Generative AI in the Computing Classroom
Symptoms and (Possible) Antidotes

Brett A. Becker

University College Dublin

- brettbecker.com Slides & Papers Available
- brett.becker@ucd.ie
- @brettabecker

CERG UCD
cerg.ucd.ie

- Cheating!!!!
- How do we assess?

Both are old problems. The first already has solutions. The second only has approximations.

AI could change:

how we teach
what we teach
when we teach
whom we teach.

Slide borrowed from ITiCSE 2023 Keynote*

It seems we are starting to move in this direction now?

Reshape the way programming (and other things) are learned, and the whole traditional computing BSc curriculum might change

- This could change who is, and who is not, attracted to computing
- This could be a big agent of change in Broadening Participation in Computing
- This could narrow the computing divide
- We may not be able to control this, but we can influence it

*www.brettbecker.com/publications/#iticse23keynote

You probably know that these tools are getting better, and accelerating in capability (and use) with no slowing down in sight

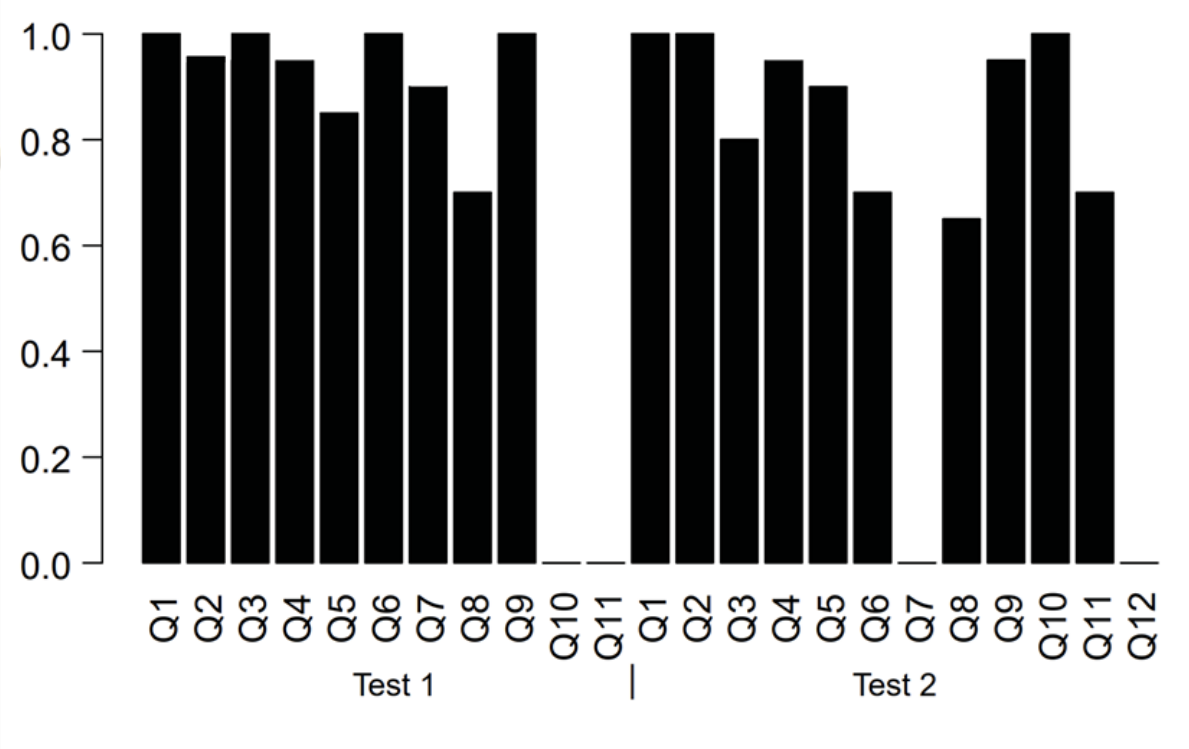
At first there was panic – Assessment! Cheating!

Then the "capabilities" papers – how good are these models?

Then the theoretical "doom and excitement" papers – what problems and opportunities will GenAI bring (maybe)

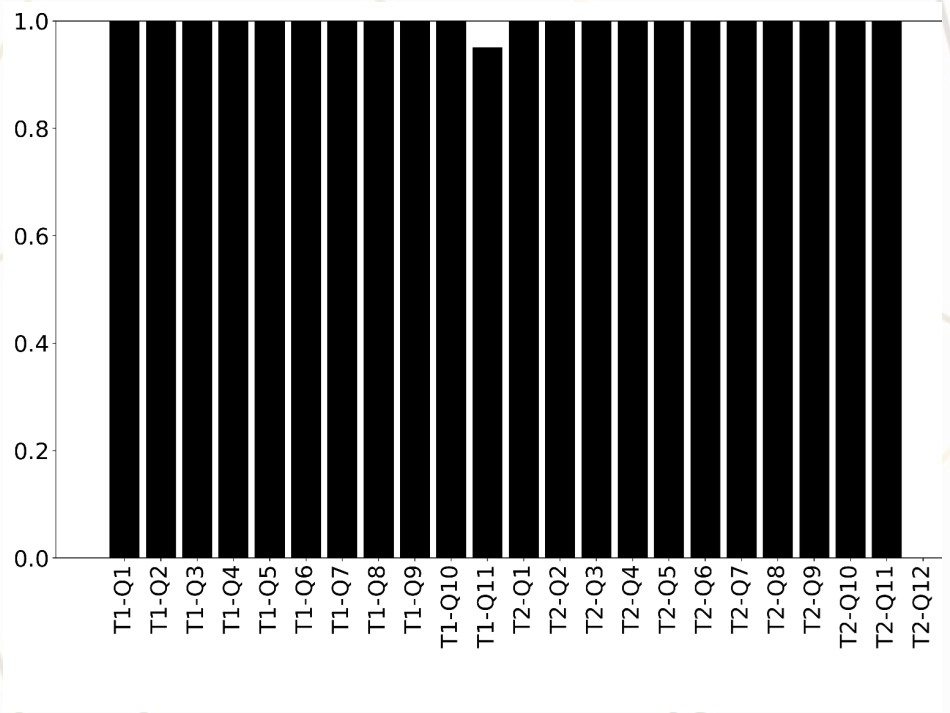
The Robots Are Coming: Exploring the Implications of OpenAI Codex on Introductory Programming

Codex, 2021 (CS1)



The Robots are Here: Navigating the Generative AI Revolution in Computing Education

GPT-4, summer 2023
(same exams as 2021)



James Prather, Paul Denny, Juho Leinonen, Brett A. Becker, Ibrahim Albluwi, Michelle Craig, Hieke Keuning, Natalie Kiesler, Tobias Kohn, Andrew Luxton-Reilly, Stephen MacNeil, Andrew Petersen, Raymond Pettit, Brent N. Reeves, and Jaromir Savelka. 2023. 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). Association for Computing Machinery, New York, NY, USA, 108–159. https://doi.org/10.1145/3623762.3633499. www.brettbecker.com/publications

Well, people (obviously) started using GenAI in the classroom, albeit in nascent and various was, What we need to do is research that.

The Robots are Here: Navigating the Generative AI Revolution in Computing Education

James Prather*
Abilene Christian University
Abilene, Texas, USA
james.prather@acu.edu

Paul Denny*
University of Auckland
Auckland, New Zealand
paul@cs.auckland.ac.nz

Juho Leinonen*
University of Auckland
Auckland, New Zealand
juho.leinonen@auckland.ac.nz

Brett A. Becker*
University College Dublin
Dublin, Ireland
brett.becker@ucd.ie

Ibrahim Albluwi
Princess Sumaya University for Technology
Amman, Jordan
i.albluwi@psut.edu.jo

Michelle Craig
University of Toronto
Toronto, Canada
mcraig@cs.toronto.edu

Hieke Keuning
Utrecht University
Utrecht, The Netherlands
h.w.keuning@uu.nl

Natalie Kiesler
DIPF Leibniz Institute for Research and Information in Education
Frankfurt am Main, Germany
kiesler@dipf.de

Tobias Kohn
Karlsruhe Institute of Technology
Karlsruhe, Germany
tobias.kohn@kit.edu

Andrew Luxton-Reilly
University of Auckland
Auckland, New Zealand
andrew@cs.auckland.ac.nz

Stephen MacNeil
Temple University
Philadelphia, Pennsylvania, USA
stephen.macneil@temple.edu

Andrew Petersen
University of Toronto Mississauga
Mississauga, Canada
andrew.petersen@utoronto.ca

Raymond Pettit
University of Virginia
Charlottesville, Virginia, USA
raymond.pettit@virginia.edu

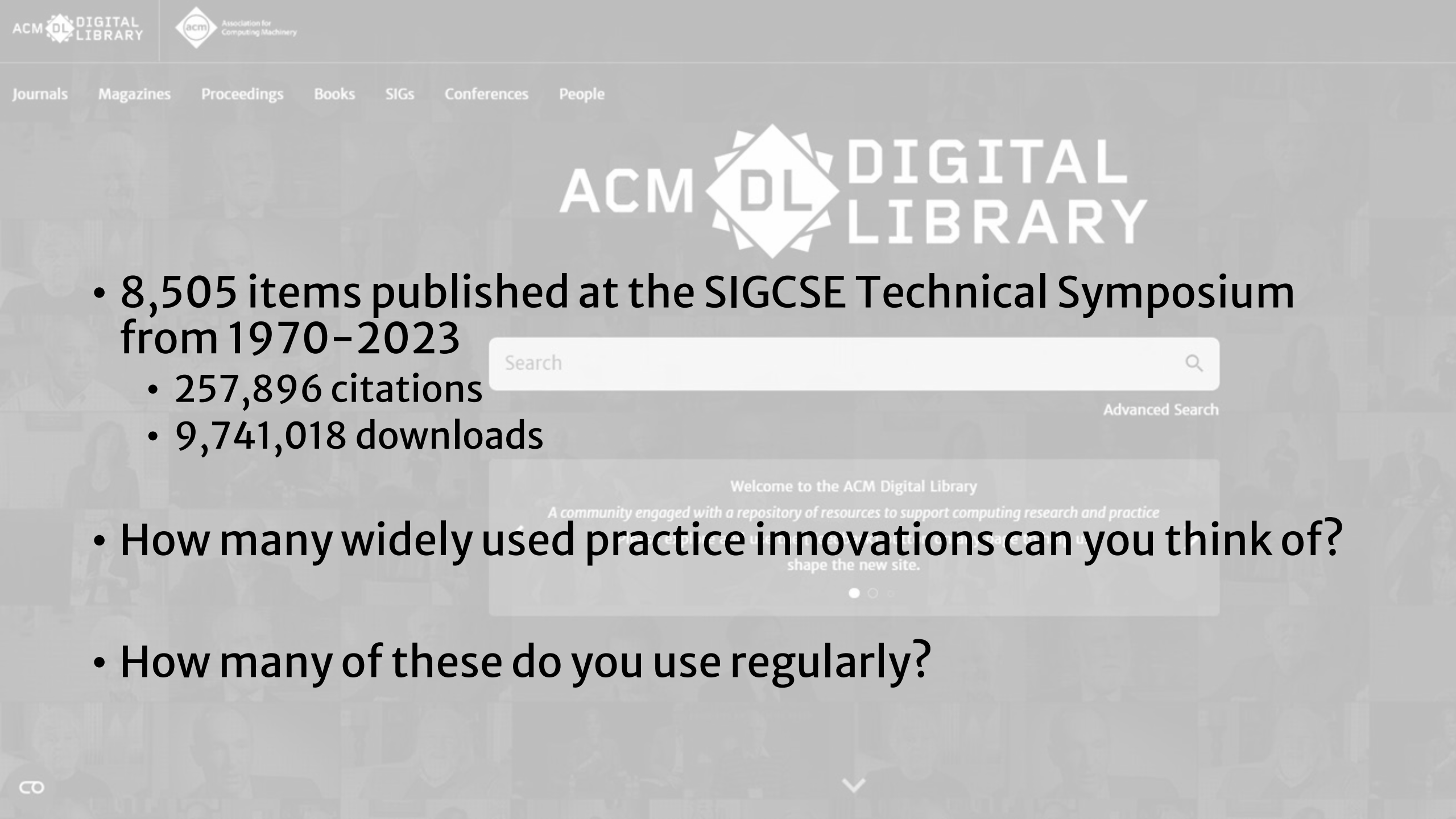
Brent N. Reeves
Abilene Christian University
Abilene, Texas, USA
brent.reeves@acu.edu

Jaromir Savelka
Carnegie Mellon University
Pittsburgh, Pennsylvania, USA
jsavelka@cs.cmu.edu

. https://doi.org/10.1145/3623762.3633499

So where are we now?

- **GenAI is probably the biggest change in education ever.**
  - And we haven’t begun to see the real effects yet.
  - We can't predict the future - especially here in this context.
- What about the printing press for instance?
  - It took generations for the impact to be felt, and it was relatively predictable.
  - Sure eventually everyone would have school books but it was not overnight. It took decades from the printing press to the average student having probably *a* book.
  - And the effect of the books took ages to take hold.



- 8,505 items published at the SIGCSE Technical Symposium from 1970–2023
  - 257,896 citations
  - 9,741,018 downloads
- How many widely used practice innovations can you think of?
- How many of these do you use regularly?

- GenAI was/is more or less instant in effect and it is improving at light speed
- We don’t know what the eventual effects will be (if there ever are “eventual effects”)
- Books ran their course. The internet largely killed them.
- Will AI ever run it’s course?

So WHERE are we now?

We are starting to see actual problems (symptoms)

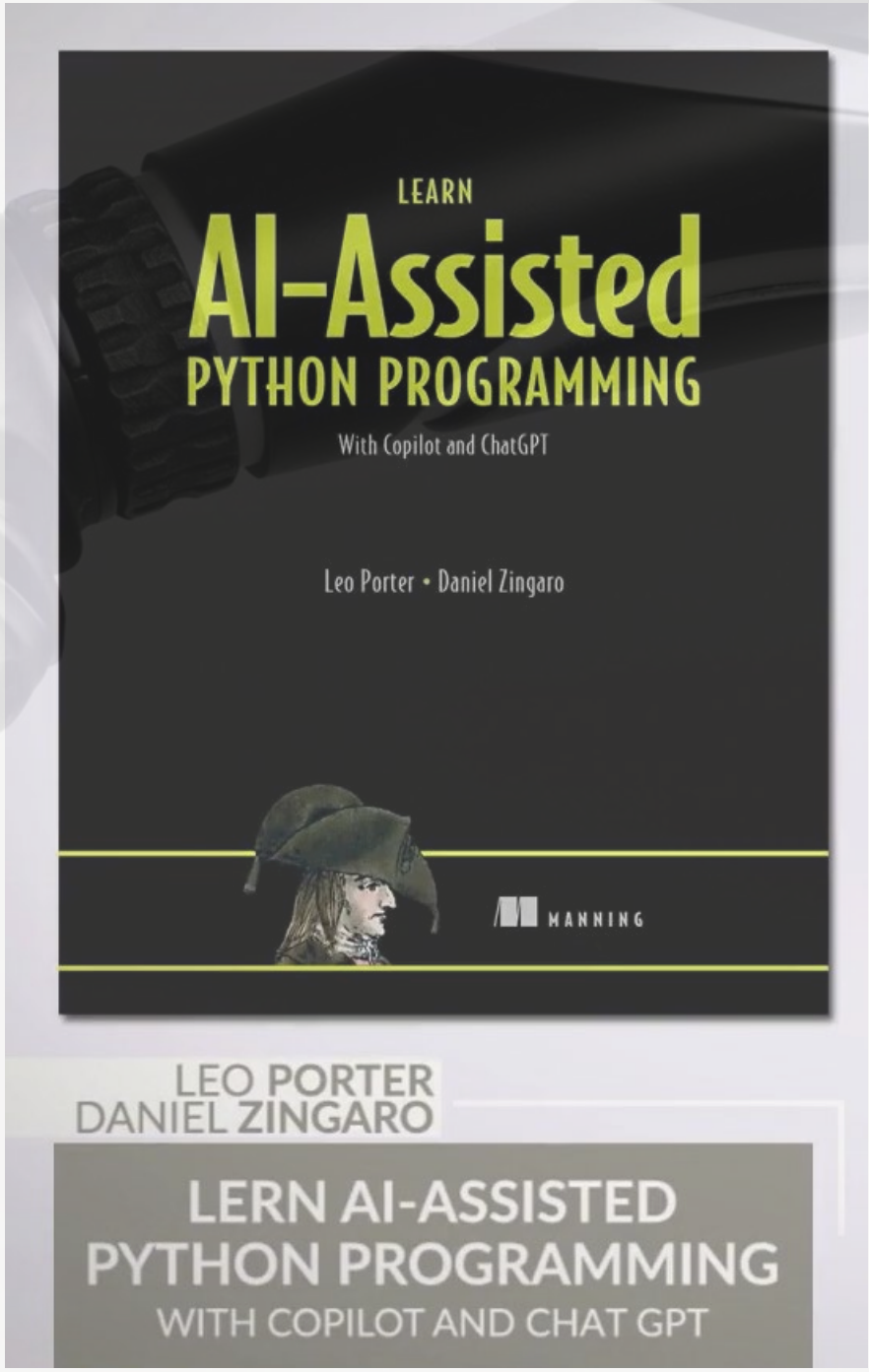
- At least we think – in real trials (Porter & Zingaro, CS50, others)

We are also starting to see actual positives (antidotes)

- At least we think – in real trials (Porter & Zingaro, CS50, others)

These are also *starting* to come to light in small research trials

- But a LOT more research is needed to confirm these
- Another But – the technology is advancing faster than these trials and research studies.
  - What do we do about that?



Harvard John A. Paulson School of Engineering and Applied Sciences

SEARCH MENU

NEWS

All News Stories

“Quacking” into computer programming

New AI bot helps students learn to code

By Matt Goisman | Press contact
January 30, 2024

Harvard students enrolled in CS50, which teaches the fundamentals of computer programming for a range of languages, had some special help with their final projects this year — an AI chatbot called the CS50 Duck.

Beginning last summer, course instructors began to integrate a suite of artificial intelligence, including the “CS50 Duck,” software built atop a large language model that helps students check their code and find answers to questions related to their coursework. The Duck is named for the class mascot, a yellow rubber duck, an allusion to “rubber duck debugging,” whereby new programmers are encouraged to talk through their programming problems with, in the absence of a fellow human, a rubber duck.

David J. Malan, Gordon McKay Professor of the Practice of Computer Science at the Harvard John A. Paulson School of Engineering and Applied Sciences (SEAS), who teaches the course, first deployed the tools with a small group of students this summer before integrating them into CS50 sections this fall.

“The Duck was my best friend,” said first-year student Hannah Nguyen, who

The yellow rubber duck is the class mascot for CS50, which this semester also integrated an artificial intelligence chatbot called the CS50 Duck to help students check their code (Eliza Grinnell)

https://seas.harvard.edu/news/2024/01/quacking-computer-programming

Teaching CS50 with AI

Leveraging Generative Artificial Intelligence in Computer Science Education

Rongxin Liu
Harvard University
Cambridge, MA, USA
rongxinliu@cs50.harvard.edu

Carter Zenke
Harvard University
Cambridge, MA, USA
carter@cs50.harvard.edu

Charlie Liu
Yale University
New Haven, CT, USA
charlie.liu@yale.edu

Andrew Holmes
Harvard University
Cambridge, MA, USA
aholmes@college.harvard.edu

Patrick Thornton
Harvard University
Cambridge, MA, USA
patrickthornton@college.harvard.edu

David J. Malan
Harvard University
Cambridge, MA, USA
malan@harvard.edu

https://cs.harvard.edu/malan/publications/V1fp0567-liu.pdf

Using Large Language Models to Enhance Programming Error Messages

Juho Leinonen
Aalto University
Espoo, Finland
juho.2.leinonen@aalto.fi

Arto Hellas
Aalto University
Espoo, Finland
arto.hellas@aalto.fi

Sami Sarsa
Aalto University
Espoo, Finland
sami.sarsa@aalto.fi

Brent Reeves
Abilene Christian University
Abilene, Texas, USA
brent.reeves@acu.edu

Paul Denny
The University of Auckland
Auckland, New Zealand
paul@cs.auckland.ac.nz

James Prather
Abilene Christian University
Abilene, Texas, USA
james.prather@acu.edu

Brett A. Becker
University College Dublin
Dublin, Ireland
brett.becker@ucd.ie

ABSTRACT

A key part of learning to program is learning to understand programming error messages. They can be hard to interpret and identifying the cause of errors can be time-consuming. One factor in this challenge is that the messages are typically intended for an audience that already knows how to program, or even for programming environments that then use the information to highlight areas in code. Researchers have been working on making these errors more novice friendly since the 1960s, however progress has been slow. The present work contributes to this stream of research by using large language models to enhance programming error messages with explanations of the errors and suggestions on how to fix them. Large language models can be used to create useful and novice-friendly enhancements to programming error messages that sometimes surpass the original programming error messages in interpretability and actionability. These results provide further evidence of the benefits of large language models for computing educators, highlighting their use in areas known to be challenging for students. We further discuss the benefits and downsides of large language models and highlight future streams of research for enhancing programming error messages.

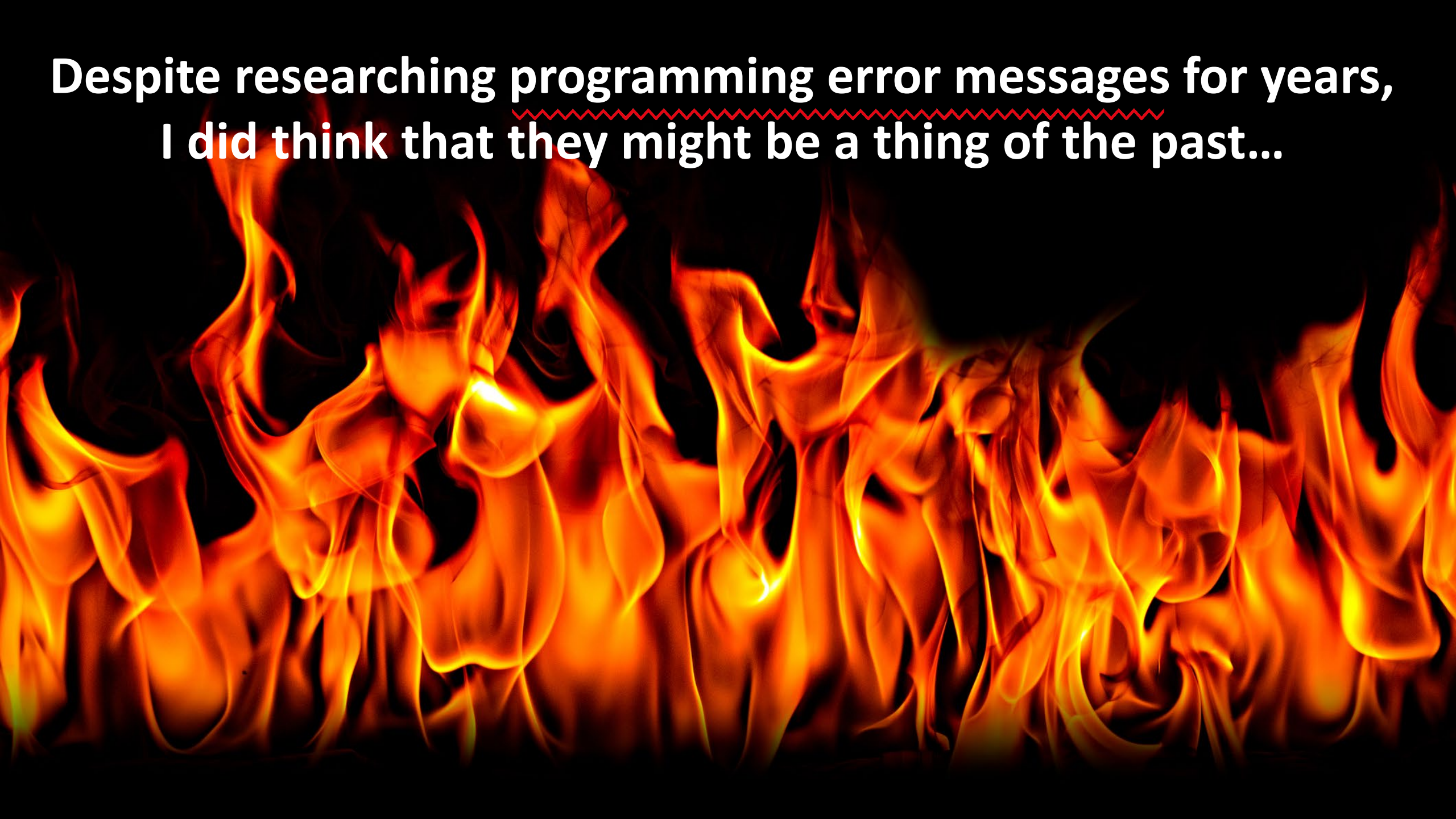
ACM Reference Format:
Juho Leinonen, Arto Hellas, Sami Sarsa, Brent Reeves, Paul Denny, James Prather, and Brett A. Becker. 2023. Using Large Language Models to Enhance Programming Error Messages. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1 (SIGCSE 2023), March 15–18, 2023, Toronto, ON, Canada.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3545945.3569770

1 INTRODUCTION

Programming Error Messages (PEMs) can be notoriously difficult to decipher, especially for novices [33], possibly to the extent that they contribute to the perception that programming is overly challenging [6]. Eye-tracking studies reveal that novices read error messages and spend a substantial amount of programming time trying to understand them [4]. Instructors report that they spend a considerable amount of time helping novices with these often cryptic messages [15, 31, 32, 37]. It is also known that error message presentation affects novice programming behavior [23]. For over six decades, researchers have attempted to improve these messages, and still there is a call for more work on the topic [7]. Some recent attempts have been made to put error messages into more natural

Juho Leinonen, Arto Hellas, Sami Sarsa, Brent Reeves, Paul Denny, James Prather, and Brett A. Becker. 2023. Using Large Language Models to Enhance Programming Error Messages. In Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1 (SIGCSE 2023). Association for Computing Machinery, New York, NY, USA, 563–569. https://doi.org/10.1145/3545945.3569770. www.brettbecker.com/publications

Despite researching programming error messages for years,
I did think that they might be a thing of the past…



- Claim (~99% confidence): Generative AI is better at programming than 99% of our students **at your institution** at the end of year 1 –
  - At least in terms of current assessment and materials

- ***Are we still teaching introductory programming the same way we were 2 years ago?***
- ***What should we do differently?***
- ***How can GenAI be leveraged for better learning?***
- ***How does this affect other courses?***

Not-bold claim (100.00% confidence): ***GenAI will cause problems and we need to deal with these***

We have developed:

- A new type of programming problem for education: **Prompt Problems**
- Prompt Problems are designed to help students learn how to write effective prompts. It's more than a copy/paste of the problem itself.

Prompt Problems: A New Programming Exercise for the Generative AI Era

Paul Denny
University of Auckland
Auckland, New Zealand
paul@cs.auckland.ac.nz

Juho Leinonen
University of Auckland
Auckland, New Zealand
juho.leinonen@auckland.ac.nz

James Prather
Abilene Christian University
Abilene, TX, USA
james.prather@acu.edu

Andrew Luxton-Reilly
University of Auckland
Auckland, New Zealand
a.luxton-reilly@auckland.ac.nz

Thezyrie Amarouche
University of Toronto Scarborough
Toronto, ON, Canada
thezyrie.amarouche@mail.utoronto.ca

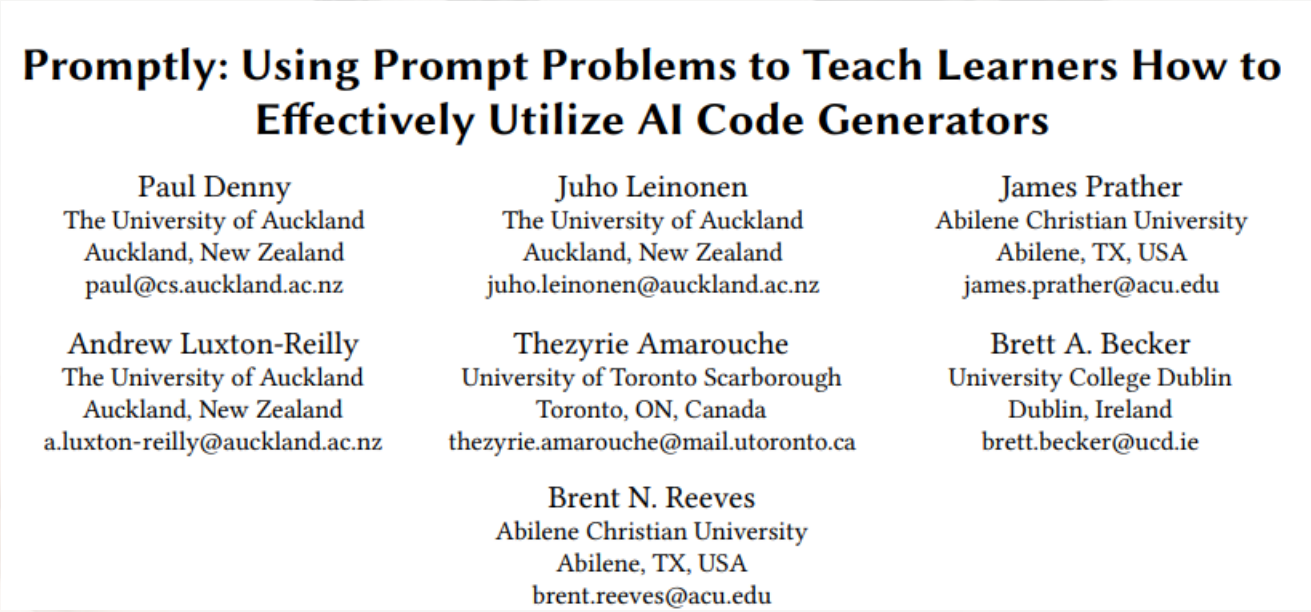
Brett A. Becker
University College Dublin
Dublin, Ireland
brett.becker@ucd.ie

Brent N. Reeves
Abilene Christian University
Abilene, Texas, USA
brent.reeves@acu.edu

https://dl.acm.org/doi/pdf/10.1145/3626252.3630909

We have developed:

- A new tool that can be used by students to practice Prompt Problems: **Promptly** (research prototype available for your use at *promptly-sigcse.web.app*)
- *Piloted* on several institutions on three continents in English, Portuguese, and Arabic with hundreds (probably thousands) of students.



Promptly: Using Prompt Problems to Teach Learners How to Effectively Utilize AI Code Generators

Paul Denny
The University of Auckland
Auckland, New Zealand
paul@cs.auckland.ac.nz

Juho Leinonen
The University of Auckland
Auckland, New Zealand
juho.leinonen@auckland.ac.nz

James Prather
Abilene Christian University
Abilene, TX, USA
james.prather@acu.edu

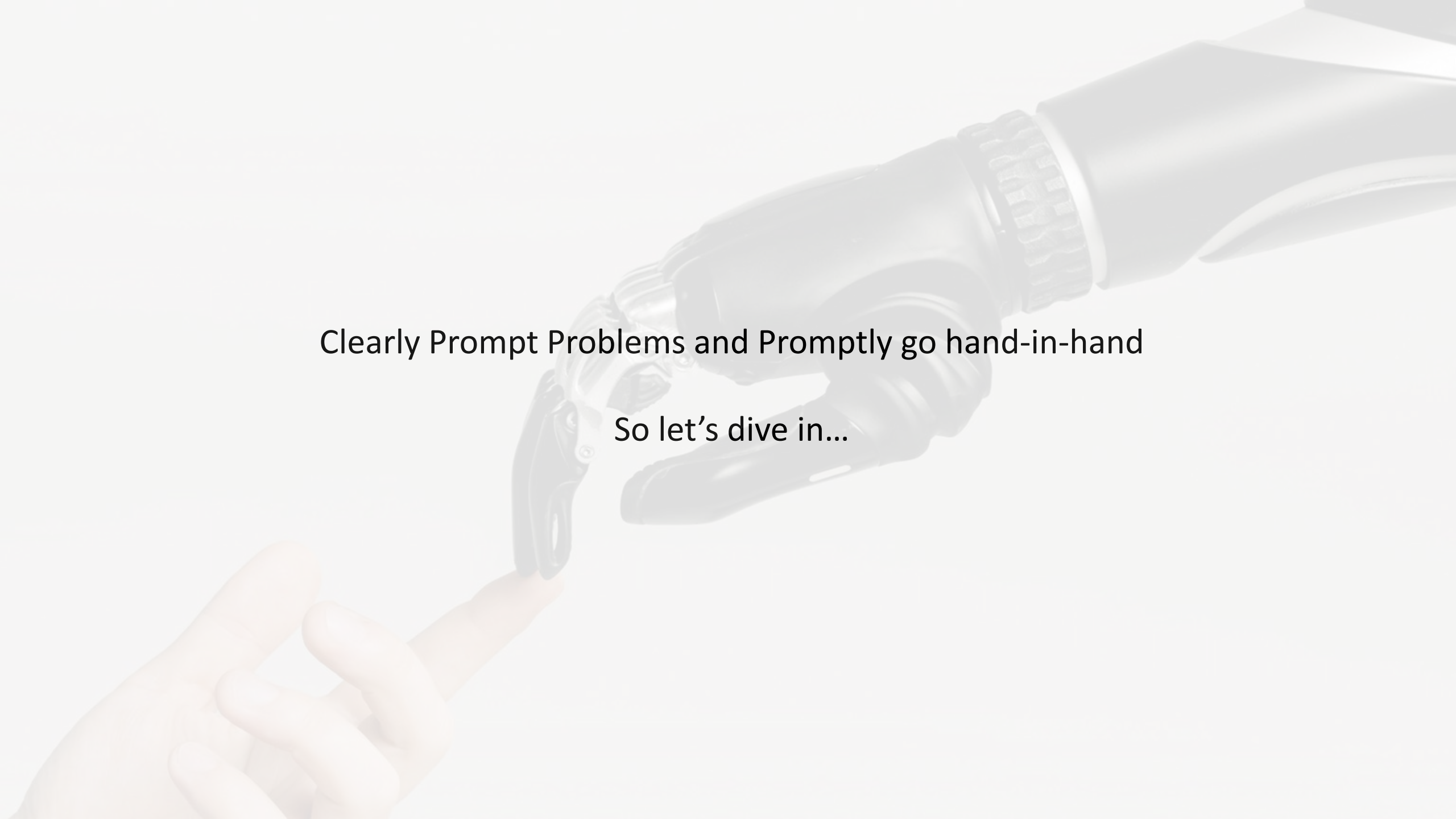
Andrew Luxton-Reilly
The University of Auckland
Auckland, New Zealand
a.luxton-reilly@auckland.ac.nz

Thezyrie Amarouche
University of Toronto Scarborough
Toronto, ON, Canada
thezyrie.amarouche@mail.utoronto.ca

Brett A. Becker
University College Dublin
Dublin, Ireland
brett.becker@ucd.ie

Brent N. Reeves
Abilene Christian University
Abilene, TX, USA
brent.reeves@acu.edu

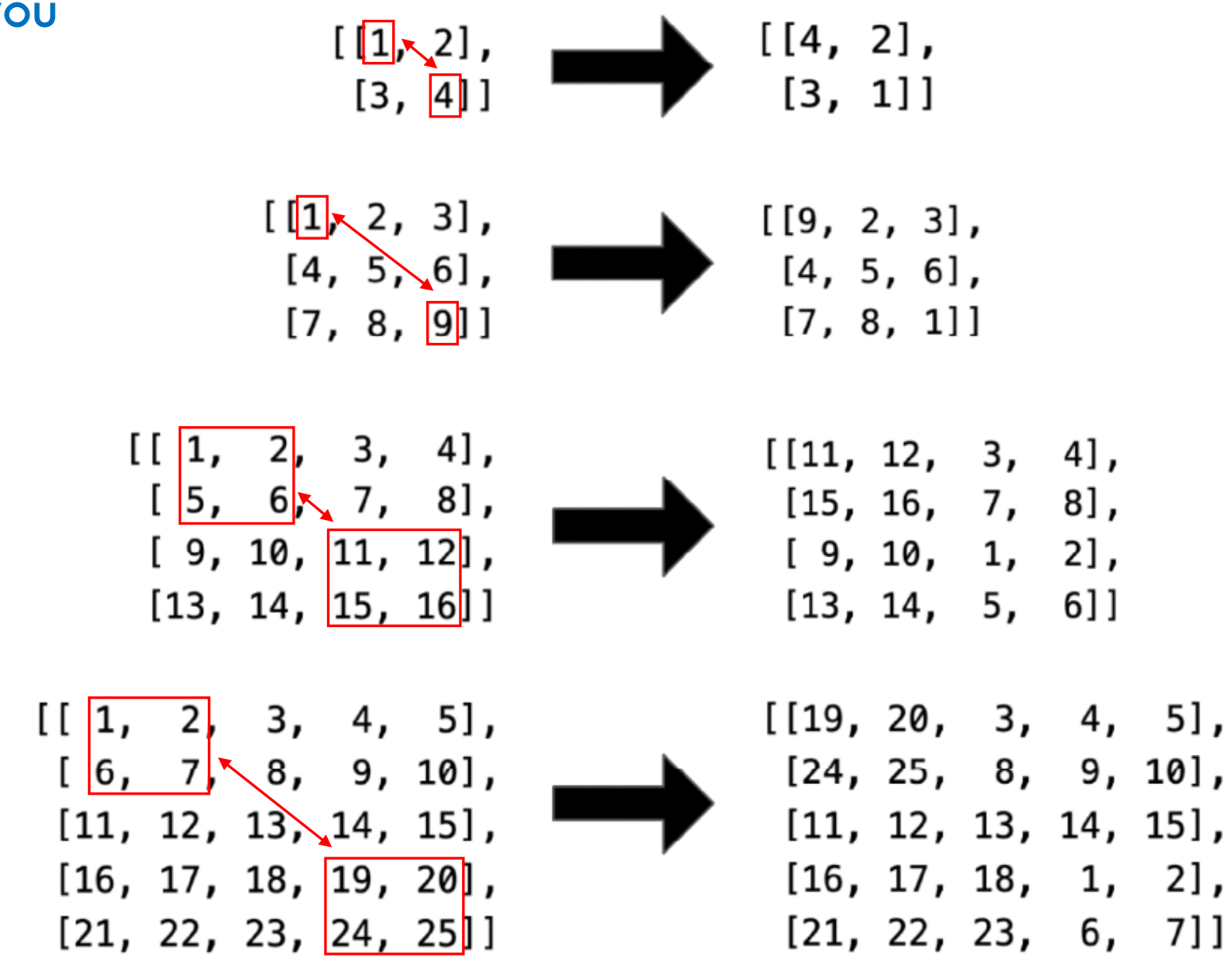
https://arxiv.org/pdf/2307.16364



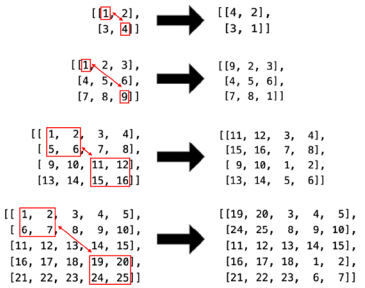
Clearly Prompt Problems and Promptly go hand-in-hand

So let’s dive in…

- How would you explain this problem?



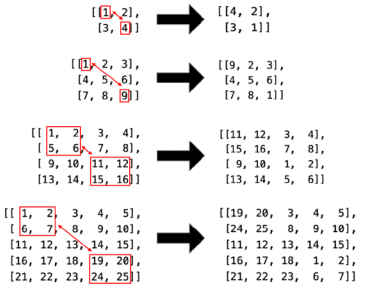
- A conversation between a (real) student and ChatGPT:



Prompt 1: Hi, I want you to write a Python program.

ChatGPT response 1: *Sure, what would you like the program to do?*

- A conversation between a (real) student and ChatGPT:



Prompt 1: Hi, I want you to write a Python program.

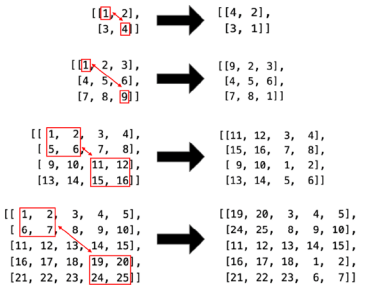
ChatGPT response 1: *Sure, what would you like the program to do?*

Prompt 2: I have a square matrix, and I want to swap the first half of the rows with the second half of the rows.

ChatGPT response 2: *Here's a Python program that should do what you're asking for:*

```
def swap_matrix_rows(matrix):
```

- A conversation between a (real) student and ChatGPT:

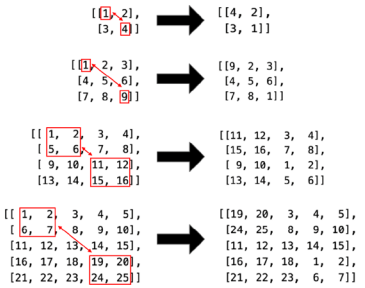


Prompt 3: That's not exactly what I meant. I actually want to swap the first half of the elements in each row with the corresponding elements in the second half of the rows.

ChatGPT response 3: *No problem, here's an updated version of the Python program that should swap the first half of the elements in each row with the corresponding elements in the second half of the rows:*

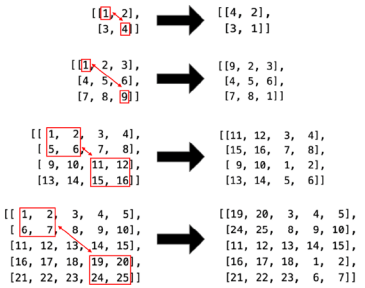
```
def swap_matrix_elements(matrix):
```

- A conversation between a (real) student and ChatGPT:



Prompt 4: Oops, sorry, I wasn't clear. I actually want to swap the first half of the elements in the first half of the rows with the corresponding elements in the last half of the rows.

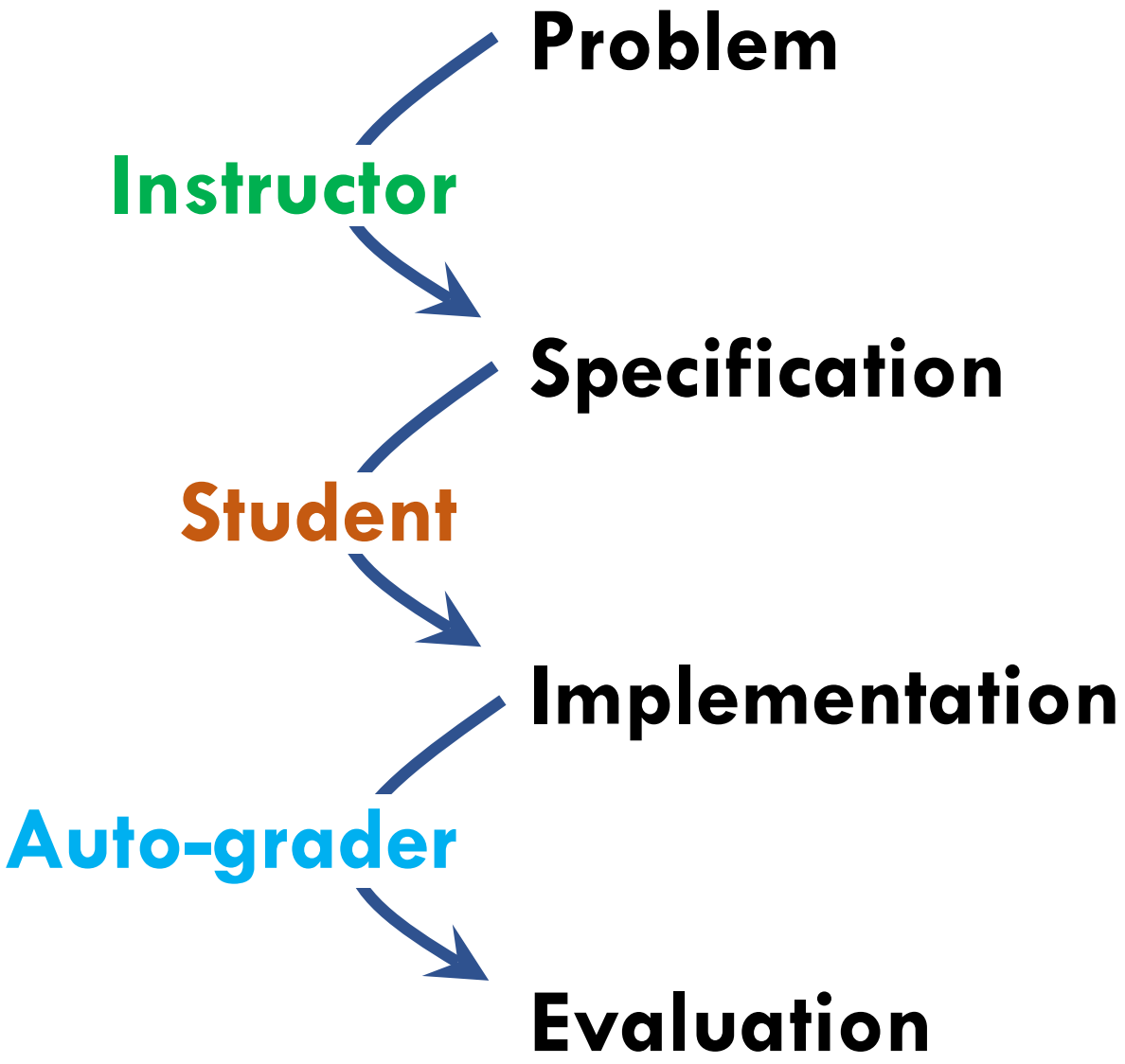
- A conversation between a (real) student and ChatGPT:

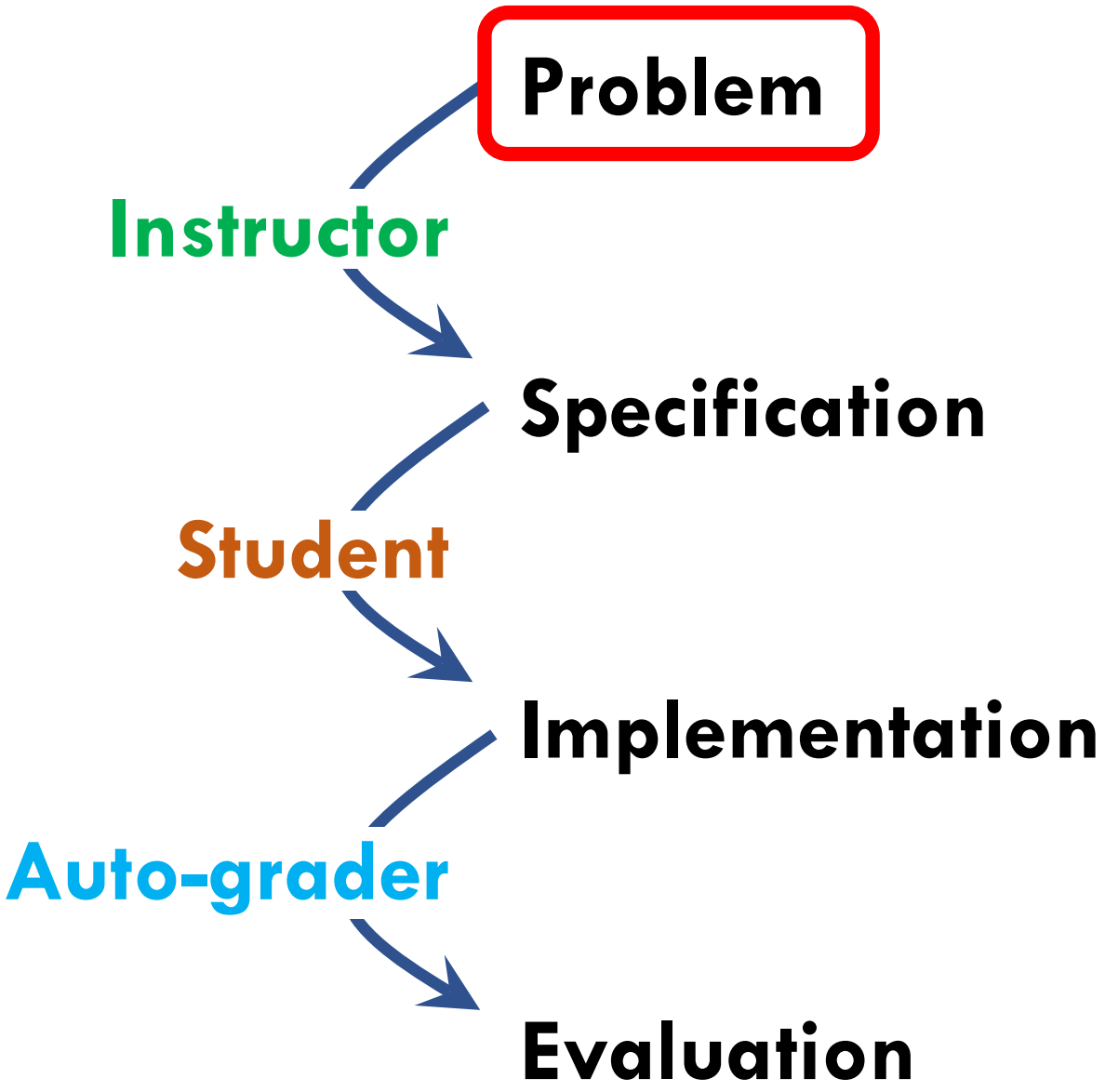


Prompt 4: Oops, sorry, I wasn't clear. I actually want to swap the first half of the elements in the first half of the rows with the corresponding elements in the last half of the rows.

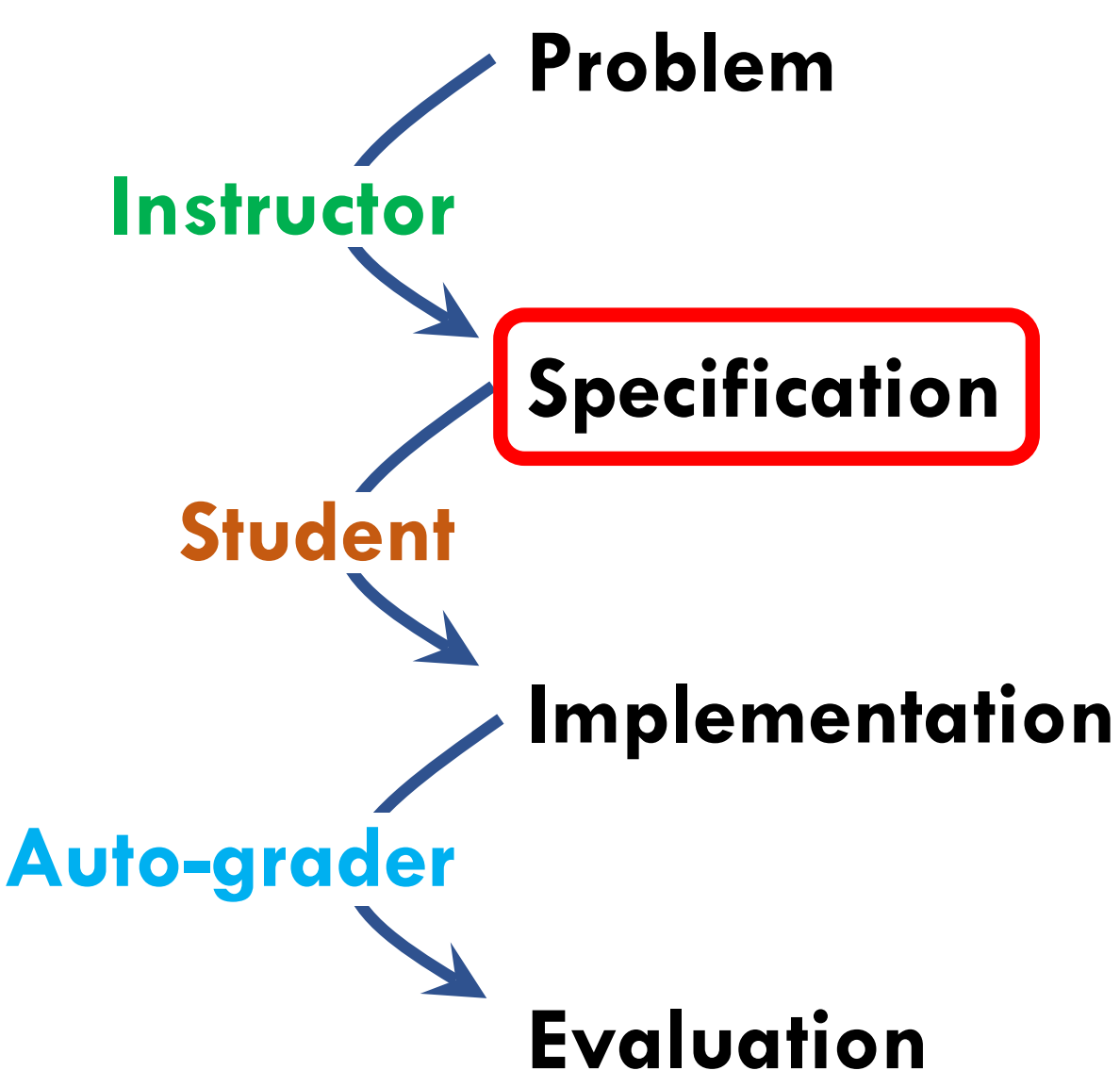
Should we be helping students develop this skill?

If so, then how?


Problem
Instructor
Specification
Student
Implementation
Auto-grader
Evaluation



"This is good" ⇨ "This is ****"
"good"



"This is good" ⇨ "This is ****"
"good"

A sentence can be "censored" by having all banned words removed. Define a function called **censor_sentence()** which takes two inputs: a sentence (this will be a string, with no punctuation, where words are separated by a single space character) and a list of banned words. The function should return a new string where all of the characters in any banned word are replaced with "*".

Problem

Instructor

Specification

Student

Implementation

Auto-grader

Evaluation

"This is good" ⇨ "This is ****"
"good"

A sentence can be "censored" by having all banned words removed. Define a function called **censor_sentence()** which takes two inputs: a sentence (this will be a string, with no punctuation, where words are separated by a single space character) and a list of banned words. The function should return a new string where all of the characters in any banned word are replaced with "*".

```
def censor_sentence(sentence, banned_words):
    sentence = sentence.split()
    for word in sentence:
        if word in banned_words:
            sentence[sentence.index(word)] = "*" * len(word)
    return " ".join(sentence)
```

Problem

Instructor

Specification

Student

Implementation

Auto-grader

Evaluation

"This is good" ⇨ "This is ****"

"good"

A sentence can be "censored" by having all banned words removed. Define a function called **censor_sentence()** which takes two inputs: a sentence (this will be a string, with no punctuation, where words are separated by a single space character) and a list of banned words. The function should return a new string where all of the characters in any banned word are replaced with "*".

```
def censor_sentence(sentence, banned_words):
    sentence = sentence.split()
    for word in sentence:
        if word in banned_words:
            sentence[sentence.index(word)] = "*" * len(word)
    return " ".join(sentence)
```

```
s = "apple banana cherry dragonfruit"
cs = censor_sentence(s, ["dragon", "fruit", "cherry"])
print(cs)
```

```
s = "a aa aaa aaaa aaaaa aaaaaa"
cs = censor_sentence(s, ["a", "aaa"])
print(cs)
```

```
s = "a a a a a a a a"
cs = censor_sentence(s, ["a"])
print(cs)
```

Prompt Problems:

Problem

~~Instructor~~ Student

Specification

~~Student~~ LLM

Implementation

Auto-grader

Evaluation

"This is good" ⇨ "This is ****"
"good"

A sentence can be "censored" by having all banned words removed. Define a function called **censor_sentence()** which takes two inputs: a sentence (this will be a string, with no punctuation, where words are separated by a single space character) and a list of banned words. The function should return a new string where all of the characters in any banned word are replaced with "*".

```
def censor_sentence(sentence, banned_words):
    sentence = sentence.split()
    for word in sentence:
        if word in banned_words:
            sentence[sentence.index(word)] = "*" * len(word)
    return " ".join(sentence)
```

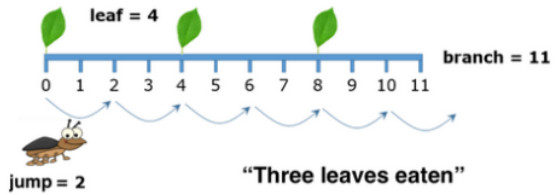
```
s = "apple banana cherry dragonfruit"
cs = censor_sentence(s, ["dragon", "fruit", "cherry"])
print(cs)
```

```
s = "a aa aaa aaaa aaaaa aaaaaa"
cs = censor_sentence(s, ["a", "aaa"])
print(cs)
```

```
s = "a a a a a a a a"
cs = censor_sentence(s, ["a"])
print(cs)
```

Class Registration — Exercise #1 — Exercise #2 — 4 Exercise #3

View the problem



LeafEater(2,4,11) => 3
LeafEater(1,1,5) => 6

Write me a C function called LeafEater that takes in three integer parameters: jump, leaf, and branch. Branch represents the length of the branch (zero indexed). Jump represents how far the leaf eater jumps each time (i.e., which array positions it checks for a leaf). Leaf represents how often leaves appear on the branch (i.e., which array positions have a leaf). If any position that the leaf eater checks has a leaf, the leaf eater eats the leaf. The leaf eater will also check the first position (i.e., index 0), and if it has a leaf, it gets eaten. The function should return the number of leaves eaten.

Write your ChatGPT prompt here

CLICK HERE TO ASK CHATGPT!

ChatGPT response:

```
int LeafEater(int jump, int leaf, int branch) {
   int eaten = 0;
   for (int i = 0; i <= branch; i += jump) {
      if (i % leaf == 0) {
         eaten++;
      }
   }
   return eaten;
}
```

- Test 1
- Test 2
- Test 3
- Test 4
- Test 5
- Test 6

Code Running response:

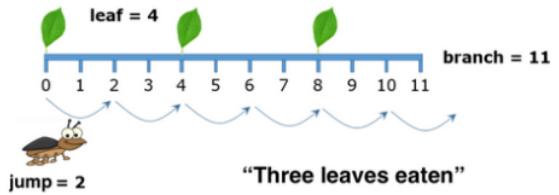
```
You pass \( ^o^ )/ !
```

Student writes prompt for LLM

Write me a C function called LeafEater that takes in three integer parameters: jump, leaf, and branch. Branch represents the length of the branch (zero indexed). Jump represents how far the leaf eater jumps each time (i.e., which array positions it checks for a leaf). Leaf represents how often leaves appear on the branch (i.e., which array positions have a leaf). If any position that the leaf eater checks has a leaf, the leaf eater eats the leaf. The leaf eater will also check the first position (i.e., index 0), and if it has a leaf, it gets eaten. The function should return the number of leaves eaten.

Class Registration — Exercise #1 — Exercise #2 — 4 Exercise #3

View the problem



LeafEater(2,4,11) => 3
LeafEater(1,1,5) => 6

Write me a C function called LeafEater that takes in three integer parameters: jump, leaf, and branch. Branch represents the length of the branch (zero indexed). Jump represents how far the leaf eater jumps each time (i.e., which array positions it checks for a leaf). Leaf represents how often leaves appear on the branch (i.e., which array positions have a leaf). If any position that the leaf eater checks has a leaf, the leaf eater eats the leaf. The leaf eater will also check the first position (i.e., index 0), and if it has a leaf, it gets eaten. The function should return the number of leaves eaten.

Write your ChatGPT prompt here

CLICK HERE TO ASK CHATGPT!

ChatGPT response:

```
int LeafEater(int jump, int leaf, int branch) {
   int eaten = 0;
   for (int i = 0; i <= branch; i += jump) {
      if (i % leaf == 0) {
         eaten++;
      }
   }
   return eaten;
}
```

Test 1
Test 2
Test 3
Test 4
Test 5
Test 6

Code Running response:

You pass \(^o^)/ !

Code is shown and evaluated

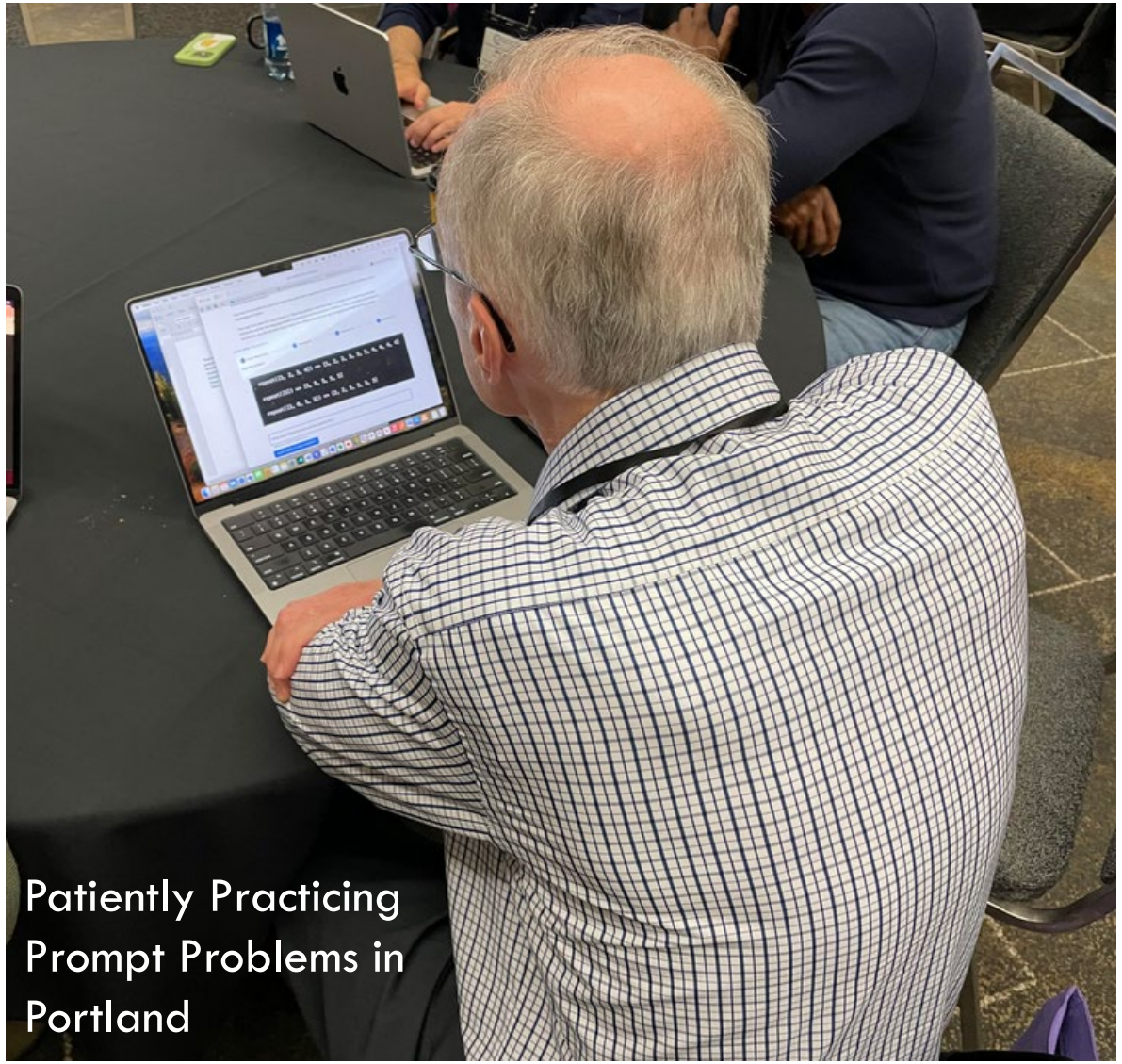
```
int LeafEater(int jump, int leaf, int branch) {
   int eaten = 0;
   for (int i = 0; i <= branch; i += jump) {
      if (i % leaf == 0) {
         eaten++;
      }
   }
   return eaten;
}
```

Test 1
Test 2
Test 3
Test 4
Test 5
Test 6

Code Running response:

You pass \(^o^)/ !

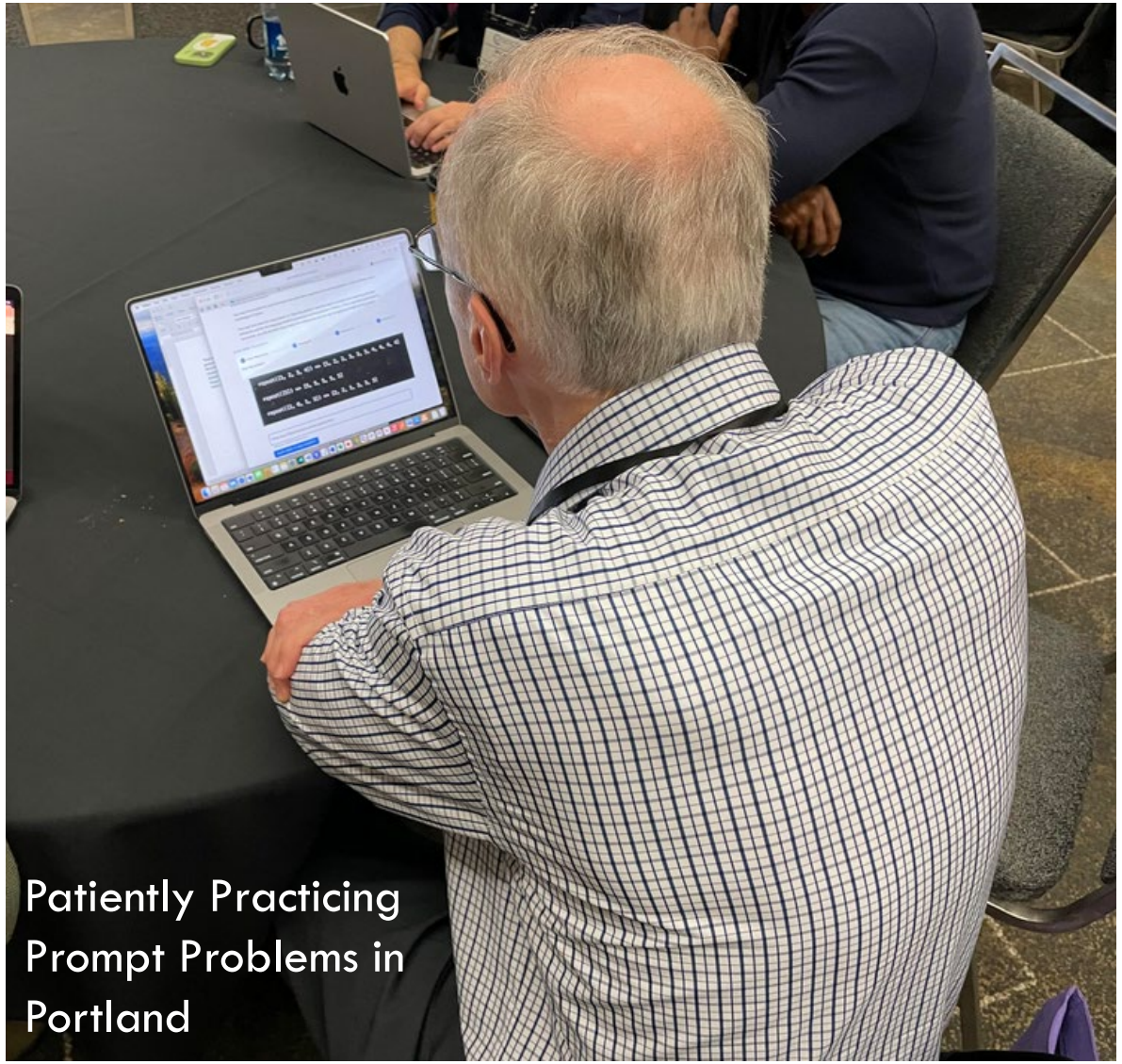
"I felt like I **needed to fully understand the question** before I could get a good answer from the language model."



Patiently Practicing Prompt Problems in Portland

"I felt like I **needed to fully understand the question** before I could get a good answer from the language model."

"I enjoyed this type of task because it made me look at the problems in a different way. **It made me think outside the box.**"

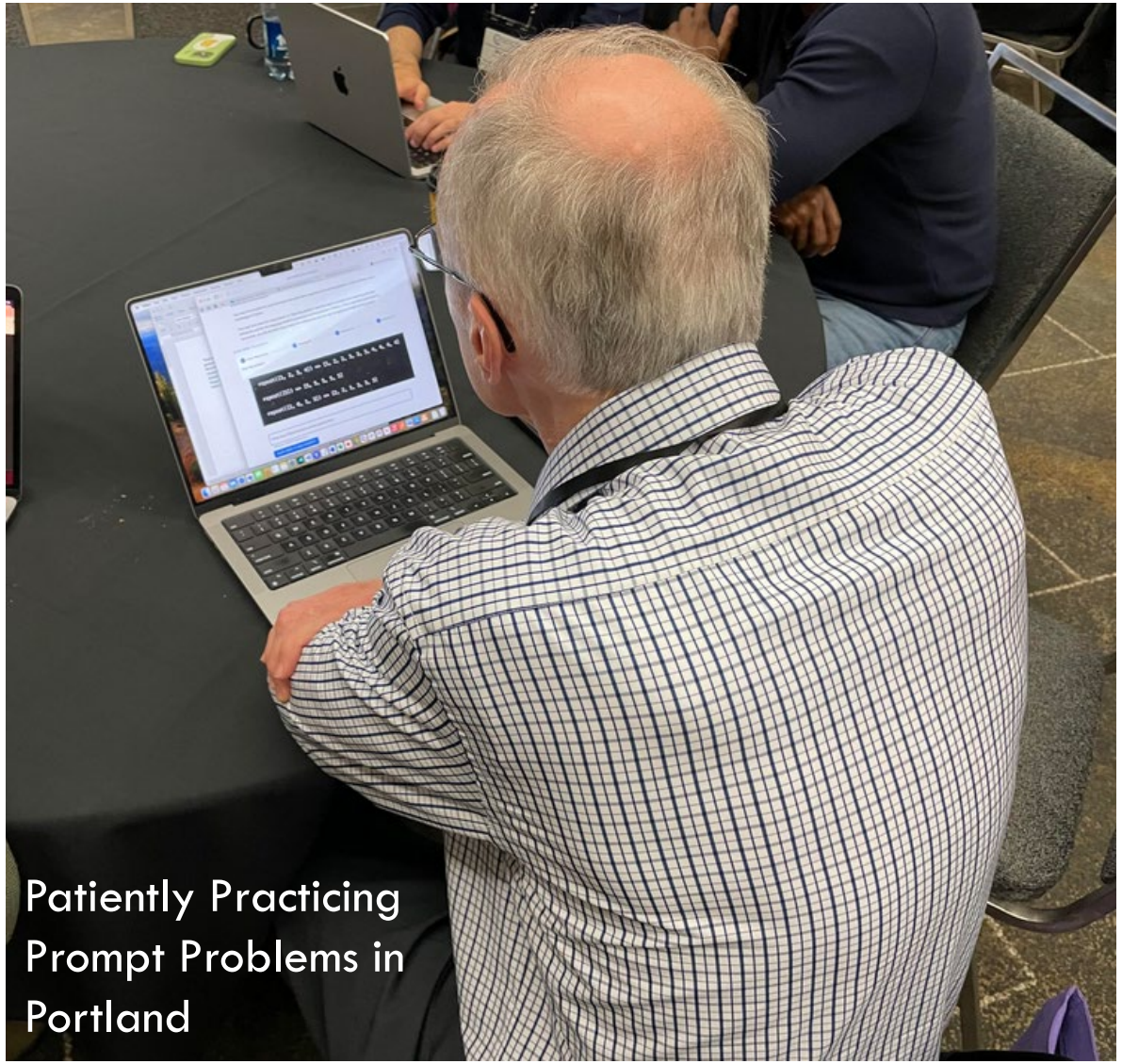


Patiently Practicing Prompt Problems in Portland

"I felt like I **needed to fully understand the question** before I could get a good answer from the language model."

"I enjoyed this type of task because it made me look at the problems in a different way. **It made me think outside the box.**"

"I really enjoyed having to define the problem as **this is often done for us in the labs and projects**. So I learned how to go through the process of understanding the problem then explaining it."

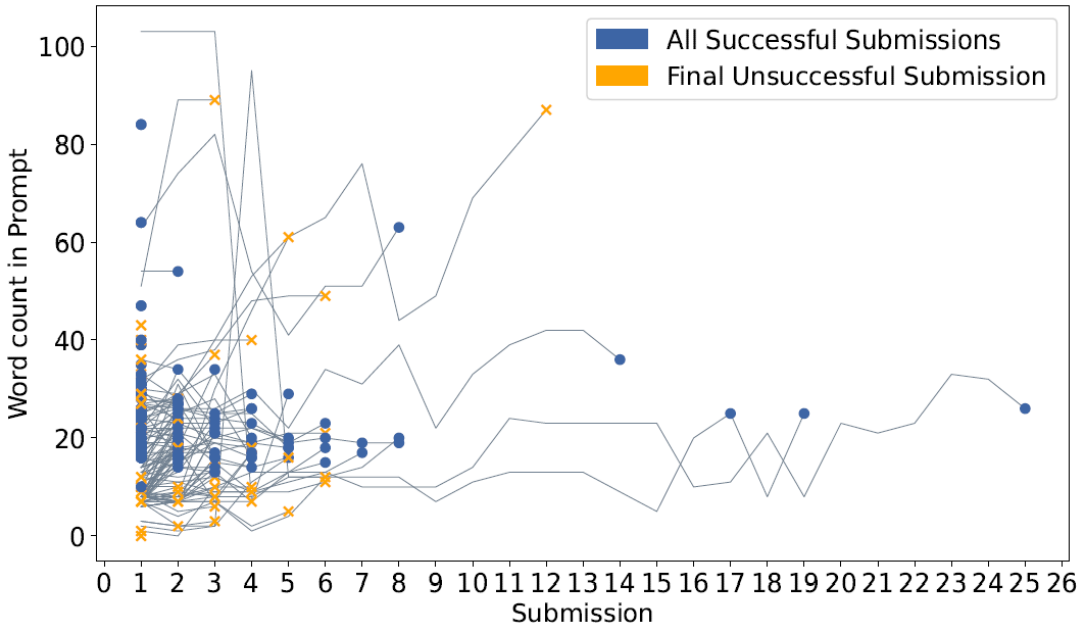


Patiently Practicing Prompt Problems in Portland

- Problem difficulty and prompt variation?
  - CS1: $n = 58$; CS2: $n = 182$
  - 3 Prompt Problems per course

Words in successful prompts

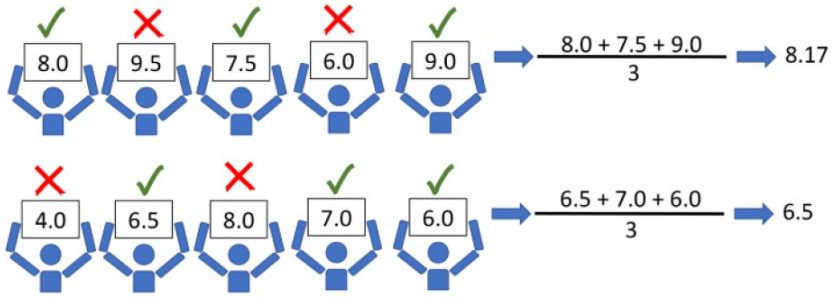
	Students	Sub	Mean	Min	Max
	44 (76%)	2.3	18.0	7	33
CS1	31 (86%)	1.8	47.9	26	85
	20 (65%)	7.5	40.7	25	66
	136 (75%)	2.4	23.0	10	84
CS2	121 (96%)	1.3	28.3	12	88
	114 (99%)	1.5	34.2	16	92



- Problem difficulty and prompt variation?
  - CS1: $n = 58$; CS2: $n = 182$
  - 3 Prompt Problems per course

Words in successful prompts

	Students	Sub	Mean	Min	Max
CS1	44 (76%)	2.3	18.0	7	33
	31 (86%)	1.8	47.9	26	85
	20 (65%)	7.5	40.7	25	66
CS2	136 (75%)	2.4	23.0	10	84
	121 (96%)	1.3	28.3	12	88
	114 (99%)	1.5	34.2	16	92



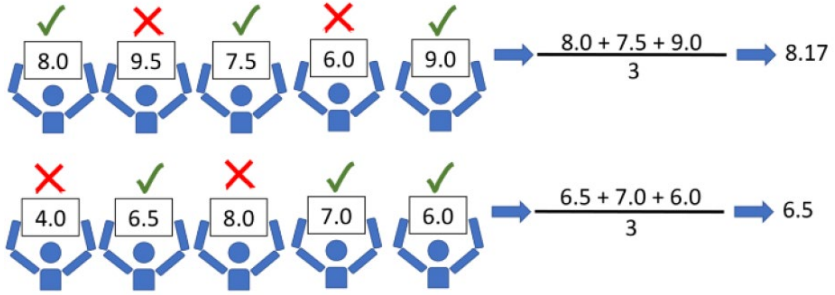
Successful

Write me a Python program that takes five decimal numbers separated by spaces, and outputs the average of the 3 median numbers rounded to 2dp.

- Problem difficulty and prompt variation?
  - CS1: $n = 58$; CS2: $n = 182$
  - 3 Prompt Problems per course

Words in successful prompts

	Students	Sub	Mean	Min	Max
CS1	44 (76%)	2.3	18.0	7	33
	31 (86%)	1.8	47.9	26	85
	20 (65%)	7.5	40.7	25	66
CS2	136 (75%)	2.4	23.0	10	84
	121 (96%)	1.3	28.3	12	88
	114 (99%)	1.5	34.2	16	92

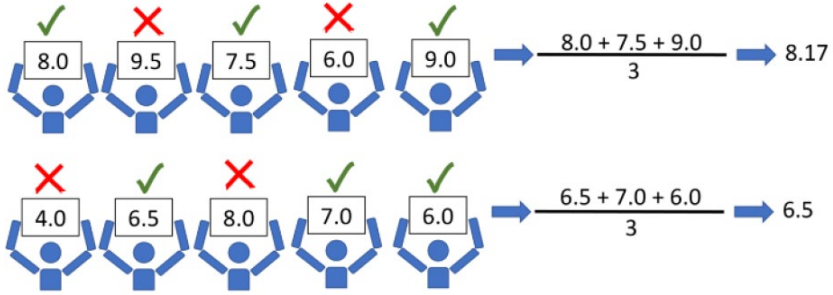


Successful

Write me a Python program that asks the user to input 5 decimal numbers separated by spaces. When the input is complete, remove the highest and lowest valued decimal numbers and then add the rest up. Once the remaining decimals have been added together, divide them by 3. Then show the user the result and that the result is rounded to two decimal places.

- Problem difficulty and prompt variation?
  - CS1: $n = 58$; CS2: $n = 182$
  - 3 Prompt Problems per course

	Students	Sub	Words in successful prompts: Mean	Min	Max
	44 (76%)	2.3	18.0	7	33
CS1	31 (86%)	1.8	47.9	26	85
	20 (65%)	7.5	40.7	25	66
	136 (75%)	2.4	23.0	10	84
CS2	121 (96%)	1.3	28.3	12	88
	114 (99%)	1.5	34.2	16	92



Unsuccessful

Write me a Python program that prompts the user to enter five decimal numbers (1dp) between 1.0 and 10.0 separated by spaces. From these values choose three, ensuring that they are all different but within 0.5 of each other. Find the average of these numbers and round the result to 2dp.

- Student reflections?
  - CS1: $n = 58$; CS2: $n = 182$
  - 3 Prompt Problems per course

Exposure to new constructs

- *"These exercises introduced me to new functions... so this method of writing code* ***could help increase my programming vocabulary****."*
- *"The main benefit I gained ... was observing the logical structure of the programs that it created. In all three cases it used functions that I was previously unaware of, allowing* ***me to gain an understanding of how they could be used*** *and the correct syntax for implementing them."*

Computational thinking

- *"I think while writing prompts for AI,* ***we actually have to have a clear logic*** *to break down the question and explain in plain words"*
- *"I do think that writing prompts for code is a good way of* ***developing analytical and problem-solving thinking and skills*** *as it forces you to think through the steps needed to take the input through to the output."*

- Student reflections?
  - CS1: $n = 58$; CS2: $n = 182$
  - 3 Prompt Problems per course

Resistance / negative feedback

- *"You have just ruined every piece of self esteem I had regarding coding. I know full well that it would have taken me around 35 minutes to figure out how to create those functions and that damn computer did it in seconds. Robots are going to own us within years."*

However, the models are always advancing! (remember)?

Can multimodal / vision models solve prompt problems? *Not trialed yet!*

Not-yet published work:

- GenAI helps good students get better, or stay the same)
- But many of the poor students get *worse*.
- Is there a real potential for an AI Divide in our programmes.
- ***What do we do about that?***

https://arxiv.org/abs/2405.17739

The Widening Gap: The Benefits and Harms of Generative AI for Novice Programmers

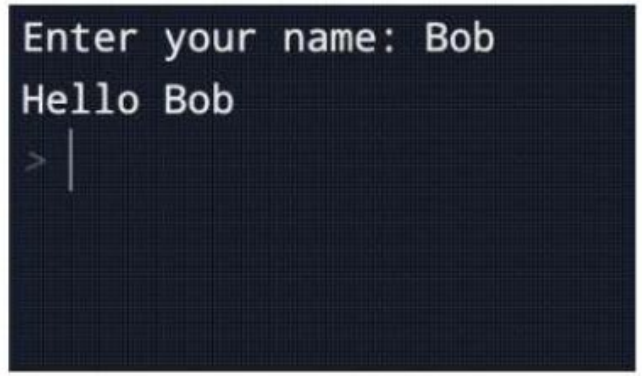
JAMES PRATHER, Abilene Christian University, TX, USA
BRENT REEVES, Abilene Christian University, TX, USA
JUHO LEINONEN, Aalto University, Finland
STEPHEN MACNEIL, Temple University, PA, USA
ARISOA S. RANDRIANASOLO, Abilene Christian University, TX, USA
BRETT BECKER, University College Dublin, Ireland
BAILEY KIMMEL, Abilene Christian University, TX, USA
JARED WRIGHT, Abilene Christian University, TX, USA
BEN BRIGGS, Abilene Christian University, TX, USA

Novice programmers often struggle through programming problem solving due to a lack of metacognitive awareness and strategies. Previous research has shown that novices can encounter multiple metacognitive difficulties while programming, such as forming incorrect conceptual models of the problem or having a false sense of progress after testing their solution. Novices are typically unaware of how these difficulties are hindering their progress. Meanwhile, many novices are now programming with generative AI (GenAI), which can provide complete solutions to most introductory programming problems, code suggestions, hints for next steps when stuck, and explain cryptic error messages. Its impact on novice metacognition has only started to be explored. Here we replicate a previous study that examined novice programming problem solving behavior and extend it by incorporating GenAI tools. Through 21 lab sessions consisting of participant observation, interview, and eye tracking, we explore how novices are coding with GenAI tools. Although 20 of 21 students completed the assigned programming problem, our findings show an unfortunate divide in the use of GenAI tools between students who accelerated and students who struggled. Students who accelerated were able to use GenAI to create code they already intended to make and were able to ignore unhelpful or incorrect inline code suggestions. But for students who struggled, our findings indicate that previously known metacognitive difficulties persist, and that GenAI unfortunately can compound them and even introduce new metacognitive difficulties. Furthermore, struggling students often expressed cognitive dissonance about their problem solving ability, thought they performed better than they did, and finished with an illusion of competence. Based on our observations from both groups, we propose ways to scaffold the novice GenAI experience and make suggestions for future work.

CCS Concepts: • **Social and professional topics → Computing education**.

Additional Key Words and Phrases: generative AI, large language models, metacognition

ACM Reference Format:
James Prather, Brent Reeves, Juho Leinonen, Stephen MacNeil, Arisoa S. Randrianasolo, Brett Becker, Bailey Kimmel, Jared Wright, and Ben Briggs. 2024. The Widening Gap: The Benefits and Harms of Generative AI for Novice Programmers. In . ACM, New York, NY, USA, 27 pages. https://doi.org/XXXXXXX.XXXXXXX



Visual representation of problem (in this case, an animation illustrates user interaction with program)

Write me a Python program that asks the user to enter their name, and then prints the word "Hello" followed by a space, followed by their name

Write your ChatGPT prompt here

Prompt entry

CLICK HERE TO ASK CHATGPT!

LLM response

ChatGPT response:

```
print("Hello " + input("Enter your name: "))
```

Code Running response:

```
You pass \( ^o^ )/ !
```

Execution output (in this case, a success message as all tests pass)

- Students described engaging in **metacognitive** aspects of learning such as **planning** their **problem solving** approach and **monitoring** whether they understood what they were doing.
- This **increased awareness** was also exemplified by students who described how the tool might better **support reflecting on their learning**.
- (We think) Prompt Problems are a useful way to teach programming concepts and encourage metacognitive programming skills.

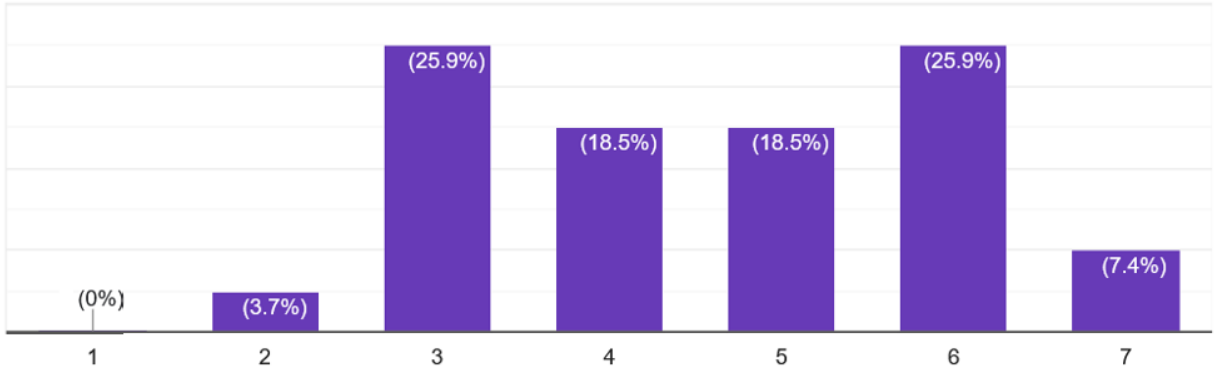
Paul Denny, Juho Leinonen, James Prather, Andrew Luxton-Reilly, Thezyrie Amarouche, Brett A. Becker, and Brent N. Reeves. 2024. Prompt Problems: A New Programming Exercise for the Generative AI Era. In Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1 (SIGCSE 2024), March 20–23, 2024, Portland, OR, USA. ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3626252.3630909 (DOI not active until March). Preprint available: https://arxiv.org/abs/2311.05943. www.brettbecker.com/publications

Paul Denny, Juho Leinonen, James Prather, Andrew Luxton-Reilly, Thezyrie Amarouche, Brett A. Becker, and Brent N. Reeves. 2024. Promptly: Using Prompt Problems to Teach Learners How to Effectively Utilize AI Code Generators. In Proceedings of the CHI Conference on Human Factors in Computing Systems (CHI '24), May 11--16, 2024, Honolulu, HI, USA. ACM, New York, NY, USA, 20 pages. https://doi.org/10.1145/3613904.3642407 (DOI not active until May). Preprint available: https://arxiv.org/abs/2307.16364. www.brettbecker.com/publications

- An “informal” survey at ITiCSE 2023

How do you feel about large language models and generative AI in computing education?

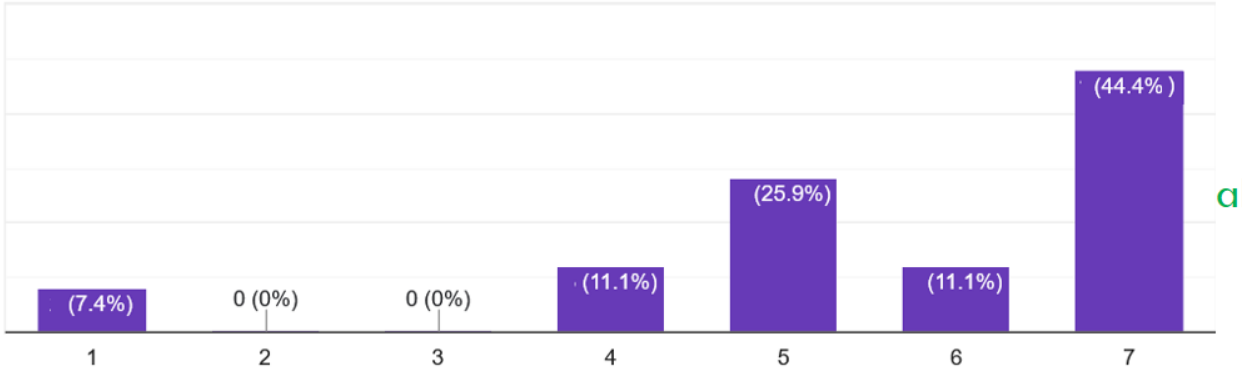
Negative, pessimistic, anxious, etc.



Positive, optimistic, excited, etc.

I plan to integrate / incorporate generative AI tools into my computing courses.

No, definitely not



Yes, absolutely