Benchmarking Large Language Models (LLMs) on Floating-Point Error Classification

David Defour Lisa Taldir Eric Petit Pablo de Oliveira

Université de Perpignan via Domitia

lisa.taldir@univ-perp.fr

Plénière FPT-4

27 mars 2026

Section 1

Existing Floating-Point Errors Debugging Tools and LLMs in Code Analysis

Existing Approaches to Floating-Point Error Detection

Static Analysis

- **Polyspace** Formal verification of numerical properties
- **PRECiSA** Proof-carrying code for floating-point accuracy

Dynamic Analysis

- **Verrou** Valgrind-based tool to estimate and locate floating-point errors
- **FPChecker** Runtime floating-point exception checking
- **CADNA** Discrete stochastic arithmetic
- **Verificarlo** Instrumentation with multiple backends

Large Language Models in Code Analysis

Strengths in Software Engineering

- Strong capabilities in **code generation**, **bug detection**, and **program repair**
- Effective for **semantic code analysis** tasks
- Operate by learning implicit patterns from data rather than enforcing formally defined semantic rules
- **Key advantage** : Ability to generalize across different coding styles and programming paradigms

Large Language Models in Code Analysis

Limitations & Complementary Role

- **Critical limitation** : Absence of soundness guarantees (no formal correctness proofs)
- LLM-based analysis is best viewed as **complementary to formal methods**
- **Practical applications** :
  - Identifying likely error patterns
  - Prioritizing suspicious code regions for review
  - Providing early feedback during development

Research Questions

1. **LLM Capability** : Can LLMs detect/classify floating-point errors (overflow, underflow, cancellation) in small C functions for given inputs ?

Research Questions

1. **LLM Capability** : Can LLMs detect/classify floating-point errors (overflow, underflow, cancellation) in small C functions for given inputs ?
2. **Agentic System Design** : How to combine LLMs with existing tools (FPChecker, etc.) into a cost-effective agentic system for HPC codebases ?

Research Questions

1. **LLM Capability** : Can LLMs detect/classify floating-point errors (overflow, underflow, cancellation) in small C functions for given inputs ?
2. **Agentic System Design** : How to combine LLMs with existing tools (FPChecker, etc.) into a cost-effective agentic system for HPC codebases ?

Key trade-offs to address :

- **Token cost** vs. **Detection quality**
- **LLM generality** vs. **Tool precision** (FPChecker, etc.)
- **Small on-premise LLMs** vs. **Large cloud LLMs** (confidentiality)
- **20-line functions** vs. **HPC-scale codebases**

Section 2

Experimental Methodology, Evaluation Metric and Validation

IEEE-754 : Floating-Point Error Patterns

Underflow Result too small to represent accurately
$x < 2^{-126}$ (float), $x < 2^{-1022}$ (double)

Overflow Result exceeds representable range
$x > (2 - 2^{-23}) \times 2^{127}$ (float), $x > (2 - 2^{-52}) \times 2^{1023}$ (double)

Div. by zero $x/0$, denominator too small
Leads to $\pm\infty$ or exception

Cancellation Loss of significance
$x - x$, $(x + \delta) - x$ with $\delta \ll x$

Comparison Risky due to binary representation
`if (x == y)` on floating-point values

NaN Not a Number from invalid operations
$\infty - \infty$, $0 \times \infty$, $0/0$, $\log(-x)$, $\sqrt{-x}$

InterFLOPBench : Floating-Point Benchmark to Test Numerical Software Analysis Tools

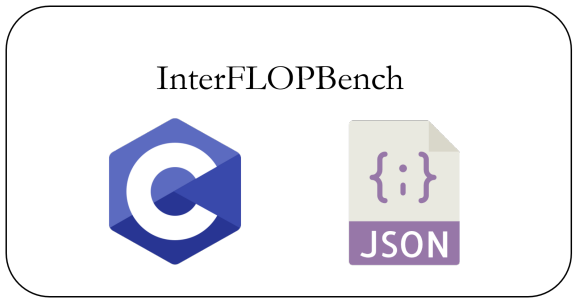


Figure – C source code with accompanying JSON metadata file.

Total :

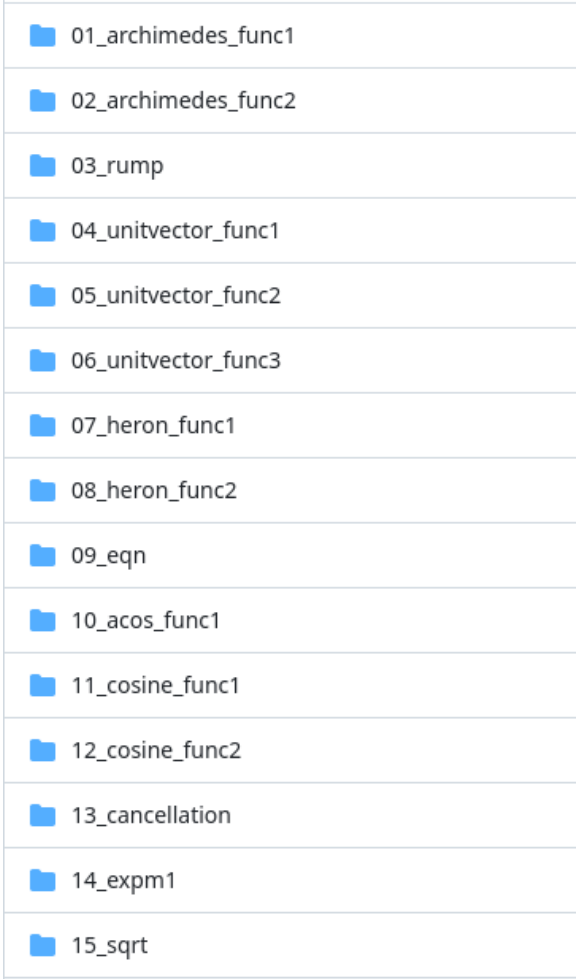
- 93 benchmarks,
- 1150 samples,
- 1315 occurrences.

Category	Occurrences
No Error	478
Comparison	255
Cancellation	214
Overflow	153
Underflow	88
Div. Zero	70
NaN	57

Table – InterFLOPBench Error Category Distribution.

InterFLOPBench : Floating-Point Benchmark to Test Numerical Software Analysis Tools

- InterFLOPBench is available at : https://github.com/interflop/InterflopBench/



InterFLOPBench : Floating-Point Benchmark to Test Numerical Software Analysis Tools

Example of kernel in InterFLOPBench.

```c
double kernel(double x, double y)
{
    double oscillator = cos(x) - y;
    double logValue = log(oscillator);
    double result = (exp(logValue / 2.0)
            + log(fabs(oscillator) + 1.0))
            / (1.0 + fabs(logValue));

    return result;
}
```

C file

```json
{"ordered_inputs": ["x", "y"],
 "ordered_outputs": ["result"],
 "datasets": [
   {"errors": ["no_error"],
    "inputs": [
       {"x": 0.0, "y": -0.5},
       {"x": 1.57, "y": -0.2}
    ]},
   {"errors": ["nan"],
    "inputs": [
       {"x": 0.0, "y": 1.5},
       {"x": 3.14, "y": 1.1}]}]}
```

Errors predicted by FPChecker for the inputs below.

JSON file

File name : 48_complex_log

This example trigger NaN if cos(x) $\leq$ y.

Micro F1 Score for Multilabel Evaluation

7 categories : cancellation, underflow, overflow, NaN, division by zero, comparison, no error

Why micro F1 ? Each (sample, label) pair weighted equally, adapted to multilabel and class imbalance.

Micro F1 Formula

$$F1_{micro} = 2 \times \frac{TP}{2TP + FP + FN}$$

Scenario	True	Predicted	Impact
Correct	overflow	overflow	TP overflow
Missing	overflow	no error	FN overflow + FP no error
False alarm	no error	overflow	FP overflow + FN no error
Under-pred	{overflow, NaN}	{overflow}	TP overflow + FN NaN
Over-pred	{NaN}	{overflow, NaN}	TP NaN + FP overflow

Table – TP/FP/FN impact.

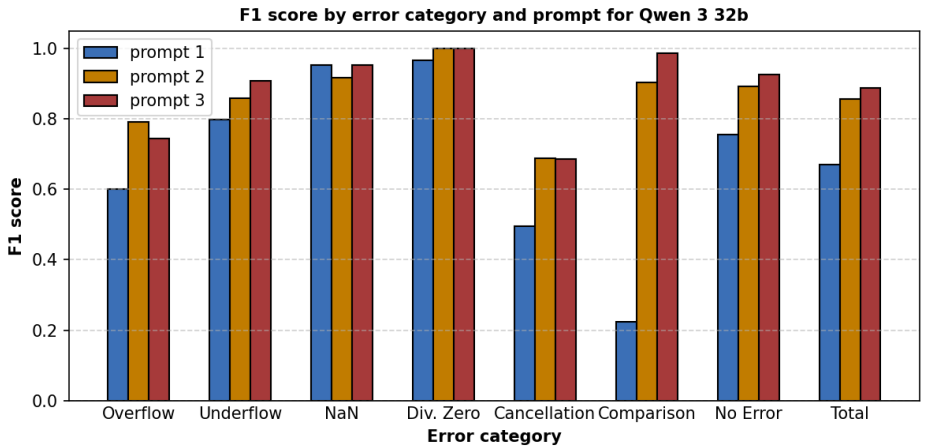
Experimental Methodology and Validation

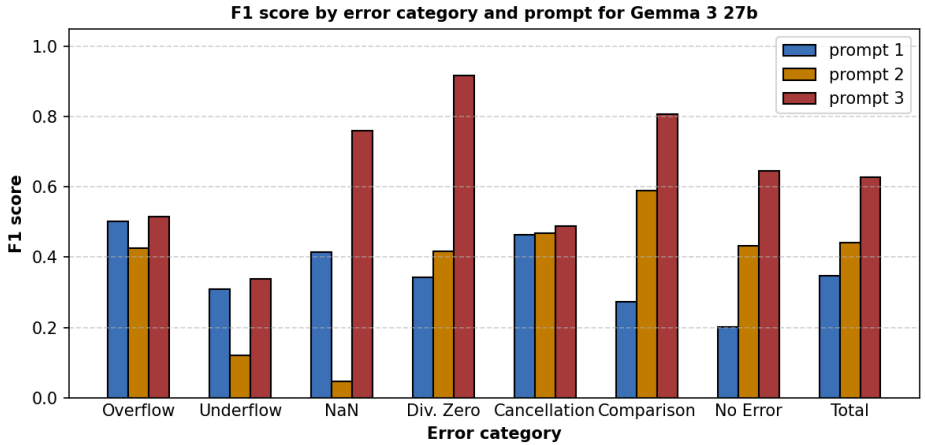
Experimental Setup :

- 14 LLMs
- Validation : FPChecker
- Metric : Micro F1-Score

Prompts :

- Prompt 1 : Provides the list of the six different-floating point error categories.
- Prompt 2 : Add definition of each error category.
- Prompt 3 : Ask the LLM for a detailed response.

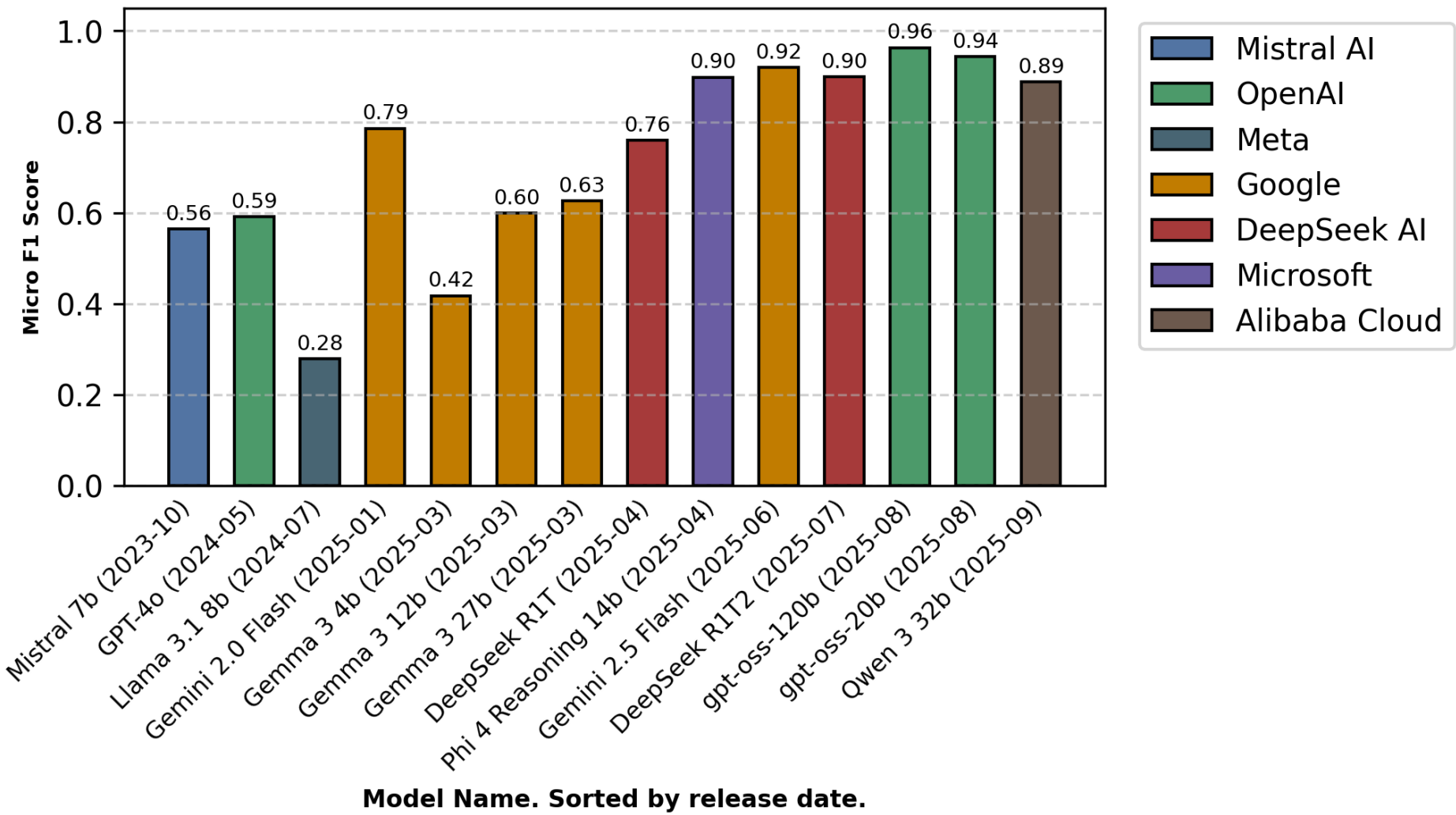




Section 3

Results

Overall Results



Discussion

Why LLMs Succeed at Detecting Floating-Point Errors

- Pattern recognition
- Perform simple computations on floating-point operations via chain-of-thought

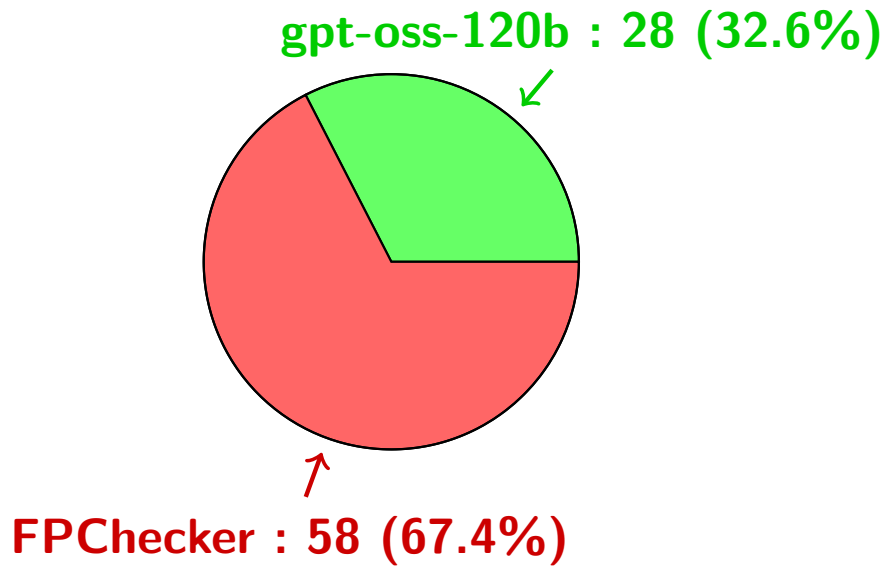
Where LLMs Fail

- Performance degrades on subtle error categories (cancellation, underflow)

Disagreement Analysis & Oracle Refinement

LLM vs. FPChecker : 86 disagreements, 28 LLM victories

- **Instruction isolation** : FPchecker cancellation checks do not handle FMA operations.
- **Explicit inequality checks** : FPChecker detects strict equality comparisons (==) but overlooks inequality checks (!=) between two floating-point variables.
- **Library function propagation** : FPChecker does not instrument `math.h` library functions (e.g. domain errors in `log` or `sqrt`).



Disagreements
(n=86 out of 1 130 test samples)

Section 4

Conclusion, Limits and Future Direction

Conclusion, Limits and Future Direction

Contribution and results :

- New benchmark : InterFLOPBench with 1 130 test samples accross 6 floating-point errors
- Best performing models score over 0.90 F1-micro score
- Prompt engineering plays a significant role (+0.19 F1-score)
- Show FPChecker limits

Limits of InterFLOPBench :

- C-only code, isolated kernel functions (not many operations, propagation of constants, known values), not all categories of errors are represented

Future Direction :

- Agentic pipeline where LLMs can leverage existing FP debugging tools and complement their coverage