THE LONDON SCHOOL OF ECONOMICS AND POLITICAL SCIENCE

Large language models in programming education

OVERVIEW OF EXISTING PRACTICES AND FUTURE DIRECTIONS

Dr. Marcos Barreto

Department of Statistics

e: m.e.barreto@lse.ac.uk

w: https://marcosebarreto.github.io/

LSE Education Symposium 2024 – Education Futures

06 June 2024

THE LONDON SCHOOL OF ECONOMICS AND POLITICAL SCIENCE

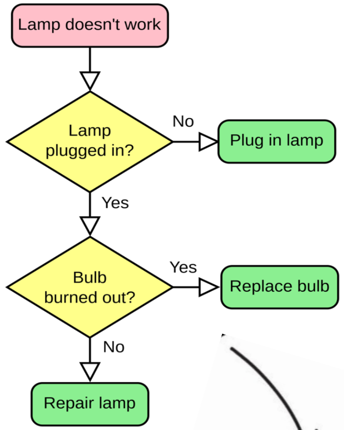
Outline

- How were we (humans) used to programming computers?
- Programming with large language models (LLMs)
- Existing practice: selected examples
- Future directions

THE LONDON SCHOOL OF ECONOMICS AND POLITICAL SCIENCE

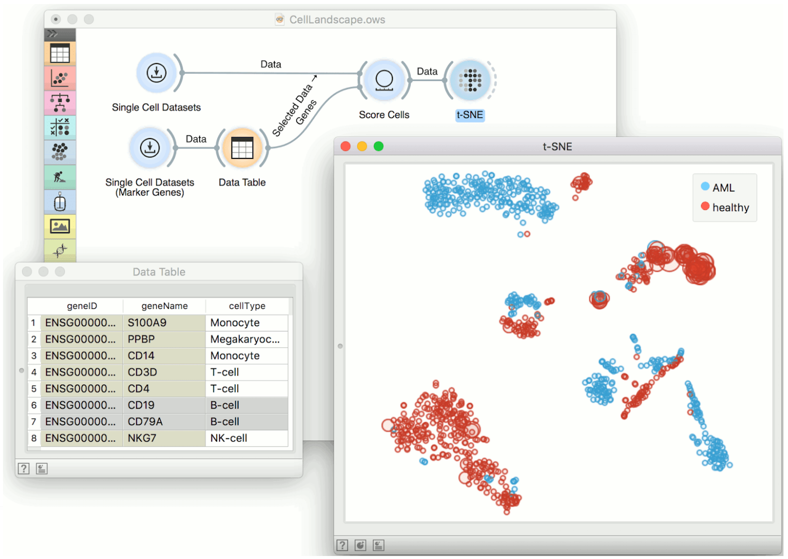
▪ Different ways we (humans) used to program computers

▪ **Programming** is writing **computer code** (**program**), based on an **algorithm**, to solve a **problem**.[1]



FLOWCHART

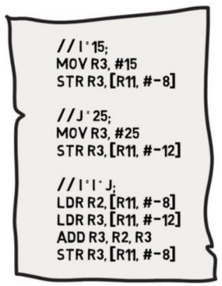
VISUAL PROGRAMMING



"NO CODE" PROGRAMMING

```
#Take the users input
words = raw_input("Enter some text to translate to pig latin: ")
print "You entered: ", words

#Now I need to break apart the words into a list
words = words.split(' ')

#Now words is a list, so I can manipulate each one usinga loop

for i in words:
    if len(i) >= 3: #I only want to translate words greater than 3 characters
        i = i + "%say" % (i[0])
        i = i[1:]
        print i
    else:
        pass
```

IMPERATIVE AND DECLARATIVE PROGRAMMING



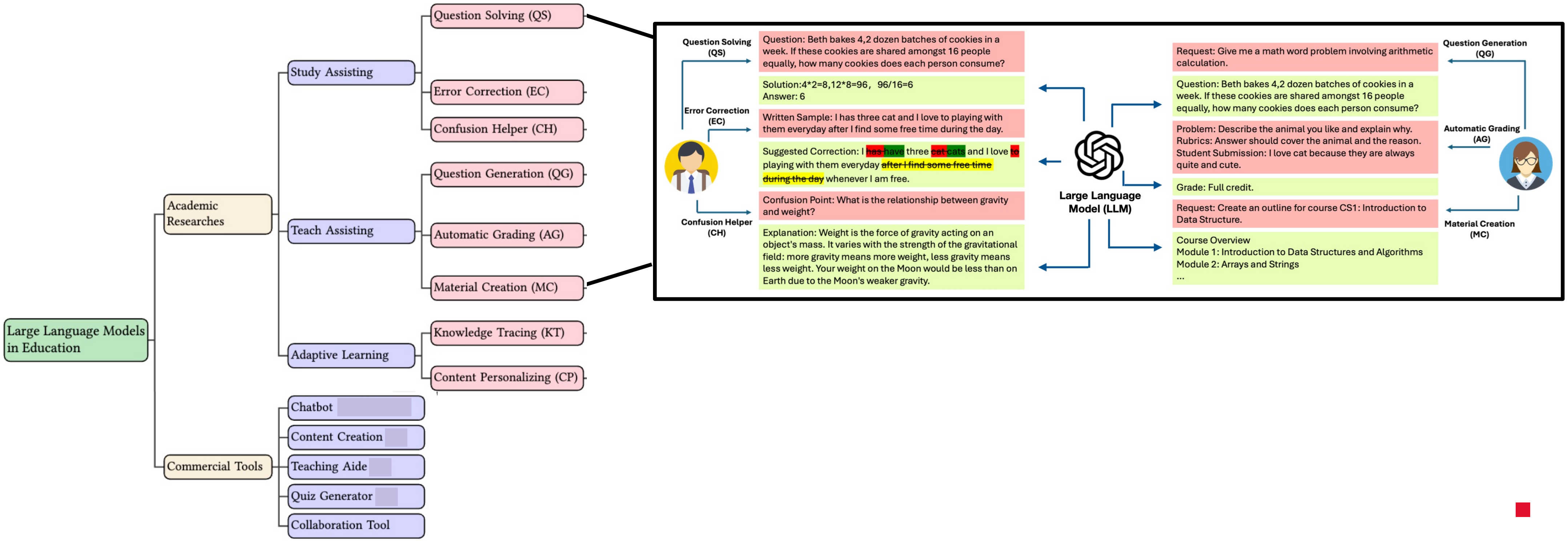
ASSEMBLY

Image credits: Android Authority, Wikipedia, CS Teaching Tips, Apifornia

THE LONDON SCHOOL OF ECONOMICS AND POLITICAL SCIENCE

▪ Large language models in education

▪ **Taxonomy** of LLMs for education applications:[2][3]



THE LONDON SCHOOL OF ECONOMICS AND POLITICAL SCIENCE

▪ Large language models in education

- Existing datasets and benchmarks are constructed for text-rich educational downstream tasks, with an emphasis on QS (question solving), EC (error correction), QG (question generation), and AG (automatic grading).[2]

Table 1: Summary of existing datasets and benchmarks in the area of LLMs for education.

Dataset&Benchmark	App	User	Subject	Level	Language	Modality	Amount
Defects4J	EC	student	computer science	professional	EN & Java	text& code	357
ManyBugs	EC	student	computer science	professional	EN & C	text& code	185
IntroClass	EC	student	computer science	professional	EN & C	text& code	998
QuixBugs	EC	student	computer science	professional	EN & multi	text& code	40
Bugs2Fix	EC	student	computer science	professional	EN & Java	text& code	2.3M
CodeReview	EC	student	computer science	professional	EN & Multi	text& code	642
CodeReview-New	EC	student	computer science	professional	EN & Multi	text& code	15

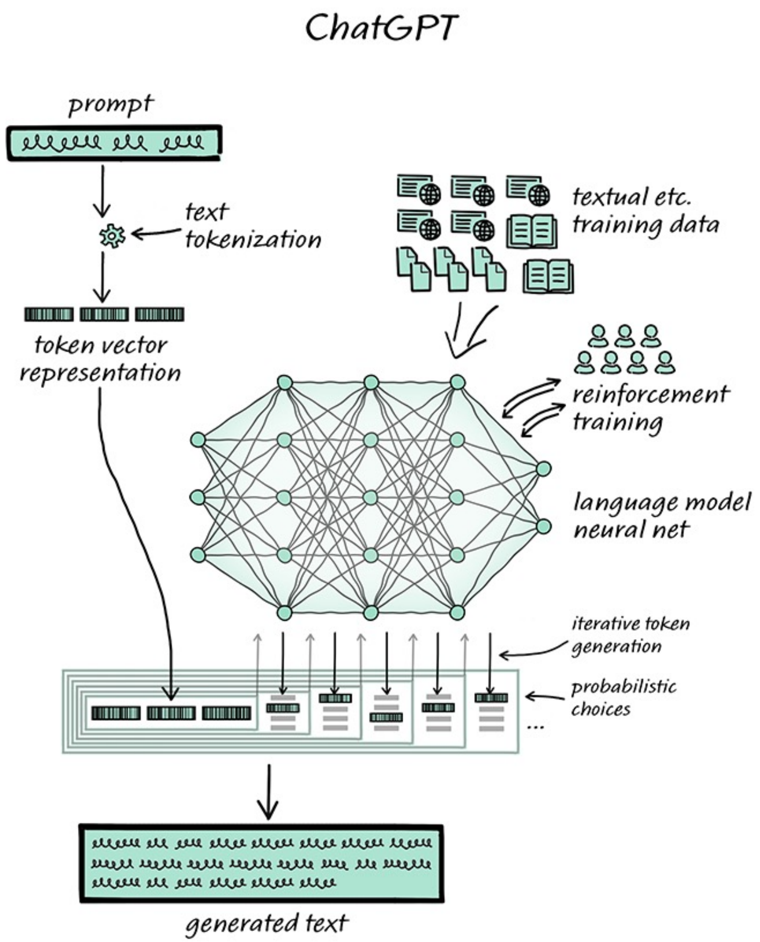
Table 1: Summary of existing datasets and benchmarks in the area of LLMs for education.

Dataset&Benchmark	App	User	Subject	Level	Language	Modality	Amount
GSM8K	QS	student	math	K-12	EN	text	8.5K
MATH	QS	student	math	K-12	EN	text	12.5K
Dolphin18K	QS	student	math	K-12	EN	text	18K
DRAW-1K	QS	student	math	comprehensive	EN	text	1K
Math23K	QS	student	math	K-12	ZH	text	23K
Ape210K	QS	student	math	K-12	EN, ZH	text	210K
MathQA	QS	student	math	K-12	EN	text	37K
ASDiv	QS	student	math	K-12	EN	text & image	2K
IconQA	QS	student	math	K-12	EN	text & table	107K
TQA	QS	student	science	K-12	EN	text & image	26K
Geometry3K	QS	student	geometry	K-12	EN	text & image	3K
AI2D	QS	student	science	K-12	EN	text & image	5K
SCIENCEQA	QS	student	science	K-12	EN	text & image	21K
MedQA	QS	student	medicine	professional	EN	text	40K
MedMCQA	QS	student	medicine	professional	EN	text	200K
TheoremQA	QS	student	science	college	EN	text	800
Math-StackExchange	QS	student	math	comprehensive	EN	text	310K
TABMWP	QS	student	math	K-12	EN	text	38K
ARC	QS	student	comprehensive	comprehensive	EN	text	7.7K
C-Eva	QS	student	comprehensive	comprehensive	ZH	text	13.9K
GAOKAO-bench	QS	student	comprehensive	comprehensive	ZH	text	2.8K
AGIEval	QS	student	comprehensive	comprehensive	EN, ZH	text	8k
MMLU	QS	student	comprehensive	comprehensive	EN	text	1.8K
CMMLU	QS	student	comprehensive	comprehensive	ZH	text	11.9K
SuperCLUE	QS	student	comprehensive	comprehensive	ZH	text	15.9K
LANG-8	EC	student	linguistic	language training	Multi	text	1M
CLANG-8	EC	student	linguistic	language training	Multi	text	2.6M
CoNLL-2014	EC	student	linguistic	language training	EN	text	58k
BEA-2019	EC	student	linguistic	language training	EN	text	686K
SIGHAN	EC	student	linguistic	language training	ZH	text	12K
CTC	EC	student	linguistic	language training	ZH	text	218K
FCGEC	EC	student	linguistic	language training	ZH	text	41K
FlaCGEC	EC	student	linguistic	language training	ZH	text	13K
GECCC	EC	student	linguistic	language training	CS	text	83K
RULEC-GEC	EC	student	linguistic	language training	RU	text	12K
Falko-MERLIN	EC	student	linguistic	language training	GE	text	24K
COWS-L2H	EC	student	linguistic	language training	ES	text	12K
UA-GEC	EC	student	linguistic	language training	UK	text	20K
RONACC	EC	student	linguistic	language training	RO	text	10K
Defects4J	EC	student	computer science	professional	EN & Java	text& code	357
ManyBugs	EC	student	computer science	professional	EN & C	text& code	185
IntroClass	EC	student	computer science	professional	EN & C	text& code	998
QuixBugs	EC	student	computer science	professional	EN & multi	text& code	40
Bugs2Fix	EC	student	computer science	professional	EN & Java	text& code	2.3M
CodeReview	EC	student	computer science	professional	EN & Multi	text& code	642
CodeReview-New	EC	student	computer science	professional	EN & Multi	text& code	15
SciQ	QG	teacher	science	MOOC	EN	text	13.7K
RACE	QG	teacher	linguistic	K-12	EN	text	100K
FairytaleQA	QG	teacher	literature	K-12	EN	text	10K
LearningQ	QG	teacher	comprehensive	MOOC	EN	text	231K
KHANQ	QG	teacher	science	MOOC	EN	text	1K
EduQG	QG	teacher	comprehensive	MOOC	EN	text	3K
MCQL	QG	teacher	comprehensive	MOOC	EN	text	7.1K
Televic	QG	teacher	comprehensive	MOOC	EN	text	62K
CLC-FCE	AG	teacher	linguistic	standardized test	EN	text	1K
ASAP	AG	teacher	linguistic	K-12	EN	text	17K
TOEFL11	AG	teacher	linguistic	standardized test	EN	text	1K
ICLE	AG	teacher	linguistic	standardized test	EN	text	4K
HSK	AG	teacher	linguistic	standardized test	ZH	text	10K

THE LONDON SCHOOL OF ECONOMICS AND POLITICAL SCIENCE

Large language model-based programming

- Three main **opportunities**:[4]
  - Generation of code from specification (**text to code**)
  - Generation of ancillary tools such as test cases (**code to code**)
  - Generation of explanations or suggestions (**code to text**)



https://writings.stephenwolfram.com/2023/02/what-is-chatgpt-doing-and-why-does-it-work/

THE LONDON SCHOOL OF ECONOMICS AND POLITICAL SCIENCE

Large language model-based programming

Pedagogical approaches for explaining code: [5]

- **“Explain in Plain English” (EiPE)**
  - Students explain the purpose of code fragments at an abstract level
- **Code tracing**
  - Students need to understand how the code executes and “predict” its behaviour (changes to memory and internal state) and output
- **Pair programming**
  - Students explain their code and their thinking process to their partner
- **Misconception-based feedback**
  - Students follow prompts based on common misconceptions to guide discussion about code

THE LONDON SCHOOL OF ECONOMICS AND POLITICAL SCIENCE

Experiences from Using Code Explanations Generated by Large Language Models in a Web Software Development E-Book

Authors: Stephen MacNeil, Andrew Tran, Arto Hellas, Joanne Kim, Sami Sarsa, Paul Denny, Seth Bernstein, Juho Leinonen Authors Info & Claims

SIGCSE 2023: Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1 • March 2023 • Pages 931–937 • https://doi.org/10.1145/3545945.3569785

Example practices: code explanation

- Setting: e-book on Web software development [5]
- Goal: integrate explanations generated by LLMs into Computer Science classrooms.
- Research questions:
  - RQ1: How do students engage with the generated explanations, and which types of explanations do they use most frequently? For what types of code snippets do students request explanations?
  - RQ2: What are the characteristics of explanations that students rate as most and least helpful?
- Approach: Use of LLMs to generate three types of explanations:
  - Line-by-line explanation
  - List of important concepts
  - High-level summary of the code
- A total of 13 code snippets * 5 code explanations * 3 types of explanations = **195 explanations** were generated and added to two chapters of the e-book.

THE LONDON SCHOOL OF ECONOMICS AND POLITICAL SCIENCE

Code snippet: JavaScript program that uses the Deno runtime to launch a Web server.

```
import { serve } from "https://deno.land/std@0
    ↪ .140.0/http/server.ts";
let count = 0;
const handleRequest = (request) => {
  if (request.method === "POST") {
    count++;
  }
  return new Response(count);
};
serve(handleRequest, { port: 7777 });
```

Line-by-line

- The first line imports the 'serve' function from the 'server' module in the 'std' library from version 0.140.0 of the Deno runtime.
- The second line declares a 'count' variable and initializes it to '0'.
- The third line defines a 'handleRequest' function that takes a 'request' parameter. This function increments the value of 'count' if the 'request' method is 'POST'.
- The fourth line returns a new 'Response' object containing the value of 'count'.
- The fifth line calls the 'serve' function, passing in the 'handleRequest' function and an options object. The options object sets the 'port' property to '7777'.

Measures: (i) explanation view time; (ii) number of views; and (iii) subjective ratings.

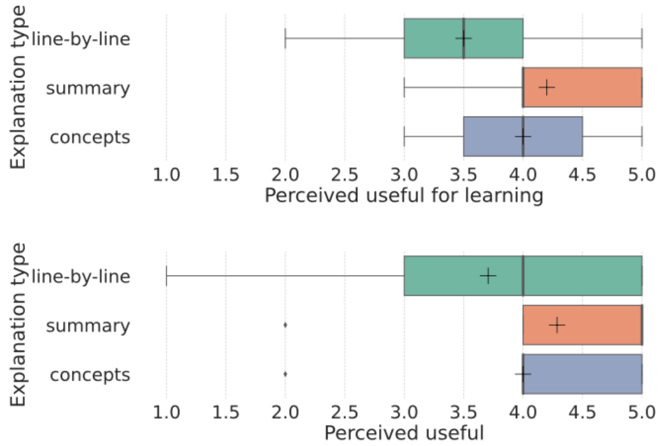


Figure 2: Boxplot of explanation usefulness ratings with + indicating mean. Although most viewed among students, line-by-line explanations were rated least helpful.

Concepts/purpose

The purpose of this code is to create a simple server that counts how many POST requests it receives.

Summary

Codex: # Summarize and explain the goal of the above code

(1) A web server is listening on port 7777. The handleRequest function is called when a request is accepted. What is the structure of the Deno module system? Deno supports ES Modules.

GPT-3: # Summarize and explain this code snippet

(1) This code snippet creates a web server that listens on port 7777 and returns "Hello world!" for every request.

176 explanations,
58 students,
Summer 2022

THE LONDON SCHOOL OF ECONOMICS AND POLITICAL SCIENCE

Comparing Code Explanations Created by Students and Large Language Models

Authors: Juho Leinonen, Paul Denny, Stephen MacNeil, Sami Sarsa, Seth Bernstein, Joanne Kim, Andrew Tran, Arto Hellas Authors Info & Claims

ITiCSE 2023: Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education V. 1 • June 2023 • Pages 124–130 • https://doi.org/10.1145/3587102.3588785

Example practices: code explanation

- Setting: 1st-year programming course, ~1,000 students [6]
- Goal: comparison of code explanations created by students vs those generated by LLMs
- Research questions:
  - RQ1: To what extent do code explanations created by students and LLMs differ in accuracy, length, and understandability?
  - RQ2: What aspects of code explanations do students value?
- Approach: two lab sessions
  - Lab A: students created explanations (purpose and summary) for three code snippets
  - Lab B (2 weeks after): students were shown a random sample of four explanations created by the students in Lab A or generated by GPT-3, and assessed them based on accuracy, understandability, and length.

THE LONDON SCHOOL OF ECONOMICS AND POLITICAL SCIENCE

Lab A: function definitions

```
double AverageNegativeValues(int values[], int length)
{
    int i, sum, count;
    i = 0;
    sum = 0;
    count = 0;

    while (i < length) {
        if (values[i] < 0) {
            sum = sum + values[i];
            count++;
        }
        i++;
    }

    return (double)sum / count;
}
```

```
int LargestValue(int values[], int length)
{
    int i, max;

    max = values[0];
    for (i = 1; i < length; i++) {
        if (values[i] > max) {
            max = values[i];
        }
    }

    return max;
}
```

```
int CountZeros(int values[], int length)
{
    int i, count;

    count = 0;
    for (i = 0; i < length; i++) {
        if (values[i] == 0) {
            count++;
        }
    }

    return count;
}
```

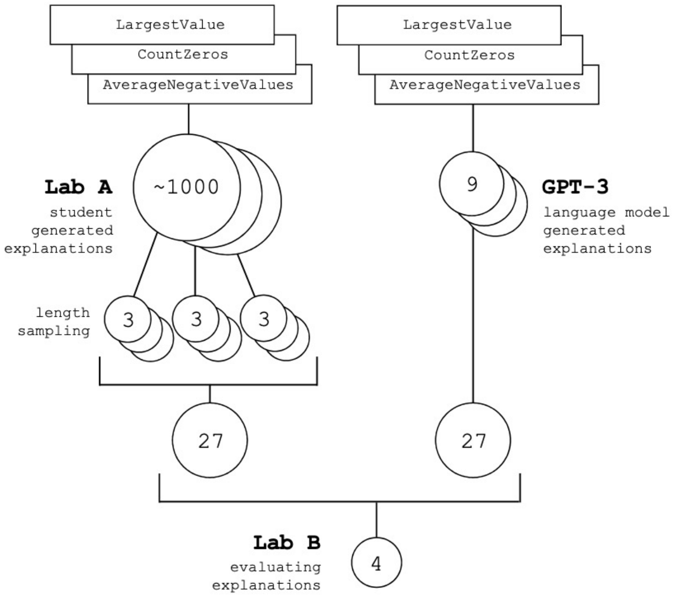


Figure 2: Overview of the data generation and sampling. In Lab B, each student was allocated four code explanations to evaluate, selected at random from a pool of 54 code explanations, half of which were generated by students in Lab A, and half of which were generated by GPT-3.

Table 1: Descriptive statistics of student responses on code explanation quality. The responses that were given using a Likert-scale have been transformed so that 1 corresponds to 'Strongly disagree' and 5 corresponds to 'Strongly agree'.

	Student-generated		LLM-generated	
	Mean	Median	Mean	Median
Easy to understand	3.75	4.0	4.12	4.0
Accurate summary	3.78	4.0	4.0	4.0
Ideal length	2.75	3.0	2.66	3.0
Length (chars)	811	738	760	731

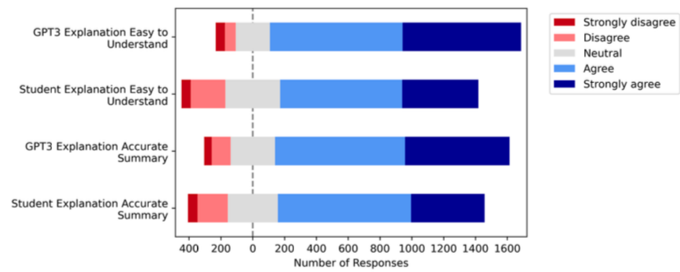


Figure 3: Distribution of student responses on LLM and student-generated code explanations being easy to understand and accurate summaries of code.

963 explanations (Lab A),
954 students (Lab B),
2022

THE LONDON SCHOOL OF ECONOMICS AND POLITICAL SCIENCE


arXiv > cs > arXiv:2401.10759

Computer Science > Human-Computer Interaction

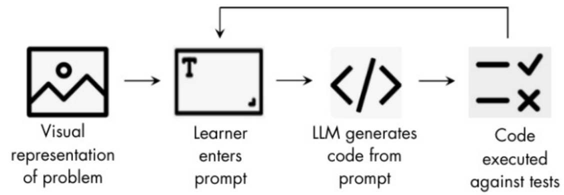
[Submitted on 19 Jan 2024]

Interactions with Prompt Problems: A New Way to Teach Programming with Large Language Models

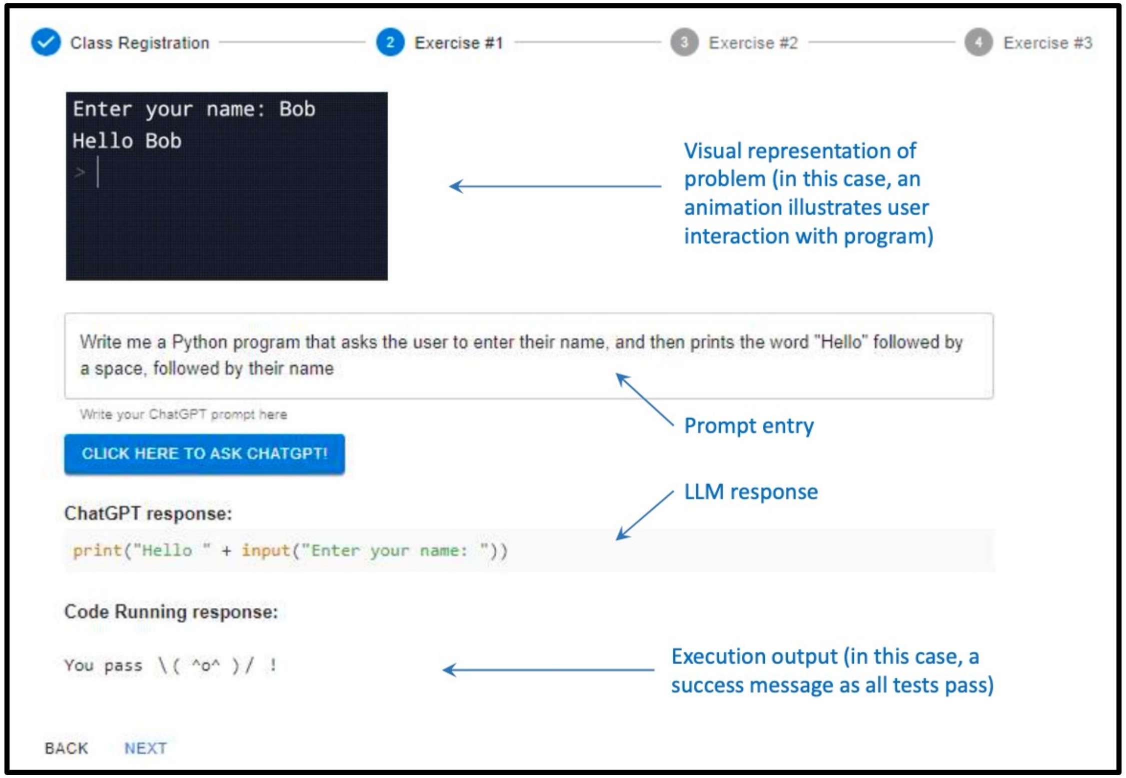
James Prather, Paul Denny, Juho Leinonen, David H. Smith IV, Brent N. Reeves, Stephen MacNeil, Brett A. Becker, Andrew Luxton-Reilly, Thezyrie Amarouche, Bailey Kimmel

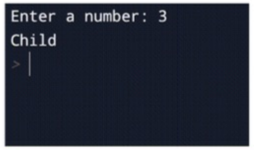
Example practices: code generation

- Setting: 1st-year programming course in Python [7]
- Goal: propose a new type of programming problem to teach coding based on Prompt Problems and related ways of assessing it
- Research questions:
  - RQ1: How do students interact with Prompt Problems while learning to program?
  - RQ2: How do students perceive Prompt Problems affecting their learning of programming concepts?
- Approach: “Promptly” tool
  - Each prompt problem consists of a visual representation of a problem (no textual description is given) and a set of associated test cases used to verify the code generated by the LLM.



Promptly interface & example exercises





AGE	CATEGORY
Below 8	Child
8-12	Tween
13-19	Teenager
20 or above	Adult

Fig. 3. Producing a categorization based on age.

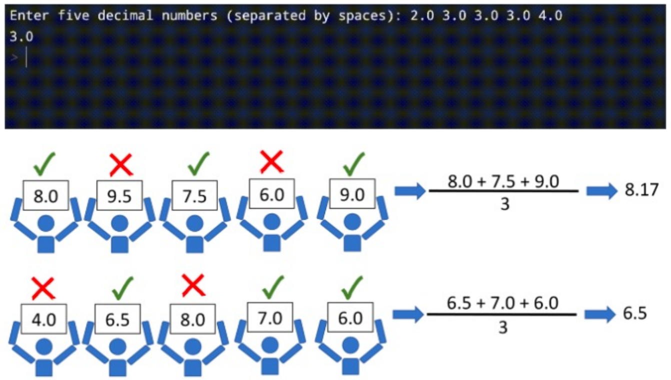


Fig. 4. Calculating the average of the "middle" values out of a set of five values (using the metaphor of judges scoring an athletic competition, where the highest and lowest values are excluded).

Results of student interactions

Pilot study
54 students
July 2023

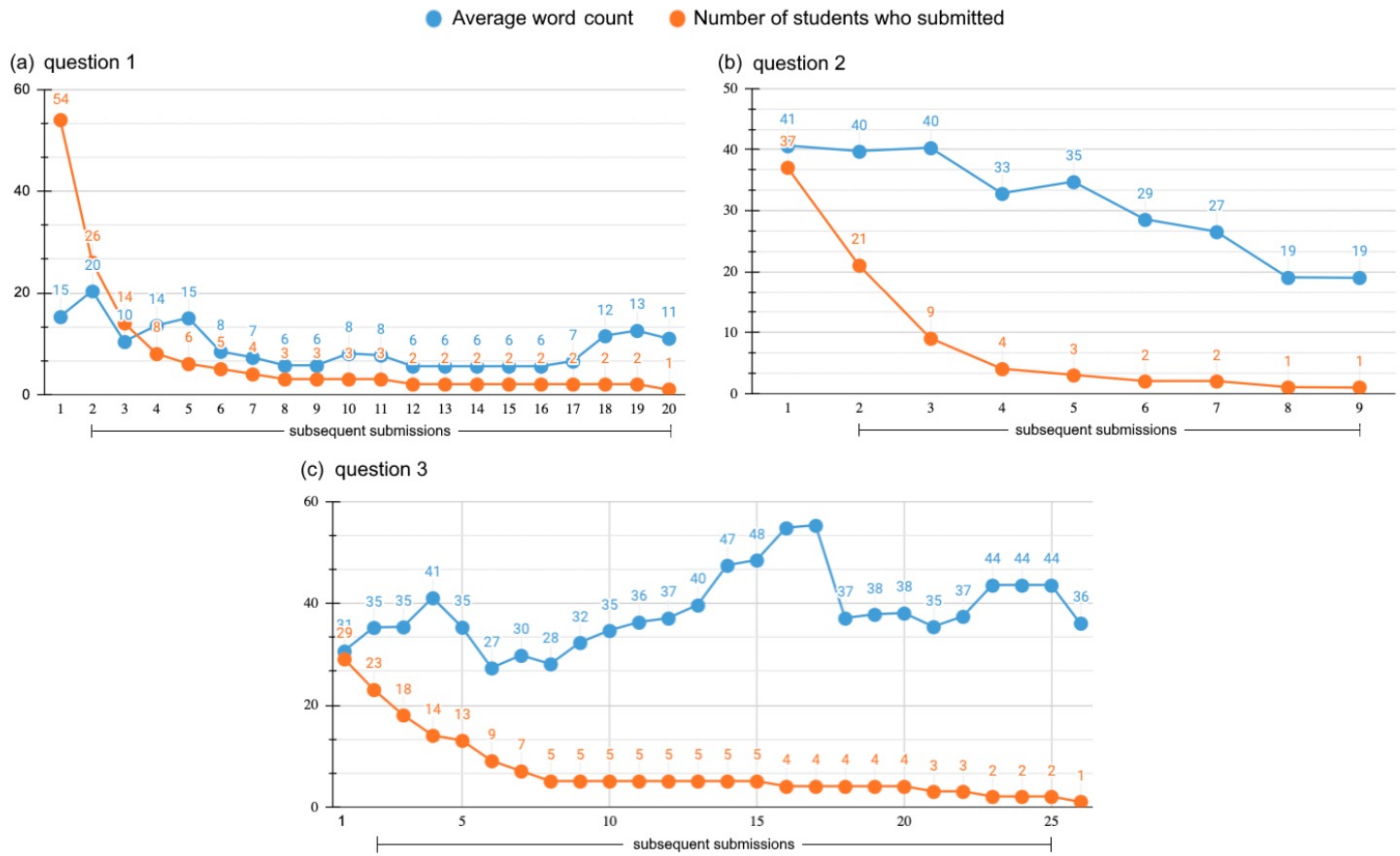


Fig. 5. The average number of words in each subsequent submission and number of participants that submitted. On the x-axis, 1 is the initial submission (attempt) per question and 2- are subsequent submissions (attempts).

THE LONDON SCHOOL OF ECONOMICS AND POLITICAL SCIENCE

Results of student interactions

Large scale study
202 students (Lab 10)
147 students (Lab 11)
2,939 prompt submissions
August 2023

Problem	Students			Submissions				Words in Prompts		
	Total	Correct	First Try	Count	Mean	Min	Max	Mean	Min	Max
Lab10-1	202	118	32	884	4.37	1	30	25.79	7	76
Lab10-2	108	108	74	212	1.96	1	10	27.39	8	93
Lab10-3	107	104	67	224	2.09	1	20	34.78	8	119
Lab11-1	147	105	39	491	3.34	1	27	41.11	9	198
Lab11-2	97	82	20	502	5.17	1	28	43.96	16	86
Lab11-3	80	60	5	626	7.82	1	47	54.07	23	115

Table 1. Summary of student interactions with the Prompt Problems in Labs 10 and 11. For students, we provide the total number of unique students that attempted each problem (Total), the number who got it correct (Correct), and the number who got it correct on the first try (First Try). For submissions, we provide the total number of prompt submissions made for that problem (Count), the mean number of submissions (Mean), the minimum number of submissions any student had to correctly solve the problem (Min), and the maximum number of submissions any student had whether correct or incorrect (Max). To describe the words in submitted correct prompts, we provide the average number of words in correct prompts (Mean), the minimum number of words in correct prompts (Min), and the maximum number of words in correct prompts (Max).

(a) Lab 10 Excercise 1

```
counter([0, 2, 3, 4, 5, 6, 0]) => 2

counter([10, 20, 30]) => 0

counter([0, 0, 0, 0, 999]) => 4
```

(b) Lab 10 Excercise 2

```
initials('abd def ghi') => 'ADG'

initials('xxx') => 'X'

initials('Hi world') => 'HW'
```

(c) Lab 10 Excercise 3

```
repeat([1, 2, 3, 4]) => [1, 2, 2, 3, 3, 3, 4, 4, 4, 4]

repeat([5]) => [5, 5, 5, 5, 5]

repeat([2, 0, 1, 3]) => [2, 2, 1, 3, 3, 3]
```

(d) Lab 11 Excercise 1

```
scramble("mossy", 1) => 'npttz'

scramble("racecar", 3) => 'udfhfdu'

scramble("hello", 0) => 'hello'

scramble("hello", -1) => 'gdkkn'

scramble("zoo", 2) => 'bqq'
```

(e) Lab 11 Excercise 2

```
arrange("AaBbCcDd") => 'ABCDdcba'

arrange("MOM DAD") => 'ADDMMO'

arrange("Mom Dad") => 'DMomda'

arrange("A Testing TEST") => 'AESTTTtsnige'

arrange("A1B2 !@ C3D4") => 'ABCD'
```

(f) Lab 11 Excercise 3

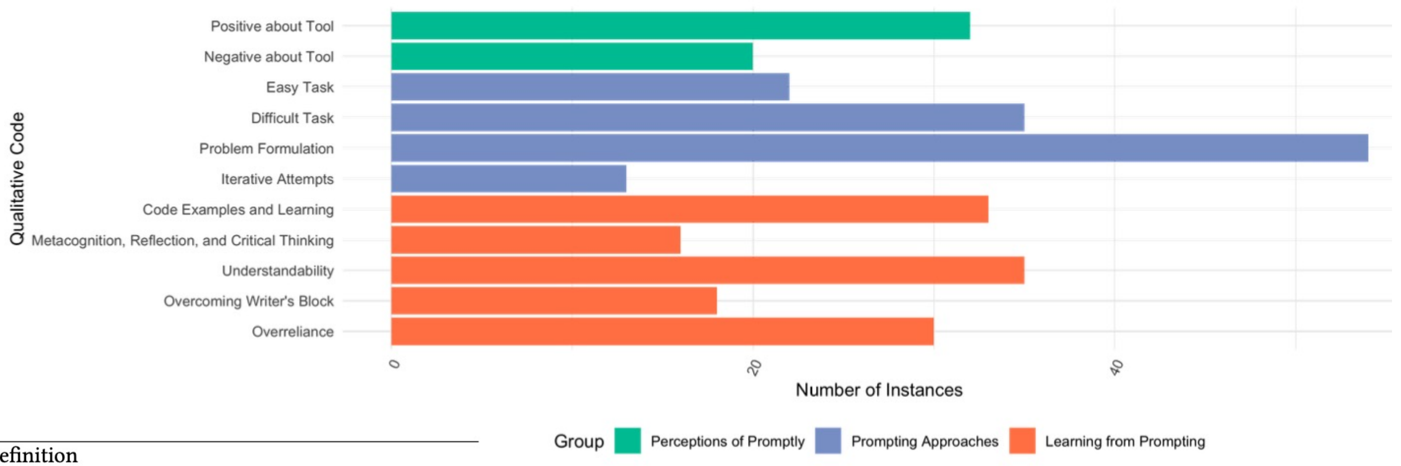
```
speak("Hello World!") => 'H3ll0 W0rld!'

speak("STEAK") => '5734K'

speak("Programming is easy") => 'Pr0gr4mm1ng 15 345y'
```

RQ1: How do students interact with Prompt Problems while learning to program?

THE LONDON SCHOOL OF ECONOMICS AND POLITICAL SCIENCE

Results of student interactions

Large scale study
202 students (Lab 10)
147 students (Lab 11)
August 2023



Theme	Code	Code Definition
Perceptions of the Tool	Positive Sentiment Towards Tool	Student expresses a form of positive sentiment towards Promptly as a tool (e.g., UI, feedback).
	Negative Sentiment Towards Tool	Student expresses a negative sentiment towards Promptly as a tool.
Prompting Approaches	Easy Task(s)	Student indicates that the task of successfully generating a prompt was easy.
	Difficult Task(s)	Student indicates that the task of successfully generating a prompt was difficult.
	Problem Formulation	Students response discusses their approach or experiences with formulating a description of the problem.
	Iterative Attempts	Student discusses their experience with or approach to iterative modifications to their initially incorrect prompt(s) or creation of successive new prompts.
Learning from Prompting	Code Examples and Learning	Student mentions learning from seeing different approaches to the solution, as generated via ChatGPT.
	Metacognition	Student mentions their thought processes.
	Understandability	Student mentions aspects of the generated code that improves or limits their ability to interpret it.
	Overcoming Writers Block	Students indicate that prompting, as a skill, is or could be useful for situations where they know the task to perform but are unsure of how to form an initial solution.
	Overreliance	Student indicates they might come to rely on prompting.

Table 2. The table of themes, codes, and code definitions.

RQ2: How do students perceive Prompt Problems affecting their learning of programming concepts?

THE LONDON SCHOOL OF ECONOMICS AND POLITICAL SCIENCE

Next-Step Hint Generation for Introductory Programming Using Large Language Models

Authors: Lianne Roest, Hieke Keuning, Johan Jeuring Authors Info & Claims

ACE '24: Proceedings of the 26th Australasian Computing Education Conference • January 2024 • Pages 144–153
• https://doi.org/10.1145/3636243.3636259

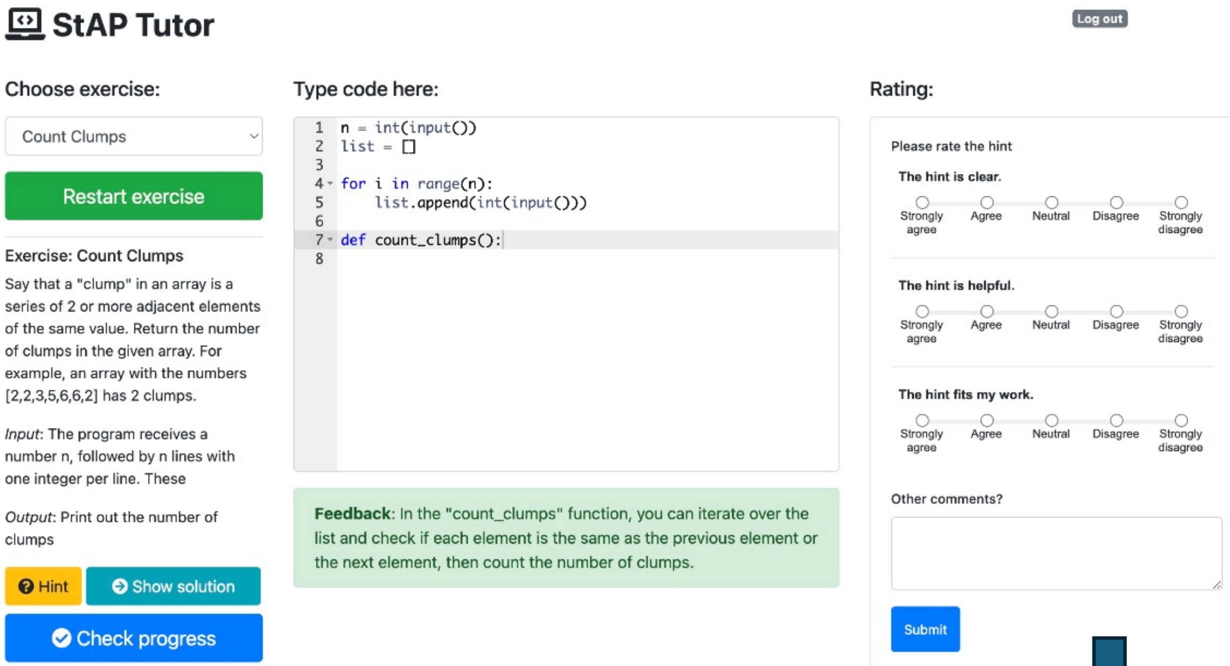
Example practices: code generation

- Setting: introductory Python exercises [8]
- Goal: how to design prompts for LLMs to produce **next-step hints** and enhance them with explanations.
- Research questions:
  - RQ1: To what extent can we use LLMs to generate informative and effective next-step hints for Python introductory programming exercises?
  - SQ1 What prompt characteristics are suitable for generating effective next-step hints with LLMs?
  - SQ2 What are students' and experts' perceptions of the quality of LLM-generated next-step hints?
- Approach: "StAP-tutor" (Step Assisted Programming tutor)
  - Input is a dataset with sequences of steps students take when solving a programming problem (148 participants)
  - Sequences are used to engineer a prompt for generating next-step hints
  - StAP-tutor allows the students to practice Python with the help of next-step hints.

THE LONDON SCHOOL OF ECONOMICS AND POLITICAL SCIENCE

Exercise for prompt engineering

Pies
A single pie costs A dollars and B cents in the cafe. Calculate how many dollars and cents one needs to pay for N pies. *Input*: The program receives three numbers A - how many dollars a pie costs; B - how many cents a pie costs; N - how many pies do you need to buy *Output*: Print out two numbers: the cost of N pies in dollars and cents.



Example hints using different prompt instructions

```
v1 = int(input())
v2 = int(input())
v3 = int(input())
v2= v2 * v3
v1= v1 * v3
```

Prompt	Hint
What is the next step	The next step is to calculate the total cost of N pies by adding v1 and v2.
Give a hint for the next step	The next step should involve calculating the total cost of N pies using the updated values of v1 and v2, and then converting the total cost into dollars and cents.
Explain the next step for a student	It seems like you are trying to calculate the total cost of N pies. You have correctly multiplied the cents by the number of pies, but you also need to multiply the dollars by the number of pies. You can do this by updating the value of v1 to be v1 multiplied by v3. Then you can use the updated values of v1 and v2 to calculate the total cost of N pies.

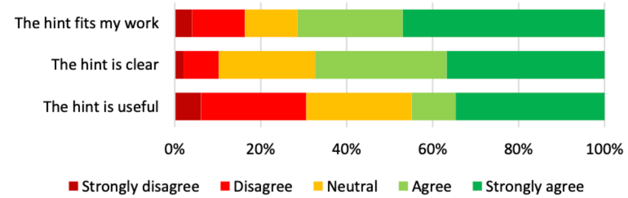


Figure 4: Student hint ratings (n=48).

THE LONDON SCHOOL OF ECONOMICS AND POLITICAL SCIENCE

Evaluating Large Language Models in Class-Level Code Generation

Authors: Xueying Du, Mingwei Liu, Kaixin Wang, Hanlin Wang, Junwei Liu, Yixuan Chen, Jiayi Feng, Chaofeng Sha, Xin Peng, Yiling Lou Authors Info & Claims

ICSE '24: Proceedings of the IEEE/ACM 46th International Conference on Software Engineering • May 2024 • Article No.: 81 • Pages 1–13 • https://doi.org/10.1145/3597503.3639219

▪ Example practices: code generation

▪ Goal: extend "simple" benchmarks for code generation to more "complex" scenarios. [9]

Table 1: Existing Benchmarks for Code Generation

Benchmark	Time	Language	Manual/Automated	Source	Granularity	#Tasks	#Tests	#LOC	#Tokens	Input Information
Concode [35]	2018	Java	Automated	Github	Function-level	2,000	-	-	26.3	NL
CoNaLA [66]	2018	Python	Automated	Stack Overflow	Statement-level	500	-	1	-	NL
APPS [32]	2021	Python	Automated	Contest Sites	Competitive	5,000	13.2	21.4	58	NL + Example Inputs/Outputs
HumanEval [21]	2021	Python	Manual	-	Function-level	164	7.7	11.5	24.4	NL + Function Signature + Example Inputs/Outputs
MBPP [15]	2021	Python	Manual	-	Function-level	974	3.0	6.8	24.2	NL
math-qa [15]	2021	Python	Manual	Math Study Sites	Statement-level	2,985	-	7.6	24.6	NL
Multi-HumanEval [14]	2022	Multilingual	Manual	-	Function-level	164	7.7	11.5	24.4	NL + Function Signature + Example Inputs/Outputs
MBXP [14]	2022	Multilingual	Manual	-	Function-level	974	3.0	6.8	24.2	NL
multi-math-qa [14]	2022	Multilingual	Manual	Math Study Sites	Statement-level	2,985	-	7.6	24.6	NL
CodeContests [43]	2022	Python, C++	Automated	Contest Sites	Competitive	165	203.7	59.8	184.8	NL + Example Inputs/Outputs
DS-1000 [40]	2022	Python	Automated	Stack Overflow	Statement-level	1,000	1.6	3.8	12.8	NL
HumanEval+ [44]	2023	Python	Manual	-	Function-level	164	774.8	11.5	24.4	NL + Function Signature + Example Inputs/Outputs
CoderEval [67]	2023	Python, Java	Automated	Github	Function-level	230	-	30	108.2	NL + Function Signature
ClassEval	**2023**	**Python**	**Manual**	-	**Class-level**	**100**	**33.1**	**45.7**	**123.7**	**Class Skeleton**

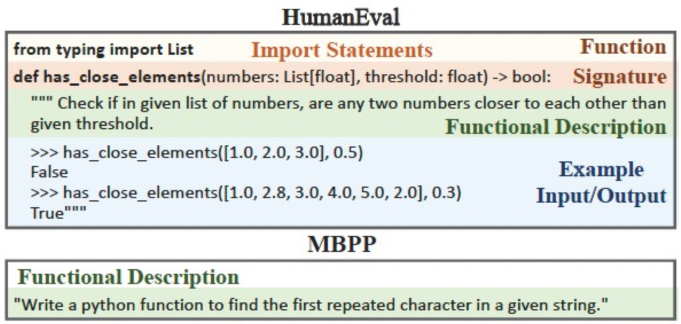


Figure 1: Examples in Existing Benchmarks

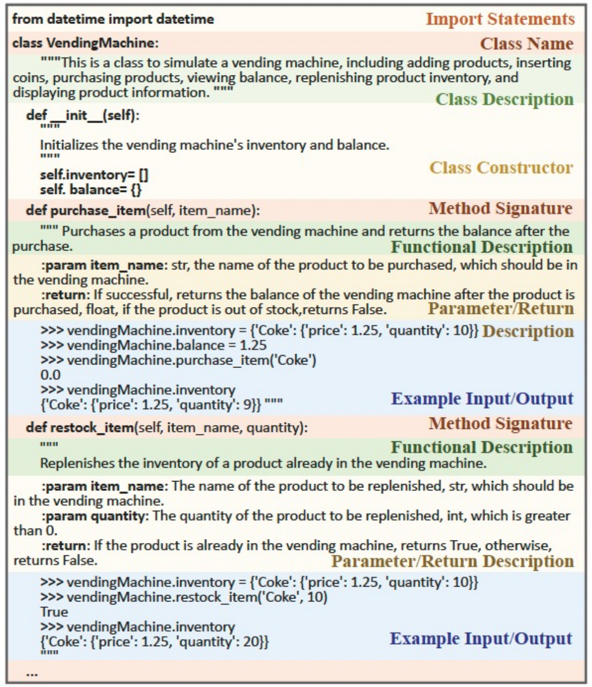
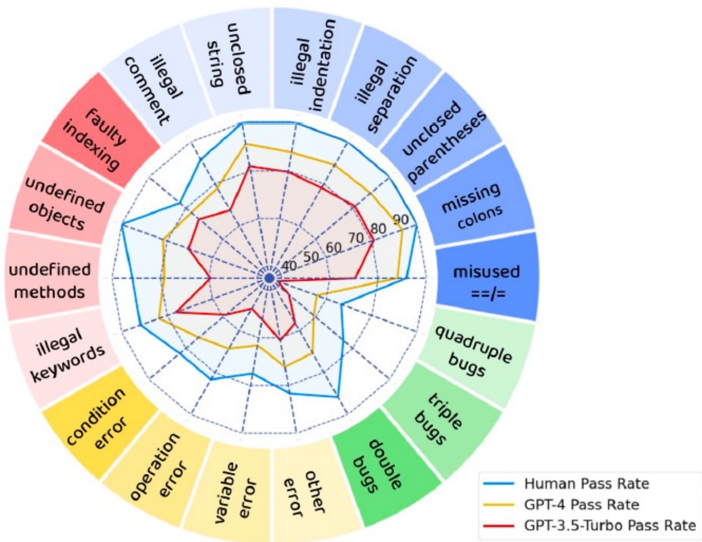


Figure 2: An Example of Class Skeleton in ClassEval

THE LONDON SCHOOL OF ECONOMICS AND POLITICAL SCIENCE

▪ Example practices: LLM debugging



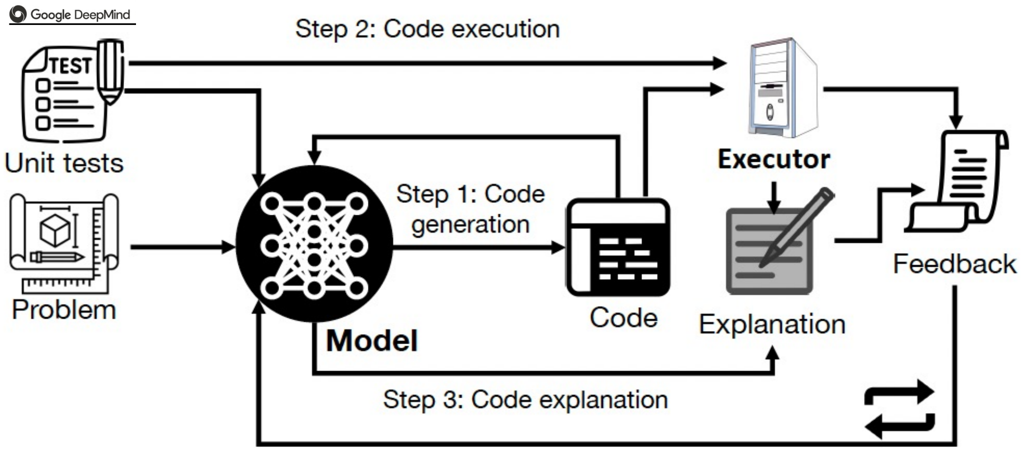
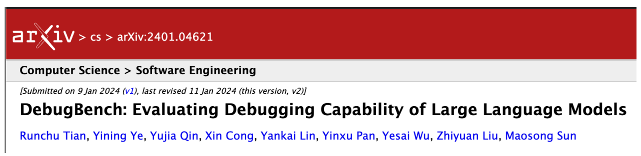


Figure 1: SELF-DEBUGGING for iterative debugging using a large language model. At each debugging step, the model first generates new code, then the code is executed and the model explains the code. The code explanation along with the execution results constitute the feedback message, based on which the model infers the code correctness and then adds this message to the feedback. The feedback message is then sent back to the model to perform more debugging steps. When unit tests are not available, the feedback can be purely based on code explanation.

- ✓ Dataset: 4,253 code examples (from LeetCode)
- ✓ 4 “bug” categories (syntax, reference. logic, misc)
- ✓ Comparative analysis (GPT-3.5, GPT-4, CodeLLama, BLOOM)

- ✓ Focus on *rubber-duck debugging* (no human intervention)
- ✓ Tested against several benchmarks, with baseline improvements of 2% - 12%.


arXiv > cs > arXiv:2401.04621

Computer Science > Software Engineering

[Submitted on 9 Jan 2024 (v1), last revised 11 Jan 2024 (this version, v2)]

DebugBench: Evaluating Debugging Capability of Large Language Models

Runchu Tian, Yining Ye, Yujia Qin, Xin Cong, Yankai Lin, Yinxu Pan, Yesai Wu, Zhiyuan Liu, Maosong Sun


arXiv > cs > arXiv:2304.05128

Computer Science > Computation and Language

[Submitted on 11 Apr 2023 (v1), last revised 5 Oct 2023 (this version, v2)]

Teaching Large Language Models to Self-Debug

Xinyun Chen, Maxwell Lin, Nathanael Schärli, Denny Zhou

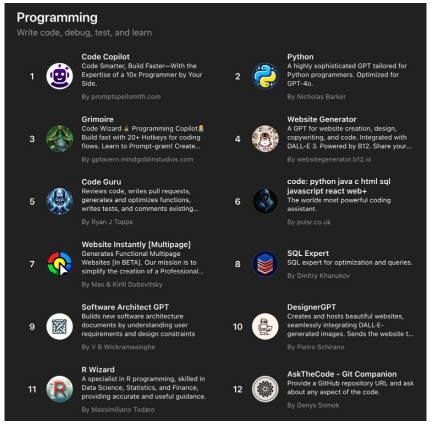
THE LONDON SCHOOL OF ECONOMICS AND POLITICAL SCIENCE

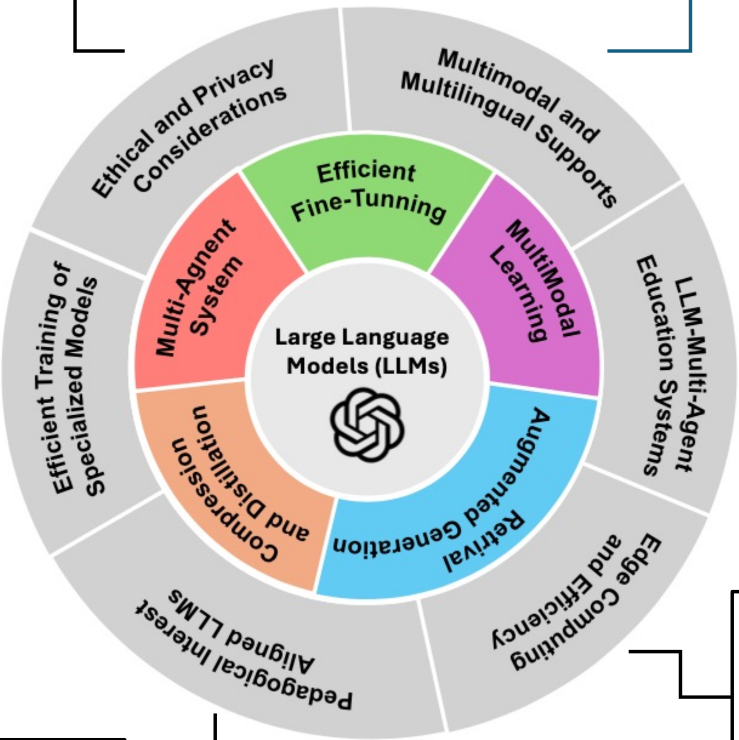
▪ Future directions [2]

- ✓ Data security, student privacy, bias mitigation
- ✓ Transparency in model training
- ✓ Inclusive models reflecting the diversity of the student population

- ✓ Tackle equity and inclusion through multilingual and multimodal LLMs
- ✓ Models that understand cultural nuances, colloquial expressions, and regional educational standards

- ✓ LLMs tailored to specific domains





- ✓ Collaboration framework for complex tasks based on conversation procedures
- ✓ LLM-based grader/critic agents + human supervision

Edge computing* and efficiency
- ✓ Lightweight models for edge deployment
- ✓ Local processing of sensitive data
- ✓ Equitable and offline access to educational tools in low-resource settings

Pedagogical interest-aligned LLMs
- ✓ Add prior information, through advanced generation techniques (such as RAG*)
- ✓ Collect large pedagogical datasets to fine-tune LLMs

RAG (retrieval-augmented generation): improve the accuracy of LLMs with facts fetched from external sources
Edge computing: processing data closer to the end user, to reduce latency and increase content delivery speed

References

- [1] Introduction to programming. BBC Bitesize. [link]
- [2] Large Language Models for Education: A Survey and Outlook. arXiv:2403.18105 [cs.CL] https://doi.org/10.48550/arXiv.2403.18105
- [3] Large language models in education: A focus on the complementary relationship between human teachers and ChatGPT. Education and Information Technologies (2023) 28:15873–15892 https://doi.org/10.1007/s10639-023-11834-1
- [4] Generative Artificial Intelligence for Education and Pedagogy. Cornell University. [link]
- [5] Experiences from Using Code Explanations Generated by Large Language Models in a Web Software Development E-Book. https://doi.org/10.1145/3545945.3569785
- [6] Comparing Code Explanations Created by Students and Large Language Models. https://doi.org/10.1145/3587102.3588785
- [7] Interactions with Prompt Problems: A New Way to Teach Programming with Large Language Models. https://doi.org/10.48550/arXiv.2401.10759
- [8] Next-Step Hint Generation for Introductory Programming Using Large Language Models. https://doi.org/10.1145/3636243.3636259
- [9] Evaluating Large Language Models in Class-Level Code Generation. https://doi.org/10.1145/3597503.3639219
- [10] DebugBench: Evaluating Debugging Capability of Large Language Models. https://arxiv.org/abs/2401.04621
- [11] Teaching Large Language Models to Self-Debug. https://arxiv.org/abs/2304.05128