Revisiting Software Diversity with LLMs

90th Meeting of the IFIP Working Group 10.4
June 27, 2026

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 (SSE)
- PhD in Informatics Engineering, ~10 years in industry before

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
- Currently focusing on leveraging genAI for (dependable and secure) software generation
- (but I also work with AI in health, biology, and space domains)

Software diversity in the LLM era

- Software diversity has been used to reduce common-mode failures
- N-version ideas rely on multiple independent implementations
- Traditional diversity is expensive: multiple teams, implementations, validation, …
- Classic work showed that independent development does not guarantee independent failures
- LLMs change the cost side: they can generate many implementations quickly, across models, temperatures, and languages

- **Can LLM-generated diversity be leveraged for reliability improvement?**

Two workstreams on LLM-generated diversity

1: Can LLMs provide software diversity that improves reliability?

- Extends classical empirical software-diversity studies to LLM-generated code
- 1-out-of-2 reliability improvement
- Compares:
  - human-written diversity (reprod.)
  - LLM-generated diversity
  - language-forced diversity
  - human–LLM heterogeneous diversity

2: Do LLM-generated implementations fail independently enough for redundancy to work in NVP?

- Methodology for evaluating failure independence in LLM-generated code
- Measures whether diversity translates into reliability under NVP
- Analyzes:
  - structural diversity
  - behavioral failure overlap
  - reliability gains under voting of homogeneous and heterogeneous
  - shared fault/root-cause patterns

1: Reliability improvement via LLMs

- **Can LLMs provide useful software diversity that improves reliability?**
- Classical EL / LM reliability-analysis framework to large populations of LLM-generated code
- Specifications: 3n+1, Factors and Factorials, Factovisors.
- Human baseline: historical UVa Online Judge programs (19,070 valid programs after filtering)
- LLM programs: generated across 14 models, temperatures*, and languages (100 p/ configuration, 49,500 overall after filtering)
- Languages: C, C++, Java, Pascal.
- Homogenous and heterogenous (forced diversity):
  - human–human (reproduction of existing studies to establish a methodological baseline)
  - LLM–LLM
  - human–LLM

- **Collaboration with City University, London, UK (Ilir Gashi and Vladimir Stankovic)**

1: Reliability improvement via LLMs

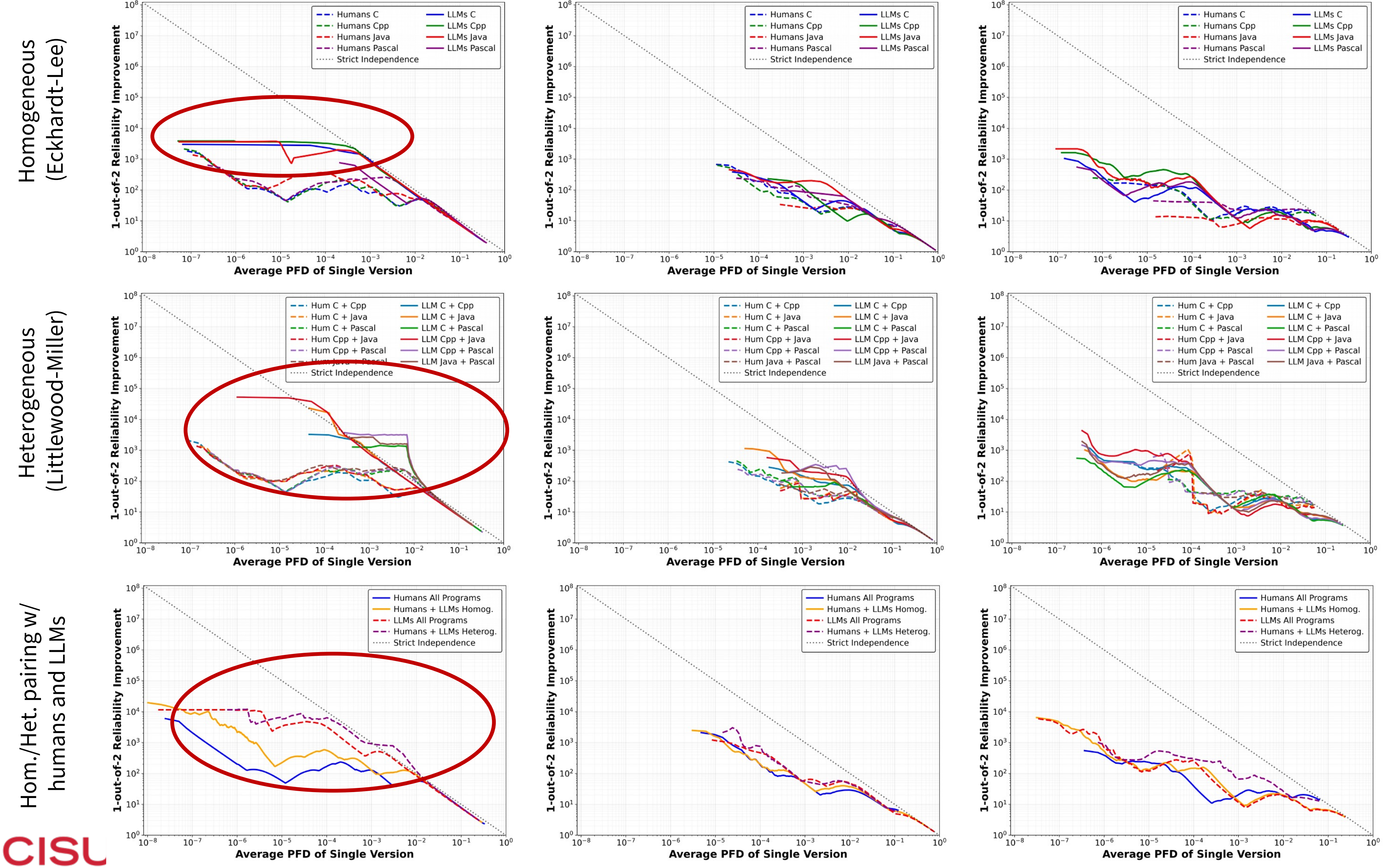
- **LLM-generated diversity appears to able to improve reliability, especially when forced**

- **LLM-generated programs can outperform human-written diversity**
  - For 3n+1, the LLM pool achieves higher reliability improvement in the highly reliable region

- **Gains are specification-dependent**
  - For Factors and Factorials, the human pool is stronger in the extreme reliability tail, while LLMs do better in some mid-range regions

- **Language forcing helps**
  - Pairing LLM programs written in different languages can yield reliability gains

- **Human–LLM pairing had interesting results (how much “human solutions” do we still write?)**
  - Enforced heterogeneous human–LLM pairs often outperform pure human, pure LLM, and mixed homogeneous pools



(a) 3n+1 (b) Factors and Factorials (c) Factovisors

CISU
SOFTWARE AND SYST
ENGINEERING

2: N-version Programming via LLMs

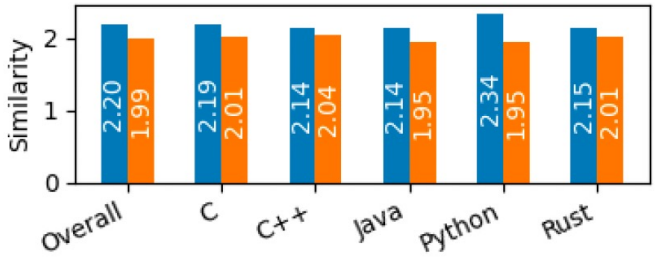
- **Do LLM-generated implementations fail independently enough for redundancy to work in NVP?**
- NVP relies on independently failing implementations; LLMs make generating versions cheap, but is there independence?
- Dataset: 224 programming problems from PROBE (originally from CodeNet)
- Generated using 12 models, 3 prompting strategies, in Python, C++, Java, C, Rust
- Diversity and reliability analyses (homogeneous vs heterogeneous):
  - structural diversity — CodeBLEU similarity
  - behavioral diversity — shared failures vs independence baseline
  - reliability gains — 3- and 5-version majority voting
  - manual fault analysis — shared root causes behind failures

- **Collaboration with UNC Charlotte, USA (Marco Vieira) and University of British Columbia, Canada (Karthik Pattabiraman)**

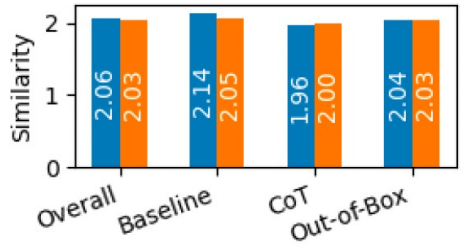
2: N-version Programming via LLMs

- **Diversity exists, but independence remains limited**
- Generated implementations show structural diversity, especially across different models
- Prompting has little effect
- Failures are still strongly correlated: implementations often fail on the same inputs
- Heterogeneous model combinations help, but do not fully solve the problem (consistent with WS1)
- NVP gains are modest: average reliability improves from about 0.88 for individual solutions to 0.90 with 3 versions and 0.91 with 5 versions
- Voting efficiency remains around 43–44% of what independence would allow
- Manual analysis shows that even different failure patterns often share root causes
  - behavioral similarity high - faults manifest as identical failures on the same inputs
  - similarity is lower - the same issues persist but appear as different failure patterns, creating the illusion of diversity

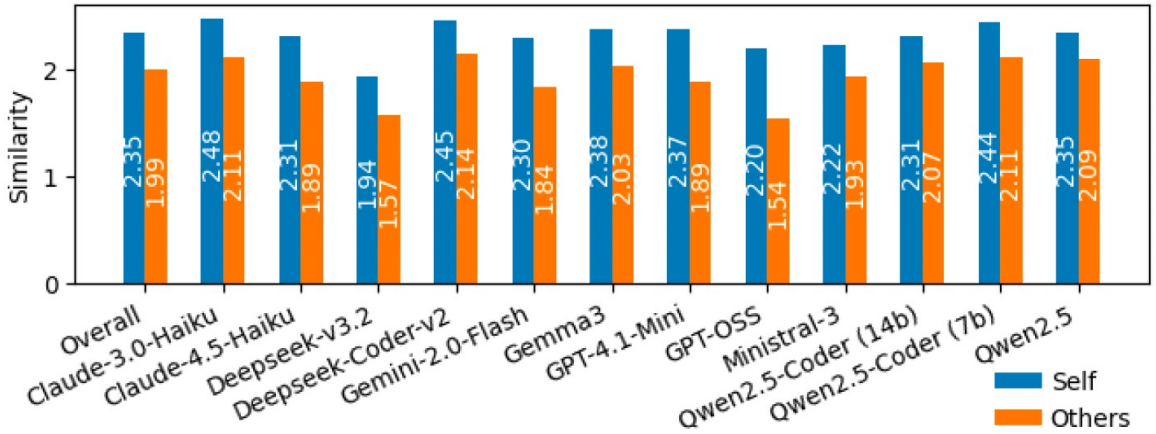
2: N-version Programming via LLMs



(a) Languages.



(b) Prompts.



(c) Models.

Fig. 5: Behavioral Sim. within the same/different entities

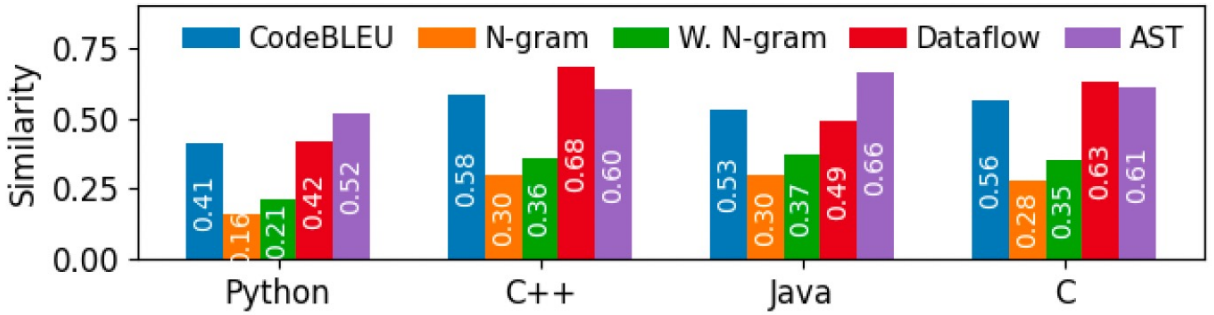


Fig. 3: Average CodeBLEU similarity across languages.

TABLE II: Overall N-version reliability / Voting efficiency

		Language		Model		Prompting	
N	Overall	Hom	Het	Hom	Het	Hom	Het
1	0.88/–	–	–	–	–	–	–
3	0.90/0.43	0.90/0.37	0.90/0.44	0.92/0.27	0.90/0.44	0.90/0.41	0.90/0.44
5	0.91/0.44	0.92/0.37	0.91/0.44	0.94/0.24	0.91/0.44	0.90/0.42	0.91/0.44

N-version reliability and gains
(compared to independence)

Related work – errors in LLM-generated code

- Large-scale study of compilation and runtime errors in LLM-generated code.
- Dataset: 79,456 erroneous code samples, 6 LLMs 4 compiled languages: C++, Java, C, Rust, 1,651 programming problems with baseline, CoT prompting, and iterative feedback
- Error patterns vary strongly across languages and models
- Even larger commercial models still make simple mistakes (missing imports/undeclared variables)
- Runtime errors reveal deeper reliability issues: out-of-bounds accesses, incorrect input processing
  - some, security-related
- Iterative feedback fixes surface-level compilation errors but not always structural/runtime errors

- **LLM-generated code still requires systematic validation and review**

(not yet) Conclusions

- LLMs make software diversity promise scalable: multiple implementations can be generated cheaply and systematically
- Reliability gains are possible, especially when forced across heterogeneous sources
- “Different-looking” implementations may still fail on the same inputs or share the same underlying causes
- Future direction: realistic problems (currently using competition-style), asymmetric fault-tolerance, failure-based selection, diversity-oriented exploration and exploitation

Revisiting Software Diversity with LLMs

90th Meeting of the IFIP Working Group 10.4
June 27, 2026

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

CISUC
SOFTWARE AND SYSTEMS ENGINEERING