Improving Quality of LLM Code Generation in Low-Resource Programming Languages via Uncertainty Estimation

Georgii Andriushchenko
Research Center of the Artificial Intelligence Institute
Innopolis University
Innopolis, Russia
georgyandryuschenko@gmail.com

***Abstract*—Large language models for source code (Code LLMs) demonstrate great performance on high-resource programming languages (HRPLs) but struggle with low-resource ones (LRPLs). Previous studies have improved LLM performance on LRPLs by continued training or tokenizer adaptation. However, they require costly data and can cause catastrophic forgetting. This paper proposes to address the poor performance of LLMs on LRPLs using uncertainty estimation (UE). UE methods have advanced LLM performance on natural language tasks, but are underexplored in source code settings. The research may provide three contributions: (1) a new code generation benchmark evaluating not only functional correctness but also readability, efficiency, and idiomatic style across Python, Java, Racket, and Elixir; (2) a new benchmark for evaluating uncertainty estimation when generating code; and (3) methods to improve LRPL code generation by leveraging UE. The methods utilizing UE include filtering synthetic training data by low uncertainty, an UE-driven curriculum learning strategy, uncertainty-aware decoding, and using uncertainty as an RL reward in alignment. The research may provide a comprehensive evaluation of uncertainty in code models, demonstrate that UE can improve LRPL generation, and open-source release of benchmarks and models as outcomes.**

***Index Terms*—large language models, code generation, low-resource programming languages, uncertainty estimation**

Research Advisor Information

Vladimir Ivanov, Head of the Lab of Natural Language Processing Methods in Software Engineering, Innopolis University. Email: v.ivanov@innopolis.ru

I. Research Problem

Large language models (LLMs) have become popular tools for programmers, powering code completion, synthesis, and refactoring. Modern LLMs for source code (Code LLMs) achieve impressive results on high-resource programming languages (HRPLs) like Python, Java, and C++ [1], [2]. However, Code LLMs struggle with the tasks in low-resource programming languages (LRPLs) [1], [3], [4]. This disparity is concerning: LRPLs risk obsolescence if AI tools fail to handle them. Indeed, recent work warns that many developers of LRPLs cannot fully leverage LLM assistance [4].

The core problem is the low performance of Code LLMs on LRPLs. General-purpose and Code LLMs are typically trained on web-mined or open-source code with heavy HRPL bias [3], [5]. Without large LRPL corpora, LLMs lack exposure to LRPL syntax and idioms. Fine-tuning on small LRPL datasets can help, but risks catastrophic forgetting of HRPL knowledge [3]. Thus, improving LRPL code generation is both an open challenge and a critical need.

The key insight introduced by the proposal is to leverage uncertainty estimation (UE) to address LRPL challenges without solely relying on large training sets. While UE has been widely applied to natural language tasks, its application to code generation is novel. There is a hypothesis that using UE can guide models to avoid generations with high uncertainty, thus improving accuracy on LRPLs that naturally have limited data.

This proposal outlines a plan to:

1) survey and synthesize (i.e., structure and analyze) related work on Code LLM adaptation and UE;
2) design new benchmarks covering non-functional code quality and UE evaluation for code;
3) develop methods that integrate UE into training and decoding to boost LRPL performance.

The proposed evaluation uses a variety of code benchmarks and metrics.

II. Background

A. Adaptation of LLMs to LRPLs

Several recent studies confirm that LLMs struggle with low-resource programming languages. Cassano et al. showed that, despite training on trillions of tokens, code models underperform on languages with sparse data [1]. For instance, StarCoder 2 [6] still lags on Racket and Elixir relative to Python [3]. McEval reports that leading models achieve high scores on popular languages (Java, C++) but much lower on niche ones [7].

The root causes include limited LRPL data and syntactic/semantic differences from HRPLs. Researchers, therefore, explore knowledge transfer techniques: they generate synthetic LRPL code via translation from Python or create tests to filter synthetic data. For example, MultiPL-T [1] generates pseudo-code in LRPLs by translating Python solutions and validating them with tests. Similarly, MultiPL-E [8] compiled multi-language benchmarks to analyze this gap.

Another approach is tokenizer adaptation: Andriushchenko and Ivanov [3] show that updating the tokenizer vocabulary improves LRPL generation quality. However, all these methods require nontrivial additional data or computation. Fine-tuning on synthetic code can lead to catastrophic forgetting of HRPL proficiency, and generating large synthetic corpora for many LRPLs is expensive.

B. Uncertainty Estimation

Uncertainty estimation quantifies how unsure a model is in its outputs. According to [9], the existing UE methods fall into several categories:

Information-based methods use internal probabilities of the model for estimation. For example, uncertainty may be estimated using maximum sequence probability by simply calculating the joint token probability of a sequence [9]. Another method is based on token entropy. Fomicheva et al. use the average or max token entropy as a confidence score [10]. More recently, attention-based measures have been proposed for building an "attention chain" to estimate token marginal probabilities [11].

Meaning-diversity methods evaluate the consistency of meaning across outputs. Semantic entropy [12] measures uncertainty over latent meaning representations. Frequency scoring [13] generates multiple answers and sees how often each appears. Infrequent answers are considered uncertain. LUQ [14] scores outputs by how many entailments they have. These methods capture conceptual uncertainty beyond raw token probabilities.

Ensemble-based methods run multiple model samples and measure disagreement. For example, Token-level Mutual Information [15] between token predictions of ensemble members can serve as uncertainty quantification.

Density-based methods use latent feature spaces. One can compute the Mahalanobis distance of a new example's hidden state from the training data's distribution [16]. In addition, there are robust density estimators to detect out-of-distribution inputs [17] or hybrid deep ensembles [18].

Reflexive or verbalized methods ask the model to assess its answer. One approach is to estimate the probability that the answer is correct [19]. Another is to have the model output a confidence token or natural-language justification of confidence [20].

Pre-trained heads train a classifier head on top of the model to predict uncertainty [21].

C. Code LLM Benchmarks

Code Generation Benchmarks. These benchmarks evaluate the functional correctness of code generated by LLMs across various programming languages. Notable examples include the BigCode Evaluation Harness [22], which supports multi-language evaluation via HumanEvalPack and MultiPL-E. Another example is McEval [7], a multilingual benchmark with 16K examples in 40 languages. Mercury [23] focuses on code efficiency, measuring runtime performance. CodeScope [24] provides a broader assessment of code quality across 43 languages, including execution-based tasks.

Repository-level Code Generation. Benchmarks like RepoBench [25] and CrossCodeEval [26] test models' ability to handle multi-file code completion and context understanding. RepoBench evaluates in-file and cross-file completion in Python and Java repositories, whereas CrossCodeEval extends this to real-world projects in Python, Java, TypeScript, and C#, emphasizing cross-file dependencies.

Agentic Benchmarks. These benchmarks simulate real-world developer tasks, such as bug fixing and issue resolution. SWE-Bench [27] and Multi-SWE-Bench [28] focus on repository-level bug fixes across multiple languages, including Java and Rust.

D. Summary

Existing literature consistently finds that Code LLMs trained on HRPLs perform poorly on LRPLs. Methods like fine-tuning on synthetic translations improve LRPL accuracy, but require additional data and can cause HRPL forgetting. Meanwhile, UE methods have not been applied to code. Regarding evaluation, most current benchmarks focus on functional correctness only: aspects like readability, efficiency, and coding style are rarely measured. These gaps motivate our new benchmarks and UE-driven approach.

III. Methodology

A. Programming Languages

The experiments may consider *Python* and *Java* as HRPLs and *Elixir* and *Racket* as LRPLs. The choice of HRPLs is motivated by the fact that these languages represent the largest portion of the training data of popular Code LLMs [6], [5]. The LRPLs are selected because they usually represent quite a small portion of the training data [6], [3].

B. Large Language Models

The proposed research involves experimentation with General-purpose and Code LLMs. General-purpose models include Llama 3 [29], DeepSeek [30], Mistral [2], and Gemma [31]. These represent the latest open-source LLMs known for strong performance and broad tasks. They are not trained specifically on code, but it may be interesting to check whether they may perform competitively on LRPL code tasks with proper uncertainty guidance. Considered Code LLMs may include StarCoder 2 [6], DeepSeek-Coder [**?**], CodeLlama [32], and Qwen2.5-Coder [33]. These are recent open models explicitly trained on code.

C. Proposed Approach

New Benchmark Proposal for Code LLMs. Existing benchmarks mostly focus on functional correctness, avoiding consideration of other aspects. A new benchmark for HRPLs and LRPLs may be proposed that considers readability, efficiency, and idiomatic style of the generated code as well as its functional correctness. Readability may be measured with code complexity (cyclomatic complexity [34], Halstead metrics [35], or line count) and documentation density (comments-to-code ratio) metrics. Measurements of code efficiency could be runtime or resource usage. To evaluate idiomatic style, language-specific style linters may be used. Functional correctness may be checked with unit tests.

Proposal of Uncertainty Estimation Benchmark. This step involves applying existing UE methods for code generation. The purpose of the benchmark is to evaluate the quality of uncertainty estimation for code. Various uncertainty estimation methods, such as information-based, meaning-diversity, ensembling, density-based, reflexive, and pre-trained heads, may be applied to code generation and evaluated. Cross-entropy between the predicted and actual uncertainties may be the metric of the benchmark. Additionally, the correlation between functional correctness and uncertainty may be analyzed.

Proposal of the Improved LLM for LRPL Code Generation Using Uncertainty Estimation. The study could try the following strategies to utilize uncertainty estimation for improved LRPL code generation:

- *Fine-tuning on synthetic data filtered by uncertainty*.
  Generate or collect synthetic LRPL code (for example, via translating HRPL code to LRPL code) and apply a model to label it. Next, compute uncertainty using various methods for each generated example. Then, filter out examples above a threshold of uncertainty. The intuition is to train only on the model's confident outputs, avoiding noisy pseudo-labels.
- *Uncertainty-driven curriculum*.
  Inspired by [36], the training examples may be sorted by uncertainty (low to high) and gradually introduced in fine-tuning. The model first sees "easy" (low-uncertainty) LRPL code, then progressively "harder" ones. This approach might stabilize training and improve convergence.
- *Uncertainty-constrained decoding*.
  This option involves modification of the generation procedure at inference. The possible techniques could be Uncertainty-Thresholded Decoding (pruning token candidates with high uncertainty), Top-k Confidence Decoding (sampling tokens with the highest negative uncertainty), Uncertainty-Aware Beam Search (using uncertainty to score the beams), and Uncertainty-weighted Sampling (sampling from the token distribution but weighting choices inversely by uncertainty).
- *LLM Alignment with Uncertainty Reward*.
  Experiments with policy optimization (Proximal Policy Optimization [37] or Reinforced Self-Training [38]), where the reward is inversely related to uncertainty, may be performed. For example, use the negative sum of token entropies (i.e., maximize confidence) plus a bonus for passing unit tests. The model is thus trained to prefer sequences of high-confidence tokens that also satisfy functionality.

IV. Expected Contributions

The following key contributions may be anticipated:

1) The new code generation and uncertainty benchmarks will bridge the existing limitations and will be introduced. Both will be open-source resources to guide future research.
2) It will be demonstrated for the first time how various existing uncertainty estimation techniques apply to Code LLMs in the LRPL setting.
3) The methods to exploit uncertainties to improve code generation performance in LRPLs will be provided and analyzed.
4) The study may shed light on code hallucination in LRPLs. For instance, it may be discovered that uncertainty spikes precisely when a model generates invalid syntax or logically incorrect code.
5) All the artifacts, including data, code, models, and benchmarks, will be open-sourced.

V. Preliminary Results Achieved

Tokenizer adaptation for LRPLs [3] is one of the achieved results so far. In this paper, it was shown that updating the tokenizer (via methods like ZeTT) significantly improves LRPL code generation by better encoding rare syntax. It sets the stage for the current proposal: having demonstrated that low-level data adjustments help, now it makes sense to seek model-level improvements via UE.

VI. Research Evaluation Plan

The evaluation involves two high-resource languages (Python, Java) and two low-resource languages (Racket, Elixir), testing both general-purpose LLMs (e.g., LLaMA-3, Gemma) and code-specialized models (e.g., StarCoder2, DeepSeek-Coder). Benchmarks will include existing frameworks like BigCode Evaluation Harness, McEval, and Mercury, as well as our new code-quality benchmark, which measures functional correctness (pass rate), readability (cyclomatic complexity, Halstead metrics, comment density), efficiency (runtime performance), and idiomatic style (adherence to language-specific linters). For uncertainty estimation, the metrics such as token entropy, semantic entropy, and ensemble variance against ground-truth correctness will be assessed. Experiments will compare baseline performance with uncertainty-driven strategies (filtered fine-tuning, curriculum learning, and modified decoding) to quantify improvements in LRPL generation quality while maintaining HRPL proficiency. Human and static analysis will validate automated metrics for readability and style.

VII. Contribution Plan

The research will be disseminated through peer-reviewed publications at top-tier conferences (e.g., ICML, NeurIPS, ICSE, ASE), release open-source benchmarks (e.g., code-quality and uncertainty benchmarks) on platforms like HuggingFace and BigCode, and share fine-tuned models (e.g., UE-enhanced StarCoder 2) alongside evaluation scripts.

Acknowledgements

The work was supported by the Ministry of Economic Development of the Russian Federation (agreement No. 139-10-2025-034 dd. 19.06.2025, IGK 000000C313925P4D0002).

References

[1] F. Cassano *et al.*, "Knowledge transfer from high-resource to low-resource programming languages for code LLMs," *Proceedings of the ACM on Programming Languages*, vol. 8, no. OOPSLA2, pp. 677–708, 2024.

[2] A. Q. Jiang *et al.*, "Mistral 7B," 2023. [Online]. Available: https://arxiv.org/abs/2310.06825

[3] G. Andriushchenko and V. V. Ivanov, "Evaluating tokenizer adaptation methods for large language models on low-resource programming languages," in *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 4: Student Research Workshop)*, 2025, pp. 823–833.

[4] S. Joel, J. J. Wu, and F. H. Fard, "A survey on LLM-based code generation for low-resource and domain-specific programming languages," 2024. [Online]. Available: https://arxiv.org/abs/2410.03981

[5] D. Guo *et al.*, "DeepSeek-Coder: when the large language model meets programming – the rise of code intelligence," 2024. [Online]. Available: https://arxiv.org/abs/2401.14196

[6] A. Lozhkov *et al.*, "StarCoder 2 and The Stack v2: The next generation," *arXiv preprint arXiv:2402.19173*, 2024.

[7] L. Chai *et al.*, "McEval: Massively multilingual code evaluation," in *The Thirteenth International Conference on Learning Representations*, 2024.

[8] F. Cassano *et al.*, "MultiPL-E: A scalable and extensible approach to benchmarking neural code generation," *arXiv preprint arXiv:2208.08227*, 2022.

[9] E. Fadeeva *et al.*, "LM-Polygraph: Uncertainty estimation for language models," in *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, 2023, pp. 446–461.

[10] M. Fomicheva *et al.*, "Unsupervised quality estimation for neural machine translation," *Transactions of the Association for Computational Linguistics*, vol. 8, pp. 539–555, 2020.

[11] Y. Li, R. Qiang, L. Moukheiber, and C. Zhang, "Language model uncertainty quantification with attention chain," *arXiv preprint arXiv:2503.19168*, 2025.

[12] L. Kuhn, Y. Gal, and S. Farquhar, "Semantic uncertainty: Linguistic invariances for uncertainty estimation in natural language generation," in *The Eleventh International Conference on Learning Representations*, 2023.

[13] C. Mohri and T. Hashimoto, "Language models with conformal factuality guarantees," in *International Conference on Machine Learning*. PMLR, 2024, pp. 36 029–36 047.

[14] C. Zhang, F. Liu, M. Basaldella, and N. Collier, "LUQ: Long-text uncertainty quantification for LLMs," in *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, 2024, pp. 5244–5262.

[15] A. Malinin and M. Gales, "Uncertainty estimation in autoregressive structured prediction," in *International Conference on Learning Representations*, 2020.

[16] K. Lee, K. Lee, H. Lee, and J. Shin, "A simple unified framework for detecting out-of-distribution samples and adversarial attacks," *Advances in Neural Information Processing Systems*, vol. 31, 2018.

[17] K. Yoo, J. Kim, J. Jang, and N. Kwak, "Detection of adversarial examples in text classification: Benchmark and baseline via robust density estimation," in *Findings of the Association for Computational Linguistics: ACL 2022*, 2022, pp. 3656–3672.

[18] A. Vazhentsev *et al.*, "Hybrid uncertainty quantification for selective text classification in ambiguous tasks," in *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2023, pp. 11 659–11 681.

[19] S. Kadavath *et al.*, "Language models (mostly) know what they know," *arXiv preprint arXiv:2207.05221*, 2022.

[20] K. Tian *et al.*, "Just ask for calibration: Strategies for eliciting calibrated confidence scores from language models fine-tuned with human feedback," in *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, 2023, pp. 5433–5442.

[21] A. Shelmanov *et al.*, "A head to predict and a head to question: Pre-trained uncertainty quantification heads for hallucination detection in llm outputs," *arXiv preprint arXiv:2505.08200*, 2025.

[22] L. Ben Allal, N. Muennighoff, L. Kumar Umapathi, B. Lipkin, and L. von Werra, "A framework for the evaluation of code generation models," https://github.com/bigcode-project/bigcode-evaluation-harness, 2022.

[23] M. Du, A. T. Luu, B. Ji, Q. Liu, and S.-K. Ng, "Mercury: A code efficiency benchmark for code large language models," *Advances in Neural Information Processing Systems*, vol. 37, pp. 16 601–16 622, 2024.

[24] W. Yan *et al.*, "CodeScope: An execution-based multilingual multitask multidimensional benchmark for evaluating llms on code understanding and generation," in *62nd Annual Meeting of the Association for Computational Linguistics, ACL 2024*. Association for Computational Linguistics (ACL), 2024, pp. 5511–5558.

[25] T. Liu, C. Xu, and J. McAuley, "RepoBench: Benchmarking repository-level code auto-completion systems," *arXiv preprint arXiv:2306.03091*, 2023.

[26] Y. Ding *et al.*, "CrossCodeEval: A diverse and multilingual benchmark for cross-file code completion," *Advances in Neural Information Processing Systems*, vol. 36, pp. 46 701–46 723, 2023.

[27] C. E. Jimenez *et al.*, "SWE-Bench: Can language models resolve real-world github issues?" *arXiv preprint arXiv:2310.06770*, 2023.

[28] D. Zan *et al.*, "Multi-SWE-bench: A multilingual benchmark for issue resolving," *arXiv preprint arXiv:2504.02605*, 2025.

[29] A. Grattafiori *et al.*, "The llama 3 herd of models," *arXiv preprint arXiv:2407.21783*, 2024.

[30] A. Liu *et al.*, "Deepseek-v3 technical report," *arXiv preprint arXiv:2412.19437*, 2024.

[31] A. Kamath *et al.*, "Gemma 3 technical report," *arXiv preprint arXiv:2503.19786*, 2025.

[32] B. Roziere *et al.*, "Code llama: Open foundation models for code," *arXiv preprint arXiv:2308.12950*, 2023.

[33] B. Hui *et al.*, "Qwen2.5-Coder technical report," *arXiv preprint arXiv:2409.12186*, 2024.

[34] T. J. McCabe, "A complexity measure," *IEEE Transactions on software Engineering*, no. 4, pp. 308–320, 1976.

[35] M. H. Halstead, *Elements of Software Science (Operating and programming systems series)*. USA: Elsevier Science Inc., 1977.

[36] Y. Zhou, B. Yang, D. F. Wong, Y. Wan, and L. S. Chao, "Uncertainty-aware curriculum learning for neural machine translation," in *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, 2020, pp. 6934–6944. [Online]. Available: https://aclanthology.org/2020.acl-main.620/

[37] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017. [Online]. Available: https://arxiv.org/abs/1707.06347

[38] C. Gulcehre, T. L. Paine, S. Srinivasan, K. Konyushkova *et al.*, "Reinforced self-training (rest) for language modeling," 2023. [Online]. Available: https://arxiv.org/abs/2308.08998