



# Evaluating AI-generated code for C++, Fortran, Go, Java, Julia, Matlab, Python, R, and Rust

Patrick Diehl, Noujoud Nader, Steve Brandt, and Hartmut Kaiser

AMTE @ EuroPar 24

August, 2024

LA-UR-24-29141



# Motivation

## Large language models are emerging for certain tasks

- Recognize and generate text
- Chat bots

However, most of these tasks are related to language processing and producing text to be read by humans. We neglect the fact here that these models can do hallucination and produce wrong facts or incorrect answers.

Our idea here was to see how good are these models to produce language understood by computers (programming languages)?

# Outline

Choice of Programming languages















Test problems

Quality of the generated software

Code metrics

Conclusion and Outlook

## Choice of Programming languages

Aug 2024	Aug 2023	Change	Programming Language	Ratings	Change
1	1		 Python	18.04%	+4.71%
2	3	▲	 C++	10.04%	-0.59%
3	2	▼	 C	9.17%	-2.24%
4	4		 Java	9.16%	-1.16%
5	5		 C#	6.39%	-0.65%
6	6		 JavaScript	3.91%	+0.62%
7	8	▲	 SQL	2.21%	+0.68%
8	7	▼	 Visual Basic	2.18%	-0.45%
9	12	▲	 Go	2.03%	+0.87%
10	14	▲	 Fortran	1.79%	+0.75%
11	13	▲	 MATLAB	1.72%	+0.67%
12	23	▲	 Delphi/Object Pascal	1.63%	+0.83%
13	10	▼	 PHP	1.46%	+0.19%
14	19	▲	 Rust	1.28%	+0.39%
15	17	▲	 Ruby	1.28%	+0.37%
16	18	▲	 Swift	1.28%	+0.37%
17	9	▼	 Assembly language	1.21%	-0.13%
18	27	▲	 Kotlin	1.13%	+0.44%
19	16	▼	 R	1.11%	+0.19%
20	11	▼	 Scratch	1.09%	-0.13%

# Test problems

## Gathering the code from ChatGPT

- We use three numerical examples: numerical integration (NI), a conjugate gradient solver (CGS), and a parallel one-dimensional heat equation solver (PHS) using finite differences.
- The code complexity increases with each example. We used the free version of ChatGPT 3.5 and the paid version ChatGPT 4.0 for our study.
- We used ChatGPT to generate the codes on 02/27/2024 for the last example and 06/05/2024 for the others.

## Example I: Numerical integration (NI)

The following queries were used to obtain the source code;

1. Write a language code to compute the area between  $-\pi$  and  $2/3\pi$  for  $\sin(x)$  and validate it.

Here, we want to validate if ChatGPT can write a code to evaluate

$$\int_{-\pi}^{2/3\pi} \sin(x) dx. \quad (1)$$



## Example 2: Conjugate gradient solver (CGS)

2. Write a conjugate gradient solver in language to solve  $A$  times  $x$  equals  $b$  and validate it.

Here, we want to evaluate whether ChatGPT can write a conjugate gradient solver and apply it to a linear system of equations, i.e.

$$A^{n \times n} \cdot x^n = b^n \text{ with } n \in \mathbb{Z}^+, A = A^T, \text{ and } x^T A x > 0, \text{ for all } X \in \mathbb{R}^n. \quad (2)$$

For more details about the conjugate gradient solver, we refer to Jonathan Richard Shewchuk et al. “An introduction to the conjugate gradient method without the agonizing pain” (1994).

## Example 3: Parallel one-dimensional heat equation solver (PHS) using finite difference

3. Write a parallel 1D heat equation solver using finite difference in language. Here, we want to evaluate whether ChatGPT can write the code to solve

$$\frac{\partial u}{\partial t} = \alpha \frac{\partial^2 u}{\partial x^2}, \quad 0 \leq x < L, t > 0 \quad (3)$$

where  $\alpha$  is the material's diffusivity. For the discretization in space a finite difference scheme

$$u(x_i, t + 1) = u(x_i, t) + dt \alpha \frac{u(t, x_{i-1}) - 2u(t, x_i) + u(t, x_{i+1}))}{2h} \quad (4)$$

We did not specify the how to generate the grid, i.e. equidistant nodal spacing with  $n$  grid points  $x = \{x_i = i \cdot h \in \mathbb{R} | i = 0, \dots, n - 1\}$ , nor what time integration method to use, e.g. the Euler method.

## Quality of the generated software

## Metric

In this section, we assess the quality of the AI-generated software:

- First, we checked whether the code compiles with a recent compiler.
- Second, we checked whether the code executed without segmentation faults or other runtime errors.
- Third, we checked whether the code produced a correct result. The results for our test cases were the following:

$$\int_{-\pi}^{2/3\pi} \sin(x) dx = -\cos(2/3\pi) + \cos(-\pi) = -0.5$$

We solve  $M \times x = b$  with the following values  $A = \begin{pmatrix} 4 & 1 \\ 1 & 3 \end{pmatrix}$  and  $b = \begin{pmatrix} 1 \\ 2 \end{pmatrix}$ . The solution  $x$  is  $(0.09090909 \quad 0.63636364)^T$ .

# Numerical integration (NI) and Conjugate gradient solver (CGS)

**Table 1.** Three aspects of code quality: Compilation (Did the code compile with a recent compiler?), Runtime (Did the code execute without errors?), and Correctness (Did the code compute correct results?).

Attribute	V	C++	Python	Julia	Java	Go	Rust	Fortran	Matlab	R
<b>Numerical integration (NI)</b>										
Compilation	3.5	✓	-	-	✓	✓	✓	✗	-	-
	4.0	✓	-	-	✓	✓	✓	✓	-	-
Runtime	3.5	✓	✓	✓	✓	✓	✓	✓	✓	✓
	4.0	✓	✗	✓	✓	✓	✓	✓	✓	✓
Correctness	3.5	✓	✓	✓	✓	✓	✓	✓	✓	✓
	4.0	✓	✗	✗	✗	✗	✗	✗	✗	✗
<b>Conjugate gradient solver (CGS)</b>										
Compilation	3.5	✓	-	-	✓	✓	✗	✗	-	-
	4.0	✓	-	-	✓	✓	✓	✓	-	-
Runtime	3.5	✓	✓	✓	✓	✓	✓	✓	✓	✓
	4.0	✓	✓	✗	✓	✓	✓	✓	✓	✗
Correctness	3.5	✓	✓	✓	✓	✗	✓	✓	✓	✓
	4.0	✓	✓	✓	✓	✓	✓	✓	✓	✗

# Parallel one-dimensional heat equation solver (PHS) using finite differences

**Table 2.** Three aspects of code quality for the **parallel heat equation solver (PHS)**: Compilation (Did the code compile with a recent compiler?), Runtime (Did the code execute without errors?), and Correctness (Did the code compute correct results?).

Attribute	V	C++	Kokkos	HPX	Python	Julia	Java	Go	Rust	Fortran	Matlab	R
Compilation	3.5	✓	✓	✓	-	-	✓	✓	✗	✗	-	-
	4.0	✗	✓	✓	-	-	✓	✓	✓	✓	-	-
Runtime	3.5	✓	✓	✗	✗	✓	✗	✗		✓	✓	✗
	4.0	✓	✓	✓	✗	✗	✓	✗	✓	✓	✓	✓
Correctness	3.5	✓	✗	✓	✗	✗	✓	✗		✗	✓	✓
	4.0	✓	✗	✗	✓	✓	✓	✗	✗	✗	✓	✓

## Common issues

Here we present the list of common issues which we observed while debugging:

- **Compilation:**

For NI and CGS, most compilation errors were minor and could be easily fixed. For PHS, most of the errors were related to parallelism, and sometimes knowledge about the parallel programming language or library was needed to fix the problem. In total, we had five compilation errors for all examples.

- **Runtime:**

Most of the runtime errors were minor and could be easily fixed. Most were type errors, undefined variables, and undefined functions for interpreted languages. Other errors were index out-of-bound exceptions for the heat equation solver for the first and last element with the stencil. Some errors were related to the parallelism and knowledge about the parallel programming language or library was required to address them.

## Common issues

Here we present the list of common issues which we observed while debugging:

- Correctness:

For NI, the ChatGPT 3.5 versions produced all the correct results. The ChatGPT 4.0 version computed the integral as 3.5 because it used absolute values for the function evaluation. After removing the absolute values all codes computed the correct result. For the conjugate gradient solver, all except two codes produced the correct result. For the single-threaded codes, overall most results were correct. For the parallel codes, 11 codes produced correct results, and 10 codes did not.



## Code metrics

## Lines of code

The lines of code were determined with the Linux tool `cloc` and the larger amount of code lines between the ChaptGPT 3.5 and ChatGPT 4.0 were chosen. The difference was between one and five lines of code.

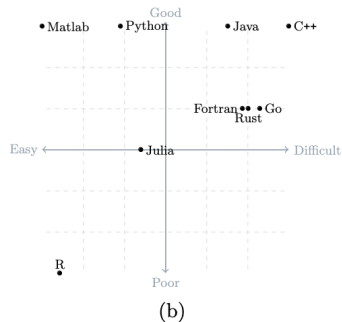
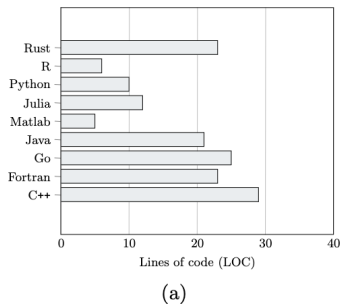
## More comprehensive metric

- The lines of code metric, however, does not measure quality well. For this, we use the Constructive Cost Model (COCOMO). COCOMO is a general model with no specialization for parallel programming. However, the HPC community never provided a similar model taking parallel and distributed computing in mind. We use this to classify if it was easy or difficult to write the code. We use the average of the COCOMO metric for version 3.5 and version 4.0 for the classification of easy and difficult.
- We use the three quantities of interest in Table 1-2 for the code quality. We use the following metric

$$q(\text{language}) := \frac{1}{2} \left( \frac{\text{comp} + \text{run} + \text{correct}}{3} + \frac{\text{comp} + \text{run} + \text{correct}}{3} \right) \quad (5)$$

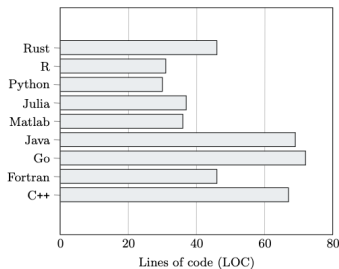
to classify the code quality from poor to good.

# Numerical integration (NI)

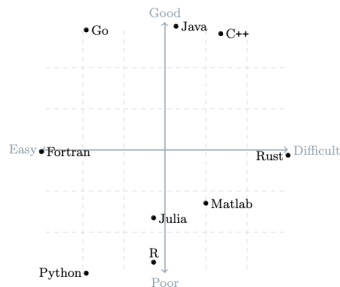


**Fig. 1.** Software engineering metrics for the **numerical integration example**: (a) Lines of code for all implementations using the maximal lines of code. The numbers were determined with the Linux tool *loc* and (b) Two-dimensional classification using the computational time and the COCOMO model.

# Conjugate gradient solver (CGS)



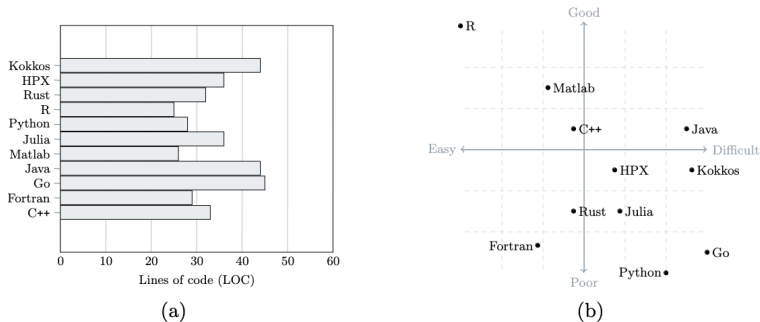
(a)



(b)

**Fig. 2.** Software engineering metrics for the **conjugate gradient solver**: (a) Lines of code for all implementations using the maximal lines of code. The numbers were determined with the Linux tool *loc* and (b) Two-dimensional classification using the computational time and the COCOMO model.

# Parallel one-dimensional heat equation solver (PHS) using finite differences



**Fig. 3.** Software engineering metrics for the parallel heat equation solver: (a) Lines of code for all implementations using the maximal lines of code. The numbers were determined with the Linux tool *cloc* and (b) Two-dimensional classification using the computational time and the COCOMO model.

## Conclusion and Outlook

## Conclusion and Outlook

### Conclusion

- Compilation: Most codes could be compiled
- Runtime: Some codes had run time errors
- Correctness: Not many codes produced correct results

Some languages worked better than others, maybe due to more training data?

Overall ChatGPT is not yet ready to produce source code even it was trained on source code!

### Outlook

- Check if ChatGPT can fix the bug encountered in the previous generated code.
- Check if ChatGPT can write GPU kernels
- use more complex examples