Revisiting Code Similarity Evaluation with Abstract Syntax Tree Edit Distance

Yewei Song[1], Cedric Lothritz[1,2], Daniel Tang[1], Tegawendé F. Bissyandé[1], and Jacques Klein[1]

[1]University of Luxembourg
[2]Luxembourg Institute of Science and Technology
[1]{yewei.song, xunzhu.tang, tegawende.bissyande, jacques.klein}@uni.lu
[2]{cedric.lothritz}@list.lu

Abstract

This paper revisits recent code similarity evaluation metrics, particularly focusing on the application of Abstract Syntax Tree (AST) editing distance in diverse programming languages. In particular, we explore the usefulness of these metrics and compare them to traditional sequence similarity metrics. Our experiments showcase the effectiveness of AST editing distance in capturing intricate code structures, revealing a high correlation with established metrics. Furthermore, we explore the strengths and weaknesses of AST editing distance and prompt-based GPT similarity scores in comparison to BLEU score, execution match, and Jaccard Similarity. We propose, optimize, and publish an adaptable metric that demonstrates effectiveness across all tested languages, representing an enhanced version of Tree Similarity of Edit Distance (TSED).

1 Introduction and Related Work

In the fields of natural language processing and software engineering, code generation tasks are gaining more and more attention. Assessing the quality of generated code is now critically important, but we still lack evaluation methods other than traditional statistical sequence evaluation methods. Widely used semantic evaluation metrics like BLEU score and Jaccard similarity rely on statistical characteristics, overlooking the intricate grammatical structures and logical relationships inherent in complex programming languages.

However, recent developments in the NLP field paved the way for novel evaluation metrics which we explore in this study. For one, the staggering number of powerful large language models (LLMs) such as GPT-3.5/4 (Achiam et al., 2023) revolutionized the NLP landscape and led to noteworthy advancements in the realm of code review and evaluation (Wang et al., 2023; Tang et al., 2024). Another recent study introduced the novel TSED metric and used it to evaluate text-to-SQL tasks (Song et al., 2023). For this study, we take advantage of these developments to (1) prompt the GPT-4 model to generate similarity scores for code, and (2) expand on the TSED metric.

We utilize these two different metrics (GPT and TSED) to evaluate the structural similarity of different programming languages and how they relate to execution matches. Furthermore, we address how these metrics are correlated to semantic similarity metrics like the BLEU score. Finally, we investigate some limitations of these metrics by delving into the impact of TSED's penalty weight of tree operations on evaluation accuracy and exploring the stability of outputs from the GPT LLMs.

As a result, we have these 3 contributions from this research: (a) we propose and publish a new tool for 48 programming languages[1], (b) we discuss 2 recent evaluation metrics and 2 traditional metrics and compare them via correlation coefficient, recall to execution match, (c) we discuss the unstable nature of GPT similarity scoring and the ways to optimize TSED.

2 Approaches

2.1 TSED on Programming Languages

Applying the TSED evaluation method, initially designed for SQL analysis, we have undergone modifications to extend its applicability to various programming languages. The fundamental TSED approach, illustrated in Figure 1, encompasses AST parsing, AST Editing Distance Calculation, and normalization, closely resembling the methodology outlined in the original paper. However, we have made modifications to both the AST parsing and normalization.

Code Parsing: Parsing in the domain of programming languages involves parsing raw code

[1]https://github.com/Etamin/TSED

Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers), pages 38–46

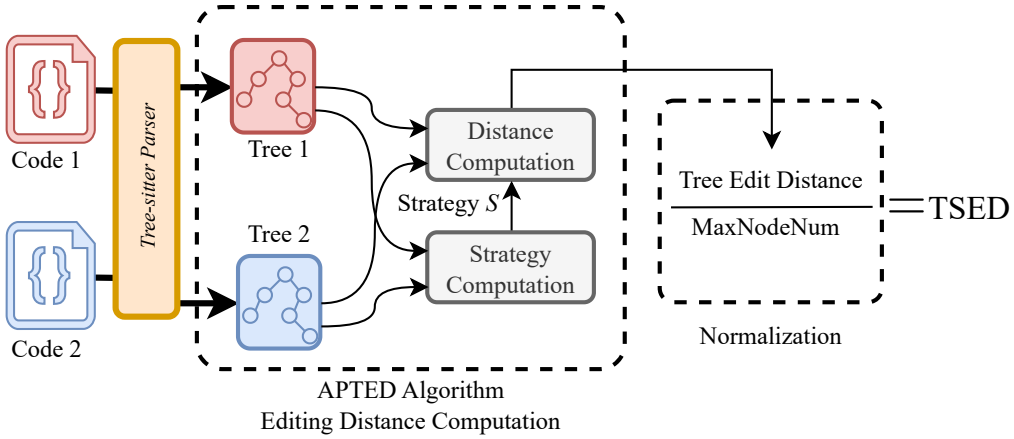

Figure 1: Pipeline of TSED Code Evaluation Metric

text into its associated AST. This parsing underscores the complexity of interpreting various programming constructs and converting them into a structured grammar tree representation.

We use tree-sitter[2] as our AST parser which is based on GLR(generalized left-to-right rightmost), a powerful parsing algorithm commonly found in the literature (Latif et al., 2023; Tomita, 1991; Clem and Thomson, 2021).

Tree Distance Computation: For calculating tree edit distance as Δ, we utilize the same function as outlined in the TSED paper, which is APTED(All Path Tree Edit Distance) algorithm (Pawlik and Augsten, 2015, 2016). Considering G_1 as predicted code's AST and G_2 as AST from ground-truth:

$$\Delta(G_1, G_2) = \min_{ops} \sum_{i=1}^{n} w(op_i) \quad (1)$$

Here, ops is a sequence of edit operations transforming G_1 into G_2, with $w(op_i)$ as the cost for the i^{th} operation.

Normalization: Normalization of tree edit distances accounts for the complexity of the code by considering the maximum number of nodes between two trees, and we add a ramp function to avoid some extreme situations:

$$TSED = \max\{1 - \frac{\delta}{MaxNodes(G_1, G_2)}, 0\} \quad (2)$$

This provides a metric for structural similarity comparison of programming code, enabling a nuanced analysis beyond mere syntactic comparison.

2.2 GPT Structure Similarity

Between 2020 and 2023, OpenAI introduced the GPT-3/3.5 and GPT-4 models, showcasing remarkable reasoning capabilities and achieving state-of-the-art performance across numerous tasks (Brown et al., 2020). Our approach involves utilizing prompts to elicit the model's output regarding the structural similarity between two code segments, resulting in a score on a scale from 0 to 1. A score of 1 indicates identical structures, while 0 signifies complete dissimilarity. Despite its effectiveness, this metric operates as a black box, leaving us unaware of the specific calculations performed by GPT or whether it consistently employs the same metric. From various research papers, we've observed that these LLMs tend to produce more unstable results with each iteration (Tian et al., 2023; Liu et al., 2023).

> Given 2 **Java** code paragraphs, please generate a similarity score from 0 to 1 (to three decimal places), by grammar parsing structure. Answer with a format like [[0.777]].
> =====Code 1=====
> [**Java** code snippet 1]
> =====Code 2=====
> [**Java** code snippet 2]
> =====End=====

This prompt above is designed to calculate and return a similarity score between two Java code snippets based on their grammatical structure. The similarity score ranges from 0 to 1, with three decimal places of precision. A score of 1 indicates identical grammatical structures, while a score of 0 indicates completely different structures. The output format [[0.777]] facilitates easy extraction and post-processing of the score.

[2]https://tree-sitter.github.io/tree-sitter/

3 Research Questions and Targets

RQ1: Can TSED be used in more programming languages? We investigate the adaptability of AST Edit Distance which is a generalized version of TSED, exploring its effectiveness in languages like Python and Java to assess its applicability for code similarity analysis.
RQ2: How are TSED and GPT similarity correlated to semantic similarity and execution match? We assess the correlation between these different metrics to understand their respective contributions in evaluating code similarity across multiple programming languages.
RQ3: What are the limits of these metrics? We assess the stability of GPT-based similarity output and analyze how parameters, particularly operation weights (delete, insert, rename), influence TSED.

4 Experiments

4.1 General Setup

In this study, our primary objective is to apply the theoretical framework to a diverse range of programming languages. To achieve this, we aim to identify executable datasets and evaluate them using predefined metrics. The experimental setup comprises two key tasks: firstly, expanding the application of TSED and GPT similarity to additional programming languages, followed by exploring the correlation between these metrics. Subsequently, we seek to assess the stability of GPT scoring and examine the impact of various parameters on the TSED metric. This structured approach allows us to comprehensively investigate the adaptability, correlations, and stability of the chosen metrics across a spectrum of programming languages.

4.2 Evaluation Metrics

- **BLEU Score** is calculated as the geometric mean of the modified precision scores for various n-gram lengths, providing a concise and standardized similarity measurement between the generated and reference text (Papineni et al., 2002).

- **Jaccard Similarity** is a measure of similarity between two sets and is calculated by dividing the size of the intersection of the sets by the size of their union, offering a quantitative assessment of the degree of overlap between the sets' elements.

- **Execution Match** Execution Match pertains to the consistency in execution outcomes between generated code and its corresponding ground truth, evaluating the equivalence in practical functionality. 1 in Execution match means they have the same execution results, and 0 means different.

- **GPT Similarity** mentioned in the Section 2.2

- **TSED** mentioned in the Section 2.1.

4.3 Datasets

Although the execution match metric is infrequently employed in programming code-related datasets, its prominence has increased in recent years. Our comparative analysis involved assessing datasets from various papers, considering factors such as dataset sizes, programming languages, and executables. As highlighted in Table 1, the **MBXP** dataset encompasses 13 different languages, serving as a function-level benchmark that effectively evaluates programming paragraphs. However, the MBXP dataset includes ground-truth solutions for only 7 languages, with C# omitted due to compilation issues. Additionally, we consider the **CoderEval** dataset to facilitate a comparison between Python and Java code generation, leveraging its longer test samples, results are in the appendix.

Table 1: Widely-used code generation benchmarks, selected from GitHub

Benchmark	Language	Samples	Executeable
CoNaLA(Yin et al., 2018)	Python	500	No
Concode(Iyer et al., 2018)	Java	2000	No
MBXP(Athiwaratkun et al., 2022)	Multilingual	974	Yes
InterCode(Yang et al., 2023)	Bash, SQL	200, 1034	Yes
CoderEval(Yu et al., 2024)	Python, Java	230	Yes
RepoEval(Liao et al., 2023)	Python	383	No

In the Bash-Shell scenarios, we reproduce results and conduct a comparative analysis using the **InterCode** dataset. Notably, we identify the SPIDER dataset within InterCode and establish it as a baseline. **SPIDER**, previously evaluated in comparison to the TSED paper, is a substantial human-labeled dataset for the text-to-SQL task. This dataset encompasses databases with intricate join solutions across diverse domains (Yu et al., 2018).

5 Results

5.1 Similarity Results

As we analyze the results presented in Table 2, our experiment demonstrates the effective performance of TSED and GPT similarity in evaluating the MBXP dataset across all 6 programming languages. No instances of parsing or scoring generation failures were observed, confirming the robustness of these metrics across languages.

Table 2: Evaluation Metrics comparison for 6 languages on MBXP dataset, prediction generated by GPT-3.5-Turbo model, ground truth from dataset

Languages	TSED	BLEU	Jaccard Sim	GPT-4	Execution
Java	0.3746	0.2041	0.2733	0.8143	0.6550
Python	0.1888	0.0843	0.2000	0.6751	0.6842
JavaScript	0.2037	0.0846	0.2037	0.6763	0.6811
Typescript	0.1360	0.0637	0.1397	0.5313	0.6642
Ruby	0.1727	0.0438	0.1810	0.7067	0.6428
Kotlin	0.3412	0.1847	0.3109	0.7073	0.5569

RQ1: Can TSED be used in more programming languages?

Answer: The exploration of TSED's adaptability beyond SQL shows promise, especially in languages like Java and Kotlin, indicating its potential for code analysis. TSED proves effective in programming languages with functional parsers, allowing for structural similarity calculation.

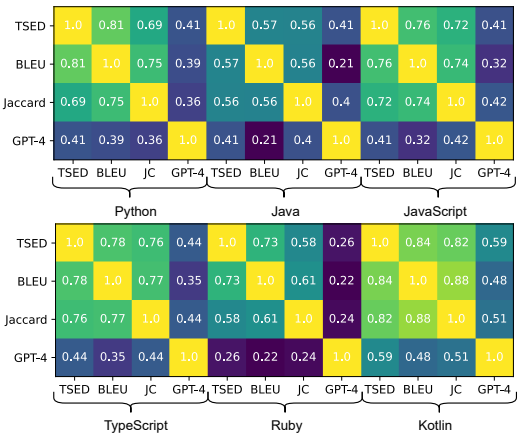

Figure 2: MBXP dataset, Pearson Correlation Heatmap between evaluation-metrics on GPT-3.5

Moreover, TSED shows a commendable correlation ranging from 0.6 to 0.8 with BLEU score and Jaccard similarity, as illustrated in Figure 2. Additionally, TSED exhibits a strong correlation with GPT similarity, especially in Java and Python during the CoderEval test, as depicted in Figure 3, underscoring its sensitivity to code structure. We employ thresholding to establish a prediction-to-execution match. If the metric value exceeds the threshold T, we assign the prediction as 1; otherwise, it is set to 0. The optimal threshold values are determined through enumeration to achieve the best match results. Based on their F1/Accuracy match to the Execution match, both TSED and GPT similarity exhibit higher accuracy compared to semantic metrics in Table 3. Notably, GPT similarity demonstrates a slightly superior F1 score and TSED gives good results on accuracy.

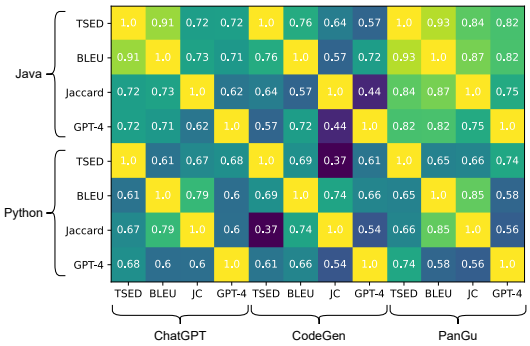

Figure 3: CoderEval Pearson Correlation Heatmap between evaluation-metrics/models/languages

RQ2: How are TSED and GPT similarity correlated to semantic similarity and execution match?

Answer: Our evaluation of TSED metrics, GPT-based similarity, and other semantic evaluation metrics revealed consistently high Pearson correlations between TSED, GPT Score, BLEU Score, and Jaccard Similarity. TSED exhibited notable accuracy in matching with Execution-Match, while GPT score demonstrated the highest F1 score, highlighting their respective strengths in capturing structural and semantic nuances in code across various programming languages.

5.2 Stability of GPT Scoring

To understand how unstable GPT scoring is, we execute the GPT-4 Similarity scoring five times on identical prediction sets, we establish the initial result as a baseline to assess differences through statistical indicators such as Mean Squared Error (MSE) or Mean Absolute Error (MAE) in comparison to the first scoring. Table 4 demonstrates that GPT scoring exhibits limited stability in the context of code similarity evaluation.

5.3 Parameter optimization of TSED

We can configure the penalty weight of 3 operations in tree distance computing: **Delete**, **Insert**, and **Rename**. Figure 4 which is from a test for the MBXP/Java dataset shows is 'Insert' has a sweet spot of 0.8. 'Delete' and 'Rename' operations just keep them in 1.0 penalty weight as the best choice. But we need to keep in mind it can be different in other programming languages.

Table 3: Execution Match F1 score & Accuracy for each thresholding metrics

Languages	TSED			GPT			BLEU			Jaccard		
	Threshold	F1	Acc	Threshold	F1	Acc	Threshold	F1	Acc	Threshold	F1	Acc
Python	0.23	0.5650	0.6057	0.83	0.6403	**0.6735**	0.07	0.5719	0.6150	0.19	0.5907	0.6253
Java	0.10	0.5108	**0.6499**	0.56	0.5693	0.6396	0.03	0.5184	0.5755	0.16	0.5612	0.6018
JavaScript	0.12	0.5494	0.6002	0.69	0.5924	**0.6205**	0.02	0.4964	0.5267	0.12	0.5245	0.5885
Typescript	0.07	0.5367	**0.5822**	0.51	0.5521	0.5708	0.01	0.4987	0.5553	0.08	0.5284	0.5708
Ruby	0.13	0.5045	0.5306	0.54	0.6051	**0.6811**	0.01	0.4375	0.4490	0.12	0.5142	0.5612
Kotlin	0.28	0.6834	**0.6823**	0.8	0.6681	0.6721	0.1	0.6441	0.6457	0.22	0.6387	0.6533

Table 4: Unstable nature of GPT-4 scoring output

Metrics	1st	2nd	3rd	4th
Mean Squared Error	0.0581	0.0583	0.0527	0.0628
Mean Absolute Error	0.1902	0.1940	0.1825	0.1996

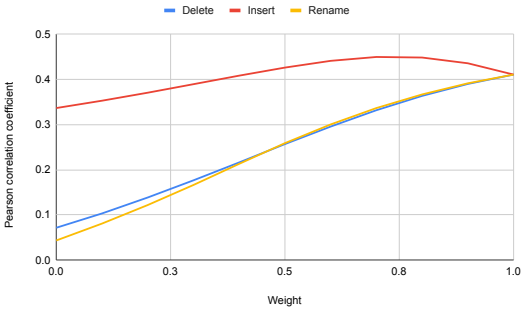

Figure 4: Change each of penalty weight influence correlation to GPT structure similarity score

RQ3: What are the limits of these metrics?

Answer: Penalty weight parameters play influential roles in the TSED metric. Besides, GPT-based similarity metrics offer higher performance at the cost of more money, leading to a bit of unstable output. This underscores the need to carefully balance performance and stability considerations in code similarity assessment across various programming languages.

5.4 Efficiency

The table 5 illustrates the computational time (in ms) required by each programming language tested, including TSED, BLEU score, Jaccard similarity, and GPT 3.5 Score. Our findings indicate that the performance of TSED is comparable to the BLEU score, with significantly lower computational time compared to GPT-3.5. This suggests that TSED is indeed efficient enough to be applied at scale.

6 Conclusion

In this paper, we applied TSED to more programming languages, compared GPT similarity and TSED to semantic metrics, and checked representation to execution match. Then we discuss limitations about the stability of GPT scoring and the penalty parameters of TSED.

Limitations

While our study provides valuable insights into code similarity assessment using TSED and GPT-based metrics, it is essential to acknowledge certain limitations. Firstly, the generalizability of our findings may be influenced by the specific datasets and programming languages employed in our analysis. Additionally, the stability of GPT-based similarity metrics, as highlighted in our results, poses a limitation in terms of consistent and reliable code assessments. Furthermore, variations in the interpretation and definition of similarity metrics across different studies may introduce inherent biases. Lastly, the effectiveness of TSED metrics may be contingent upon the quality of the employed parsers and the fine-tuning of penalty parameters. These limitations underscore the need for caution when extrapolating our results to diverse contexts and emphasize the necessity for further research to address these challenges.

Ethics Statement

Our research adheres to ethical standards, prioritizing integrity and respect for all involved parties. We ensured data privacy, obtained informed consent

Table 5: Average execution time(ms) of metrics and programming languages

	Python	**Java**	**JavaScript**	**TypeScript**	**C#**	**Ruby**	**Kotlin**
TSED	0.0227	0.0645	0.0315	0.0697	0.0373	0.0092	0.0307
BLEU	0.0075	0.0113	0.0155	0.0163	0.0160	0.0116	0.0144
Jaccard	1.6e-5	2.9e-5	1.9e-5	2.4e-5	2.7e-5	1.5e-5	1.8e-5
GPT3.5ß Score	1304	1860	1231	1339	1470	1044	1681

where applicable, and maintained transparency in our methodologies. The study was conducted with the utmost consideration for ethical guidelines and the welfare of participants, upholding the principles of fairness, accountability, and academic integrity throughout the research process.

Acknowledgment

This research was funded in whole, or in part, by the Luxembourg National Research Fund (FNR), grant references NCER22/IS/16570468/NCER-FT and BRIDGES2021/IS/16229163/LuxemBERT. We extend our heartfelt appreciation to our collaborator, BGL BNP PARIBAS, for their invaluable support and special thanks to Saad Ezzini from Lancaster University for his advisory contributions.

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Ben Athiwaratkun, Sanjay Krishna Gouda, Zijian Wang, Xiaopeng Li, Yuchen Tian, Ming Tan, Wasi Uddin Ahmad, Shiqi Wang, Qing Sun, Mingyue Shang, et al. 2022. Multi-lingual evaluation of code generation models. *arXiv preprint arXiv:2210.14868*.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners.

Timothy Clem and Patrick Thomson. 2021. Static analysis at github: An experience report. *Queue*, 19(4):42–67.

Srinivasan Iyer, Ioannis Konstas, Alvin Cheung, and Luke Zettlemoyer. 2018. Mapping language to code in programmatic context. *arXiv preprint arXiv:1808.09588*.

Afshan Latif, Farooque Azam, Muhammad Waseem Anwar, and Amina Zafar. 2023. Comparison of leading language parsers–antlr, javacc, sablecc, tree-sitter, yacc, bison. In *2023 13th International Conference on Software Technology and Engineering (ICSTE)*, pages 7–13. IEEE.

Dianshu Liao, Shidong Pan, Qing Huang, Xiaoxue Ren, Zhenchang Xing, Huan Jin, and Qinying Li. 2023. Context-aware code generation framework for code repositories: Local, global, and third-party library awareness. *arXiv preprint arXiv:2312.05772*.

Xiao Liu, Yanan Zheng, Zhengxiao Du, Ming Ding, Yujie Qian, Zhilin Yang, and Jie Tang. 2023. Gpt understands, too. *AI Open*.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, ACL '02, page 311–318, USA. Association for Computational Linguistics.

Mateusz Pawlik and Nikolaus Augsten. 2015. Efficient computation of the tree edit distance. *ACM Transactions on Database Systems (TODS)*, 40(1):1–40.

Mateusz Pawlik and Nikolaus Augsten. 2016. Tree edit distance: Robust and memory-efficient. *Information Systems*, 56:157–173.

Yewei Song, Saad Ezzini, Xunzhu Tang, Cedric Lothritz, Jacques Klein, Tegawendé Bissyandé, Andrey Boytsov, Ulrick Ble, and Anne Goujon. 2023. Enhancing text-to-sql translation for financial system design. *arXiv preprint arXiv:2312.14725*.

Daniel Tang, Zhenghan Chen, Kisub Kim, Yewei Song, Haoye Tian, Saad Ezzini, Yongfeng Huang, and Jacques Klein Tegawende F Bissyande. 2024. Collaborative agents for software engineering. *arXiv preprint arXiv:2402.02172*.

Haoye Tian, Weiqi Lu, Tsz On Li, Xunzhu Tang, Shing-Chi Cheung, Jacques Klein, and Tegawendé F Bissyandé. 2023. Is chatgpt the ultimate programming assistant–how far is it? *arXiv preprint arXiv:2304.11938*.

Masaru Tomita. 1991. *Generalized LR parsing*. Springer Science & Business Media.

Junjie Wang, Yuchao Huang, Chunyang Chen, Zhe Liu, Song Wang, and Qing Wang. 2023. Software testing with large language model: Survey, landscape, and vision. *arXiv preprint arXiv:2307.07221*.

John Yang, Akshara Prabhakar, Karthik Narasimhan, and Shunyu Yao. 2023. Intercode: Standardizing and benchmarking interactive coding with execution feedback. *arXiv preprint arXiv:2306.14898*.

Pengcheng Yin, Bowen Deng, Edgar Chen, Bogdan Vasilescu, and Graham Neubig. 2018. Learning to mine aligned code and natural language pairs from stack overflow. In *Proceedings of the 15th international conference on mining software repositories*, pages 476–486.

Hao Yu, Bo Shen, Dezhi Ran, Jiaxin Zhang, Qi Zhang, Yuchi Ma, Guangtai Liang, Ying Li, Qianxiang Wang,

and Tao Xie. 2024. Codereval: A benchmark of pragmatic code generation with generative pre-trained models. In *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering*, pages 1–12.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, et al. 2018. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. *arXiv preprint arXiv:1809.08887*.

A Additional Experiment Details

A.1 Parser Comparison

The ANTLR[3] (ANother Tool for Language Recognition) tool, serving as a distinct AST parser compared to tree-sitter, demonstrated notable differences. Following our evaluation using identical settings for TSED metrics, as Figure 5 shows, it became evident that the correlation with other metrics was inferior to the original solutions. This experiment underscores the crucial role of parser performance in the computation procedure, highlighting the significance of selecting an appropriate parser for accurate and reliable code similarity assessments.

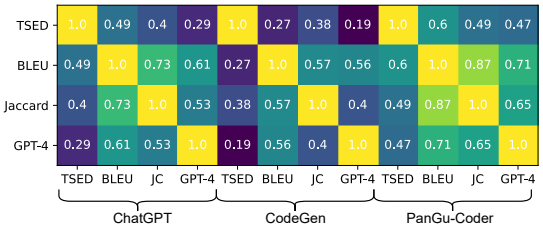

Figure 5: CoderEval Java Pearson Correlation Heatmap between evaluation-metrics/models/languages on TSED with ANTLR parser

A.2 Other experiment results

Due to space constraints, a subset of experimental data is provided in the appendix. A comprehensive evaluation of CoderEval and InterCoder is detailed in Table 6, while specific original sample data from the MBXP dataset is presented in Table 7.

CoderEval, designed for class-level code generation tasks, proves to be a challenging test. Utilizing Pass@10 data as a test sample, TSED demonstrates a robust correlation with semantic indicators in both Java and Python languages. Additionally, a noteworthy correlation is observed between TSED and GPT Similarity.

In the case of InterCoder, we confirm that TSED calculations extend to Bash scripts. Also, the correlation in Figure 6 between TSED to semantic metrics is acceptable, the GPT score doesn't have a good correlation to others. We also replicate the performance of the SPIDER dataset, noting differences from the original paper but not to a significant extent.

Despite the notably low semantic similarity between the MBXP built-in samples and the ground

[3]https://www.antlr.org/

truth, a relatively high execution match is observed. We acknowledge this disparity and plan to address it through optimization in future research endeavors.

Table 6: 4 Evaluation Metrics compared to Ground Truth on CoderEval(Java&Python) / InterCode(Bash) / SPIDER(SQL)

Languages	Model	TSED	BLEU	Jaccard Sim	GPT-4	Execution
Java	ChatGPT	0.4971	0.3655	0.3384	0.7392	0.3539
	CodeGen	0.3616	0.2871	0.2506	0.6603	0.1391
	PanGu	0.5029	0.3722	0.3849	0.6778	0.2543
Python	ChatGPT	0.2840	0.1285	0.1763	0.5883	0.2104
	CodeGen	0.2703	0.1778	0.1821	0.5604	0.0948
	PanGu	0.2829	0.0868	0.1567	0.5086	0.1183
Shell	GPT-4	0.5853	0.2816	0.3567	0.8511	0.4851
	starchat	0.4065	0.1594	0.2081	0.6740	0.2374
	vicuna	0.4755	0.1621	0.2295	0.7164	0.2451
SQL	ChatGPT-3.5	0.6824	0.3304	0.3710	0.9461	0.6482
	nsql-6B	0.8022	0.4493	0.4356	0.9265	0.5483
	RESDSQL	0.7422	0.2084	0.1868	0.9629	0.7756

Table 7: 4 Evaluation Metrics compare to Ground Truth on 7 languages MBXP Dataset Samples

Languages	TSED	BLEU	Jaccard Sim	GPT-4	Execution
Java	0.2218	0.1046	0.1960	0.4248	0.853
Python	0.1550	0.0255	0.1222	0.3396	0.822
JavaScript	0.1870	0.0573	0.1685	0.4005	0.786
Typescript	0.1186	0.0288	0.1260	0.4247	0.872
Ruby	0.2073	0.0235	0.1796	0.4830	0.589
Kotlin	0.1720	0.0336	0.1877	0.3976	0.637

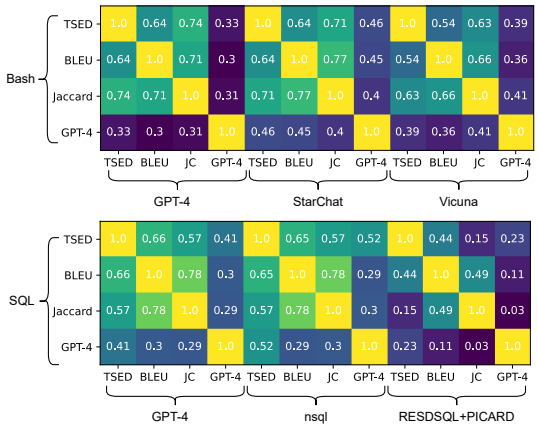

Figure 6: InterCode/SPIDER Pearson Correlation Heatmap between evaluation-metrics/models/languages

B Case Studies

B.1 A. Low BLEU, but high TSED

```
### Code Paragraph 1
int result = 0;
        for(int i = 0; i < n; i++) {
            result = n * (7 * n - 5) / 2;
}
        return result;
    }
}
### Code Paragraph 2
int jacobsthalNumber = 1;
        for(int i = 2; i <= n; i++){
            jacobsthalNumber =
                ↪ jacobsthalNumber + (n
                ↪ - i) * (i - 1);
        }
        return jacobsthalNumber;
    }
}
```

In the provided code snippets, both segments involve loops for performing calculations, which contributes to their high structural similarity. However, the semantic similarity is relatively low due to the significant disparity in variable names, which occupy a considerable portion of the tokens. Despite the differences in semantics, the BLEU score, a metric commonly used for evaluating text similarity, yields a score of 0.359, indicating some level of similarity. In contrast, the Tree Similarity of Edit Distance (TSED) metric, which accounts for structural differences, produces a higher score of 0.8, highlighting the effectiveness of TSED in capturing structural similarities even when semantic differences exist.

B.2 BLEU and TSED similar

```
### Code Paragraph 1
def max_of_two(a, b):
    if a > b:
        return a
    else:
        return b
### Code Paragraph 2
def max_of_two(a, b):
    return max(a, b)
```

Both the BLEU score (0.408) and the TSED (0.444) score suggest that while the two code snippets achieve somehow similar functionality, they do so using different structural approaches.

Token Similarity: Despite structural differences, the two snippets share many common tokens, leading to a moderate BLEU score. This reflects their similar semantic intent and consistent use of basic programming constructs.

Structural Difference: The TSED score highlights the structural changes. The first snippet uses

explicit control flow statements (if-else), while the second snippet uses a built-in function (max). This difference is significant enough to lower the structure similarity score but not so drastic as to suggest entirely different functionalities.

Overall, these scores indicate that the snippets are functionally similar (performing the same task) with some commonality in token usage but exhibit distinct structural approaches in their implementations. This case illustrates how different coding styles and structures can be used to achieve the same outcome in programming.

B.3 Let ChatGPT explain in detail

> **Prompt:** Given 2 Java code paragraphs, please generate a similarity score from 0 to 1 (to three decimal places), by grammar parsing structure. Answer with a format like [[0.777]]. And give us 3 dimensions of similarity in integer score 1 5, which is "functional", "tree structure", and "semantical"; Then give the reason of the score you think if it's high or low.

```
int result = 0;
        for(int i = 0; i < n; i++) {
            result = n * (7 * n - 5) / 2;
}
        return result;
    }
}
int jacobsthalNumber = 1;
        for(int i = 2; i <= n; i++){
            jacobsthalNumber =
                ↪ jacobsthalNumber + (n
                ↪ - i) * (i - 1);
        }
        return jacobsthalNumber;
    }
}
```

Here is the similarity score and the three dimensions of similarity for the given code snippets: [[0.875]] Functional Similarity: 4 Tree Structure Similarity: 5 Semantical Similarity: 3 Explanation:

- Functional Similarity (4): Both code snippets perform calculations within a loop structure, albeit with different mathematical operations. They both iterate over the variable i to calculate a result based on the value of n.
- Tree Structure Similarity (5): The overall structure of both code snippets is highly similar, with both containing a loop structure followed by a calculation and a return statement.
- Semantical Similarity (3): While the overall structure is similar, the semantics differ significantly. The variable names (result vs. jacobsthalNumber) and the mathematical operations performed within the loop differ, leading to a moderate level of semantic dissimilarity.