Measuring LLM Code Generation Stability via Structural Entropy

Yewei Song, Tiezhu Sun, Xunzhu Tang, Prateek Kumar Rajput,
Tegawendé F. Bissyandé and Jacques Klein,
The Interdisciplinary Centre for Security, Reliability and Trust
University of Luxembourg

***Abstract*—Assessing the stability of code generation from large language models (LLMs) is essential for judging their reliability in real-world development. We extend prior "structural-entropy" concepts to the program domain by pairing entropy with abstract-syntax-tree (AST) analysis. For any fixed prompt, we collect the multiset of depth-bounded subtrees of AST in each generated program and treat their relative frequencies as a probability distribution. We then measure stability in two complementary ways: (i) *Jensen–Shannon divergence*, a symmetric, bounded indicator of structural overlap, and (ii) a *Structural Cross-Entropy* ratio that highlights missing high-probability patterns. Both metrics admit structural-only and token-aware variants, enabling separate views on control-flow shape and identifier-level variability. Unlike pass@k, BLEU, or CodeBLEU, our metrics are reference-free, language-agnostic, and execution-independent. We benchmark several leading LLMs on standard code generation tasks, demonstrating that AST-driven structural entropy reveals nuances in model consistency and robustness. The method runs in $O(n, d)$ time with no external tests, providing a lightweight addition to the code-generation evaluation toolkit.**

***Index Terms*—Large Language Models, Code Generation, Structural Entropy, Evaluation Metrics, Stability Test**

I. Introduction

The advent of powerful Large Language Models (LLMs) has enabled remarkable capabilities in automated code generation. However, a notable challenge is the high variability of generated code: identical prompts can yield substantially different code snippets across runs or across models. Prior studies observe that even with fixed inputs and hyperparameters, state-of-the-art LLMs are rarely deterministic at the output level [1]. For example, ChatGPT produces completely different code on repeated queries for the same programming task in the majority of cases (e.g. 75.8% of tasks showed zero identical test outputs across runs) [2]. Crucially, setting the sampling temperature to zero (greedy decoding) did not guarantee consistency. This output instability undermines developer trust and makes reproducibility of code-generation research difficult. In safety-critical or collaborative software settings, unpredictable LLM suggestions can harm reliability. Hence there is a pressing need to rigorously quantify the structural stability of generated code, beyond simply assessing correctness.

Current evaluation of LLM code outputs largely focuses on functional correctness or textual similarity to reference solutions. The pass@k metric, for example, checks if any of k samples passes the unit tests [3], thereby measuring functional success. While pass@k and related metrics (e.g. the unbiased pass-ratio@n) are valuable for overall performance [4], they do not address variability: different outputs may all pass the tests but differ substantially in structure. Likewise, traditional NLP-style metrics (BLEU, ROUGE, METEOR, etc. [5]–[7]) assess n-gram overlap with a reference solution, but have known limitations for code [8], [9]. Ren et al. in the CodeBLEU work show that BLEU correlates poorly with code semantics and cannot account for the many functionally equivalent programs that do not share surface tokens [10]. Code-specific metrics such as CodeBLEU have been proposed to address syntax and semantics: CodeBLEU augments n-gram matches with AST-based syntax weighting and data-flow features [10]. These improvements help correlate with human judgments, but CodeBLEU and similar metrics still measure pairwise similarity to a single reference output rather than the consistency across multiple samples. We list some metrics that are currently used for measuring LLM stability in Table I. In short, existing metrics focus on correctness or reference fidelity, but they do not capture structural/topological similarity of different outputs from the same prompt.

Although structural entropy—measuring uncertainty or variability in structural choices—is not widely used in LLM code-generation literature, related concepts have been explored in software engineering contexts. For example, Torres et al. applied structural entropy metrics to study software evolution, quantifying how changes affect the organization and complexity of code structures over time [11]. Their method captures how transformations impact predictability and structural information in software systems. A closely analogous idea in LLM text generation is Semantic Entropy (SE), proposed by Kossen et al., which assesses model uncertainty by clustering multiple generated answers by meaning; high SE indicates diverse or inconsistent outputs, signaling possible hallucinations [12]. Inspired by this, our work adapts the entropy concept to code generation, analyzing variability in generated code AST structures. Low structural entropy reflects consistent outputs, whereas high entropy indicates the model frequently alternates between different programming structures or approaches.

II. Structural Entropy and Similarity

Our pipeline consists of three phases: subtree extraction from ASTs, constructing empirical distributions from these subtrees,

[1]Proposed by this paper. Readers can check the code with the link: https://github.com/Etamin/SCE.git

2643-1572/25/$31.00 ©2025 IEEE
DOI 10.1109/ASE63991.2025.00343

TABLE I
COMPARISON OF CURRENT STABILITY METRICS WITH OUR APPROACHES.

Evaluation Method	Description	Structure	Ref.
BLEU/ ROUGE-L/ METEOR	N-gram overlap or sequence matching to reference code.	No	[5]–[7]
Exact Match	Checks if generated code exactly matches the reference.	No	-
Pass@k	Runs up to k generated outputs; succeeds if any pass all tests.	Indirect	[13]
CodeBLEU	Combines BLEU with AST subtree match and data-flow match.	Yes	[10]
RUBY	Compare Program Dependency Graph of output vs. reference.	Yes	[14]
TSED	Computes the tree edit distance between AST of output and AST of reference.	Yes	[15]
Semantic Entropy	Clusters multiple answers to the same question, then calculates entropy.	Indirect	[16]
Structural Entropy[1]	Proposed: Parse the answers' AST structure and computes the subtree entropy between answers, with KL or JS divergence.	Yes	-

and computing similarity metrics. We parse code outputs into ASTs and extract depth-bounded subtrees, transforming them into canonical encodings. These encodings are then used to construct empirical distributions that capture structural variations between code outputs. Finally, we calculate stability using two entropy-based metrics: Jensen–Shannon divergence, providing a symmetric measure of structural similarity, and Structural Cross-Entropy, which emphasizes missing high-probability patterns.

Phase 1 : Depth-bounded Subtree Extraction: Let $\mathcal{T}_A$ and $\mathcal{T}_B$ denote the ASTs parsed from two code snippets. Fix a depth parameter $d \in \mathbb{N}$. For every node v in an AST, we consider the rooted subtree $\mathrm{sub}(v; d)$ which is the fragment of $\mathcal{T}$ rooted at v containing all descendants up to depth d. To transform such subtrees into hashable symbols, we define two canonical encodings:

(a) *Structure-only* encoding $\sigma_{\text{struct}} : \mathrm{sub}(v; d) \mapsto (\text{node-type}(v), \big(\text{node-type}(c_1), \ldots, \text{node-type}(c_k)\big))$, where $c_1, \ldots, c_k$ are the immediate children of v.

(b) *Structure+value* encoding $\sigma_{\text{value}} : \mathrm{sub}(v; d) \mapsto (\text{node-type}(v),\ \text{lexeme}(v),\ \big(\text{node-type}(c_1), \ldots, \text{node-type}(c_k)\big))$, where $\text{lexeme}(v)$ is the exact source text covered by v (for leaves) or a sentinel $\varnothing$ (for internal nodes).

These two approaches trade off generality vs. specificity. Structure-only patterns may capture common coding patterns and yield higher overlap between different programs, but they lose fine-grained information. Structure-with-value patterns are more discriminative (sensitive to exact code), but they also increase the size of the pattern vocabulary and may yield sparser overlap. In practice, one can choose the representation depending on whether value information is important to the similarity task.

For either choice of σ, we enumerate the multisets of depth-bounded subtrees extracted from $\mathcal{T}_A$ and $\mathcal{T}_B$:

$$\begin{aligned} S_A &= \big\{\sigma\big(\mathrm{sub}(v; d)\big) \;\big|\; v \in \mathcal{T}_A\big\}, \\ S_B &= \big\{\sigma\big(\mathrm{sub}(v; d)\big) \;\big|\; v \in \mathcal{T}_B\big\}. \end{aligned} \tag{1}$$

Let $n_A = |S_A|$ and $n_B = |S_B|$ be the total numbers of (possibly repeated) subtree symbols harvested. In (1) we treat S_A, S_B as multisets symbols may appear with multiplicity.

Phase 2 : Constructing Empirical Distributions: Let the multisets of subtree symbols extracted in Phase 1 be $S_A \subseteq \Sigma^*$ and $S_B \subseteq \Sigma^*$, where Σ^* denotes the countable universe of canonical subtree encodings (either structure–only or structure–with–value). Define the *joint support*

$$U = S_A \cup S_B = \big\{u_1, \ldots, u_m\big\}, \qquad m = |U|. \tag{2}$$

Each $u_i \in U$ is a distinct subtree symbol.

Multiplicity functions.: For $u \in U$ define

$$\begin{aligned} c_A(u) &= \#\big\{\, s \in S_A : s = u \,\big\}, \\ c_B(u) &= \#\big\{\, s \in S_B : s = u \,\big\}, \end{aligned} \tag{3}$$

i.e. the number of occurrences of u in the respective multiset. Let

$$n_A = \sum_{u \in U} c_A(u), \qquad n_B = \sum_{u \in U} c_B(u) \tag{4}$$

be the total numbers of (depth-bounded) subtrees harvested from $\mathcal{T}_A$ and $\mathcal{T}_B$, respectively.

Empirical probability distributions.: We define the frequency distributions $P, Q : U \to [0, 1]$ by

$$\begin{aligned} P(u) &= \frac{c_A(u)}{n_A}, \\ Q(u) &= \max\Big(\frac{c_B(u)}{n_B}, \varepsilon\Big), \\ \sum_{u \in U} P(u) &= \sum_{u \in U} Q(u) = 1, \end{aligned} \tag{5}$$

where $0 < \varepsilon \ll 1$ is a fixed smoothing constant ensuring $Q(u) > 0$ for every $u \in U$. (The summation constraint on Q can be restored by a final renormalisation, but in practice $\varepsilon \ll 1/n_B$ suffices and the effect on $\sum_u Q(u)$ is negligible.)

In vector notation let

$$\begin{aligned} \mathbf{p} &= \big(P(u_1), \ldots, P(u_m)\big)^{\mathsf{T}}, \\ \mathbf{q} &= \big(Q(u_1), \ldots, Q(u_m)\big)^{\mathsf{T}} \in [0, 1]^m \end{aligned} \tag{6}$$

so $\|\mathbf{p}\|_1 = \|\mathbf{q}\|_1 = 1$. These two probability vectors fully characterize the structural 'vocabularies' of AST $\mathcal{T}_A$ and $\mathcal{T}_B$ and constitute the input for the similarity metrics of Phase 3.

Remark. The inclusion of a smoothing parameter ε is only required for the directed cross-entropy in Phase 3 A to avoid undefined $\log Q(u)$ when $Q(u) = 0$. The symmetric Jensen–Shannon divergence of Phase 3 B remains finite without explicit smoothing as long as P and Q are defined on the common support U.

Phase 3 : Entropy–Based Similarity Metrics: With P and Q defined in (5), we present two similarity measures.

(A) Structural Cross-Entropy (SCE): Define the cross-entropy of P relative to Q and the Shannon entropy of Q as

$$\begin{aligned} H(P, Q) &= -\sum_{u \in U} P(u) \log Q(u), \\ H(Q) &= -\sum_{u \in U} Q(u) \log Q(u). \end{aligned} \tag{7}$$

We normalise by taking the *ratio*

$$S_{\mathrm{CE}}(P,Q) \;=\; \frac{H(Q)}{H(P,Q)}, \qquad 0 < S_{\mathrm{CE}} \le 1.$$
$$S_{\mathrm{CE}}(P,Q) = 1 \iff P = Q; \tag{8}$$
$$S_{\mathrm{CE}} \to 0 \text{ as } Q \text{ fails to explain } P.$$

Because the numerator and denominator share the same logarithmic base, the score is scale–independent and monotone in the KL divergence $D_{\mathrm{KL}}(P \parallel Q) = H(P,Q) - H(P)$. In code, compute $H(Q)$ and $H(P,Q)$ directly with base-2 logs; skip terms with $P(u) = 0$ and apply ε smoothing on $Q(u)$. This ratio acts as a normalized **structural cross-entropy**.

(B) Jensen–Shannon Divergence (JSD): Let $M = \frac{1}{2}(P+Q)$ denote the midpoint distribution. The Jensen–Shannon divergence is

$$D_{\mathrm{JS}}(P \parallel Q) \;=\; \tfrac{1}{2}D_{\mathrm{KL}}(P \parallel M) \;+\; \tfrac{1}{2}D_{\mathrm{KL}}(Q \parallel M)$$
$$\;=\; H(M) - \tfrac{1}{2}\big[H(P) + H(Q)\big]. \tag{9}$$

With logarithm base 2, $D_{\mathrm{JS}} \in [0,1]$ bits and $\sqrt{D_{\mathrm{JS}}}$ is a metric [17]. We use

$$S_{\mathrm{JSD}}(P,Q) \;=\; 1 - D_{\mathrm{JS}}(P \parallel Q) \in [0,1], \tag{10}$$

where larger values again indicate greater similarity. Unlike S_{CE}, JSD is *symmetric* and finite without smoothing so long as P, Q share support.

a) Computational cost: Both scores require $O(|U|)$ arithmetic once P and Q are formed. When depth d is fixed, $|U|$ grows at most linearly in the number of AST nodes, the overall pipeline $O(n\,d)$ for subtree extraction plus $O(|U|)$ for scoring.

b) Summary: S_{CE} (Equation 8) offers a directed measure—"how efficiently does distribution Q encode P?"—while S_{JS} (Equation 10) gives a symmetric, bounded divergence–to–similarity conversion. Either can be deployed depending on whether directionality or symmetry is desired; both map naturally into the $[0,1]$ range, facilitating thresholding and comparative analysis of program structures.

III. Experiments and Results

To evaluate the effectiveness and interpretability of both entropy-based stability metrics we proposed, (1) Jensen–Shannon Divergence (JSD) and (2) Structural Cross Entropy (SCE), we conducted experiments focusing on the stability of LLM-generated code. Specifically, we provided each prompt to the LLM five times independently, generating five distinct outputs per prompt. We then measured stability by computing similarity or divergence scores pairwise among these five outputs, using standard evaluation metrics (BLEU, CodeBLEU, TSED) and our proposed entropy-based metrics (JSD and SCE). The final stability score for each metric was obtained by averaging all pairwise comparisons.

For comparison, we also included pass@k, a metric evaluating functional correctness, computed by executing each generated code output against provided test cases. Unlike other metrics, pass@k directly reflects correctness rather than stability but serves as an important baseline for understanding practical implications of structural stability. Additionally, we analyzed structural-only variants – JSD (structural) and SCE (structural) – (cf. Section II.1(a)) by abstracting away token values, thus isolating structural differences to examine their effect on stability.

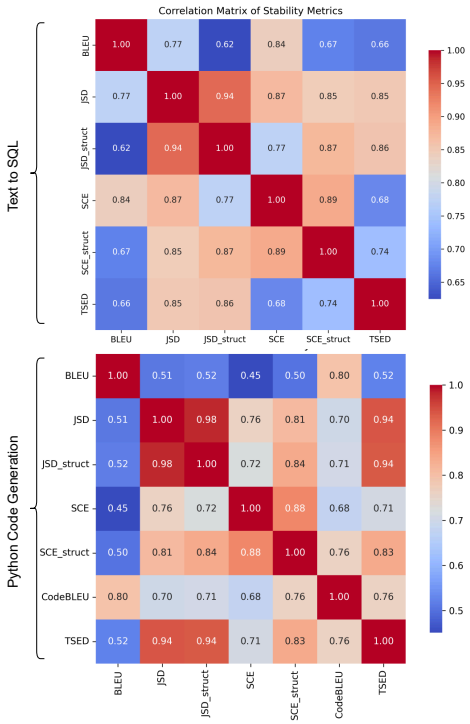

Fig. 1. Heatmap of Pearson Correlation Coefficient between Stability Evaluation Metrics, JSD: Jensen-Shannon Divergence; SCE: Structural Cross Entropy, Structural means value in subtree is ignored.

A. Experimental Setup

We performed our evaluations using two widely recognized benchmarks across two programming languages:

- **Python (BigCodeBench):** A dataset widely adopted for general code generation tasks [18].
- **SQL (Spider):** A comprehensive benchmark for SQL query generation [19].

We evaluated three representative models: LLaMA 3.1 (8B, instruct), Qwen-2.5(7B, instruct), and Qwen-2.5-Coder(7B). Our analysis focuses on understanding how our entropy-based measures reflect code stability and their correlations with existing metrics.

B. Quantitative Results

The results are summarized in Table II. For the Python tasks, while the Qwen-2.5 models exhibit stronger stability in lexical metrics such as BLEU and CodeBLEU compared to LLaMA 3.1, a notable discrepancy between their pass@1 and pass@5 indicates considerable variability. Specifically, Qwen-2.5-Coder achieves pass@1 of 0.373 and pass@5 of 0.517, suggesting

TABLE II
COMPARISON OF CURRENT STABILITY METRICS. JSD: JENSEN-SHANNON DIVERGENCE; SCE: STRUCTURAL CROSS ENTROPY; STRUCTURAL MEANS ONLY COMPARES THE SUBTREE STRUCTURE.

Model	Language(Task)	Avg. BLEU	Code BLEU	Pass@1	Pass@5	TSED	SCE(structural)	SCE	JSD(structural)	JSD
LLaMA-3.1 8B it	Python BigCodeBench	0.428	0.669	0.289	0.481	0.785	0.798	0.656	0.940	0.898
Qwen-2.5 7B it		0.596	0.716	0.339	0.482	0.765	0.823	0.726	0.947	0.913
Qwen-2.5-Coder 7B		0.614	0.715	0.373	0.517	0.764	0.823	0.722	0.951	0.918
LLaMA-3.1 8B it	SQL Spider	0.495	N/A	0.673	0.824	0.832	0.729	0.669	0.934	0.905
Qwen-2.5 7B it		0.770	N/A	0.716	0.790	0.921	0.886	0.850	0.971	0.958
Qwen-2.5-Coder 7B		0.664	N/A	0.787	0.860	0.927	0.822	0.781	0.962	0.946

that multiple sampled outputs differ substantially in correctness, reflecting inherent instability.

For SQL (Spider), this instability is also evident. Although Qwen models achieve higher overall correctness and stability, they demonstrate substantial differences between pass@1 and pass@5 (e.g., Qwen-2.5-Coder: pass@1 = 0.787, pass@5 = 0.860), further highlighting output variability.

Our proposed entropy-based stability metrics (JSD and SCE) complement these findings. JSD values remain consistently high (above 0.9), suggesting that structurally, outputs are broadly similar across samplings. However, SCE scores (especially when considering token values explicitly) are consistently lower, indicating sensitivity to subtle variations overlooked by purely structural metrics. This contrast highlights the utility of SCE in detecting fine-grained token-level variability.

Moreover, the structural-only variants of both metrics generally produce higher scores, as expected, reflecting their insensitivity to identifier or literal changes. Thus, these structural-only scores represent a baseline stability that isolates deeper syntactic patterns from superficial token-level differences.

Overall, our metrics successfully quantify and clarify the nature of stability in generated code, complementing traditional correctness-based metrics like pass@k, which indirectly reflect variability through the disparity between single- and multi-sample correctness evaluations.

C. Correlation Analysis

Figure 1 illustrates the Pearson correlation coefficient across stability metrics:

a) Python (BigCodeBench): We observe high correlations between JSD and JSD(structural) (0.94), indicating structural abstraction maintains strong stability signals. The correlation between SCE and JSD metrics is moderately strong (0.85-0.87), though lower than the near-perfect correlation between JSD and its structural variant (0.94). TSED shows good correlation with JSD (0.85–0.86), reinforcing its utility for capturing structural differences. Notably, SCE structural variants correlate less strongly with TSED (0.74), highlighting differences in sensitivity to structural variations.

b) SQL (Spider): On SQL tasks, the correlations between JSD and JSD(structural) are even stronger (0.98), underscoring the robustness of structural abstraction in capturing stability. Interestingly, BLEU's correlation is lower with JSD metrics (around 0.51), emphasizing the limitation of token-based metrics in SQL's structural context. CodeBLEU is not valid for SQL, hence excluded. TSED correlates strongly with JSD (0.94), confirming their structural similarity evaluation alignment. The SCE metrics correlate moderately (0.72–0.88) with others, again highlighting their distinctive sensitivity to token-level discrepancies.

IV. THREATS TO VALIDITY

Several threats may impact the validity of our findings. Firstly, our experiments were conducted on specific benchmarks (Python BigCodeBench and SQL Spider), potentially limiting generalizability across other languages and tasks. Second, the chosen depth parameter in subtree extraction might influence stability measures; different values could yield varied results. Additionally, while our entropy-based metrics effectively capture syntactic variability, they currently do not explicitly address semantic equivalences or execution behaviors. Lastly, the limited number of models evaluated and the constrained dataset size may affect the robustness and external validity of our conclusions. Future work will address these limitations by expanding evaluations across diverse programming languages, incorporating broader semantic and behavioral analyses, and employing larger-scale benchmarks.

V. CONCLUSION

In this paper, we introduced entropy-based metrics—Jensen-Shannon divergence (JSD) and structural cross-entropy (SCE)—to quantify the stability of LLM-generated code using Abstract Syntax Trees (ASTs). Our experiments demonstrate that these metrics:

1) Provide insights beyond functional correctness (pass@k) and lexical-syntactic metrics (BLEU, CodeBLEU).
2) Correlate strongly with AST-based structural metrics (e.g., TSED), effectively capturing structural stability.
3) Offer significant advantages with structural-only abstraction by reducing token-value noise.

Distinguishing structural-only from full-subtree encodings highlights deep syntactic stability versus token-specific variations. These metrics are lightweight, language-agnostic, and generalizable, complementing existing stability assessments.

Future work will incorporate semantic relationships, such as data- and control-flow dependencies, to extend analysis to behavioral stability. We also plan adaptive subtree-weighting schemes and benchmarking on larger, multi-module codebases to study long-range structural variability.

ACKNOWLEDGEMENT

The FNR funded this research under grants NCER22/IS/16570468/NCERFT.

REFERENCES

[1] B. Atil, A. Chittams, L. Fu, F. Ture, L. Xu, and B. Baldwin, "Llm stability: A detailed analysis with some surprises," *arXiv preprint arXiv:2408.04667*, 2024.

[2] S. Ouyang, J. M. Zhang, M. Harman, and M. Wang, "An empirical study of the non-determinism of chatgpt in code generation," *ACM Transactions on Software Engineering and Methodology*, vol. 34, no. 2, pp. 1–28, 2025.

[3] M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. D. O. Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman *et al.*, "Evaluating large language models trained on code," *arXiv preprint arXiv:2107.03374*, 2021.

[4] J. Jiang, F. Wang, J. Shen, S. Kim, and S. Kim, "A survey on large language models for code generation," *arXiv preprint arXiv:2406.00515*, 2024.

[5] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu, "Bleu: a method for automatic evaluation of machine translation," in *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, 2002, pp. 311–318.

[6] S. Banerjee and A. Lavie, "METEOR: An automatic metric for MT evaluation with improved correlation with human judgments," in *Proceedings of the ACL Workshop on Intrinsic and Extrinsic Evaluation Measures for Machine Translation and/or Summarization*, J. Goldstein, A. Lavie, C.-Y. Lin, and C. Voss, Eds. Ann Arbor, Michigan: Association for Computational Linguistics, Jun. 2005, pp. 65–72. [Online]. Available: https://aclanthology.org/W05-0909/

[7] C.-Y. Lin, "Rouge: A package for automatic evaluation of summaries," in *Text summarization branches out*, 2004, pp. 74–81.

[8] M. Evtikhiev, E. Bogomolov, Y. Sokolov, and T. Bryksin, "Out of the bleu: how should we assess quality of the code generation models?" *Journal of Systems and Software*, vol. 203, p. 111741, 2023.

[9] N. Tran, H. Tran, S. Nguyen, H. Nguyen, and T. Nguyen, "Does bleu score work for code migration?" in *2019 IEEE/ACM 27th International Conference on Program Comprehension (ICPC)*. IEEE, 2019, pp. 165–176.

[10] S. Ren, D. Guo, S. Lu, L. Zhou, S. Liu, D. Tang, N. Sundaresan, M. Zhou, A. Blanco, and S. Ma, "Codebleu: a method for automatic evaluation of code synthesis," *arXiv preprint arXiv:2009.10297*, 2020.

[11] A. Torres, S. Baltes, C. Treude, and M. Wagner, "Applying information theory to software evolution," in *2023 IEEE/ACM 2nd International Workshop on Natural Language-Based Software Engineering (NLBSE)*. IEEE, 2023, pp. 48–55.

[12] J. Kossen, J. Han, M. Razzak, L. Schut, S. Malik, and Y. Gal, "Semantic entropy probes: Robust and cheap hallucination detection in llms," *arXiv preprint arXiv:2406.15927*, 2024.

[13] S. Kulal, P. Pasupat, K. Chandra, M. Lee, O. Padon, A. Aiken, and P. S. Liang, "Spoc: Search-based pseudocode to code," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[14] D. G. Paul, H. Zhu, and I. Bayley, "Benchmarks and metrics for evaluations of code generation: A critical review," in *2024 IEEE International Conference on Artificial Intelligence Testing (AITest)*. IEEE, 2024, pp. 87–94.

[15] Y. Song, C. Lothritz, X. Tang, T. Bissyandé, and J. Klein, "Revisiting code similarity evaluation with abstract syntax tree edit distance," in *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, 2024, pp. 38–46.

[16] A. Chakraborty and A. Agarwal, "Evaluating llm using semantic entropy," *ThoughtWorks Insights*, 2024. [Online]. Available: https://www.thoughtworks.com/insights/blog/generative-ai/Evaluating-LLM-using-semantic-entropy

[17] J. Lin, "Divergence measures based on the shannon entropy," *IEEE Transactions on Information theory*, vol. 37, no. 1, pp. 145–151, 2002.

[18] T. Y. Zhuo, M. C. Vu, J. Chim, H. Hu, W. Yu, R. Widyasari, I. N. B. Yusuf, H. Zhan, J. He, I. Paul *et al.*, "Bigcodebench: Benchmarking code generation with diverse function calls and complex instructions," *arXiv preprint arXiv:2406.15877*, 2024.

[19] T. Yu, R. Zhang, K. Yang, M. Yasunaga, D. Wang, Z. Li, J. Ma, I. Li, Q. Yao, S. Roman *et al.*, "Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task," in *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, 2018, pp. 3911–3921.