings, how it can impede their learning experiences. While using them may be sorely tempting, they appear to be open to several strategies to help mitigate the detrimental effects of LLMs, and these require further study.

References

- [1] Mohammad Akyash, Kimia Zamiri Azar, and Hadi Mardani Kamali. “StepGrade: Grading Programming Assignments with Context-Aware LLMs”. In: *IEEE ISEC*. 2025, pp. 1–6. DOI: 10.1109/ISEC64801.2025.11147374.
- [2] Rina Azoulay, Tirza Hirst, and Shulamit Reches. “Large Language Models in Computer Science Classrooms: Ethical Challenges and Strategic Solutions”. In: *Appl. Sci.* 15 (2025).
- [3] Stella Biderman and Edward Raff. “Fooling MOSS detection with pre-trained language models”. In: *Proceedings of the 31st ACM international conference on information & knowledge management*. 2022, pp. 2933–2943.

- [4] Cecilia Ka Yuk Chan. “Students’ perceptions of AI-giarism: investigating changes in understandings of academic misconduct”. In: *Educ. Inf. Technol.* 30 (2025), pp. 8087–8108. DOI: 10.1007/s10639-024-13151-7.
- [5] Basak Demirok and Mucahid Kutlu. “AIGCodeSet: A New Annotated Dataset for AI Generated Code Detection”. In: *SIU*. 2024, pp. 1–4.
- [6] Ethan Dickey. *The Failure of Plagiarism Detection in Competitive Programming*. 2025. arXiv: 2505.08244 [cs.CY]. URL: <https://arxiv.org/abs/2505.08244>.
- [7] Alexander Hong and Gongbing Hong. “The Effectiveness of Coding LLMs and the Challenges in Teaching CS1/2: A Case Study”. In: *J. Comput. Sci. Coll.* 40 (2024), pp. 122–131.
- [8] Gregor Jošt, Viktor Taneski, and Sašo Karakatič. “The impact of large language models on programming education and student learning outcomes”. In: *Appl. Sci.* 14 (2024), p. 4115.
- [9] Lina Kloub and Aayush Gupta. In: *ASEE NE Section*. Apr. 2024.
- [10] Charles Koutcheme et al. “Evaluating Language Models for Generating and Judging Programming Feedback”. In: *Proc. SIGCSE*. 2025, pp. 624–630. DOI: 10.1145/3641554.3701791.
- [11] Wenhan Lyu et al. “Evaluating the Effectiveness of LLMs in Introductory Computer Science Education: A Semester-Long Field Study”. In: *Proc. L@S*. 2024, pp. 63–74. DOI: 10.1145/3657604.3662036.
- [12] André Nitze. *Future-proofing Education: A Prototype for Simulating Oral Examinations Using Large Language Models*. 2023. arXiv: 2401.06160 [cs.CY]. URL: <https://arxiv.org/abs/2401.06160>.
- [13] Marc Oedingen et al. “ChatGPT Code Detection: Techniques for Uncovering the Source of Code”. In: *AI 5* (2024), pp. 1066–1094. DOI: 10.3390/ai5030053.
- [14] J. Offutt. “Preventing Cheating Is Better Than Curing Cheating”. In: *EDULEARN*. 2024, pp. 6984–6991. DOI: 10.21125/edulearn.2024.1655.
- [15] Junghwan Rhee et al. “An Evaluation on the Impact of Large Language Models on Computer Science Curricula”. In: *J. Comput. Sci. Coll.* 40 (2024), pp. 70–80.
- [16] Stack Overflow. *2025 Stack Overflow Developer Survey*. 2025. URL: <https://survey.stackoverflow.co/2025/>.

- [17] Annapurna Vadaparty et al. *Integrating Large Language Models and Evaluating Student Outcomes in an Introductory Computer Science Course*. 2025. arXiv: 2510.18806 [cs.CY]. URL: <https://arxiv.org/abs/2510.18806>.
- [18] J.D. Zamfirescu-Pereira et al. “61A Bot Report: AI Assistants in CS1 Save Students Homework Time and Reduce Demands on Staff”. In: *Proc. SIGCSE*. 2025, pp. 1309–1315.

How Students Use AI in Project Development*

Catherine Stringfellow, N. (Thor) Lang, R. (Cooper) Ricketts,

Jose Marquez and Mekhi Prospere

Computer Science Department

Midwestern State University

Wichita Falls, TX 76308

`catherine.stringfellow@msutexas.edu`

`{thorseph4, rcricketts}@gmail.com`

`jmarquez1705@hotmail.com, meprospere0707@my.msutexas.edu`

Abstract

AI-assisted programming tools are increasingly embedded in programming environments, influencing how professionals and students write, understand, and debug code. As the use of these tools is becoming commonplace, it is important to examine their potential to support learning and productivity and the challenges they introduce for conceptual understanding, skill development, and academic integrity. This paper describes research on developer use of AI-assisted coding tools and presents student-reported experiences from coursework, offering insight into how AI is being incorporated into coding practices by students in educational settings.

1 Introduction

AI-assisted programming tools have become integrated into modern software development environments and are increasingly utilized in industry. These

*Copyright ©2026 by the Consortium for Computing Sciences in Colleges. Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the CCSC copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Consortium for Computing Sciences in Colleges. To copy otherwise, or to republish, requires a fee and/or specific permission.

tools can significantly improve developer productivity, particularly for repetitive and routine coding tasks. At the same time, important limitations exist, and as such require human judgment and oversight to ensure correctness and maintain software quality.

Students are also increasingly using AI tools, such as ChatGPT, GitHub Copilot, and IntelliSense, for their class projects. In a recent course on visual programming in C#, senior-level students were permitted to use AI as a tool to assist in both code-writing and GUI development. Course projects included a game (Battleship), a visual drawing tool, and a book inventory control system. Concepts covered inheritance and polymorphism, event driven programming visual components, such as text boxes, forms, buttons, menus, listboxes, etc. All students in the course were required to document their use of AI and to write an essay at the end of the semester on how they used it and to describe the benefits and limitations they recognized. Based on their essay, several students (the co-authors on this paper) were asked to submit their specific examples with screenshots to the instructor. These examples are described in this paper. This paper analyzes student-reported coursework experiences from their essays and the submitted examples to better understand how they are incorporating AI tools into their own coding practices in educational settings.

This paper is organized as follows: Section 2 describes the use of AI in code development, first by professional developers and then by students. Section 3 describe the use of AI in developing GUIs, again by developers and then by students. Sections 4 and 5 identifies some of the common benefits and limitations that students encountered. Finally, Section 6 presents a summary and some ideas for future work.

2 AI in Code Development

AI-assisted programming tools, such as GitHub Copilot, Visual Studio IntelliSense, and ChatGPT, are increasingly being integrated into modern software development activities, providing real-time support for code generation, auto-completion, debugging, and documentation. Understanding how these tools are used, as well as their benefits and limitations, has become important in both professional and educational contexts. The following subsections present complementary perspectives on AI use in coding, first summarizing findings from recent research and industry studies on developer adoption and impact, and then examining student-reported experiences that highlight practical usage patterns, learning considerations, and concerns related to over-reliance and academic integrity.

2.1 Developers' Use of AI Tools in Coding

Recent research and industry reports have examined the role of AI-assisted programming tools, particularly GitHub Copilot, in supporting software development through code generation, autocompletion, and algorithmic assistance [2, 4, 9, 11]. Copilot and IntelliSense, both integrated into Visual Studio, support a wide range of development tasks, including generating entire functions from natural-language prompts, providing context-aware code completions, and offering refactoring suggestions and test ideas [7]. A systematic review of 395 research papers published between 2017 and 2024 further identifies numerous software engineering tasks supported by large language models, such as code generation, bug fixing, documentation, code translation, and requirements analysis [6].

Visual Studio delivers many of these capabilities through IntelliSense's inline suggestion interface, which presents recommendations as developers type and allows them to be adopted immediately [13]. Studies show that the introduction of inline suggestions significantly increased the use of AI-assisted code changes by a factor of 3.5, compared to earlier code changes that were made based on hidden light-bulb menus. For example, when IntelliSense suggests constructor completions inline, most developers accept the recommendation and bypass manual typing altogether.

Empirical evaluations comparing Copilot-generated code to human-written solutions demonstrate that while Copilot can produce solutions for a broad set of fundamental algorithmic problems, including sorting, data structures, graph algorithms, and greedy algorithms, its results often exhibit lower correctness, optimality, and reproducibility than those created by human programmers [4]. Although Copilot-generated code frequently contains bugs or inconsistencies, experienced developers are typically able to repair these issues with relatively low effort, reinforcing Copilot's value as a productivity aid rather than a fully autonomous coding solution. In contrast, novice developers struggle to identify and correct AI-generated errors.

Complementary qualitative and survey-based studies of early Copilot users indicate that developers commonly use AI assistance to learn new programming languages, generate unit tests, and accelerate routine coding tasks [2]. High acceptance rates of AI-generated suggestions were positively correlated with perceived productivity, even when developers modified the suggested code. However, some users reported diminished understanding of code behavior and expressed concerns related to security, defensive coding practices, and overall code quality. Industry studies further report substantial productivity benefits from AI-assisted development, including faster task completion, reduced cognitive load during repetitive work, and increased job satisfaction through improved focus and sustained flow during complex problem-solving activities

[10, 12]. Taken together, the literature suggests that AI-based coding tools can meaningfully enhance developer productivity and experience when used critically and with human expertise. At the same time, these tools introduce trade-offs related to correctness, comprehension, and reliance, underscoring the continued need for human judgment and oversight in software development.

2.2 Students' Use of AI in Coding

Prather, et al. identify four interaction modes that students adopt when using AI code generators: exploration, acceleration, shepherding, and drifting [10].

- Exploration occurs when programmers are unsure of their next step and use AI to explore possible approaches.
- Acceleration describes situations where programmers know what they want to implement and use AI to complete tasks more quickly. A common example is accepting an IntelliSense or Copilot suggestion and then editing it to match specific requirements, reducing manual typing time.
- Shepherding involves actively guiding the AI through iterative prompts to achieve a desired result.
- Drifting occurs when programmers repeatedly follow incorrect AI-generated suggestions, reflecting difficulty in understanding the generated code.

Together, these modes illustrate the range of ways students interact with AI tools, from effective productivity support to less productive reliance that hinders understanding.

In a recent senior-level course, students reported using AI across their projects, primarily as a coding support and productivity tool, rather than a replacement for their own problem-solving. Tools such as GitHub Copilot, Visual Studio IntelliSense, and ChatGPT, were commonly used to generate starting points, such as function headers, class structures, inheritance hierarchies, algorithms, and documentation comments, as well as to assist with code completion and syntax recall. Several students emphasized using AI to reduce time spent brainstorming or searching documentation, while still reviewing, editing, and validating all generated code to avoid errors and academic misconduct. ChatGPT was often viewed as more successful for explaining concepts, generating higher-quality code drafts, or debugging persistent issues, while IntelliSense and Copilot supported in-line suggestions. Students consistently noted the importance of their own human oversight, expressing concern that overreliance on AI could weaken their problem-solving skills or reduce their conceptual understanding of the code, if they just accepted the code uncritically.

One student stated, "... I use AI with almost every programming assignment. However, this is mostly in ways of helping me organize and get an idea of what my program will need; very rarely will I ever actually use code generated

by AI. Copilot can be a great help in debugging as it is capable of spotting all sorts of errors that humans can easily miss. I typically use Copilot to clean up my code and reformat it as well as add in missing comments when I am done with programming.”

Overall, students primarily used AI productively through acceleration, using AI to speed up work they already understood and relying on Copilot or IntelliSense for syntax recall, formatting, commenting, and quick code completion, often accepting suggestions and then editing them. Next, students used AI for exploration, particularly when they felt stuck or as a “thinking partner” to explore options rather than produce final code. Students occasionally engaged in shepherding, using AI to clean up, refactor, or document code after initial implementation. Finally, students were largely aware of drifting and attempted to avoid it.

3 AI in GUI Development

Recent studies suggest that AI-assisted tools can play a meaningful role in supporting GUI development by helping developers move beyond an empty canvas and toward an initial design direction. The following sections describe how professional and student developers use these tool for GUI development.

3.1 AI Use by Developers in GUI Development

In much of the existing literature, AI use in graphical user interface (GUI) development is discussed only indirectly, most often framed as support for repetitive or routine coding tasks, rather than as a primary or creative design activity. For example, professional developers report using AI assistance to generate repetitive UI-related code, such as event wiring and control setup [2]. Similarly, studies indicate that AI-assisted coding tools such as Copilot are most effective when applied to repetitive or boilerplate tasks, an area that aligns closely with common GUI implementation activities, including configuring control properties and initializing forms, while remaining less reliable for complex logic or design decisions [4, 9, 11]. Notably, many of the studies do not explicitly describe AI being used for GUI layout design, visual aesthetics, or user-centered interface development. When prompted effectively, AI systems are able to reference user-defined variables and apply contextual reasoning to the placement of visual elements, sometimes offering descriptive layout guidance rather than direct code generation. For example, if a form requires a button, AI may either propose a hard-coded implementation or instead suggest where the button should be placed within the interface, such as positioning it “at the bottom left of the right-most panel, under the ‘help’ checkbox,” depending on how the prompt is framed [7] This flexibility allows developers to

maintain creative control, while still benefiting from AI-generated suggestions, although AI-produced GUI code often relies on explicit, hard-coded layouts rather than the drag-and-drop design approaches commonly used in Visual Studio C#.

Beyond individual suggestions, research indicates that AI can significantly accelerate initial GUI design by generating draft layouts or design prototypes that reduce the time spent brainstorming ideas. A 2025 study of professional UX/UI designers found that generative AI was particularly useful for producing first-draft prototypes that designers could refine through iterative improvement [3]. Similarly, other work suggests that AI supports not only repetitive UI planning tasks, but also creative exploration by proposing alternative or innovative interface designs [8]. These findings translate well to GUI development in environments such as C#, where AI can generate candidate sets of controls and layout configurations that developers can incrementally adapt. By enabling rapid iteration and collaborative refinement, AI helps programmers converge on effective GUI designs more quickly, while retaining full control over implementation decisions. AI-generated mockups and layout drafts can meaningfully reduce early layout effort by providing grouped controls and structural organization, replacing the need to start from a blank form and saving time on alignment, spacing, and naming [1]. This allows developers to focus more on event handling, logic, and user-centered decisions, rather than low-level layout concerns [5].

At the same time, researchers caution that AI's capabilities in visual programming remain limited. Studies evaluating ChatGPT and GPT-4 for visual programming report that these models struggle to integrate the spatial, logical, and programming skills required for robust visual interface development [12]. As a result, while AI can serve as a powerful assistant for planning and prototyping GUIs, overreliance on it for complete visual programming solutions may introduce risks, again underscoring the continued importance of human judgment and design expertise.

3.2 AI Use by Students in GUI Development

Notably missing from the literature is how students use AI for GUI development. The students in this study were required to explicitly describe their use of AI in developing GUIs for Visual Studio-based, form-application projects in C# in an essay at the end of the term. Primarily, they described their experiences were more about ideation and layout generation. Students described using AI tools to overcome early design challenges, particularly when deciding on layout and control placement. For example, one student reported writing all event handlers, input validation, and file operations independently, but turning to an AI tool for layout inspiration after struggling with the visual

arrangement of textboxes and labels. By prompting the AI with a description of the application's purpose, which was a book manager system, the student received multiple layout suggestions, which helped broaden their perspective, even though the final design remained intentionally simple. Students consistently characterized this as one of AI's key benefits: providing alternative viewpoints when they reached a design impasse.

Across projects, students also reported using AI to generate initial GUI layouts or working form examples from textual descriptions, refine prototypes through iterative suggestions, and identify potential controls or interface elements they had not previously considered. AI was particularly helpful for completing repetitive GUI-related code, such as configuring control properties, while leaving higher-level design decisions to the student. At the same time, some students deliberately avoided using AI for full front-end or visual design, noting that AI-generated interfaces often lacked human-centered aesthetics or usability. Instead, students selectively incorporated AI-inspired ideas to enhance their final GUIs, using AI as a source of inspiration for refinement, rather than as an authoritative designer. This pattern highlights a human-in-the-loop approach, where AI supports ideation and iteration, while students retain responsibility for usability, design quality, and implementation.

Overall, AI is most effectively used in GUI development as a planning and ideation aid rather than a complete design solution. Developers and students leverage AI to generate initial layouts and prototypes, translate textual descriptions into interface components, and rapidly explore multiple design alternatives during early brainstorming stages. AI also supports customization by helping tailor interfaces to different user types and suggesting creative or functional enhancements, such as improved control placement or additional features. By accelerating early design decisions and enabling quick iteration, AI reduces the friction of starting from a blank interface, while allowing developers to retain control over refinement, usability, and implementation details.

4 Common Benefits Reported by Students

Students largely viewed AI tools as productivity-enhancing assistants, rather than replacements for their own work. Personally, students felt AI helped save them significant time, reported that AI reduced frustration, sped up development, and helped them overcome mental blocks by providing ideas, alternative approaches, or explanations when they were stuck.

In GUI development, students found AI particularly beneficial for brainstorming layouts and discovering controls they might not have otherwise considered, helping them move forward when uncertain about design decisions. Several students also noted that AI reduced reliance on office hours or extensive

documentation searches, making their workflow more efficient and self-directed.

5 Limitations and Frustrations

At the same time, students expressed clear personal limitations and concerns. A recurring issue was that AI-generated code was often incorrect, overly complex, or misaligned with assignment requirements, requiring careful review, debugging, and repeated prompting. Some students felt that reliance on AI could create gaps in their understanding, especially when Copilot completed code too quickly or obscured the reasoning behind solutions. Ethical concerns about academic misconduct were prominent, leading many students to intentionally avoid submitting AI-generated code verbatim and to use AI mainly for hints (even though they were permitted to use it with proper documentation). In GUI development, several students felt AI performed poorly at producing human-centered or aesthetically pleasing interfaces, limiting its usefulness beyond ideation. Finally, students acknowledged the risk of over-reliance on AI tools, noting that while these systems made development easier, they were worried that this could weaken their long-term problem-solving skills and confidence, if used without restraint or critical judgment.

One student stated, “Speaking in terms on the over-reliance on the use of AI in writing code, I can say from personal experience that the integration of LLMS with programming has been both a blessing and a curse. However, this has come at the cost of developing problem-solving skills as it is now easier to rely on Copilot or ChatGPT to solve issues with the code.”

6 Conclusion

This paper examined student-reported experiences on how AI tools complement their workflows in C# projects, particularly in code writing and GUI development. The findings show that AI was used to assist rather than replace student problem-solving by helping streamline repetitive tasks, generate initial structures, and support early-stage design decisions. Students reported gains in efficiency and creativity, as AI made it easier to start tasks and enabled rapid exploration of ideas, while still requiring their judgment to ensure correctness, usability, and alignment with their own learning objectives.

At the same time, the results highlight the importance of intentional and guided use of AI in student projects. Effective use of AI was characterized by treating it as a source of suggestions, scaffolding, and inspiration, rather than as an authoritative solution generator. Based on the observed experiences, recommended practices include encouraging students to critically review and modify AI-generated output, use AI primarily for brainstorming, prototyping,

and repetitive tasks, and reflect on how AI suggestions influence their design and implementation choices. Explicit discussion of academic integrity, transparency in AI use, and alignment with course learning outcomes can further help ensure that AI enhances learning rather than undermines it.

Looking ahead, AI tools are likely to become more deeply integrated into educational programming environments, offering richer context awareness, improved support for visual and spatial reasoning, and stronger alignment with pedagogical goals. As these tools evolve, they may better support not only code generation but also higher-level design reasoning, adaptive feedback, and personalized learning support. However, this evolution underscores the continued need for instructional guidance and curriculum design that emphasizes foundational skills, conceptual understanding, and ethical use. Overall, AI presents significant opportunities to enhance visual development activities and student learning, provided it is adopted thoughtfully, critically, and in ways that preserve the central role of human expertise and creativity.

References

- [1] S. Abrahão et al. “Software Engineering by and for Humans in an AI Era”. In: *ACM Transactions on Software Engineering and Methodology* 34.5 (2025), pp. 1–46. DOI: <https://doi.org/10.1145/3715111>.
- [2] C. Bird et al. “Taking Flight with Copilot: Early Insights and Opportunities of AI-Powered Pair Programming Tools”. In: *Communications of the ACM* 66.6 (2023), pp. 56–62.
- [3] V. Chen et al. “Need Help? Designing Proactive AI Assistants for Programming”. In: *Proceedings of the CHI Conference on Human Factors in Computing Systems*. CHI '25. Yokohama, Japan: ACM, 2025, pp. 1–18. DOI: 10.1145/3706598.3714002. URL: <https://doi.org/10.1145/3706598.3714002>.
- [4] A. Dakhel et al. “GitHub Copilot AI Pair Programmer: Asset or Liability?” In: *Journal of Systems and Software* 203 (2023), p. 111734. DOI: <https://doi.org/10.1016/j.jss.2023.111734>.
- [5] Forasoft. *How AI Tools Assist Designers in Creating UI/UX Designs for Software Products*. <https://forasoft.medium.com/how-ai-tools-assist-designers-in-creating-ui-ux-designs-for-software-products-ba187581b9b3>. Aug. 2025.
- [6] A. Hou et al. “Large Language Models for Software Engineering: A Systematic Literature Review”. In: *ACM Transactions on Software Engineering and Methodology* 33.8 (2024), pp. 1–79.

- [7] Microsoft. *IntelliSense in Visual Studio*. <https://learn.microsoft.com/en-us/visualstudio/ide/using-intellisense?view=visualstudio> (accessed February 9, 2026). Oct. 2025.
- [8] B. Okpala. *Examining the Impact of Generative AI on UX/UI Design*. <https://dx.doi.org/10.2139/ssrn.5312384> (accessed November 24, 2025). Nov. 2024.
- [9] S. Peng et al. *The Impact of AI on Developer Productivity: Evidence from GitHub Copilot*. <https://doi.org/10.48550/arXiv.2302.06590> (accessed January 9, 2026). Feb. 2023.
- [10] J. Prather et al. “The Robots are Here: Navigating the Generative AI Revolution in Computing Education”. In: *Proceedings of the 2023 Working Group Reports on Innovation and Technology in Computer Science Education*. ITiCSE-WGR ’23. Turku, Finland: ACM, 2023, pp. 1–52.
- [11] D. Ramel. “A Year In, GitHub Measures AI-Based Copilot’s Productivity Boost”. In: *Visual Studio Magazine* (Sept. 2022). Published Sep 13, 2022.
- [12] A. Singla. “Evaluating ChatGPT and GPT-4 for Visual Programming”. In: *Proceedings of the 2023 ACM Conference on International Computing Education Research*. ICER ’23. Chicago, IL, USA: ACM, 2023.
- [13] P. Vaithilingam et al. “Towards More Effective AI-Assisted Programming: A Systematic Design Exploration to Improve Visual Studio IntelliSense’s User Experience”. In: *Proceedings of the 45th International Conference on Software Engineering: Software Engineering in Practice*. ICSE-SEIP ’23. New York, NY, USA: ACM, 2023, pp. 185–195.

Reviewers — 2026 CCSC South Central Conference

Syed Ahmed	<i>York Technical College, Rock Hill, SC</i>
Eduardo Colmenares-Diaz ..	<i>Midwestern State University, Wichita Falls, TX</i>
Jason Dill	<i>Webster University, Webster Groves, MO</i>
Anne Marie Eubanks ..	<i>Stephen F. Austin State University, Nacogdoches, TX</i>
Dushanthi Herath	<i>Maryville University, Creve Coeur, MO</i>
Essa Imhmed	<i>Eastern New Mexico University, NM</i>
Tina Johnson	<i>Midwestern State University, Wichita Falls, TX</i>
Shyam Karrah	<i>The University of Texas at Dallas, TX</i>
Michael Kart	<i>St. Edward's University, Austin, TX</i>
Jennifer Lavergne	<i>McNeese State University, LA</i>
Vipin Menon	<i>McNeese State University, Lake Charles, LA</i>
José Carlos Metrolho	<i>Polytechnic Institute of Castelo Branco, Castelo Branco, Portugal</i>
Mika Morgan	<i>Midwestern State University, Wichita Falls, TX</i>
Muhammad Rahman	<i>Clayton State University, Morrow, GA</i>
Kenneth Rouse	<i>LeTourneau University, Longview, TX</i>
Christian Servin	<i>El Paso Community College, El Paso, TX</i>
Bilal Shebaro	<i>St. Edward's University, Austin, TX</i>
Bill Siever	<i>Washington University in St. Louis, St. Louis, MO</i>
Fei Zuo	<i>University of Central Oklahoma, Edmond, OK</i>