Benchmarking GenAI for Software Engineering: Challenges and Insights

Marco Vieira

marco.vieira@charlotte.edu

UNIVERSITY OF NORTH CAROLINA CHARLOTTE

Based on what you know about me…

... draw a picture of what you think my current life looks like

What about my social life?

Based on what I know, your professional commitments—including teaching, research, conference organizing, and writing papers—seem to dominate your schedule. While this indicates a highly productive and intellectually stimulating life, it might leave limited time for social activities.

Why this talk?

LLMs are transforming software engineering

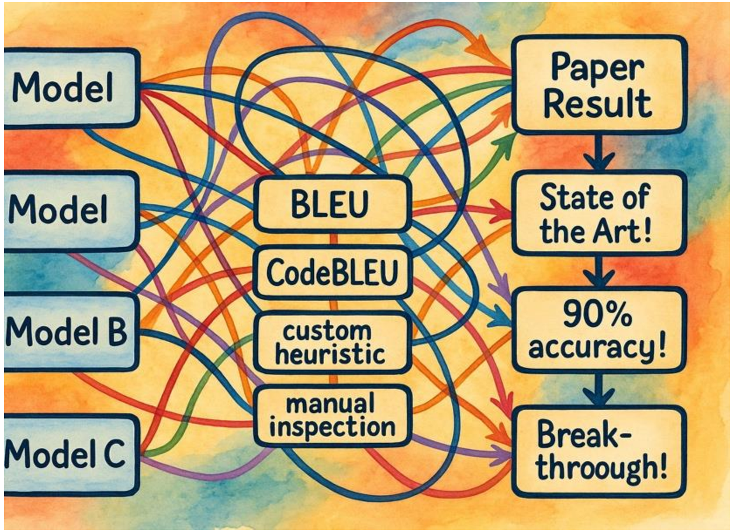
– Generate code, translate between languages, write tests, detect vulnerabilities, ….

The scientific foundation for evaluation is immature

No rigorous benchmarking

– Wrong conclusions, unsafe adoption, misleading comparisons, …

The benchmarking crisis…



Metrics vary wildly

Different teams / different prompts

Unreported inference parameters

Not run in realistic contexts

Datasets lack coverage or realism

Incomparable results!

Consequences of poor evaluation

BLEU “high score”, but the code fails to compile

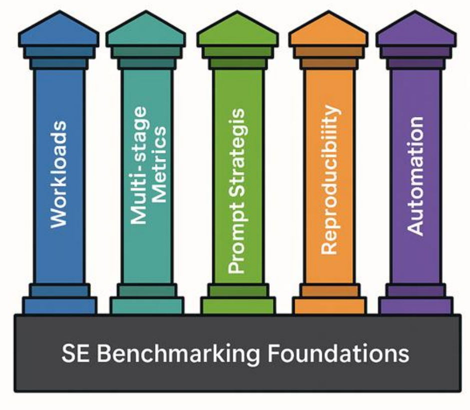
High pass@1 but ignores runtime errors

Generated tests that run but do not exercise code

Semantic drift undetected by lexical metrics

“LLM produces perfect-looking code but fails on edge cases”

What do we actually need?



Realistic, diverse workloads

Multi-stage metrics

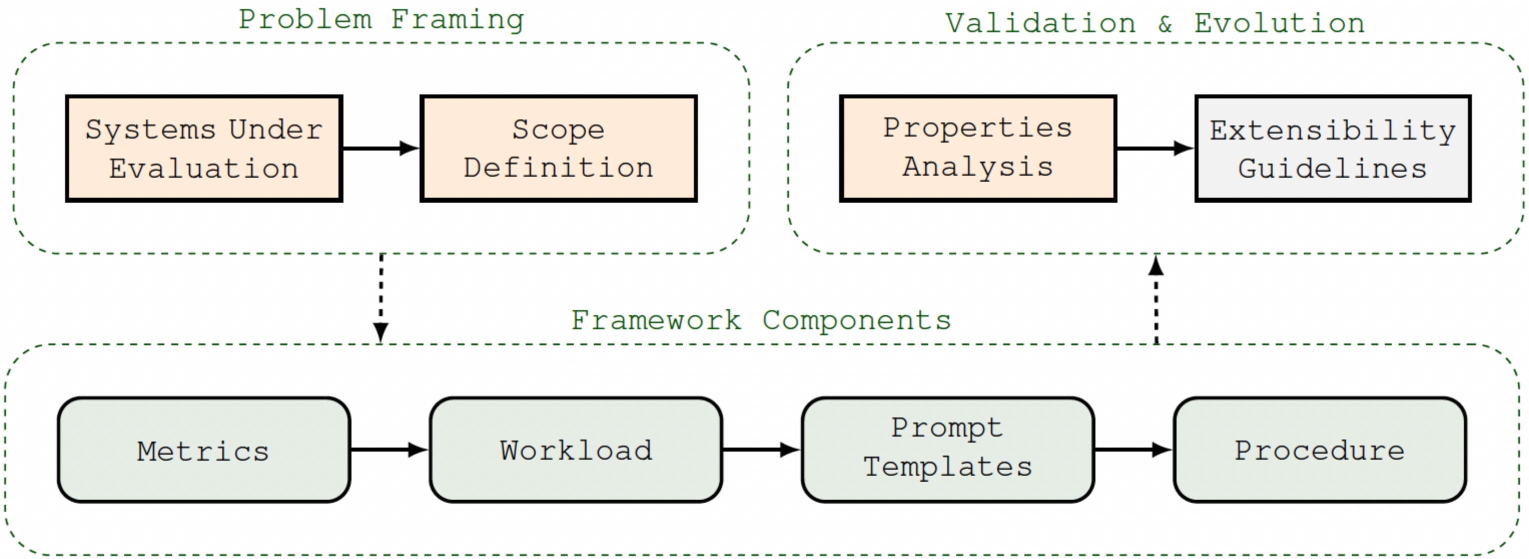
- Syntax → runtime → semantics → structure

Standardized, documented prompt strategies

Automation to avoid human bias

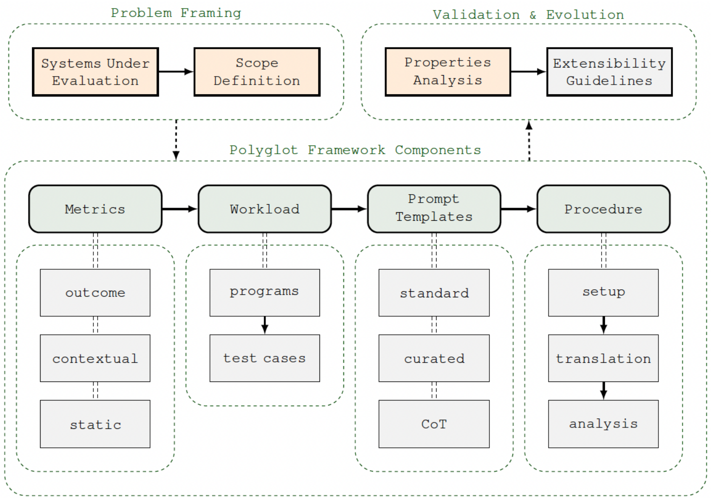
Fully reproducible experimental pipeline

Benchmarking Framework



Focus on the properties!

- Representativeness
- Repeatability / Reproducibility
- Non-Intrusiveness
- Scalability
- Portability
- …



POLYGLOT: BENCHMARKING CODE TRANSLATION WITH LLMs

Why code translation?

Legacy systems need modernization

– To improve security and maintainability

Manual translation

– Time-consuming, error-prone

Why is code translation hard?

– Paradigm shifts: memory model, I/O, typing, …

– Recursion, loops, error handling, etc.

– Need to preserve functional behavior, not just syntax

Research gap

Lack of standardized procedure and metrics

– Makes results hard to compare

Limited language coverage

– Most work focuses on 1-to-1 pairs like C ↔ Python

No extensible evaluation pipeline

– Prevents adding new languages or models easily

Polyglot

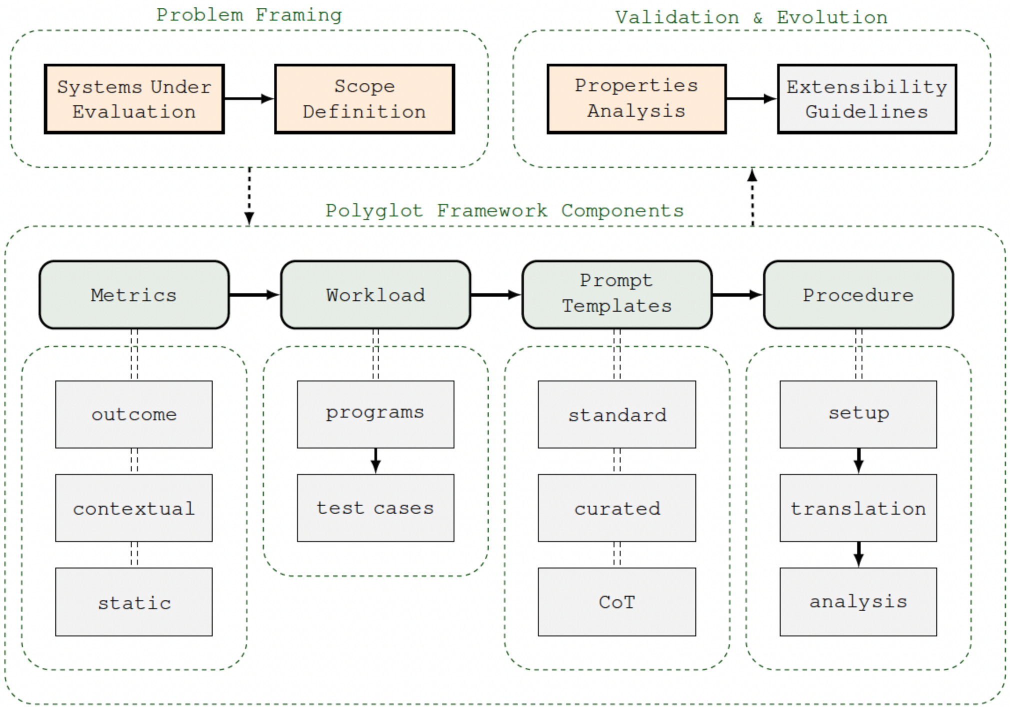
Fully automated, reproducible evaluation framework

Measures translation across four axes:

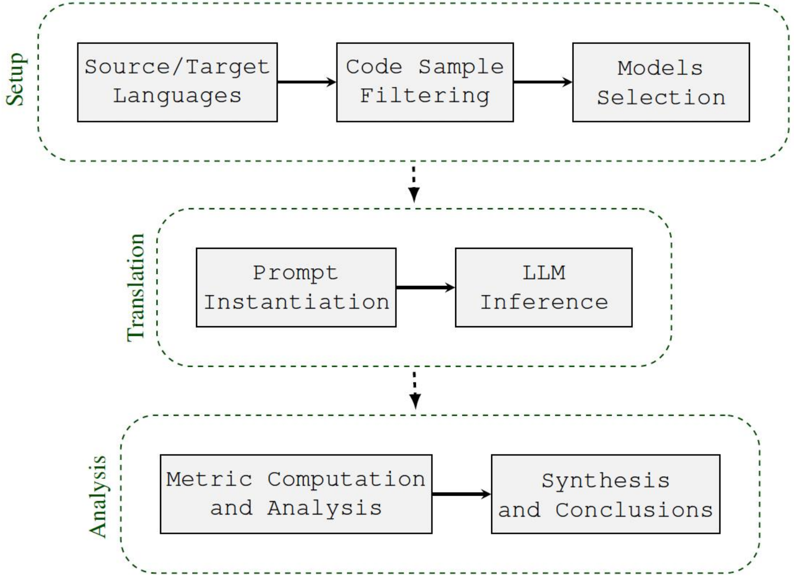
- Syntactic correctness
- Execution reliability
- Semantic preservation
- Static code metrics

polyglotweb.site

Architecture of Polyglot



Well defined evaluation pipeline



Experimental setup

Source: C (from IBM CodeNet)

Targets: Python, Java, Rust

Seven LLMs

Prompts: S-ZS, C-ZS, CoT

Selection		**Value**		
Problems (per complexity / total):				43 / 129
Solutions (per problem / per complexity / total):				6 / 300 / 900
		Avg	**Min**	**Max**
Simple	SLoC	18.1	3	264
	CC	5.7	1	86
Moderate	SLoC	46.0	7	400
	CC	13.7	2	191
Complex	SLoC	75.6	14	618
	CC	25.9	4	268

LLMs vs. Target Languages

Lang.	LLM	Fail Comp.	Fail Run.	Fail Tests	Pass Tests
Java	llama3.1_8b	36.78%	14.31%	26.99%	21.91%
	llama3.1_70b	17.02%	11.72%	21.02%	50.24%
	qwen2.5_32b	19.50%	11.12%	16.02%	53.36%
	qwen2.5-coder_7b	27.18%	12.27%	16.43%	44.12%
	qwen2.5-coder_32b	11.75%	10.90%	15.57%	61.77%
	deepseek-coder_33b	15.39%	13.83%	19.58%	51.21%
	deepseek-coder-v2_16b	21.13%	11.23%	16.17%	51.46%
Python	llama3.1_8b	5.78%	56.06%	17.98%	20.17%
	llama3.1_70b	2.82%	41.19%	17.54%	38.45%
	qwen2.5_32b	1.11%	39.75%	12.01%	47.13%
	qwen2.5-coder_7b	3.00%	54.76%	13.13%	29.11%
	qwen2.5-coder_32b	1.08%	35.93%	13.57%	49.43%
	deepseek-coder_33b	3.89%	43.79%	16.54%	35.78%
	deepseek-coder-v2_16b	4.63%	42.86%	14.28%	38.23%
Rust	llama3.1_8b	80.13%	9.75%	4.23%	5.90%
	llama3.1_70b	59.07%	16.57%	8.68%	15.68%
	qwen2.5_32b	51.35%	18.87%	10.72%	19.06%
	qwen2.5-coder_7b	67.33%	12.53%	8.97%	11.16%
	qwen2.5-coder_32b	35.22%	17.09%	15.39%	32.30%
	deepseek-coder_33b	74.45%	11.42%	5.04%	9.08%
	deepseek-coder-v2_16b	58.29%	15.31%	8.82%	17.58%

Java → up to 62% pass rate

Python → up to 49%

Rust → only 32%

Code-specialized > general models

Bigger ≠ Better

- Smaller, trained models beat LLaMA 70B

Prompting matters?

Lang.	LLM	Prompt	Fail Comp.	Fail Run.	Fail Tests	Pass Tests
Java	llama3.1_70b	S-ZS	14.91%	11.35%	16.35%	57.40%
		C-ZS	14.79%	11.57%	17.35%	56.28%
		CoT	21.36%	12.24%	29.37%	37.04%
	qwen2.5-coder_32b	S-ZS	10.34%	11.01%	14.68%	63.96%
		C-ZS	13.46%	10.57%	13.24%	62.74%
		CoT	11.46%	11.12%	18.80%	58.62%
	deepseek-coder-v2_16b	S-ZS	21.13%	10.79%	15.02%	53.06%
		C-ZS	22.58%	10.23%	15.80%	51.39%
		CoT	19.69%	12.68%	17.69%	49.94%
Python	llama3.1_70b	S-ZS	1.22%	41.05%	15.35%	42.38%
		C-ZS	2.45%	36.82%	13.35%	47.39%
		CoT	4.78%	45.72%	23.92%	25.58%
	qwen2.5-coder_32b	S-ZS	0.78%	36.04%	12.90%	50.28%
		C-ZS	1.78%	34.71%	12.35%	51.17%
		CoT	0.67%	37.04%	15.46%	46.83%
	deepseek-coder-v2_16b	S-ZS	4.67%	43.83%	13.68%	37.82%
		C-ZS	5.23%	36.93%	16.35%	41.49%
		CoT	4.00%	47.83%	12.79%	35.37%

S-ZS ≈ C-ZS → stable

CoT → more reasoning but lower reliability

What about the impact of complexity?

Lang.	LLM	Complex.	Fail Comp.	Fail Run.	Fail Test	Pass Test
Java	llama3.1_70b	Simple	4.01%	7.69%	7.69%	80.60%
		Moderate	12.24%	9.86%	22.11%	55.78%
		Complex	28.10%	16.34%	19.28%	36.27%
	qwen2.5-coder_32b	Simple	2.34%	6.35%	10.03%	81.27%
		Moderate	10.54%	9.18%	14.29%	65.99%
		Complex	17.97%	17.32%	19.61%	45.10%
	deepseek-coder-v2_16b	Simple	6.02%	8.03%	9.70%	76.25%
		Moderate	19.05%	8.84%	18.71%	53.40%
		Complex	37.91%	15.36%	16.67%	30.07%
Python	llama3.1_70b	Simple	0.33%	23.08%	10.03%	66.56%
		Moderate	0.68%	44.22%	18.71%	36.39%
		Complex	2.61%	55.56%	17.32%	24.51%
	qwen2.5-coder_32b	Simple	0.00%	25.42%	4.35%	70.23%
		Moderate	1.02%	33.33%	14.97%	50.68%
		Complex	1.31%	49.02%	19.28%	30.39%
	deepseek-coder-v2_16b	Simple	1.34%	31.44%	5.69%	61.54%
		Moderate	3.40%	43.54%	19.73%	33.33%
		Complex	9.15%	56.21%	15.69%	18.95%

Reliability ↓ as complexity ↑

Easy tasks → mostly correct

Complex logic → runtime & semantic failures

All models struggle, regardless of size or specialization

Translation failures

Syntax errors: missing braces, types, …

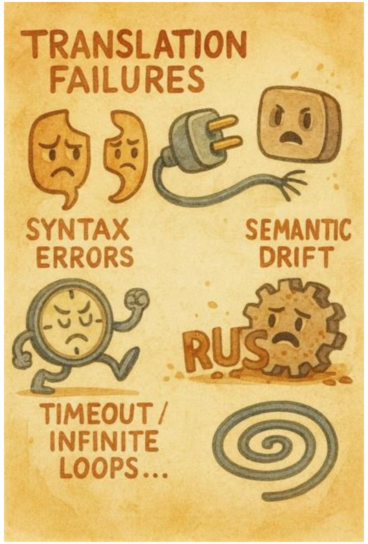
I/O mismatch

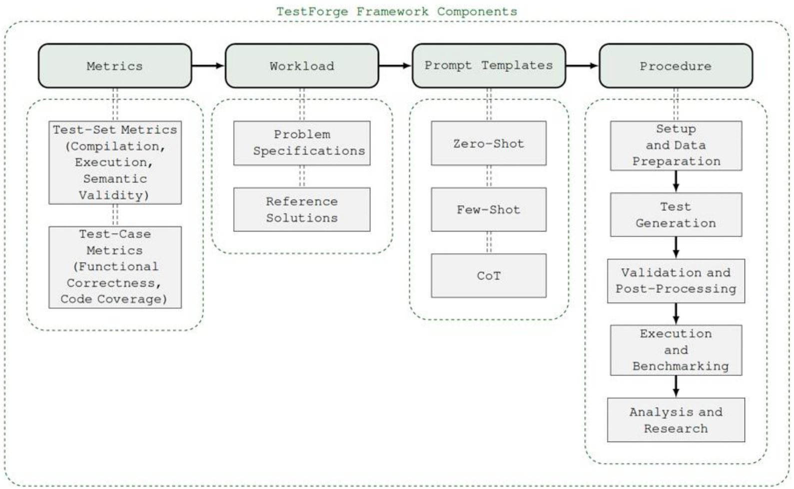
Semantic drift / incorrect logic changes

Memory/ownership errors in Rust

Timeout / infinite loops

…





TESTFORGE: BENCHMARKING LLM-BASED TEST CASE GENERATION

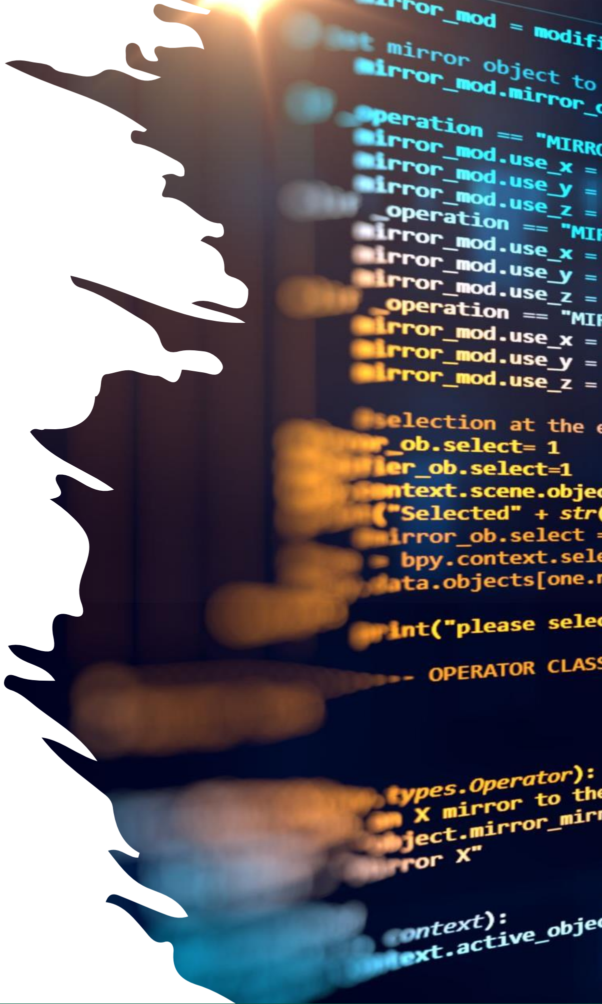
Test case generation

Ensures functional correctness of software

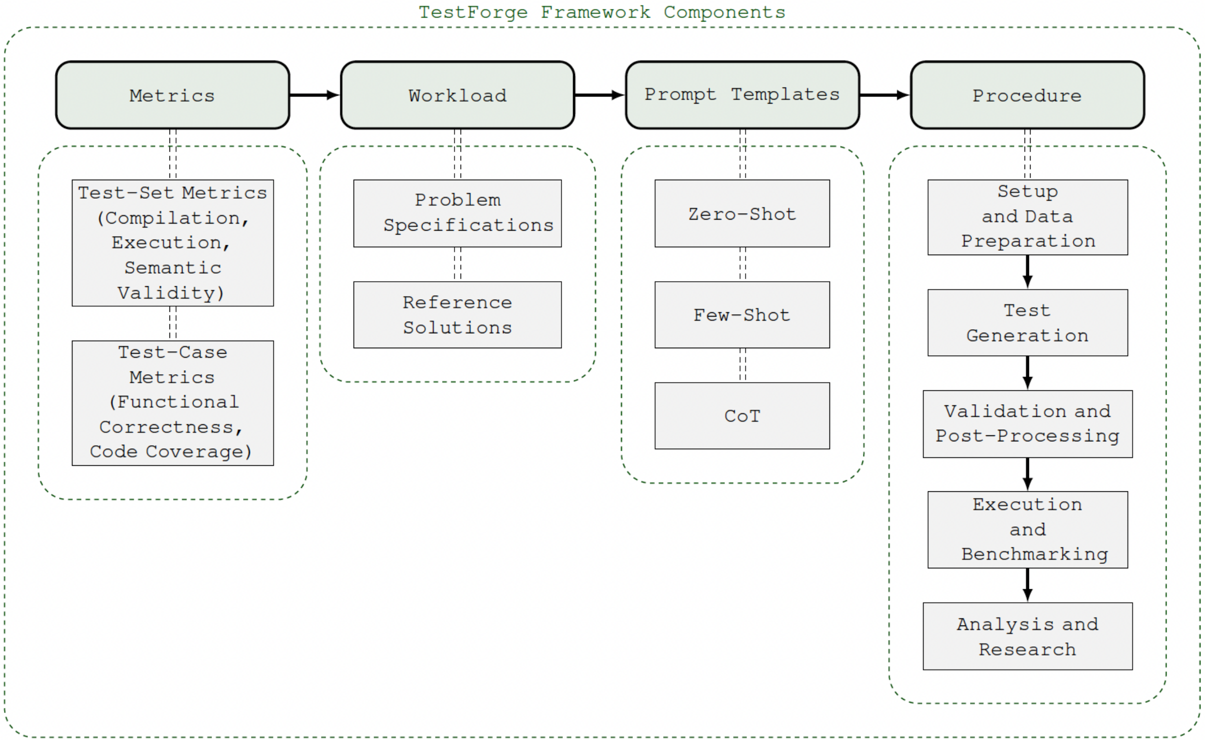
– Reveals hidden bugs and edge cases

Strengthens confidence in refactoring and modernization

Automates one of the most time-consuming tasks in software development



Architecture of TestForge



Experimental setup

NL descriptions + Java implementations (IBM CodeNet)

Target: Java 11 + JUnit 5

43 problems (simple/moderate/complex)

20 tests/problem (standard, boundary, mixed)

4 LLMs

Prompts: ZS, FS, CoT

Overall results…

Specialized coder models outperform larger general LLMs

Model size is not the dominant factor

Semantic correctness (O_4) varies widely across models

LLM	O_1 (%)	O_2 (%)	O_3 (%)	O_4 (%)
`Llama3.3:70b`	3.79	6.34	0.00	**89.87**
`Qwen2.5-coder:14b`	4.93	2.34	0.00	**92.73**
`Qwen3:32b`	22.95	1.73	1.52	**73.80**
`Qwen3:4b`	47.74	2.73	5.59	**43.94**

Prompting also matters here!

CoT often yields the best final correctness (O_4)

Few-shot is not reliably better than zero-shot

Prompting effects vary by model: no universal winner

LLM	Prompt Type	O_1 (%)	O_2 (%)	O_3 (%)	O_4 (%)
`Llama3.3:70b`	Zero-Shot	2.04	8.79	0.00	89.16
	Few-Shot	8.11	2.93	0.00	88.96
	Chain-of-Thought	1.23	7.29	0.00	**91.48**
`Qwen2.5-coder:14b`	Zero-Shot	4.98	2.32	0.00	92.71
	Few-Shot	5.11	2.59	0.00	92.30
	Chain-of-Thought	4.70	2.11	0.00	**93.18**
`Qwen3:32b`	Zero-Shot	28.49	2.32	2.86	66.33
	Few-Shot	20.31	1.77	0.55	77.37
	Chain-of-Thought	20.04	1.09	1.16	**77.71**
`Qwen3:4b`	Zero-Shot	47.31	1.30	6.75	44.65
	Few-Shot	50.92	3.54	5.39	40.15
	Chain-of-Thought	44.99	3.34	4.64	**47.03**

LLM vs. Test Type vs. Complexity

LLM	Test Type	Easy				Moderate				Hard			
		O_1 (%)	O_2 (%)	O_3 (%)	O_4 (%)	O_1 (%)	O_2 (%)	O_3 (%)	O_4 (%)	O_1 (%)	O_2 (%)	O_3 (%)	O_4 (%)
`Llama3.3:70b`	Standard	0.00	0.91	0.00	**99.09**	0.40	2.02	0.00	**97.58**	0.00	1.53	0.00	**98.47**
	Boundary	4.55	12.73	0.00	82.73	2.42	10.89	0.00	86.69	2.29	9.16	0.00	88.55
	Mixed	0.00	6.36	0.00	93.64	0.81	10.08	0.00	89.11	0.76	10.69	0.00	88.55
`Qwen2.5-coder:14b`	Standard	4.55	0.91	0.00	94.55	4.44	1.61	0.00	93.95	6.11	2.29	0.00	**91.60**
	Boundary	1.82	0.00	0.00	**98.18**	3.63	1.61	0.00	**94.76**	7.63	1.53	0.00	90.84
	Mixed	4.55	0.91	0.00	94.55	4.84	4.84	0.00	90.32	5.34	3.05	0.00	**91.60**
`Qwen3:32b`	Standard	15.45	0.00	0.91	**83.64**	22.98	0.40	0.40	76.21	22.90	0.00	3.82	73.28
	Boundary	19.09	0.91	0.00	80.00	21.77	2.42	0.40	75.40	24.43	3.05	5.34	67.18
	Mixed	16.36	0.00	0.00	**83.64**	15.32	1.61	0.40	**82.66**	20.61	0.00	0.76	**78.63**
`Qwen3:4b`	Standard	26.36	0.00	0.91	**72.73**	41.53	2.82	3.23	**52.42**	56.49	5.34	9.16	**29.01**
	Boundary	38.18	0.00	3.64	58.18	43.15	2.82	2.42	51.61	51.91	9.92	10.69	27.48
	Mixed	39.09	0.00	0.00	60.91	48.39	2.02	3.23	46.37	56.49	7.63	11.45	24.43

Easy tasks mask fundamental weaknesses

Boundary and mixed tests reduce correctness for most models

Only top models remain stable across complexity levels

Common failures in test case generation

Assertions compare identical values (always passing)

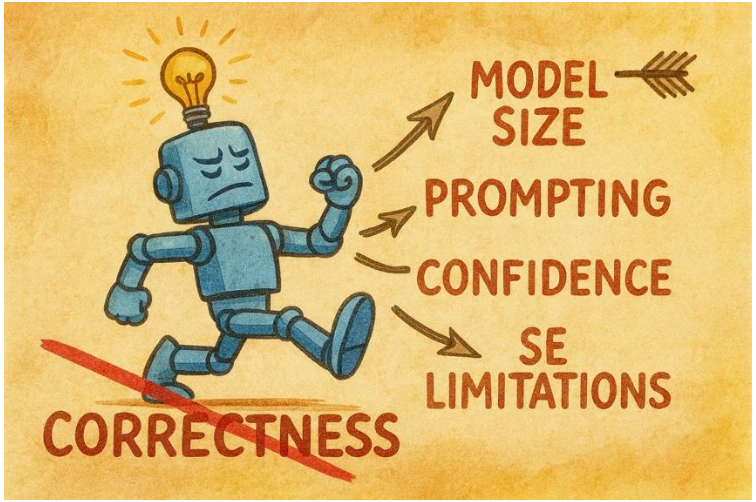
Wrong expected outputs

Tests not reaching the target method

Mis-parsed input formats

Runtime exceptions

…



INSIGHTS AND FUTURE…

Cross-cutting insights…

Reasoning ≠ correctness

Model size is not the dominant factor

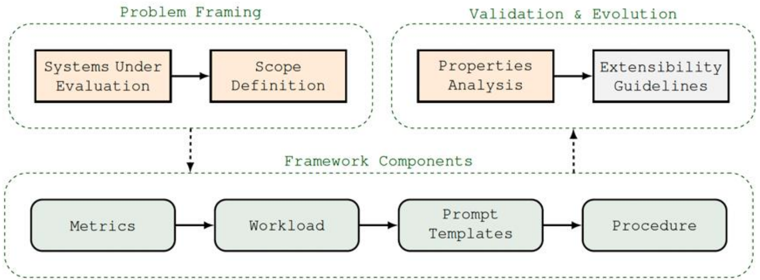
LLMs overestimate their confidence in hard tasks

Prompting interacts nonlinearly with accuracy

SE tasks show deeper AI limitations

– Specification fidelity, semantics, edge-case handling

Unified benchmarking blueprint!?



Explicit task scope

Representative workload selection

Adequate prompt design & documentation

Diverse, but comparable metrics

Controlled procedures, automation, and reproducibility

Public result repositories

Looking ahead

Combine metrics with trustworthiness attributes

– Reliability, explainability, robustness, …

How to evaluate Human/AI interaction?

Long-context (full project) reasoning as the next frontier

Specialized LLMs for SE will outperform general-purpose!?

Take-aways

GenAI is powerful but brittle

Naïve evaluation gives a false sense of progress

Rigorous, transparent benchmarking is essential

Polyglot & TestForge show how to build such benchmarks

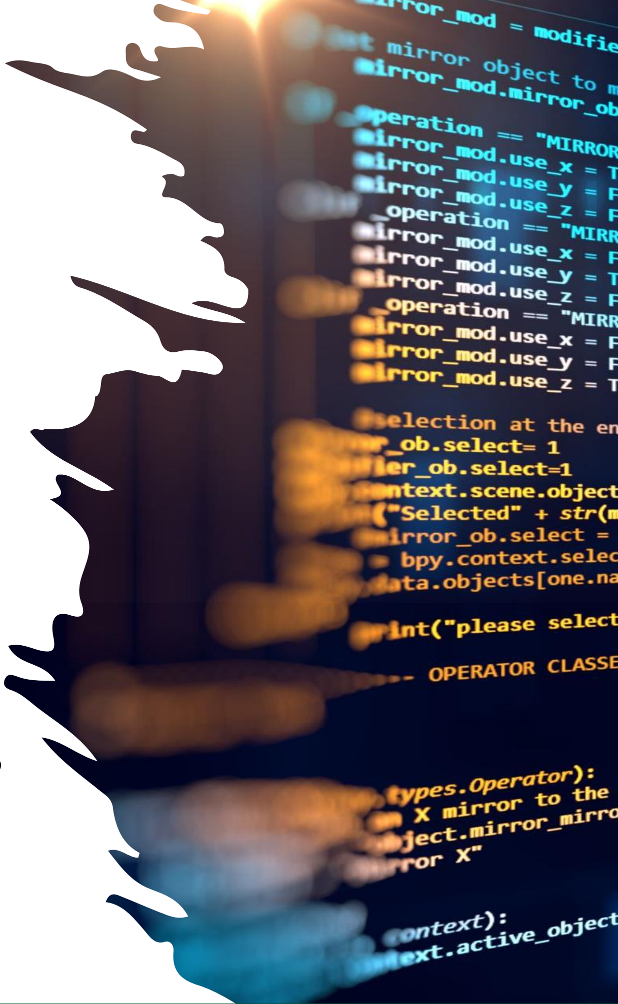
Future of trustworthy SE depends on the measurement discipline!

Join us and contribute

[PolyglotWeb.site]()
– Code translation results and datasets

[TestForgeWeb.site]()
– Test generation benchmark

Looking for contributors for new workloads, metrics, languages, …



Benchmarking GenAI for Software Engineering: Challenges and Insights

Marco Vieira
marco.vieira@charlotte.edu

UNIVERSITY OF NORTH CAROLINA CHARLOTTE