Semantic vs. Structural Signals: Log-Probability and LLM-as-a-Judge for Reference-Free Code Evaluation

Dmitriy Fedrushkov[1], Yulong He[2], Ivan Smirnov[1], Artem Aliev[2], Sergey Kovalchuk[1]
[1]ITMO University, Saint Petersburg, Russia
[2]St. Petersburg State University, Saint Petersburg, Russia
Correspondence: fedrushkov@niuitmo.ru

Abstract

Reference-free evaluation of LLM-generated code is essential when execution-based testing is unavailable or costly. We compare two paradigms: **explicit LLM-as-a-Judge** scoring, which assigns a quality score to a solution, and **log-probability scoring**, which uses $\log P_\theta(\text{code} \mid \text{task})$ as an instruction-free signal.

Across HumanEval-X, we find that the two approaches capture *qualitatively different aspects* of code correctness. Explicit judges—particularly larger models—perform strongly on generated code, reflecting their ability to reason about task–solution alignment, but fail to distinguish correct solutions from minimally mutated ones. Log-probability exhibits the opposite pattern: weaker performance on generated code, but consistent pairwise separation of canonical from mutated solutions.

These results reveal a **discrimination–ranking dissociation** and show that the two paradigms provide complementary, non-interchangeable signals: explicit judges capture semantic correctness, while log-probability captures local structural consistency.

1 Introduction

Automated evaluation of LLM-generated code is essential in benchmark construction, reinforcement learning from AI feedback (RLAIF), and iterative code-generation pipelines. Execution-based evaluation (pass@k) provides a reliable ground-truth signal, but requires test harnesses and sandboxed execution, making it costly or unavailable in many settings. This motivates *reference-free* evaluation methods.

Two such approaches have recently gained attention. **Explicit LLM-as-a-Judge** scoring prompts a language model to assign a quality score to a candidate solution, leveraging its ability to reason about the task and the code. **Log-probability scoring** instead uses $\log P_\theta(\text{code} \mid \text{task})$ from a frozen model as an instruction-free signal, reflecting how expected the code is under the model.

While both approaches are widely used, their relationship remains unclear, as prior work evaluates them in isolation without controlled perturbations that preserve fluency while breaking correctness.

In this paper, we place explicit judges and log-probability scoring side by side under a unified experimental setup on HumanEval-X. We evaluate both approaches on generated code with execution-based ground truth, and on a controlled **canonical-vs-mutated** corpus where correct solutions are transformed by minimal functional mutations.

Several local models are evaluated under both paradigms—as explicit instruction-conditioned judges and as log-probability evaluators— allowing partial disentanglement of evaluation paradigm effects from the underlying model.

The two paradigms succeed in complementary settings, producing a **discrimination–ranking dissociation**: explicit judges excel at global task–solution assessment, while log-probability captures local structural consistency. Token-level analysis further shows that log-probability penalties concentrate at mutation sites, providing diagnostic localisation unavailable from score-only judges.

2 Related Work

LLM-as-a-Judge. Zheng et al. (2023a) demonstrate that strong LLMs can achieve over 80% agreement with human preferences, establishing LLM-as-a-Judge as a scalable proxy for human judgment. Zhu et al. (2025) show that fine-tuned judge models can match larger teacher models efficiently. Recent studies highlight systematic limitations: Li et al. (2024) document position bias in pairwise comparisons, and other work identifies sensitivity to prompt formulation. These findings confirm that LLM-as-a-Judge is inherently *instruction-conditioned*, with evaluations shaped

Proceedings of the Fifth Workshop on Generation, Evaluation and Metrics (GEM), pages 574–581

by prompt design and evaluation protocols rather than solely by intrinsic solution quality.

Reference-free code evaluation. Execution-based metrics (pass@k, Chen et al. 2021) provide reliable ground truth but require test harnesses. Surface metrics (CodeBLEU, Ren et al. 2020) avoid execution but correlate weakly with correctness (Fedrushkov et al., 2026). LLM-based reference-free approaches (Zhuo, 2024; Tong and Zhang, 2024) address this gap via instruction-conditioned scoring, while log-probability provides an instruction-free signal derived directly from the model.

Log-probability as a quality signal. Holtzman et al. (2020) show that high-probability text can still exhibit undesirable properties, highlighting the limitations of likelihood-based evaluation. In the context of code generation, log-probability reflects how consistent a solution is with patterns learned during pretraining, capturing syntactic and local structural consistency. However, its relationship to functional correctness remains unclear, as semantically incorrect solutions may still be locally plausible under the model.

Gap addressed. Existing work evaluates these two paradigms in isolation and does not examine their behaviour under *structurally minimal but functionally incorrect* perturbations. We provide, to our knowledge, the first head-to-head comparison with execution-based ground truth and a controlled mutation probe, revealing a qualitative (rather than merely quantitative) difference between semantic and structural evaluation signals.

3 Methodology

3.1 Dataset and Evaluation Protocol

We use **HumanEval-X** (Zheng et al., 2023b), a multilingual extension of HumanEval (Chen et al., 2021) covering 164 programming problems across five languages: Python, Java, Go, JavaScript, and C++. We use five decoder-only models (Qwen3, Llama-3.1, DeepSeek-V2, Gemma3, Phi4), all in the 8–16B parameter range, enabling fully local inference. These models are used for code generation, log-probability evaluation, and explicit instruction-conditioned judging. Ground-truth correctness labels are obtained by executing the generated solutions against the provided test suites (pass@1, binary). We evaluate two corpora:

- **Generated** — solutions produced by five local models which also serve as both log-probability and explicit judges (Section 3.2), using a standard zero-shot prompt, yielding a 5-language $\times$ 5-generator dataset with pass@1 labels from test execution. This design produces a 5×5 generator–judge cross-evaluation matrix for log-probability scoring, with the diagonal representing self-evaluation. The reuse of the same local models across generation and evaluation settings enables controlled comparison of evaluation paradigms within the same underlying models.
- **Mutated** — the benchmark's canonical (reference) implementations, each differing from the original by a single local transformation (wrong operator, off-by-one, swapped comparison, or incorrect return value), most of which fail at least one test, although a small fraction may still pass, reflecting the non-exhaustiveness of the test suites.

The Mutated corpus serves two purposes: (1) it approximates an all-fail condition (pass@1 ≈ 0), as the vast majority of mutated solutions fail, which allows us to test whether evaluation signals degrade when there is no within-condition correctness variation; and (2) it furnishes the paired canonical–mutated corpus for the discrimination experiment in Section 4.3. Mutations are generated with `universalmutator` (Groce et al., 2018; Deb et al., 2024) and are *not* limited to syntactic errors. They include semantically incorrect yet locally plausible transformations: wrong operators (`+`$\rightarrow$`-`), swapped comparisons (`<`$\rightarrow$`>`), off-by-one offsets, and incorrect return values—code that is syntactically valid but in most cases fails functional tests. This design ensures that the mutation experiment probes *structural* sensitivity, not merely detection of unparseable code. Log-probability evaluation covers all five languages including C++; explicit judge evaluation covers all five languages for the generated corpus, but only four languages (Python, Java, Go, JavaScript) for the mutated corpus. C++ is excluded from mutation-based evaluation due to safety considerations, as small automatic transformations may introduce undefined behaviour.

3.2 Evaluation Signals

Log-probability judges. Given a task description x and a candidate solution $y = (y_1, \ldots, y_T)$, the judge treats the solution as a *continuation* of the task description under the standard causal

language-modelling objective. A single forward pass over the concatenation $[x; y]$ yields token-level log-probabilities $\log P_\theta(y_t \mid x, y_{<t})$. Only code tokens are scored; the task prefix is masked (labels set to -100), so the signal reflects how expected the solution is *given the task*, not the likelihood of the task itself. We aggregate the token-level values in three ways:

$$S_{\text{sum}} = \sum_{t=1}^{T} \log P_\theta(y_t \mid x, y_{<t}), \tag{1}$$

$$S_{\text{norm}} = \frac{S_{\text{sum}}}{T^\alpha}, \quad \alpha{=}0.5, \tag{2}$$

$$S_{\text{avg}} = \frac{S_{\text{sum}}}{T} \quad (\alpha{=}1). \tag{3}$$

S_{sum} is the raw sequence log-likelihood; S_{norm} reduces the dominance of sequence length with a softer correction (we set $\alpha = 0.5$ to provide a softer length normalisation that preserves local log-probability deviations), though it can inflate scores for short, fluent but incorrect solutions—a risk examined in Section 5; S_{avg} ($\alpha{=}1$) is the standard per-token mean.

Explicit LLM-as-a-Judge scoring. We evaluate six explicit judges: two large-scale models (**gpt-oss-120B** and **Qwen3-Coder-80B**) and four local 8–16B models (**Llama-3.1**, **DeepSeek-V2**, **Gemma3**, and **Phi4**). The larger models are referred to as *gpt-oss* and *Qwen3-Coder* in tables for brevity.

Each judge is prompted to estimate, on a $[0, 1]$ scale, a score reflecting how likely the candidate solution is to correctly solve the task. The prompt presents the task description and the candidate code, and asks for a single floating-point estimate with no chain-of-thought or execution feedback. Task descriptions follow the HumanEval-X benchmark specification without modification.

Explicit judges were applied to the Generated corpus (all five languages) and to the Mutated corpus (four languages; C++ excluded). The four local explicit judges are reused from the generation and log-probability evaluation settings, enabling controlled within-model comparison between instruction-conditioned and instruction-free evaluation modes.

Token-level attribution. For the canonical-vs-mutated analysis (Section 4.4), we obtain token-level log-probability vectors from each judge model. We align the canonical and mutated token sequences using `difflib.SequenceMatcher` and classify each token as *changed* (differs between canonical and mutated) or *unchanged*. This allows us to measure mean log-probability at mutation sites separately from background positions.

3.3 Metrics

We report **ROC-AUC** for binary pass-vs-fail discrimination on the Generated corpus, and for canonical-vs-mutated discrimination on the Mutated corpus. For the mutation experiment we additionally report **pairwise accuracy** (fraction of per-task pairs where the canonical solution's *sequence-level aggregate* score exceeds its mutated counterpart's, computed independently for each aggregation) and **mean score difference** $\overline{\Delta} = \overline{\text{score}_{\text{canon}}} - \overline{\text{score}_{\text{mut}}}$. Token-level attribution uses $\Delta_{\textbf{changed}}$, the difference in mean log-probability at changed vs. unchanged token positions between canonical and mutated code.

This design enables direct comparison between semantic instruction-conditioned evaluation and structural likelihood-based evaluation using shared local models.

4 Experiments

4.1 LLM-as-a-Judge on Generated Code

We first evaluate explicit LLM-as-a-Judge scoring on generated code. Table 1 reports ROC-AUC for six instruction-conditioned judges: two large models (gpt-oss-120B and Qwen3-Coder-80B) and four local 8–16B models (Llama3, DeepSeek, Gemma3, Phi4). Results are broken down by generator model, with the Mutated corpus AUC shown as the final row to preview the contrast developed in Section 4.3.

The two larger judges achieve the strongest and most consistent performance across all five generators. **gpt-oss-120B** reaches an overall ROC-AUC of 0.971, with per-generator values ranging from 0.954 (Phi4) to 0.971 (Gemma3). **Qwen3-Coder-80B** also performs strongly (AUC 0.905 overall), uniformly below gpt-oss-120B but well above chance across all generators.

The four local explicit judges exhibit substantially weaker but still above-random discrimination ability. Among them, **Phi4** performs best (0.855 overall), followed by **Gemma3** (0.817), while **Llama3** (0.653) and **DeepSeek** (0.532) remain substantially below the larger judges. These results suggest that explicit instruction-conditioned

Table 1: Explicit LLM-as-a-Judge ROC-AUC by generator model. *Mutated*: canonical-vs-mutated AUC (overall).

Generator	gpt-oss	Qwen3-Coder	Llama3	DeepSeek	Gemma3	Phi4
Qwen3	0.968	0.886	0.647	0.554	0.796	0.837
Llama3	0.969	0.915	0.669	0.479	0.849	0.867
DeepSeek	0.969	0.923	0.667	0.564	0.837	0.873
Gemma3	0.971	0.876	0.586	0.523	0.746	0.798
Phi4	0.954	0.869	0.624	0.538	0.801	0.808
Overall	**0.971**	**0.905**	**0.653**	**0.532**	**0.817**	**0.855**
Mutated	**0.503**	**0.525**	**0.521**	**0.516**	**0.518**	**0.494**

evaluation benefits from stronger judge capability and, to some extent, model scale.

4.2 Log-Probability on Generated Code

Despite their structural similarity to the judge models, generated solutions are not well-distinguished by log-probability. ROC-AUC for binary pass-vs-fail discrimination ranges from 0.499 (Llama3) to 0.619 (Gemma3) across the same five models. These results remain substantially below the strongest explicit judges (gpt-oss-120B and Qwen3-Coder-80B), although some local instruction-conditioned judges achieve comparable performance. Performance varies across programming languages, but no log-probability judge approaches the level of the strongest explicit instruction-conditioned evaluators on any language.

A notable self-evaluation failure emerges from the generator–judge cross-evaluation matrix: models that evaluate their *own* outputs perform near-randomly or negatively (e.g., Gemma3 on Gemma3-generated code: AUC 0.539), consistent with the distributional self-confirmation identified in Pan et al. (2023). Cross-model evaluation is therefore essential when using log-probability as a quality signal.

4.3 Canonical vs. Mutated Discrimination

We now probe both paradigms with a controlled perturbation: replacing fluently generated code with canonical benchmark solutions and their single-token mutations.

Explicit judges fail to distinguish canonical from mutated code. As shown in the Mutated column of Table 1, gpt-oss-120B drops from AUC 0.971 to 0.503; Qwen3-Coder-80B drops from 0.905 to 0.525—near-random performance in our experimental setting. Single-token structural mutations preserve the overall task–solution alignment visible to an instruction-following judge, so the error is invisible at the semantic level.

Notably, this behaviour persists even for the four local models that also serve as log-probability evaluators. When used as explicit instruction-conditioned judges, these models remain near chance on the mutation task despite achieving strong canonical-vs-mutated discrimination under likelihood-based scoring.

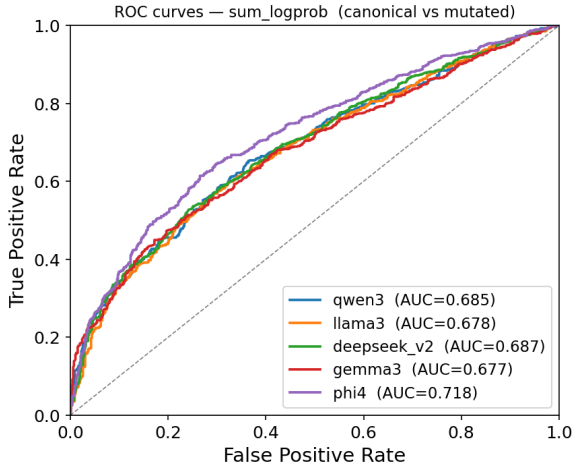
Log-probability scores decrease sharply on mutated code. Table 2 shows the mean score difference $\overline{\Delta} = \overline{\text{score}_{\text{canon}}} - \overline{\text{score}_{\text{mut}}}$ for each judge and aggregation. All differences are large and positive across all three aggregations. For S_{avg}, the gap ranges from 0.355 (DeepSeek-V2) to 0.548 (Qwen3) nats per token, producing consistent within-task pairwise ordering ($> 97\%$ accuracy) across all three aggregations.

Table 2: Log-probability mean score difference (canonical − mutated) by aggregation and judge model.

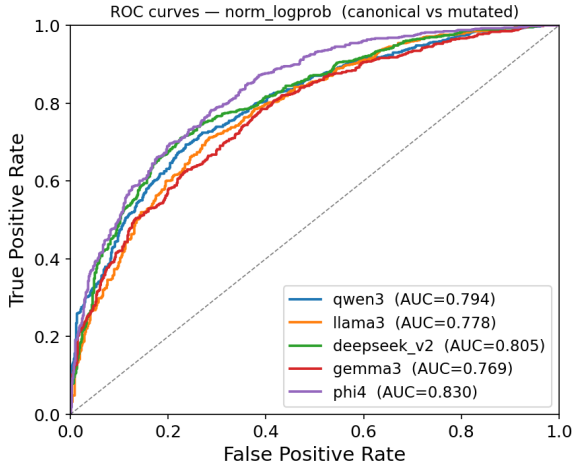
Model	sum logprob	norm logprob	avg logprob
Qwen3	20.5525	2.7563	0.5483
Llama3	15.6727	2.0691	0.4175
DeepSeek-V2	15.1424	1.8487	0.3546
Gemma3	16.8279	2.1081	0.4247
Phi4	17.7108	2.3569	0.4675

ROC analysis confirms systematic discrimination. Figure 1 shows the ROC curves for canonical-vs-mutated discrimination under all three aggregations. All models produce ROC curves consistently above the diagonal, despite high within-task pairwise accuracy (>97%), with overall AUC 0.66–0.70 for S_{avg} and up to 0.83 for length-normalised variants—far exceeding the explicit judge range of 0.503–0.525. All aggregations show consistent trends; we focus on S_{sum} and S_{norm} for visualisation. All models achieve $> 97\%$ pairwise accuracy on per-task comparisons. Go achieves the highest per-language AUC (0.728–0.749), while JavaScript is the most variable (0.668–0.704).

Figures 2 and 3 confirm the same pattern through



(a) sum_logprob



(b) norm_logprob

Figure 1: ROC curves for canonical vs. mutated discrimination. Both aggregations clearly separate canonical from mutated solutions across all five models.

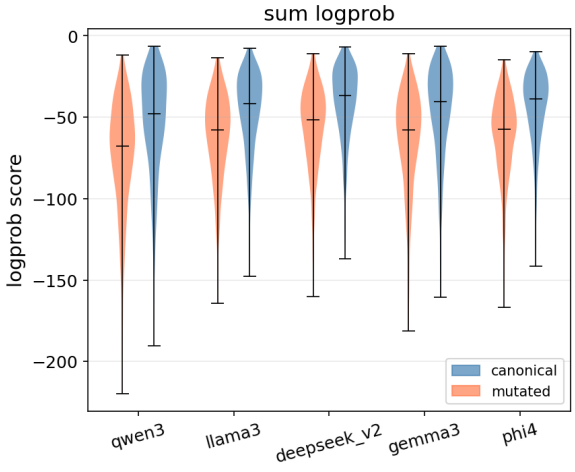


Figure 2: Score distributions for canonical (blue) and mutated (orange) code under sum log-probability.

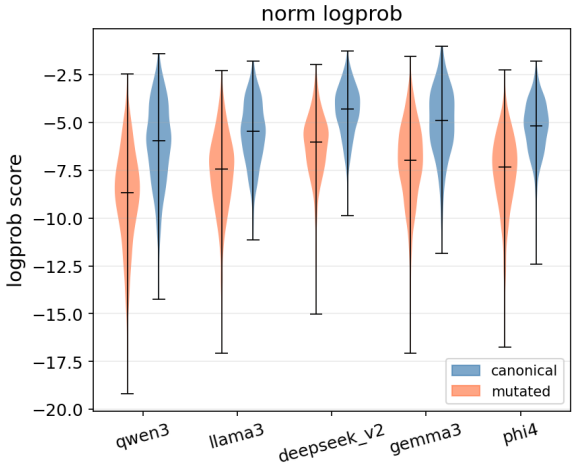


Figure 3: Score distributions for canonical (blue) and mutated (orange) code under length-normalised log-probability.

full score distributions: the canonical distribution is shifted upward relative to the mutated distribution across all judges. The gap between per-task pairwise accuracy ($> 97\%$) and cross-task AUC (0.66–0.70 for S_{avg}, up to 0.83 for normalised variants) is discussed in Section 5.

Taken together, these results show that the observed semantic–structural dissociation persists even within the same local model family: the same models exhibit near-random behaviour under instruction-conditioned judging yet strong mutation discrimination under likelihood-based evaluation.

4.4 Token-Level Attribution

The high pairwise accuracy raises a key question: is the log-probability penalty global or localised at mutation sites?

We measure mean log-probability at changed (mutation) vs. unchanged positions for canonical and mutated code across all models (Figure 4). **Unchanged positions** have comparable mean log-probability in canonical and mutated code (≈ -0.43 to -0.67 nats), indicating that overall fluency is similar. **Changed positions** show a substantial split: canonical changed tokens -0.43 to -0.71 nats; mutated changed tokens -2.35 to -3.85 nats. The difference Δ_{changed} ranges from 1.88 (DeepSeek-V2) to 3.13 (Qwen3) nats, localised to the exact mutation sites.

Figure 4 confirms this pattern across all five models. The effect is most pronounced for Qwen3 ($\Delta_{\text{changed}} = 3.13$ nats, corresponding to $23\times$ higher perplexity at mutation sites).

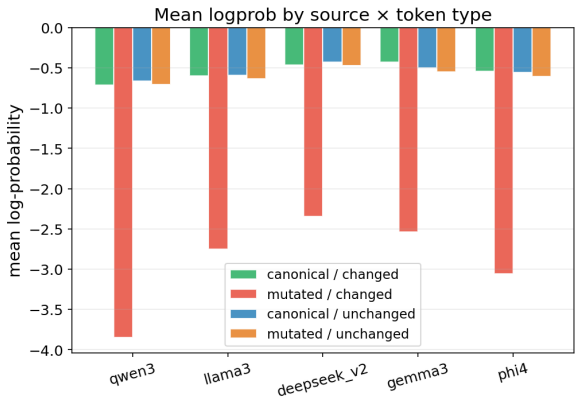


Figure 4: Mean log-probability by token type (changed/unchanged) and source (canonical/mutated), showing a sharp drop at mutation sites.

4.5 Token-Level Examples

To illustrate the token-level behaviour qualitatively, we include the original HumanEval-X prompts corresponding to the examples shown in Figure 5. The figure demonstrates representative cases in which the single changed token receives -8.38 to -11.00 nats while surrounding tokens remain near-zero in both versions.

Java/13 prompt (HumanEval-X).

```
import java.util.*;
import java.lang.*;

class Solution {
    /**
    Return a greatest common divisor of two
    integers a and b
    >>> greatestCommonDivisor(3, 5)
    1
    >>> greatestCommonDivisor(25, 15)
    5
     */
   public int greatestCommonDivisor(int a, int b) {
```

JavaScript/70 prompt (HumanEval-X).

```
/*
  Given list of integers, return list in strange
  order. Strange sorting, is when you start with
  the minimum value, then maximum of the remaining
  integers, then minimum and so on.

  Examples:
  strangeSortList([1, 2, 3, 4]) == [1, 4, 2, 3]
  strangeSortList([5, 5, 5, 5]) == [5, 5, 5, 5]
  strangeSortList([]) == []
*/
const strangeSortList = (lst) => {
```

5 Discussion

5.1 Discrimination–Ranking Dissociation

The results reveal a clear **inversion**: gpt-oss-120B achieves AUC 0.971 on generated code and 0.503 on structured mutations; Gemma3 (log-probability) achieves AUC 0.619 on generated code and 0.673 on mutations. No single evaluator consistently dominates across both settings, indicating sensitivity to different aspects of code correctness. The results therefore suggest that evaluation behaviour depends not only on model capability, but also on the evaluation paradigm through which the model is applied.

Explicit scoring is sensitive to global task–solution alignment and often reflects errors such as incorrect algorithms, missed edge cases, and wrong abstractions. Single-token mutations preserve this alignment, making the error invisible at the semantic level, likely due to reliance on high-level plausibility and learned priors over valid code, which can overlook local token-level deviations when global control flow remains intact.

Log-probability, by contrast, penalises local inconsistencies—as confirmed by token-level attribution (Section 4.4)—but may fail on semantically incorrect solutions that remain locally consistent, as such errors do not violate token-level expectations learned during pretraining.

The gap between per-task pairwise accuracy ($>97\%$) and cross-task AUC (0.66–0.70 for S_{avg}, up to 0.83 for normalised variants) reflects cross-task variability and score aggregation effects: strong local penalties may be diluted at sequence level, and easier tasks can dominate ranking. These results are conditioned on mutation-based errors; more complex logical bugs may not exhibit the same local probability patterns.

5.2 Practical Implications

Neither paradigm alone provides a reliable evaluation signal across settings. Explicit judges provide a strong global quality signal but lack sensitivity to structured local errors, while log-probability offers stable, instruction-free scoring with token-level interpretability and high pairwise discrimination accuracy.

The comparison includes both cross-scale and same-scale settings: the four local 8–16B models are evaluated both as explicit judges and as log-probability evaluators, while gpt-oss-120B and Qwen3-Coder-80B provide large-model reference points. Nevertheless, the observed complementarity suggests a hybrid approach: explicit judges for global ranking, combined with log-probability for pairwise filtering and local error detection, for example via simple score aggregation or reranking

Figure 5: Token-level log-probability for two mutation examples (Qwen3-8B judge). Colour encodes log-probability: green = consistent, red = inconsistent.

schemes. Token-level attribution can further guide targeted regeneration at low-probability positions.

6 Conclusion

We compare explicit LLM-as-a-Judge scoring and log-probability evaluation on HumanEval-X across five languages. Large explicit judges achieve strong performance on generated code (AUC up to 0.971), while smaller local explicit judges show weaker but still above-random discrimination. However, all explicit judges—including the local models reused as log-probability evaluators—fail to discriminate canonical solutions from single-token mutations, remaining near chance (AUC 0.49–0.53). Log-probability shows the opposite pattern: weaker global ranking (AUC 0.50–0.62) but high pairwise accuracy ($> 97\%$) on the mutation task, with penalties localised to mutation sites ($\Delta_{\text{changed}} = 1.9$–$3.1$ nats).

The two paradigms are complementary rather than interchangeable: explicit judges for semantic task–solution assessment, log-probability for pairwise structural discrimination and token-level diagnostics.

Future work includes extending token-level attribution to real generation errors beyond synthetic mutations, evaluating whether the observed semantic–structural dissociation persists in larger model classes comparable to explicit judges, and studying hybrid evaluation schemes that combine both signals (e.g., via weighted aggregation or pairwise reranking).

Limitations

We see the following limitations of our study which are considered as further directions with our ongoing research.

- Explicit judge scores on the Mutated corpus are not available for C++, precluding a complete five-language comparison in the mutation experiment.

- pass@1 is a coarse binary label; richer quality rubrics (readability, efficiency, partial credit) may reveal different relative strengths.

- The mutation strategy follows HumanEval conventions, producing specific patterns of defects; real-world bugs may follow different distributions.

- All log-probability judges are relatively small (8–16B parameter) decoder-only causal LMs; larger models, encoder-decoder architectures, or fill-in-the-middle models may behave differently.

- We do not investigate prompt sensitivity for explicit judges, which can significantly affect scores (Eiras et al., 2025).

- Results are based on HumanEval-style tasks (short, self-contained functions) and may not generalise to larger or real-world codebases.

Ethics statement

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

This work benefited from the use of generative AI tools for language editing and text refinement. All generated content was carefully reviewed and verified by the authors.

References

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, and 39 others. 2021. Evaluating large language models trained on code. *Preprint*, arXiv:2107.03374.

Sourav Deb, Kush Jain, Rijnard van Tonder, Claire Le Goues, and Alex Groce. 2024. Syntax is all you need: A universal-language approach to mutant generation. *Proc. ACM Softw. Eng.*, 1(FSE).

Francisco Eiras, Eliott Zemour, Eric Lin, and Vaikkunth Mugunthan. 2025. Know thy judge: On the robustness meta-evaluation of llm safety judges. *Preprint*, arXiv:2503.04474.

Dmitry Fedrushkov, Sergey Kovalchuk, and Artem Aliev. 2026. Natural language processing metrics efficiency for evaluating a generated code: facing the challenge. *Scientific and Technical Journal of Information Technologies, Mechanics and Optics*, 26:135–144.

Alex Groce, Josie Holmes, Darko Marinov, August Shi, and Lingming Zhang. 2018. An extensible, regular-expression-based tool for multi-language mutant generation. In *Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings*, ICSE '18, page 25–28, New York, NY, USA. Association for Computing Machinery.

Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. 2020. The curious case of neural text degeneration. *arXiv preprint arXiv:1904.09751*.

Zongjie Li, Chaozheng Wang, Pingchuan Ma, Daoyuan Wu, Shuai Wang, Cuiyun Gao, and Yang Liu. 2024. Split and merge: Aligning position biases in LLM-based evaluators. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 11084–11108, Miami, Florida, USA. Association for Computational Linguistics.

Liangming Pan, Michael Saxon, Wenda Xu, Deepak Nathani, Xinyi Wang, and William Yang Wang. 2023. Automatically correcting large language models: Surveying the landscape of diverse self-correction strategies. *Preprint*, arXiv:2308.03188.

Shuo Ren, Daya Guo, Shuai Lu, Long Zhou, Shujie Liu, Duyu Tang, Neel Sundaresan, Ming Zhou, Ambrosio Blanco, and Shuai Ma. 2020. Codebleu: a method for automatic evaluation of code synthesis. *Preprint*, arXiv:2009.10297.

Weixi Tong and Tianyi Zhang. 2024. Codejudge: Evaluating code generation with large language models. *Preprint*, arXiv:2410.02184.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. 2023a. Judging llm-as-a-judge with mt-bench and chatbot arena. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23, Red Hook, NY, USA. Curran Associates Inc.

Qinkai Zheng, Xiao Xia, Xu Zou, Yuxiao Dong, Shan Wang, Yufei Xue, Lei Shen, Zihan Wang, Andi Wang, Yang Li, Teng Su, Zhilin Yang, and Jie Tang. 2023b. Codegeex: A pre-trained model for code generation with multilingual benchmarking on humaneval-x. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, KDD '23, page 5673–5684, New York, NY, USA. Association for Computing Machinery.

Lianghui Zhu, Xinggang Wang, and Xinlong Wang. 2025. Judgelm: Fine-tuned large language models are scalable judges. *Preprint*, arXiv:2310.17631.

Terry Yue Zhuo. 2024. ICE-score: Instructing large language models to evaluate code. In *Findings of the Association for Computational Linguistics: EACL 2024*, pages 2232–2242, St. Julian's, Malta. Association for Computational Linguistics.