Information Parity for Code: The Scope of Transfer in Multilingual Code Models

Alexander Tsvetkov [1] Alon Kipnis [1]

[1]School of Computer Science, Reichman University

Paper, code & data
github.com/tsalex1992/information-parity-for-code

Reichman University

Problem

Status quo
Multilingual code benchmarks show uneven performance across languages.

Question
Do benchmark gaps reflect language-level difficulty or task structure?

Goal
Map when Code IP tracks multilingual performance, and when task structure takes over.

Key Contributions

- We adapt **Information Parity (IP)** to code, defining Code Information Parity (Code IP) on an audited corpus of 440 semantically aligned implementations across 20 languages.
- We show that Code IP is **regime-dependent**: it tracks short-horizon generation, weakens on execution prediction, and does not transfer to reverse reasoning or whole-program synthesis.
- We clarify what Code IP captures by separating per-token cost, token count, and public code-volume effects.

Method

Audited parallel code → Negative log-likelihood → Code Information Parity → Task-regime comparison

$$\text{Code IP}(\ell) = \frac{\text{NLL}(\ell)}{\text{NLL}(\text{Python})}$$

Code IP asks how many bits a model assigns to the implementation in language ℓ, relative to the aligned Python implementation. Code IP > 1 means higher surprisal than Python; lower Code IP means greater model-relative efficiency.

Controlled Corpus

22	20	440	Passed
classic algorithms	languages	implementations	execution checks

Each implementation preserves the same task, algorithmic family, intermediate work, and output semantics as the Python reference. We remove benchmark harnesses, examples, unnecessary imports, boilerplate, and standard-library shortcuts.

Evaluation Setup

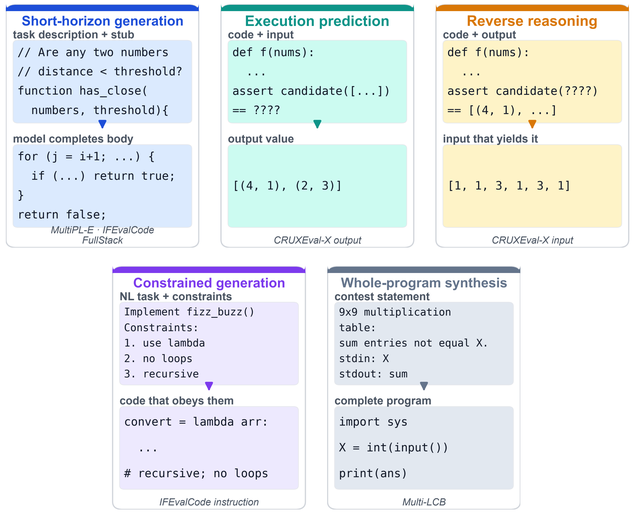
- **Scoring:** raw code only, native tokenizer, total next-token surprisal in bits.
- **Benchmarks:** MultiPL-E, CRUXEval-X, IFEvalCode, FullStack Bench, and Multi-LCB.
- **Statistic:** within-model Spearman correlation across non-Python languages, pooled with Fisher-z intervals.
- **Coverage:** Code IP is scored on 80 models; benchmark correlations use slice-specific overlap pools of 15–28 models and 5–18 non-Python languages.

What Code IP Captures

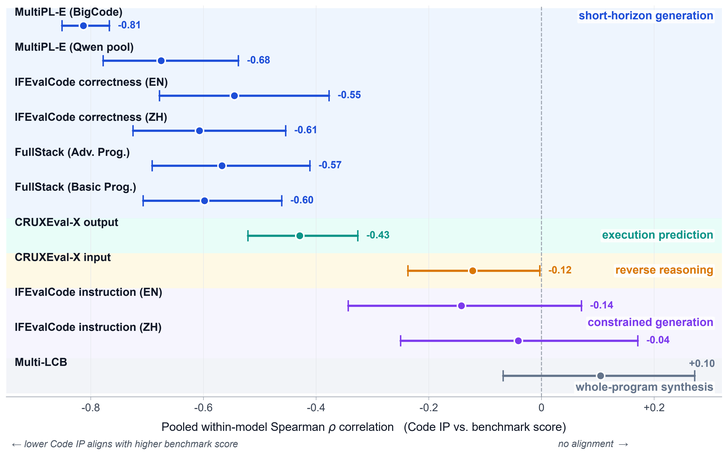
$$\text{Code IP} = \underbrace{\frac{\text{bits/token}(\ell)}{\text{bits/token}(\text{Python})}}_{\text{relative per-token cost}} \times \underbrace{\frac{\#\text{tokens}(\ell)}{\#\text{tokens}(\text{Python})}}_{\text{relative token count}}$$

- **Combines:** tokenization and verbosity with model-dependent representation cost.
- **Aligns with:** tasks where success depends on continuing or completing code-like context.
- **Weakens when:** tasks move beyond bounded forward completion into broader reasoning, planning, or synthesis.

Task Regimes



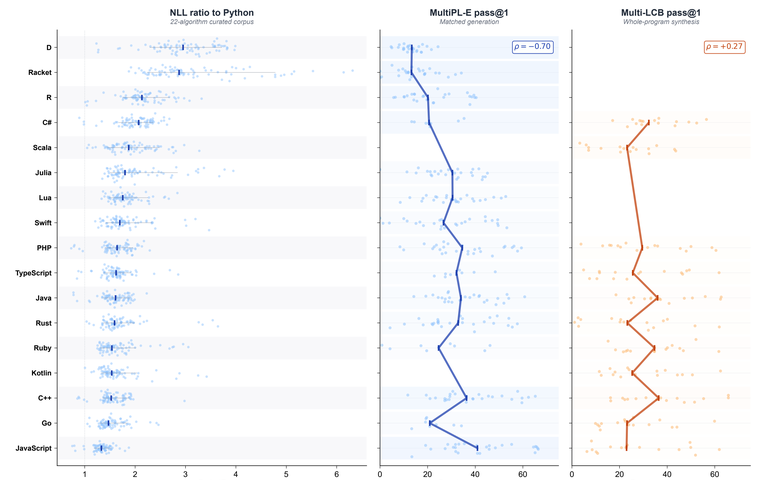
Main Result



Pooled within-model Spearman correlations; bars show descriptive 95% Fisher-z intervals. Slice overlaps span 15–28 models and 5–18 non-Python languages.

- **Strong** on short-horizon generation (ρ from -0.55 to -0.81).
- **Remains negative though moderated** on execution prediction ($\rho = -0.429$).
- **No clear alignment** for reverse reasoning, constrained generation, or whole-program synthesis; `Multi-LCB` disappears at $\rho = +0.105$.

Language Ordering Evidence



Languages are ordered by median Code IP. MultiPL-E follows that ordering; Multi-LCB does not.

- The same language ordering is informative for `MultiPL-E` but not `Multi-LCB`.

Robustness: Is It Just Training-Data Volume?

Code IP still predicts short-horizon benchmark performance after controlling for public code volume.

Slice	Code IP ρ after volume control
MultiPL-E BigCode	-0.550
MultiPL-E Qwen	-0.550
CRUXEval-X output	-0.262
Multi-LCB	$+0.214$

Partial Spearman correlations after controlling for $\log_{10}$ language volume in The Stack v2 SWH-full; Stack v2 covers 16/19 non-Python corpus languages.

Takeaway

Code IP is a regime diagnostic, not a global language-ability score: it predicts language gaps for bounded forward code generation, but not when tasks require reverse reasoning, instruction following, or whole-program synthesis.