Agentic AI for Floating-Point Error Detection

David Defour Lisa Taldir Eric Petit Pablo de Oliveira

Université de Perpignan via Domitia
ESPACE-DEV
France

July 6, 2026

Université Perpignan Via Domitia

Section 1

Introduction

Motivations for this talk : LLMs strengths

- Strong capabilities in code generation
- Effective for semantic code analysis task
- Operate by learning implicit patterns data

Figure – Overall score of DevQualityEval. Data from `https://devqualityeval.com/`.

Université Perpignan Via Domitia

Section 2

Can LLMs classify floating-point exceptions in small C functions for given inputs ?

InterFLOPBench [1] : Floating-Point Benchmark to Test Numerical Software Analysis Tools

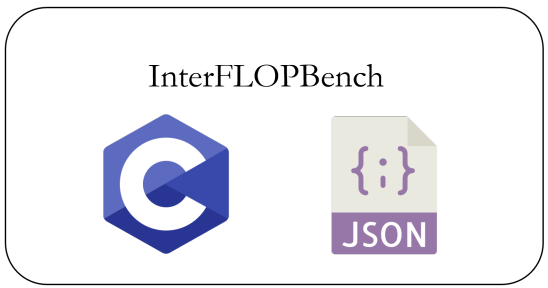


Figure – C source code with accompanying JSON metadata file. A total of **93** benchmarks, **1 150** samples (inputs) and **1 315** errors occurrences.

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

Table – InterFLOPBench Error Category Distribution.

Université Perpignan Via Domitia

1. https://github.com/interflop/InterflopBench/

Example of kernel in InterFLOPBench

C file

```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;
}
```

JSON 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.

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, and no error.

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

Figure – TP/FP/FN impact. The "True" is obtained from the dynamic exception-based tool FPChecker.

Overall Results

Full per-benchmark disagreement analysis is available online at : github.com/interflop/InterflopBench/ground_truth_analysis_table.html

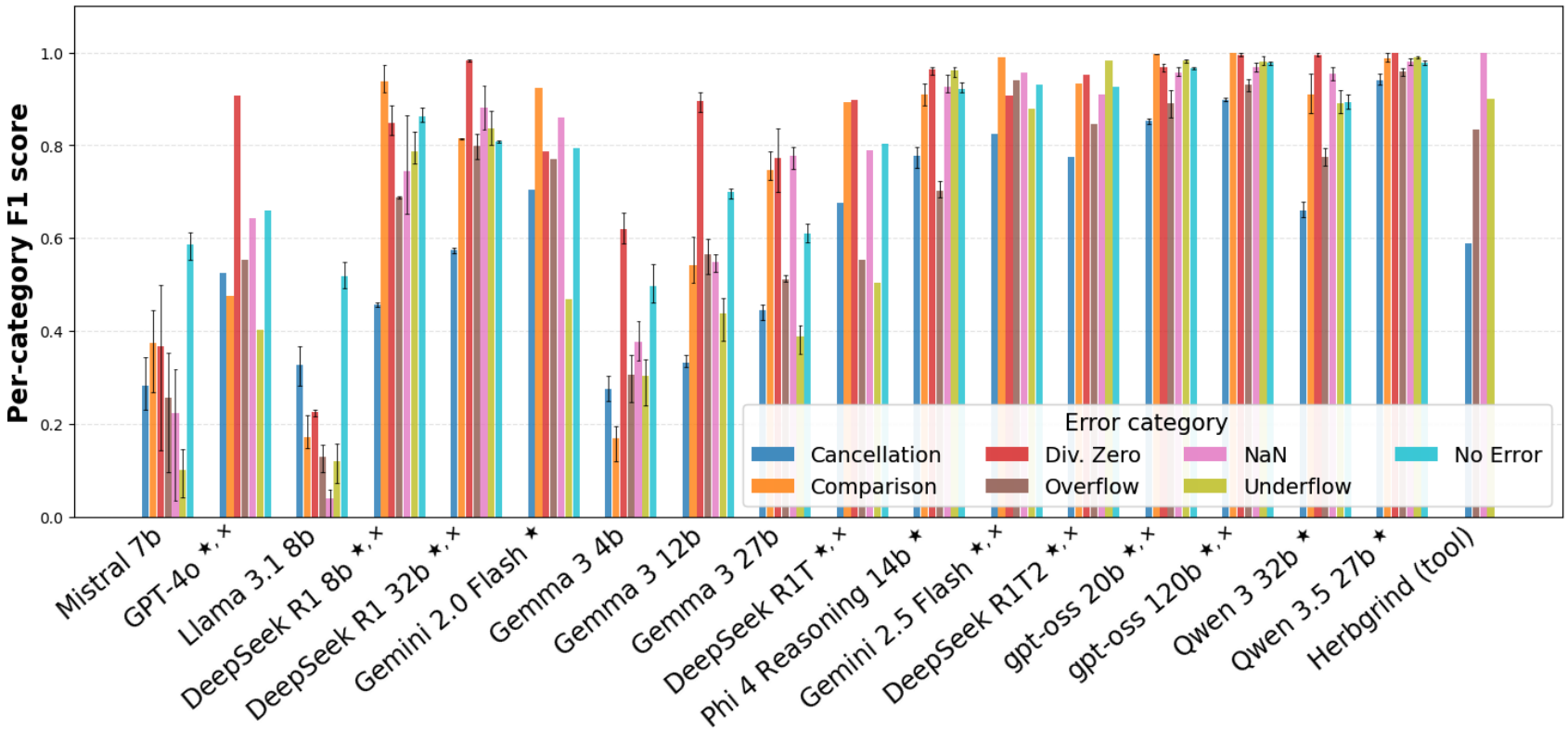


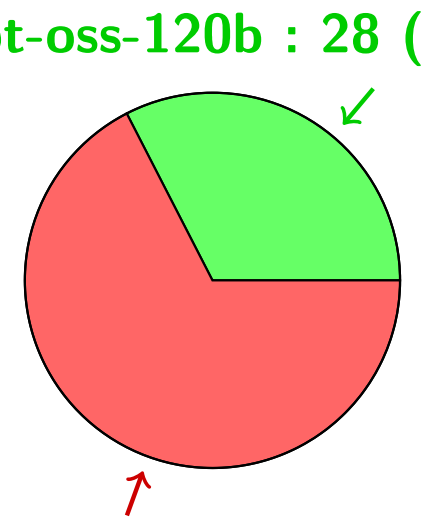
Figure – Per-category F1-score across 17 LLMs sorted by release date from the oldest (left), to the latest (right). ⋆ Chain-of-Thought (CoT) reasoning ; × Mixture-of-Experts (MoE) architecture.

Université Perpignan Via Domitia

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.



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

Conclusion : LLM already have some basic numerical knowledge

Limitations :

- No formal correctness proof
- Absence of guarantees

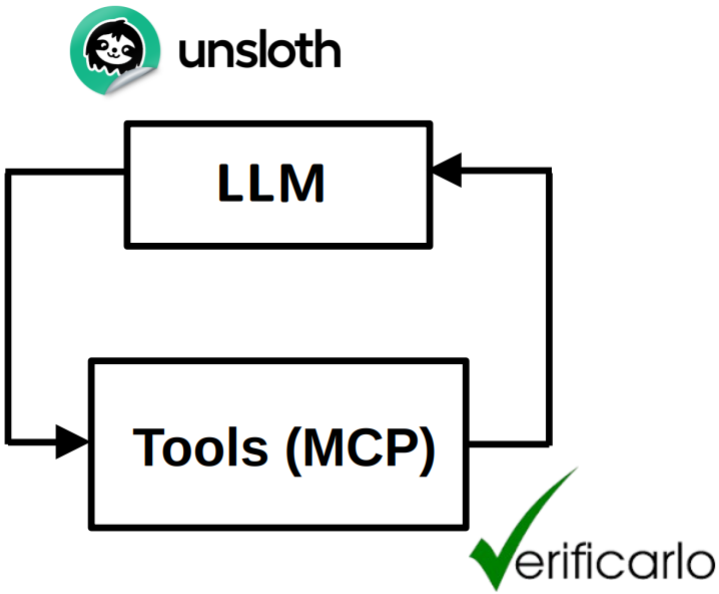
Pratical applications :

- Identifying likely error patterns
- Prioritizing suspicious code regions for review

Section 3

How to combine LLMs with existing tools into a local agentic system for HPC codebases ?

MCP Servers with Local LLMs using Unsloth



MCP Verificarlo : compile, instrument, and analyze numerical code

- Browse project files, read source code, and detect Makefiles
- Compile C/C++/Fortran code through Verificarlo compiler
- Run binaries under different floating-point backends
- Compare numerical stability across optimization levels or algorithm variants
- Explore reduced-precision effects
- Detect catastrophic cancellation at runtime
- Delta-debug to detect the exact source lines causing numerical instability

Use Case : Kahan Summation

```
1  REAL sum = f[0];
2  REAL c = 0.0, y, t;
3  for (i = 1; i < N; i++) {
4      y = f[i] - c;
5      t = sum + y;
6      c = (t - sum) - y;
7      sum = t;
8  }
9  return sum;
```

Kahan Use Case : Two Reports

Claude Code (VS Code) vs Local model (Qwen 3.5 9b)

	Validation	**Local system**
IEEE baseline	5.08125470021-33238e+02	5.08125470021-33238e+02
MCA samples	30 runs ; mean 508.1254700213325	30 runs ; "very close to baseline"
Std. dev.	5.55×10^{-13}	*not reported*
Significant digits	CNH : 14.54 General : 14.45	"typically 15–16" *(no method named/ exact value)*
Verdict	Stable, no further debugging needed	Stable, no issues detected

Execution Time

Metric	Time	Speed
Prompt eval (s)	130.35	35 tok/s
Generation (s)	401.23	5.4 tok/s
Tokens		2,173
Total wall time		694.91 s (11.6 min)

Use Case : Riemann–Siegel

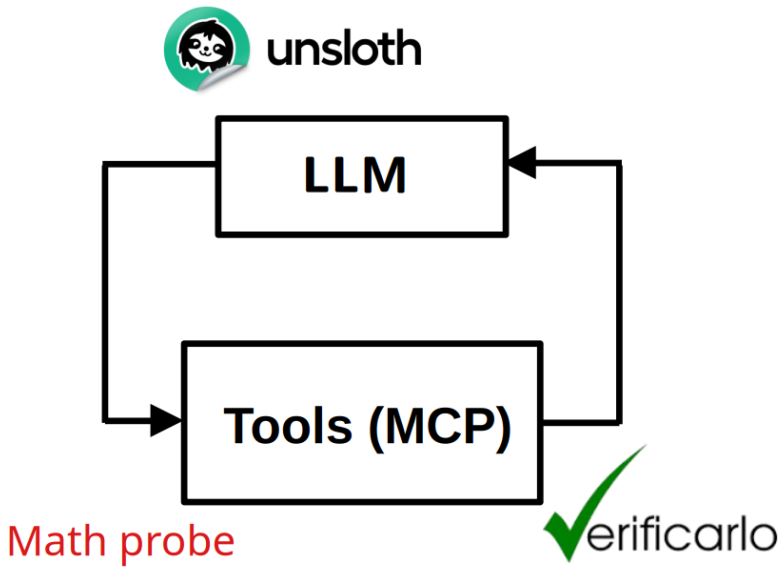
```
$ ./RiemannSiegel 10 100000 10
I found 137931 Zeros in 6.934 seconds      # INCORRECT

$ ./RiemannSiegel 10 100000 100
I found 138069 Zeros in 56.035 seconds     # OK
```

Expected answer

Counting sign changes of $Z(t)$ on a uniform grid is only correct if the grid is fine enough to resolve *every individual* zero.

MCP Math Probe



- Will this expression overflow the target dtype ?
- Is this subtraction losing decimal digits ?
- Is this sampling step too coarse for the underlying oscillation ?

Qwen 3.5's Conclusion : A Mixed Signal

The final answer contains **two different mechanisms** :

The correct sentence

"The coarser sampling either misses the exact crossing point due to step size = 0.1, ***evaluates both points on the same side of zero due to undersampling****...the finer sampling catches the intermediate sign change."*

The incorrect sentence, right next to it

"Numerical noise in $Z(t)$ *causes sign flips at slightly different t-values...within numerical noise (*$\sim 10^{-12}$*)."*

Execution Time

Phase	Time	Speed
Prompt eval (time to first token)	600.28 s (10.0 min)	36.7 tok/s
Generation	2185.51 s (36.4 min)	4.4 tok/s
Tokens generated		9,544
Total wall time	**4127.14 s (68.8 min)**	

Section 4

Conclusion & Future Directions

Conclusion & Future Directions

- The fix that matters most for efficient computation : localize before instrumenting
- For scalability, a major open question is how an LLM reads and navigates large source files
- Tools that can localize floating-point-relevant computation in code before deep instrumentation is applied

Thank you for your attention