Beyond Functional Correctness

An Empirical Evaluation of Large Language Models for Text-to-Code Generation

87th Meeting of the IFIP Working Group 10.4

João R. Campos
jrcampos@dei.uc.pt
CISUC, University of Coimbra

CISUC
SOFTWARE AND SYSTEMS ENGINEERING

Myself — João R. Campos

- Assistant Professor at the University of Coimbra, CISUC, Software and Systems Engineering group (SSE)
- PhD in Informatics Engineering, ML-based OS-level Online Failure Prediction
  - ~10 years in industry before
- Researcher at Teaching: Software Security, Software Automation, Advanced Machine Learning Laboratory, Databases, Introduction to Programming, Advanced Machine Learning , Project Management (…)

Advancing dependable and secure systems by developing and tailoring state-of-the-art AI, grounded in a deep understanding of AI principles

- Devil is in the details, AI/ML will always output something and positive results look good ☺
- Recent studies observed that a high percentage of ML-based research does not hold in practice
- (but I also work with AI/ML in health, biology, and space domains - non safety-critical tasks)

Context

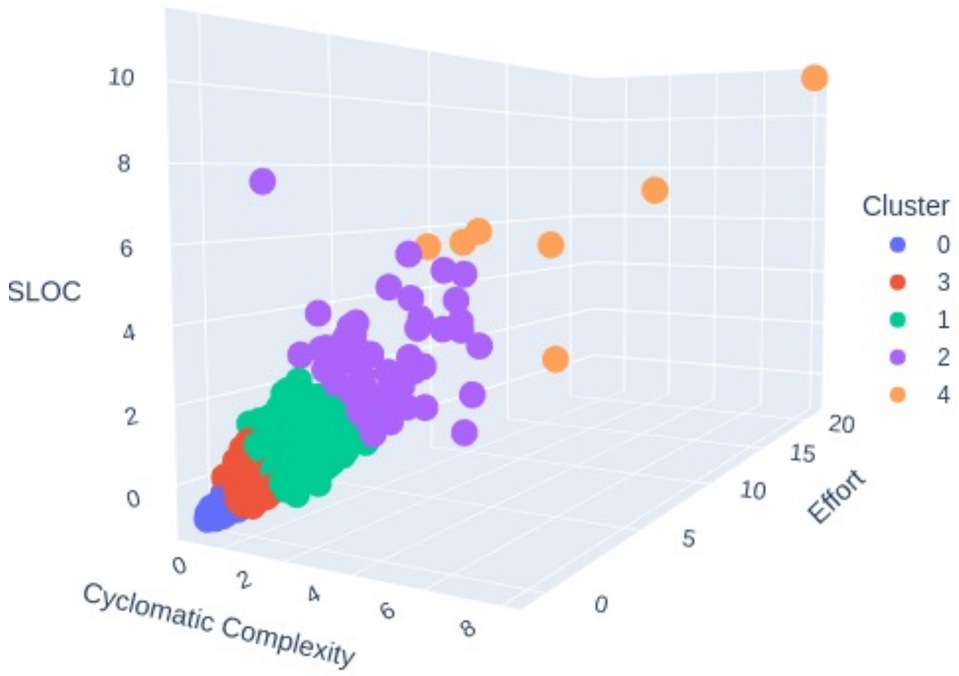
- LLMs advances in the generation of code from natural language
- LLMs are significantly limited for complex problems
- Existing benchmarking works are limited
- Structured benchmarks/processes are needed
- **Goal: define a systematic framework for assessing code generation capabilities of LLMs**

Dataset and Metrics

- CodeNet (IBM), ~4k problems, 55 languages, > 13M reference solutions

(a) Dataset Details

Languages	Python, C++
Number of problems	1651
Problem Difficulty Dist.	**0:** 796, **1:** 520, **2:** 261, **3:** 74
Test Cases per Problem	3-10



(b) Metrics

Execution-based	pass@k, outcome rate
Static Analysis	cyclomatic complexity, LLOC, SLOC

(c) LLMs

Model	Training	MoE	Params	Quant.
Qwen2.5	General	No	14b	4b
Qwen2.5-Coder	Gen.+Code	No	7b, 14b	4b, 16b (7b)
StarCoder2:Instruct	Code	No	15b	4b
Deepseek-coder-v2	Code	Yes	16b	4b

(d) Experimental settings

ICL	0-shot, 1-shot
Model Hyper-parameters	Temperature = 0.6 Top-k = 50, Top-p = 1.0
Improvement Iterations	2

Workflow

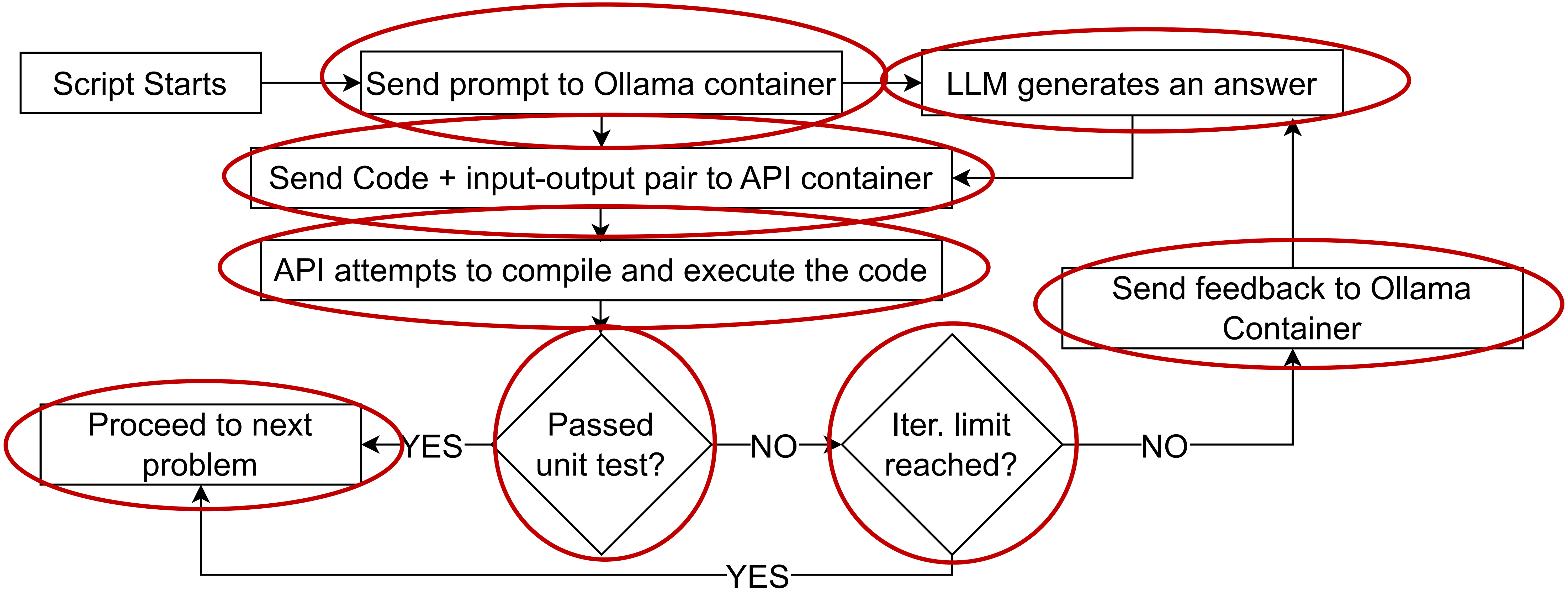
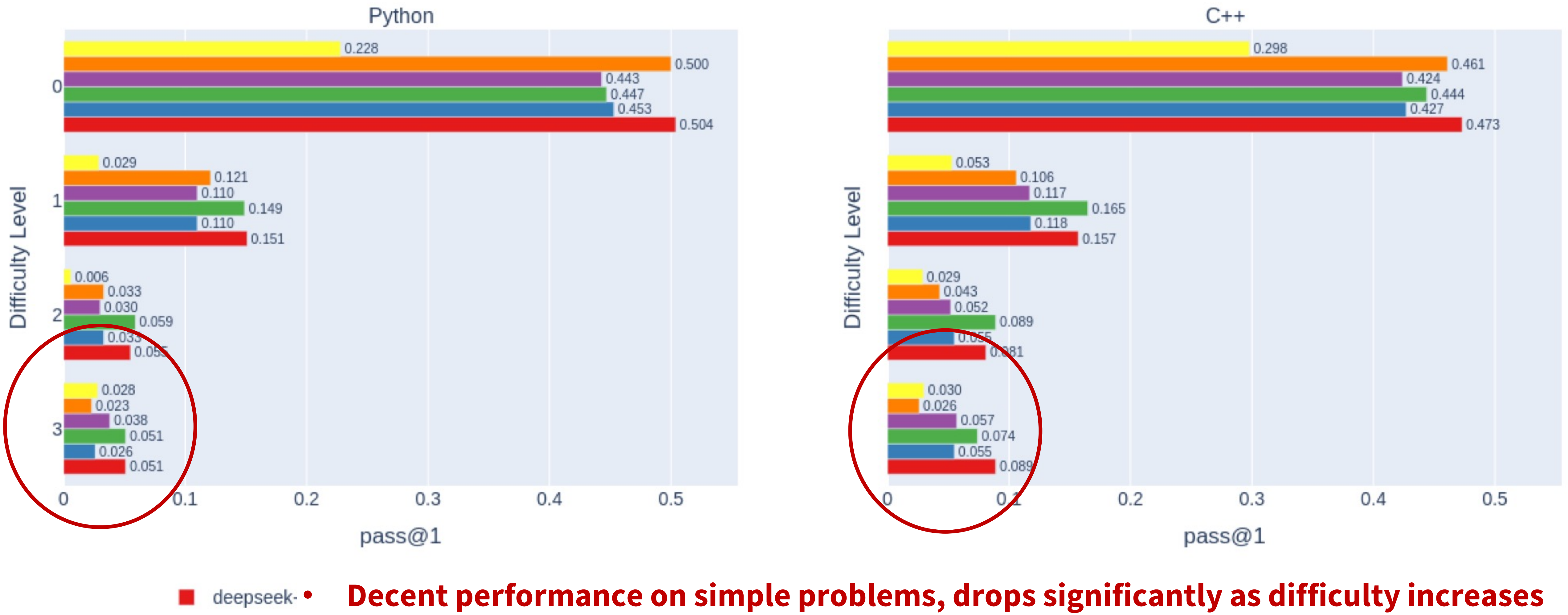


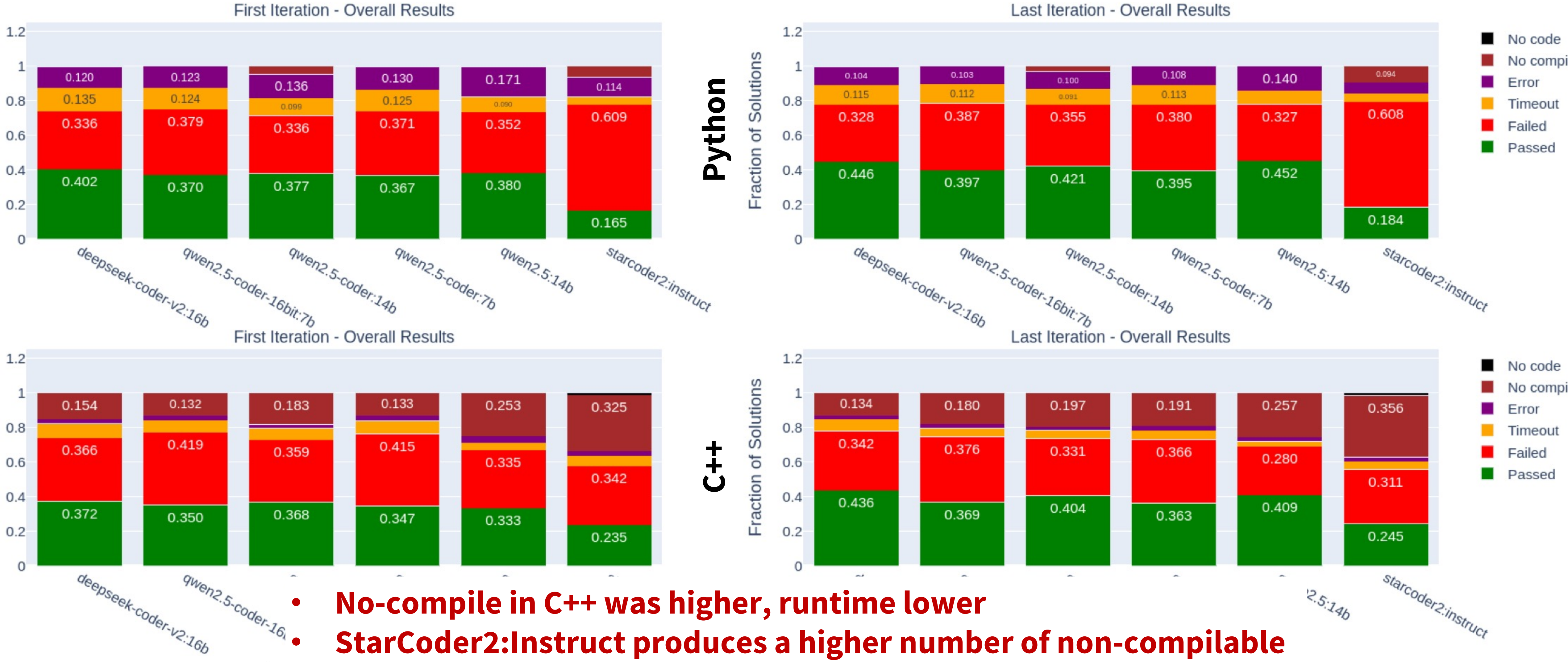
Fig. 3: Overview of the code generation procedure

pass@1 by difficulty



- **Decent performance on simple problems, drops significantly as difficulty increases**
- **Code-tuned LLMs lower perf on simple tasks but better on complex**
- **ICL showed no significant gains (the prompt was already detailed?)**

Outcome rates



- No-compile in C++ was higher, runtime lower
- StarCoder2:Instruct produces a higher number of non-compilable
- Allows understanding where the various LLMs solutions fail

Static analysis - Python

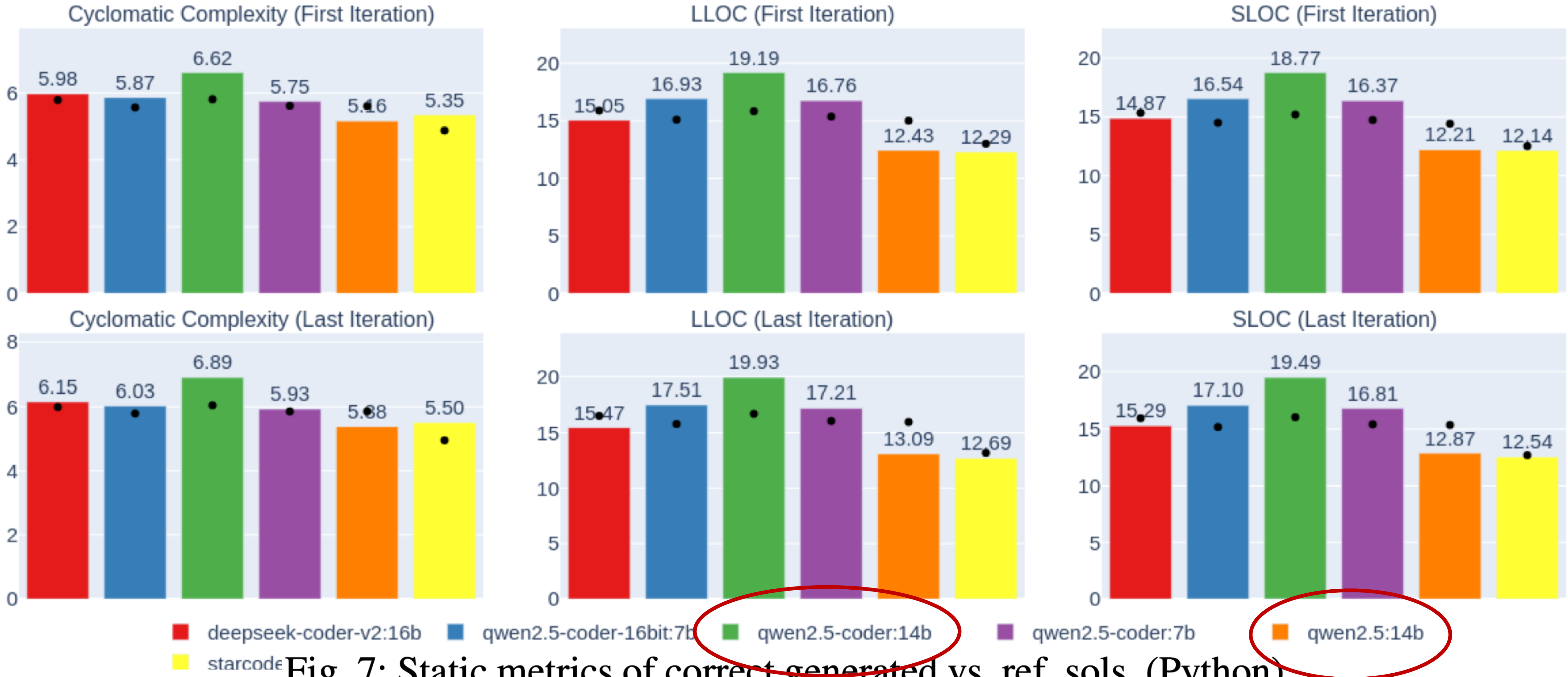


Fig. 7: Static metrics of correct generated vs. ref. sols. (Python)

Static analysis - C++

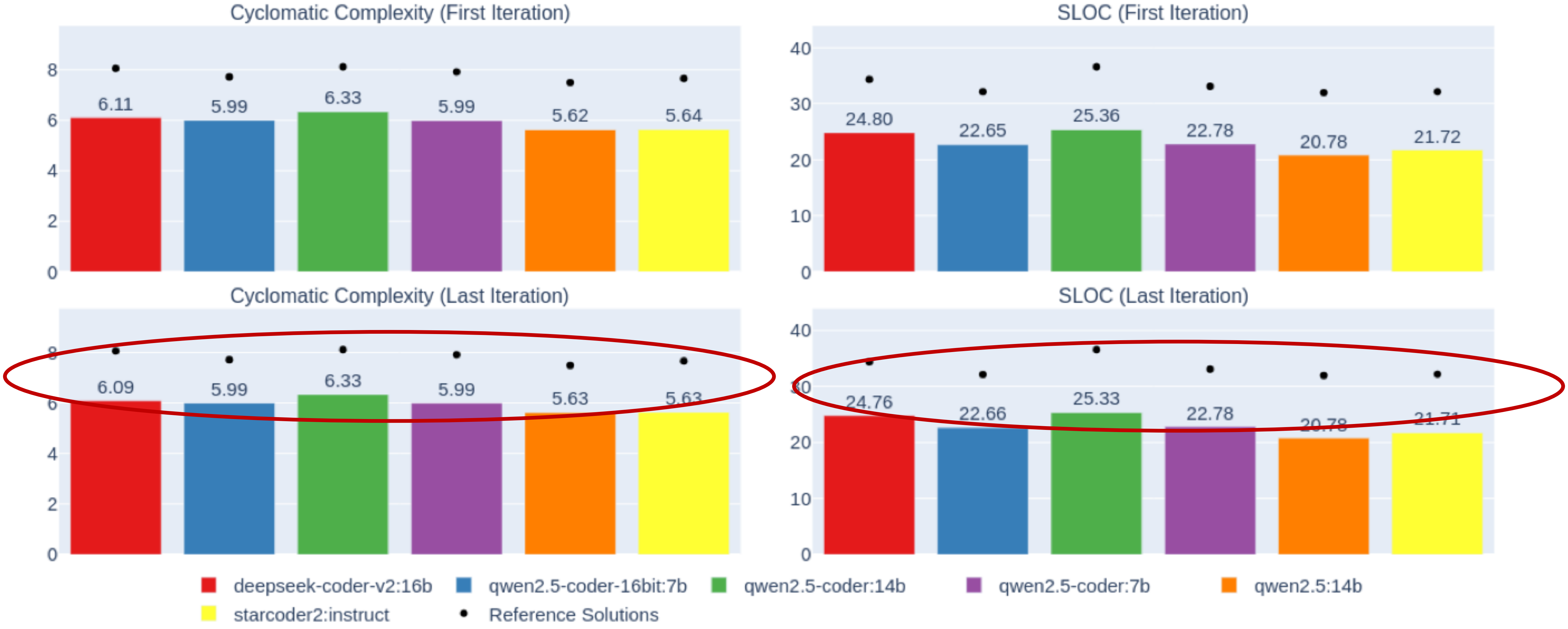


Fig. 8: Static metrics of correct generated vs. ref. sols. (C++)

Commercial – Gemini 2.0-flash & GPT 4.1-mini

Python

C++

Previous Best:
Python – (0) 0.505 ; (3) 0.051
C++ - (0) 0.473 ; (3) 0.089

Common errors – Python / C++

- Runtime errors are often caused by missing or **insufficient input validations**
  - unguarded memory access and arithmetic overflows
  - high memory allocation without checking input sizes

- Lack of robustness is particularly concerning, as with AI-assisted code generation the programmer will rely more and more on the system

- LLMs frequently omits essential checks, increasing the risk of bugs and **vulnerabilities**

João R. Campos
jrcampos@dei.uc.pt

CISUC, University of Coimbra

CISUC
SOFTWARE AND SYSTEMS ENGINEERING