Beyond Lexical Similarity: A Benchmark for Evaluating Code Documentation Agents

Anonymous Authors[1]

Abstract

The rise of automated documentation generation tools has created a critical need for objective evaluation methods. Current approaches to assessing documentation quality include lexical similarity metrics (e.g., BLEU) that compare generated text to human-written references, as well as more recent multi-faceted metrics that evaluate structural completeness or helpfulness. However, these methods do not directly measure whether documentation enables accurate code synthesis - a key aspect of its practical utility. This paper introduces a novel benchmark that overcomes this limitation by assessing documentation based on its utility in a downstream doc-to-code task. We propose a benchmark that automatically evaluates the quality of generated documentation at a repository level by assessing its utility for code regeneration using associated unit tests. To validate the proposed benchmark, we evaluate documentation produced by popular open-source agent systems. Our results show that the benchmark evaluates documentation quality differently from standard reference-based and reference-free metrics. The code of benchmark is available here: https://doi.org/10.5281/zenodo.19253895

1. Introduction

Automated documentation generation has become an important task in modern software engineering workflows, driving advances in both developer productivity and code maintainability. As large language models (LLMs) and agent-based systems increasingly participate in code authoring and comprehension processes, the quality of automatically generated documentation directly affects downstream performance in code synthesis and understanding tasks. However, the evaluation of documentation quality remains an open challenge. Existing evaluation methods for generated documentation can be broadly categorized into two types. The first comprises lexical similarity metrics, such as BLEU (Papineni et al., 2002), ROUGE (Lin, 2004), CodeBLEU (Ren et al., 2020), and BERTScore (Zhang et al., 2019), that compare generated documentation against human-written references. While widely used, these metrics are limited in their ability to reflect the practical utility of documentation for program synthesis (Naik, 2024). The second category includes more recent multi-faceted evaluation metrics, such as those introduced in the DocAgent framework (Yang et al., 2025), which assess documentation across dimensions including completeness (structural presence of required sections), helpfulness (practical utility and clarity via LLM-as-judge), and truthfulness (verification of repository-specific entity mentions). These metrics provide richer assessments than lexical overlap alone but remain proxy measures: they evaluate structural or stylistic properties rather than directly measuring whether documentation enables a downstream system to correctly implement the documented functionality.

We introduce a novel benchmark that shifts evaluation from lexical comparison to functional adequacy. Rather than measuring textual resemblance to a reference, our benchmark evaluates whether documentation enables an independent LLM to regenerate functionally correct code that passes the repository's unit tests. The benchmark operates in two stages: (1) an agent system generates function-level docstrings across a repository; (2) a fixed evaluator LLM uses each docstring with file-level context to regenerate the function body, which is executed against unit tests and scored via pass@k.

This test-based evaluation captures informational completeness and clarity as perceived by an autonomous coding agent, rather than stylistic or narrative similarity to human references. Unlike reference-based methods, our approach admits that AI-generated documentation can functionally outperform human-written baselines, particularly when measured by downstream code generation success rather than subjective readability. By extending evaluation to repository scope, the benchmark enables large-scale, cross-domain

[1]Anonymous Institution, Anonymous City, Anonymous Region, Anonymous Country. Correspondence to: Anonymous Author <anon.email@domain.com>.

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

comparison of documentation systems in realistic settings where AI agents operate as both consumers and producers of technical content.

This work makes the following **main contributions**:

- We introduce a new benchmark for evaluating repository-level documentation generation based on functional correctness in a downstream doc-to-code task.
- We propose a novel reference-free repository-level metric that measures the utility of documentation through unit test success, providing a practical alternative to lexical similarity metrics.
- We provide empirical evidence that this metric captures quality distinctions between documentation systems that traditional lexical and heuristic methods fail to detect.

2. Related Works

2.1. Documentation generation agents and frameworks

Automated documentation generation has been a prominent topic in recent years due to the significant amount of developer working hours that can be saved through subsequent tools. For example, DocuMint (Poudel et al., 2024) analyzes the usage of small language models in the area of docstring generation, with results showing potential for lightweight model usage, but highlighting gaps in consistency compared to human-written references. Recently, RepoAgent (Luo et al., 2024) has been proposed, capable of operating on repository scale, integrating code, comments, and structural metadata to assess and improve documentation quality in context. DocAgent (Yang et al., 2025) is another multi-agent framework that goes beyond single-shot generation, using code processing in a topological order through a specialized agent, which precedes generation, refinement, and verification. Current research shows emerging trends moving from one-shot documentation generation to context-aware, flexible, and systematic solutions.

2.2. Metrics for generated docs

The use of linguistic similarity metrics such as BLEU, METEOR, ROUGE-L, CIDEr or SPICE for docstring evaluation usually leads to unsatisfactory results, because they correlate poorly with human evaluations (Hu et al., 2022; Evtikhiev et al., 2023). For this reason, the evaluation of generated docstrings has transitioned from these traditional metrics, to models such as CoCC (Huang et al., 2024) which detect consistency and alignment between code logic and its docstring, CIDRe (Dziuba & Malykh, 2025) and SIDE (Mastropaolo et al., 2023) which are able to compute complex characteristics without an additional reference with multi-language support, and round-trip evaluation (Allamanis et al., 2024) methods which measure completeness by generating code from docstring and checking its validity via unit tests.

While round-trip evaluation methods establish a promising direction by measuring completeness through code regeneration from documentation, they differ from our benchmark in one aspect. Existing round-trip approaches (e.g., RTC (Allamanis et al., 2024)) operate on isolated code regions or functions without incorporating cross-file repository context (imports, class hierarchies, inter-module dependencies), which is precisely the context that distinguishes high-quality repository-level documentation from function-level summaries.

Recent works also explore LLM-as-Judge approaches, where large language models serve as automated evaluators of documentation quality by scoring generated texts along dimensions such as accuracy, consistency, completeness, and clarity (Dasgupta & Shankar, 2025; Zhu et al., 2023; Mao et al., 2023; Zheng et al., 2023). These systems have demonstrated near-human performance on enterprise documents (Dasgupta & Shankar, 2025) and moderate correlation with human judgment for code documentation (Hu et al., 2022), although reproducibility and potential evaluator bias remain ongoing challenges.

In the field of code generation, for instance, the benchmark c-CRAB (Zhang et al., 2026) evaluates LLM-based agents on code review tasks through the automated generation of unit tests. However, it does not analyse for the quality of code documentation.

3. Proposed benchmark

3.1. Problem Statement

Existing evaluation methods for generated documentation serve different purposes but share a common limitation: they do not directly measure whether documentation enables *accurate code reconstruction*. We address this gap by defining documentation quality in terms of functional utility.

Definition. **Good documentation** for a function f_i is a text that, when provided together with file-level context C_i to a fixed evaluator LLM M, enables regeneration of a function body that passes all associated unit tests T_i.

Notation. Let R be a repository with target functions $F_R = \{f_i\}_{i=1}^{N_R}$. A documentation agent A produces $d_i = A(f_i, C_i)$, where C_i is the file-level context (imports, signature, class scope). The evaluator M generates m candidate implementations $\hat{f}_{i,j} = M(d_i, C_i; \xi_j)$, $j = 1, \ldots, m$,

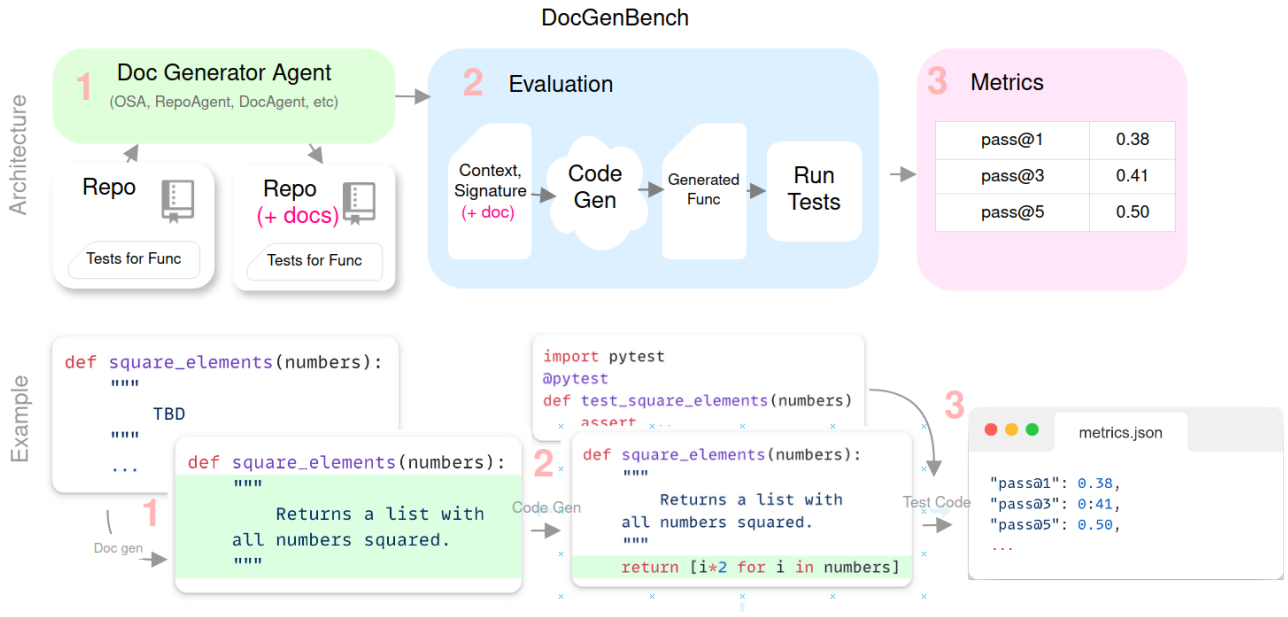

Figure 1. Benchmark workflow: (1) **Doc Generator Agent**: generates function-level documentation (docstrings) for the repository. (2) **Evaluation**: an independent evaluator uses the generated docstring + file context to regenerate the function body, which is then executed against unit tests. (3) **Metrics**: documentation quality is quantified via `pass@k` based on test pass rates.

where ξ_j denotes sampling randomness. We define:

$$\text{score}(\hat{f}_{i,j}) = \mathbb{I}\big[\hat{f}_{i,j} \text{ passes all tests in } T_i\big],$$
$$c_i = \sum_{j=1}^{m} \text{score}(\hat{f}_{i,j}).$$

Documentation quality is quantified by the unbiased pass@k estimator (Chen, 2021):

$$\text{pass@}k(f_i) = 1 - \frac{\binom{m-c_i}{k}}{\binom{m}{k}}. \quad (1)$$

3.2. Benchmark Pipeline

Figure 1 illustrates the three-stage workflow.

3.2.1. Stage 1: Documentation Generation

An agent framework (e.g., RepoAgent, OSA) traverses the repository and emits a docstring d_i for each target function f_i, utilizing available repository context (e.g., cross-file dependencies). The benchmark treats the agent as a black box.

3.2.2. Stage 2: Code Regeneration and Testing

The fixed evaluator M receives (d_i, C_i) and produces m candidate function bodies. Each candidate is integrated into the codebase and executed against the unit-test suite T_i in a sandboxed environment. Binary outcomes are recorded per the notation in Section 3.1.

3.2.3. Stage 3: Metric Computation

Repository-level pass@k is computed from per-function scores using Equation 1.

3.3. Dataset Construction

The benchmark comprises 2,924 tasks drawn from 28 Python repositories of varying size, including 10 popular projects (>100 GitHub stars) and 18 randomly sampled repositories. Full dataset statistics are provided in Appendix A.3.

3.3.1. Task Construction Pipeline

The construction pipeline consists of the following steps:

1. **Repository cloning and build.** Each candidate repository is cloned and built in a Dockerized environment.
2. **Test suite filtering.** Only repositories whose original test suites pass with per-function execution time < 10 s are retained.
3. **Target function selection.** Tested functions are identified via a two-step procedure: (a) all test files in the `tests/` directory are discovered and functions executed by `pytest` are listed; (b) dynamic tracing during test execution pinpoints which repository functions are actually invoked.
4. **Mock implementation filtering.** For each traced func-

tion, the body is replaced with a minimal stub (`pass`, or a type-appropriate default). The full test suite is rerun; only functions whose associated tests *fail* with the stub but *pass* with the original implementation are retained. This two-sided check serves a dual purpose: it eliminates *vacuous tests* (which pass regardless of the implementation) and ensures that each retained task carries a non-trivial functional contract that the test suite actively verifies. As a result, pass@k scores on retained tasks reflect genuine behavioral requirements rather than test-suite artifacts.

5. **Task formulation.** The file-level context C_i (imports, class scope, signature) and the existing docstring form the input; the original function body is the target. During evaluation, agent-generated documentation replaces the original docstring, and regenerated bodies are tested via `pytest`.

The entire pipeline is fully automated through the open-source `repotest` library, ensuring reproducible execution limits and standardized logging.

3.4. Design Decisions and Scope

Code leakage. A potential concern is that generated docstrings may trivially embed the target code, reducing regeneration to copy-paste. Two observations argue against this. First, the correlation between CodeBERTScore(code, doc) and pass@k is below 0.12 across all configurations (Figure 2), indicating that high-scoring docstrings do not systematically resemble the target code. Second, we ranked all tasks by pass@k, manually inspected the top-10 highest-scoring tasks for each agent, and found no instances of verbatim or near-verbatim code embedded in the generated docstrings.

Metric scope. The metric is reference-free and measures functional utility as perceived by an LLM performing code synthesis. Style, tone, and human readability are explicitly out of scope.

Unit tests as a proxy for functional correctness. A fundamental assumption of the benchmark is that unit tests T_i adequately capture the behavioral contract of each target function. We acknowledge that unit tests are an imperfect proxy: weak tests may accept degraded implementations, while overly strict tests may reject correct but stylistically different solutions. Three design choices mitigate this risk. First, the mock-filtering step (Section 3.3) guarantees that every retained task has at least one test that *discriminates* between a trivial stub and the reference implementation, eliminating vacuous test suites by construction. Second, the task construction pipeline retains only functions that are *directly invoked* during test execution (verified via dynamic tracing), ensuring that the test-to-function mapping is concrete rather than incidental. Third, we manually inspected a random sample of 50 retained tasks and confirmed that in all cases the associated tests exercise the core input–output contract of the function (e.g., return values, side effects, exception behavior) rather than implementation-internal details such as variable names or call order. While computing formal coverage metrics (statement or branch coverage per function) would further strengthen this argument, the combination of mock filtering, dynamic tracing, and manual inspection provides reasonable confidence that pass@k scores reflect meaningful functional requirements. Extending the benchmark with mutation testing or property-based test generation is a promising direction for future work.

4. Experimental studies

This section presents a comprehensive empirical evaluation of the proposed benchmark on the task of automated repository-level documentation generation.

4.1. Documentation Methods Compared

The experimental study focuses on benchmarking and generating docstrings for open-source LLM-based agents OSA (Nikitin et al., 2025) and RepoAgent (Luo et al., 2024) using 2924 tasks from 28 repositories of varying sizes. For both tools, we developed scripts to iteratively process each repository in the dataset. The total cost for generating docstrings across all repositories using OSA and RepoAgent was less than $100. In contrast, the more expensive DocAgent framework required approximately the same budget to generate docstrings for a single repository (e.g. cryptography-suite), which is why experiments with DocAgent were not conducted. Furthermore, computing the pass@k metrics from our proposed benchmark cost less than $10, making it roughly ten times cheaper than calculating the multi-faceted metrics of Completeness, Helpfulness, and Truthfulness described in Section 4.2.2.

4.1.1. OSA

OSA is a multi-agent open-source tool for improving repository quality (Nikitin et al., 2025). It generates READMEs, documentation, CI/CD scripts, and a report analyzing the strengths and weaknesses of a repository. To speed up the process, it employs asynchronous documentation generation. The tool follows a multi-stage refinement process: it first generates initial docstrings for functions, methods, and classes, then generates a high-level "main idea" summary of the repository, and finally uses this summary to refine and update the previously generated docstrings for improved consistency and quality. Because OSA is a complex tool, its command-line interface (CLI) was configured with appropriate parameters to generate only the docstrings.

4.1.2. REPOAGENT

RepoAgent is an open-source tool designed to generate repository-level documentation. It analyzes repository contents at various levels of abstraction — including the project tree, code structure, and topology — to produce high-quality, understandable, and complete documentation (Luo et al., 2024). The generated documentation is produced in Markdown format. To integrate this output as docstrings directly into the source code, we developed a dedicated conversion script, which is included in the replication package. The correctness of the insertion was verified using the standard Python `compileall` command, ensuring that all modified files remained syntactically valid. The documentation generation language was set to English.

4.2. Evaluation Metrics

4.2.1. LINGUISTIC SIMILARITY METRICS

To analyze the possible correlation with the proposed metrics, classical metrics of linguistic similarity were calculated between the original and generated docstring pairs.

BLEU (Bilingual Evaluation Understudy) is a metric that compares a candidate to references via modified n-gram precision with a brevity penalty (Papineni et al., 2002).

ROUGE1 is a summarization metric measuring unigram overlap between a candidate summary and reference summaries; commonly reported as recall (optionally with precision and F1) (Lin, 2004).

ROUGE-Lsum is a summarization metric based on the longest common subsequence (LCS), computed at summary level by aggregating sentence-level LCS matches (no double-counting of candidate tokens across sentences). Reported as recall, precision, and F-measure (Lin, 2004).

BERTScore is a token-level similarity via contextual embeddings (Zhang et al., 2019).

CodeBERTScore is a token-level similarity metric for code generation that matches candidate and reference code tokens via contextual embeddings from pretrained code models and can additionally encode the natural-language input context (Zhou et al., 2023).

4.2.2. ROBUST MULTI-FACETED EVALUATION METRICS

Beyond lexical similarity, we also report metrics introduced in the DocAgent paper covering three dimensions: **Completeness** (AST- and pattern-based structural check that required sections such as Summary/Args/Returns/Raises/Examples are present when implied by the code), **Helpfulness** (LLM-as-judge rubric score of practical utility/clarity and non-trivial guidance), and **Truthfulness** (grounding via verifying repository-specific entity mentions against a repository dependency graph, e.g., Existence Ratio)(Yang et al., 2025).

5. Results

Table 1 reports execution-based documentation utility measured by pass@1 and pass@3 under two fixed evaluator models (gpt-5-mini and qwen3-235b-a22b), both with default hyperparameters (available in Appendix A.1). The same two models were used as code generators during evaluation, also with default hyperparameters. The evaluation prompt is presented in Appendix A.2. The choice of default hyperparameters is deliberate: since our goal is to evaluate documentation quality rather than to optimize code generation performance, fixing hyperparameters at their default values eliminates any confound that could arise from tuning the evaluator to a specific agent's output style.

Across both evaluators, agent-generated documentation substantially improves downstream functional reconstruction relative to the default docstrings, indicating that the benchmark is sensitive to documentation quality rather than merely textual overlap. In particular, RepoAgent consistently yields the strongest performance (best pass@1 and pass@3 in most settings), and the gap between RepoAgent and OSA is large, demonstrating that the benchmark can reliably distinguish documentation-generation frameworks. Finally, we observe non-trivial interactions between the documentation generator and the evaluator model (e.g., RepoAgent+gpt-5-mini reaches 0.83 pass@1 under the qwen evaluator), suggesting that the benchmark captures robustness across different code-synthesis backends.

Notably, the ranking of documentation systems remains stable regardless of which evaluator model is used: both gpt-5-mini and qwen3-235b-a22b assign the same relative ordering to OSA and RepoAgent (Figure 4 for the visualization). This consistency suggests that the benchmark's discriminative power is not an artifact of evaluator bias. Although it is well established that models tend to favor outputs from their own family (Wataoka et al., 2024), no such self-preference effect is apparent here: neither gpt-5-mini nor qwen3-235b-a22b shows a systematic advantage for the documentation generated by the model of the same family, indicating that pass@k scores reflect genuine functional utility rather than stylistic alignment with the evaluator.

To assess whether benchmark scores are driven by data leakage through pre-training memorization, we stratify results by repository popularity. Table 3 in the Appendix reports per-repository pass@k for all configurations. We split the 28 repositories into popular ($>$100 GitHub stars; 10 repos) and non-popular ($\leq$100 stars; 18 repos) subsets. The ranking of documentation systems (RepoAgent $>$ OSA $>$ default) is preserved in both subsets, and the absolute pass@k differences between subsets are small (within 0.05 on average),

Table 1. pass@1, pass@3 Evaluation metrics across dataset samples. gpt is for gpt-5-mini evaluation model, qwen is for qwen3-235b-a22b evaluation model. Default is for initial docstrings.

doc_system	doc_model	pass@1(gpt)	pass@1(qwen)	pass@3(gpt)	pass@3(qwen)
OSA	gpt-5-mini	0.73	0.69	0.8	0.72
	qwen3-235b-a22b	0.63	0.58	0.72	0.62
RepoAgent	gpt-5-mini	**0.75**	**0.83**	**0.89**	**0.86**
	qwen3-235b-a22b	0.74	0.74	0.84	0.77
default	-	0.55	0.48	0.64	0.52

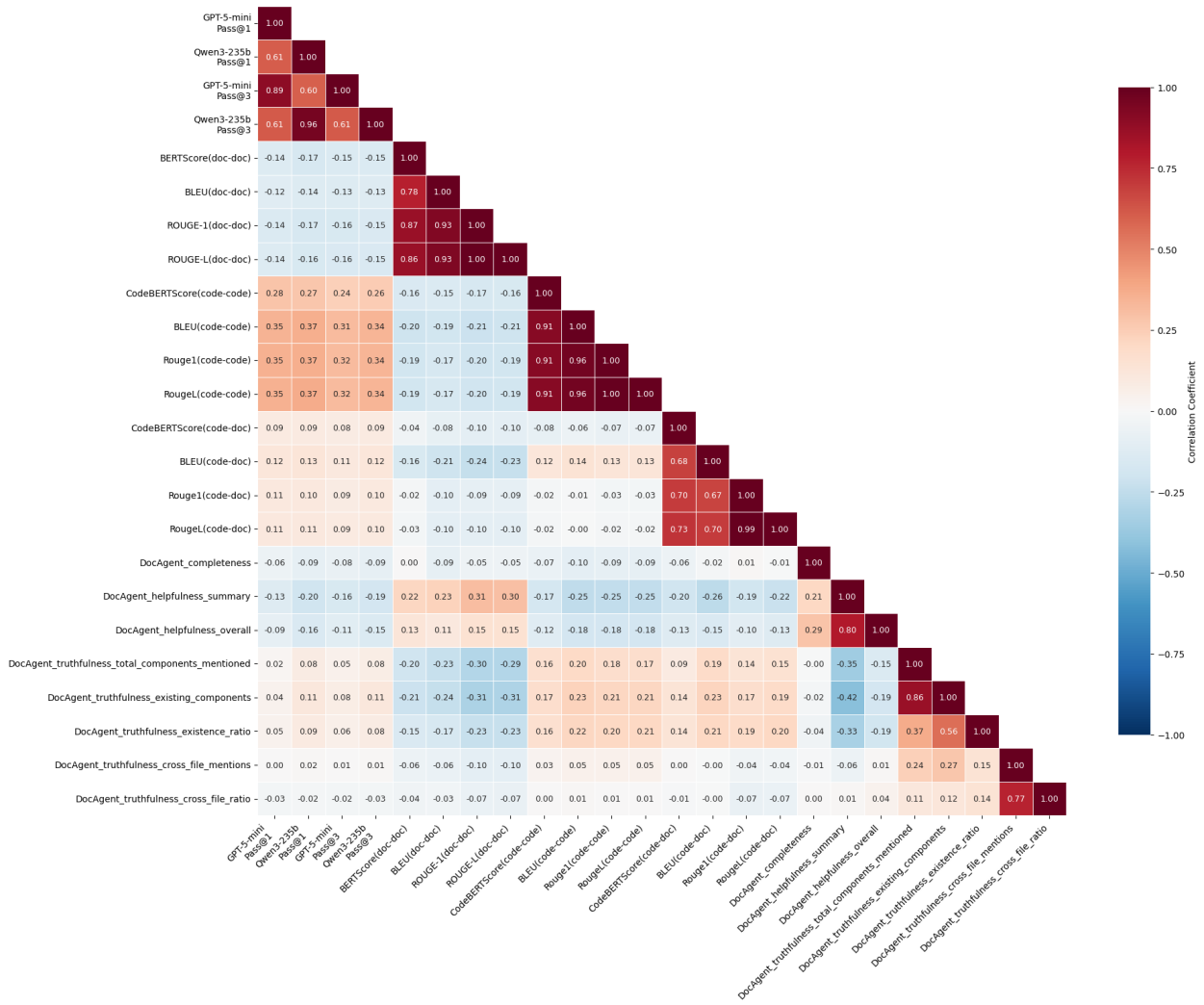

Figure 2. Correlation Matrix between Functional Utility and Reference-Based Documentation Metrics.

suggesting that the metric's discriminative power is not an artifact of the evaluator having memorized popular codebases during pre-training. Per-repository variance remains substantial (interquartile range of 0.15–0.25 across repositories), confirming that documentation quality interacts with repository-specific factors such as code complexity and API surface; however, the *relative* ordering of systems is stable across the vast majority of repositories.

Figure 3 presents biplots for metrics. Figure 2 presents correlations between pass@k, linguistic similarity metrics and multi-faceted evaluation metrics. Doc-doc implies for comparison between original and generated docstring, code-

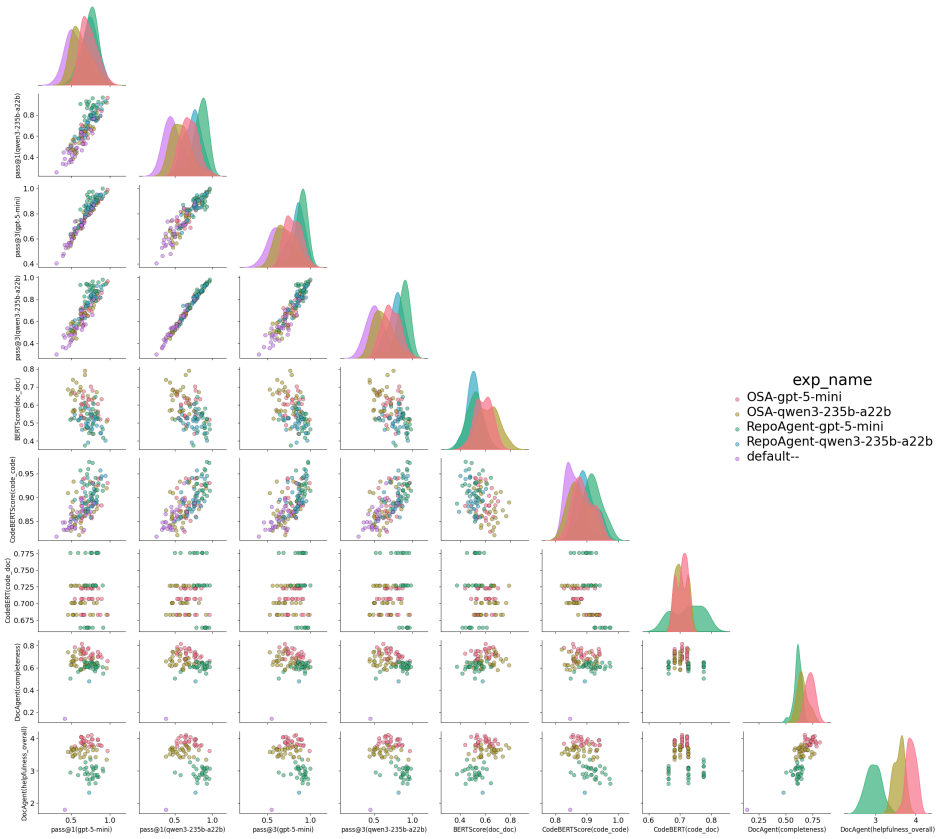

Figure 3. Scatter plots showing pairwise metric correlations across repositories. The analysis reveals strong positive correlations among all pass@1 and pass@3 metrics, independent of the evaluator model. Weak negative correlations are observed between pass@1 and BERT(doc_to_doc). This likely occurs when pass@1 without documentation is zero and the document remains unchanged (BERT(doc_doc) $\approx$ 1), suggesting that pass@1 with changed documentation will also be zero.

code implies for comparison between original code and code generated by the evaluation model, code-doc implies for comparison between original code and generated doc.

The observed weak negative correlation (from -0.17 to -0.12) between linguistic similarity doc-doc metrics and pass@k metrics suggests that higher textual similarity to the original documentation does not correspond to improved functional utility in downstream code regeneration, which is a finding contrary to conventional assumptions underlying linguistic similarity metrics (including BLEU, ROUGE, and BERTScore). Instead, these metrics may be poorly aligned with practical documentation quality. The weak (from -0.20 to 0.11) correlation between DocAgent metrics and pass@k metrics also indicate that heuristic-based and LLM-as-judge metrics do not correspond to improved documentation quality.

We hypothesize that, in the context of LLM-based agents, greater divergence from the original reference can be associated with better test-suite performance, further motivating the need for functional, outcome-based evaluation. We can see that the ranking ability of the proposed metrics does not depend on the task evaluation model.

We report both pass@1 and pass@3 to assess the robustness of execution-based evaluation to sampling randomness. Since code generation is stochastic, single-sample performance (pass@1) can exhibit higher variance; measuring pass@3 provides a small-budget repeated-sampling estimate and allows us to verify that the observed trends and metric-induced rankings remain stable across reruns rather than fluctuating due to random sampling.

6. Limitations

Dependence on Test Quality. The metric assumes that unit tests T_i capture the functional requirements of each target function. Although mock-based filtering removes vacuous tests and manual inspection confirms behavioral relevance for a sample of tasks, residual risks remain. Tests with narrow input coverage may inflate pass@k by accepting implementations that handle only the tested cases. Conversely, tests that assert implementation-specific details (e.g., exact internal state or string formatting) may penalize func-

tionally correct alternatives. Quantitative coverage analysis (e.g., per-function statement or branch coverage) is not included in the current version of the benchmark; adding such analysis would enable stratified evaluation across coverage levels and further validate the metric's reliability. We leave this extension to future work.

Evaluator Dependence. The evaluation relies on a fixed LLM M and a fixed prompt template. Absolute scores and difficulty may shift with stronger or weaker M, even if relative ordering across agents remains similar. This introduces evaluator-induced bias that should be acknowledged when comparing results across different evaluator versions.

Scope: Style and Readability. The metric measures functional utility rather than human-centric qualities (style, tone, pedagogy, narrative flow). Documentation that scores highly may still be suboptimal for human readers. Conversely, excellent human-oriented documentation may not maximize the metric if it prioritizes exposition over operational specificity.

Edge Cases. A known edge case occurs when documentation includes or paraphrases the target code, reducing the task to trivial reconstruction. Empirical analysis suggests this is not a significant factor in practice (Section 3.4); however, the benchmark does not include an automated filter for code-in-docstring detection, which remains a direction for future work.

Context Boundary. Results depend on what is allowed in the "file context" C_i (e.g., imports, signature, class scope). Expanding or shrinking C_i may change the difficulty and alter pass rates. Ambiguity in C_i definition can lead to inconsistencies; thus, a clear, fixed policy for C_i is required for comparability.

Repository Coverage and Representativeness. Benchmark outcomes depend on the selection of repositories, languages, and domains. Skewed or narrow coverage can bias conclusions about general documentation quality. Extending the corpus and reporting per-domain breakdowns can mitigate this limitation.

Human vs. AI Orientation. The metric emphasizes usefulness for an AI consumer performing code synthesis. Documentation beneficial to human understanding (examples, rationale, historical notes) may be undervalued. Conversely, terse, implementation-oriented text may overperform for AI while offering less human readability.

Aggregation Choices. Aggregation from function to file to repository (and across repositories) assumes uniform weighting by default. Alternative weightings (e.g., by number of tests or estimated difficulty) can change aggregate scores. Reported rankings may shift under different, yet reasonable, aggregation policies.

7. Discussion

The proposed benchmark addresses a gap between how documentation is traditionally evaluated (lexical overlap with a reference) and how it is consumed in practice (as input to an LLM performing code synthesis). The weak correlation between pass@k and both lexical metrics and multi-faceted heuristic scores (Section 5) confirms that these two perspectives measure fundamentally different aspects of documentation quality. A practical implication is that documentation agents can be improved by optimizing directly for functional utility rather than reference similarity. The benchmark's pass@k signal could serve as a reward in self-improvement pipelines, enabling agents to iteratively refine docstrings via doc-to-code feedback without requiring human annotation. While our metric does not assess human readability, it is complementary to style-oriented evaluations. Future work could combine functional utility with readability scores to provide a holistic assessment covering both AI and human consumers of documentation.

8. Conclusion

In this work, we introduced the first repository-level benchmark for evaluating documentation quality through functional correctness in a downstream doc-to-code task. The proposed reference-free pass@k metric provides a complementary evaluation dimension that enables objective comparison of agent-based systems based on the practical utility of their generated documentation. Our results demonstrate that this functional approach identifies quality distinctions between documentation generation methods that lexical similarity metrics and multi-faceted evaluation metrics cannot capture, providing a robust and reproducible framework for future research in automated documentation generation.

The primary strength of the proposed approach is that it directly measures whether generated documentation enables functionally correct code reconstruction, providing an operationally grounded assessment complementary to existing lexical and heuristic metrics.

9. Data Availability Statement

The replication package for this paper is publicly available at https://anonymous.4open.science/r/DocGenBench-6365/ and by DOI https://doi.org/10.5281/zenodo.19253895. All scripts necessary to reproduce the reported results are included.

Impact Statement

This paper presents work whose goal is to advance the field of Machine Learning. There are many potential societal consequences of our work, none which we feel must be specifically highlighted here.

References

Allamanis, M., Panthaplackel, S., and Yin, P. Unsupervised evaluation of code llms with round-trip correctness, 2024. URL https://arxiv.org/abs/2402.08699.

Chen, M. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

Dasgupta, S. and Shankar, H. Ai agents-as-judge: Automated assessment of accuracy, consistency, completeness and clarity for enterprise documents. *arXiv preprint arXiv:2506.22485*, 2025.

Dziuba, M. and Malykh, V. Cidre: A reference-free multi-aspect criterion for code comment quality measurement, 2025. URL https://arxiv.org/abs/2505.19757.

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

Hu, X., Chen, Q., Wang, H., Xia, X., Lo, D., and Zimmermann, T. Correlating automated and human evaluation of code documentation generation quality. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 31(4):1–28, 2022.

Huang, Y., Chen, Y., Chen, X., and Zhou, X. Are your comments outdated? towards automatically detecting code-comment consistency, 2024. URL https://arxiv.org/abs/2403.00251.

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

Luo, Q., Ye, Y., Liang, S., Zhang, Z., Qin, Y., Lu, Y., Wu, Y., Cong, X., Lin, Y., Zhang, Y., Che, X., Liu, Z., and Sun, M. Repoagent: An llm-powered open-source framework for repository-level code documentation generation, 2024.

Mao, R., Chen, G., Zhang, X., Guerin, F., and Cambria, E. Gpteval: A survey on assessments of chatgpt and gpt-4. *arXiv preprint arXiv:2308.12488*, 2023.

Mastropaolo, A., Ciniselli, M., Penta, M. D., and Bavota, G. Evaluating code summarization techniques: A new metric and an empirical characterization, 2023. URL https://arxiv.org/abs/2312.15475.

Naik, A. On the limitations of embedding based methods for measuring functional correctness for code generation. *arXiv preprint arXiv:2405.01580*, 2024.

Nikitin, N., Getmanov, A., Popov, Z., Alekseevna, U. E., Aksenkin, Y., Sokolov, I., and Boukhanovsky, A. An LLM-powered tool for enhancing scientific open-source repositories. In *Championing Open-source DEvelopment in ML Workshop @ ICML25*, 2025. URL https://openreview.net/forum?id=lrWseP67ab.

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

Poudel, B., Cook, A., Traore, S., and Ameli, S. Documint: Docstring generation for python using small language models, 2024. URL https://arxiv.org/abs/2405.10243.

Ren, S., Guo, D., Lu, S., Zhou, L., Liu, S., Tang, D., Sundaresan, N., Zhou, M., Blanco, A., and Ma, S. Codebleu: a method for automatic evaluation of code synthesis, 2020.

Wataoka, K., Takahashi, T., and Ri, R. Self-preference bias in llm-as-a-judge. *arXiv preprint arXiv:2410.21819*, 2024.

Yang, D., Simoulin, A., Qian, X., Liu, X., Cao, Y., Teng, Z., and Yang, G. Docagent: A multi-agent system for automated code documentation generation, 2025.

Zhang, T., Kishore, V., Wu, F., Weinberger, K. Q., and Artzi, Y. Bertscore: Evaluating text generation with bert. *arXiv preprint arXiv:1904.09675*, 2019.

Zhang, Y., Pan, Z., Yusuf, I. N. B., Ruan, H., Shariffdeen, R., and Roychoudhury, A. Code review agent benchmark, 2026. URL https://arxiv.org/abs/2603.23448.

Zheng, L., Chiang, W.-L., Sheng, Y., Zhuang, S., Wu, Z., Zhuang, Y., Lin, Z., Li, Z., Li, D., Xing, E., et al. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in neural information processing systems*, 36: 46595–46623, 2023.

Zhou, S., Alon, U., Agarwal, S., and Neubig, G. CodeBERTScore: Evaluating code generation with pretrained models of code. In Bouamor, H., Pino, J., and Bali, K. (eds.), *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 13921–13937, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.859. URL https://aclanthology.org/2023.emnlp-main.859/.

Zhu, L., Wang, X., and Wang, X. Judgelm: Fine-tuned large language models are scalable judges. *arXiv preprint arXiv:2310.17631*, 2023.

A. Appendix

A.1. Hyperparameters

Hyperparameters of models: gpt-5-mini: temperature fixed at 1, reasoning effort medium, context window 400k tokens qwen3-235b-a22b: non-thinking mode, temperature 0.7, top_p 0.8, top_k 20, context window 32,768 tokens

A.2. Evaluation Prompt

Listing 1. Prompt for evaluation stage function-body completion. The `left_context` includes the function signature and its docstring (if present), while the model must output only the indented body.

```
Given the context before the function:
```python
{left_context}
```
Generate the continuation of the body of one method, without a header.
Important: format your answer in a code block:
```python
<code>
```
Observe indentation as in Python.
The function body will be inserted after the context and will be tested.
```

A.3. Dataset

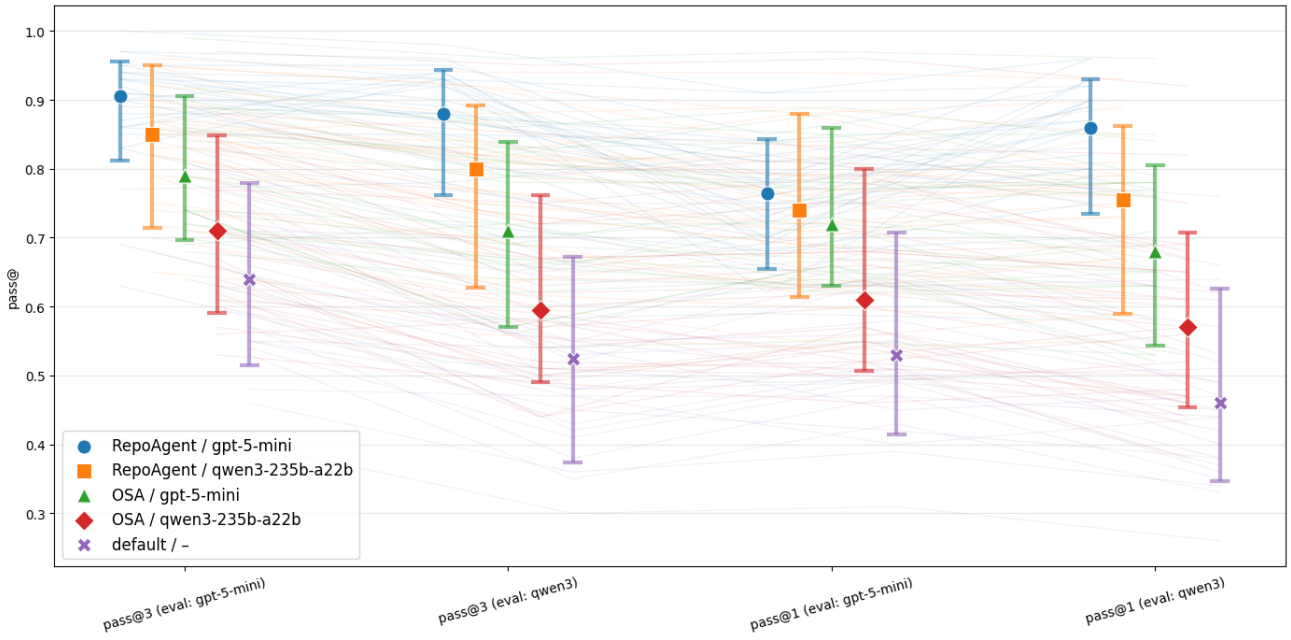

Figure 4. Distribution of pass@k scores across evaluation configurations. Points denote medians over 28 repositories; vertical bars span the 0.1–0.9 interquantile range; faint lines trace individual repository trajectories.

The aggregate ranking of documentation systems (RepoAgent > OSA > default) is invariant to both the evaluator model (gpt-5-mini vs. qwen3-235b-a22b) and the metric strictness ($k \in \{1, 3\}$). At the per-repository level, however, pairwise ranking consistency degrades: in fewer than 40% of repositories do all configuration pairs preserve their global ordering. No evaluator self-preference bias is observed—switching the evaluator between the GPT-5 and Qwen families does not alter the ranking of documentation models from those same families, contrary to the commonly reported tendency of LLMs to favor architecturally similar outputs (Zheng et al., 2023).

Table 2. Repository statistics. The parameter `n_candidates` reflects the overall number of docstrings, `n_all_tasks` - the number of valid tasks, `n_tasks` - the number of valid tasks which are completed less than in 10 seconds.

repo	n_candidates_tasks	n_all_tasks	n_tasks	forks	stars
SySS-Research/hashcathelper	146	106	106	7	120
pytube/pytube	486	152	142	2521	13066
explosion/radicli	178	55	55	7	108
explosion/confection	202	64	64	15	193
citrusvanilla/tinyflux	463	224	224	12	191
psf/requests	661	156	63	9707	53682
astanin/python-tabulate	404	360	360	180	2508
robobenjie/posetree	155	60	60	3	161
probabilists/zuko	335	129	129	34	441
keleshev/schema	188	178	178	215	2940
openqasm/oqpy	274	165	165	14	29
pyiron/structuretoolkit	322	90	62	1	8
fox-it/dissect.esedb	189	114	66	11	22
bridgecrewio/jsonpath-ng	539	114	91	3	8
google/pycnite	167	78	78	3	26
furiosa-ai/cocotbext-fcov	251	119	119	1	9
trezor/upysize	144	79	79	4	6
dajiaji/pyhpke	144	82	82	6	13
balancap/arrowbic	199	75	75	0	6
ecmwf/troika	250	70	70	6	11
MaxAtkinson/tilted	108	69	69	0	16
kodejuice/arithmetic-compressor	144	88	88	3	19
edsaav/nail	152	67	67	2	4
dmayo3/mocksafe	237	91	91	0	13
CepstrumLabs/pylox	334	176	176	0	3
ravin-d-27/PyDeepFlow	446	154	154	31	31
marcpage/pylavi	226	187	133	4	11
kolonialno/oida	219	48	48	1	5

Table 3. Pass@1 / Pass@3 evaluation metrics across repos. Format: pass@1(gpt-5-mini) / pass@1(qwen3-235b-a22b) // pass@3(gpt-5-mini) / pass@3(qwen3-235b-a22b)

doc_system	OSA		RepoAgent		default
doc_model	gpt-5-mini	qwen3-235b-a22b	gpt-5-mini	qwen3-235b-a22b	–
CepstrumLabs/pylox	0.85 / 0.85 0.89 / 0.87	0.82 / 0.75 0.87 / 0.79	0.79 / 0.89 0.87 / 0.93	0.88 / 0.90 0.95 / 0.92	0.78 / 0.62 0.84 / 0.66
MaxAtkinson/tilted	0.77 / 0.78 0.87 / 0.81	0.71 / 0.62 0.84 / 0.65	0.81 / 0.83 0.94 / 0.87	0.79 / 0.81 0.90 / 0.84	0.50 / 0.38 0.64 / 0.42
SySS-Research/hashcathelper	0.76 / 0.65 0.83 / 0.69	0.63 / 0.60 0.71 / 0.62	0.82 / 0.75 0.92 / 0.78	0.71 / 0.68 0.82 / 0.75	0.67 / 0.62 0.72 / 0.65
astanin/python-tabulate	0.64 / 0.58 0.70 / 0.62	0.55 / 0.47 0.64 / 0.50	0.59 / 0.58 0.69 / 0.58	0.60 / 0.58 0.70 / 0.62	0.55 / 0.40 0.74 / 0.48
balancap/arrowbic	0.65 / 0.63 0.72 / 0.67	0.60 / 0.53 0.71 / 0.57	0.78 / 0.82 0.85 / 0.85	0.75 / 0.78 0.83 / 0.83	0.47 / 0.44 0.57 / 0.47
bridgecrewio/jsonpath-ng	0.97 / 0.96 0.99 / 0.96	0.96 / 0.92 0.97 / 0.95	0.91 / 0.92 0.94 / 0.93	0.94 / 0.93 0.96 / 0.94	0.69 / 0.53 0.77 / 0.57
citrusvanilla/tinyflux	0.72 / 0.69 0.79 / 0.72	0.62 / 0.57 0.68 / 0.60	0.85 / 0.92 0.95 / 0.95	0.79 / 0.79 0.88 / 0.81	0.64 / 0.57 0.71 / 0.60
dajiaji/pyhpke	0.85 / 0.77 0.90 / 0.79	0.78 / 0.68 0.83 / 0.72	0.78 / 0.96 0.97 / 0.96	0.89 / 0.85 0.92 / 0.88	0.65 / 0.52 0.69 / 0.55
dmayo3/mocksafe	0.69 / 0.65 0.78 / 0.66	0.51 / 0.50 0.61 / 0.52	0.69 / 0.93 0.86 / 0.94	0.72 / 0.75 0.86 / 0.77	0.47 / 0.35 0.57 / 0.38
ecmwf/troika	0.72 / 0.58 0.81 / 0.65	0.57 / 0.46 0.74 / 0.54	0.74 / 0.80 0.93 / 0.89	0.69 / 0.61 0.87 / 0.74	0.49 / 0.37 0.54 / 0.41
edsaav/nail	0.63 / 0.71 0.69 / 0.72	0.49 / 0.52 0.56 / 0.58	0.82 / 0.87 0.89 / 0.88	0.62 / 0.67 0.75 / 0.70	0.40 / 0.44 0.48 / 0.47
explosion/confection	0.66 / 0.55 0.74 / 0.57	0.54 / 0.46 0.60 / 0.50	0.77 / 0.86 0.86 / 0.92	0.69 / 0.66 0.82 / 0.74	0.42 / 0.38 0.55 / 0.45
explosion/radicli	0.63 / 0.65 0.72 / 0.66	0.50 / 0.47 0.57 / 0.51	0.75 / 0.80 0.90 / 0.81	0.64 / 0.62 0.72 / 0.64	0.50 / 0.51 0.61 / 0.55
fox-it/dissect.esedb	0.65 / 0.53 0.74 / 0.56	0.57 / 0.44 0.67 / 0.44	0.70 / 0.86 0.95 / 0.87	0.63 / 0.67 0.74 / 0.70	0.43 / 0.33 0.53 / 0.35
furiosa-ai/cocotbext-fcov	0.65 / 0.69 0.74 / 0.72	0.46 / 0.48 0.53 / 0.49	0.62 / 0.90 0.83 / 0.93	0.55 / 0.59 0.65 / 0.61	0.31 / 0.26 0.40 / 0.30
google/pycnite	0.88 / 0.79 0.92 / 0.82	0.75 / 0.69 0.80 / 0.72	0.73 / 0.93 0.97 / 0.93	0.88 / 0.79 0.95 / 0.82	0.63 / 0.61 0.70 / 0.64
keleshev/schema	0.55 / 0.53 0.64 / 0.54	0.52 / 0.47 0.63 / 0.51	0.49 / 0.50 0.63 / 0.52	0.54 / 0.55 0.68 / 0.56	0.55 / 0.45 0.69 / 0.53
kodejuice/arithmetic-compressor	0.77 / 0.78 0.83 / 0.79	0.66 / 0.61 0.81 / 0.63	0.78 / 0.89 0.94 / 0.90	0.83 / 0.89 0.95 / 0.92	0.51 / 0.42 0.60 / 0.48
kolonialno/oida	0.63 / 0.63 0.68 / 0.64	0.55 / 0.43 0.66 / 0.49	0.91 / 0.96 1.00 / 0.98	0.77 / 0.76 0.85 / 0.79	0.39 / 0.34 0.46 / 0.36
marcpage/pylavi	0.78 / 0.76 0.86 / 0.78	0.68 / 0.65 0.77 / 0.67	0.68 / 0.75 0.77 / 0.78	0.72 / 0.73 0.84 / 0.75	0.58 / 0.49 0.66 / 0.54
openqasm/oqpy	0.67 / 0.65 0.74 / 0.69	0.53 / 0.57 0.61 / 0.58	0.72 / 0.90 0.93 / 0.92	0.71 / 0.73 0.79 / 0.77	0.49 / 0.45 0.55 / 0.48
probabilists/zuko	0.78 / 0.78 0.86 / 0.81	0.68 / 0.65 0.78 / 0.67	0.68 / 0.70 0.83 / 0.72	0.73 / 0.78 0.87 / 0.82	0.64 / 0.47 0.75 / 0.52
psf/requests	0.84 / 0.76 0.90 / 0.83	0.79 / 0.67 0.83 / 0.75	0.84 / 0.77 0.93 / 0.85	0.82 / 0.74 0.90 / 0.81	0.75 / 0.66 0.80 / 0.75
pyiron/structuretoolkit	0.66 / 0.52 0.74 / 0.57	0.52 / 0.38 0.62 / 0.44	0.67 / 0.79 0.86 / 0.83	0.68 / 0.59 0.79 / 0.63	0.51 / 0.36 0.59 / 0.43
pytube/pytube	0.73 / 0.67 0.79 / 0.70	0.62 / 0.61 0.71 / 0.67	0.84 / 0.87 0.91 / 0.88	0.81 / 0.76 0.89 / 0.82	0.56 / 0.49 0.64 / 0.55
ravin-d-27/PyDeepFlow	0.75 / 0.77 0.84 / 0.83	0.68 / 0.69 0.77 / 0.75	0.81 / 0.80 0.91 / 0.87	0.79 / 0.76 0.86 / 0.82	0.63 / 0.64 0.71 / 0.70
robobenjie/posetree	0.89 / 0.84 0.92 / 0.86	0.83 / 0.82 0.88 / 0.85	0.76 / 0.86 0.88 / 0.88	0.76 / 0.85 0.85 / 0.87	0.79 / 0.76 0.87 / 0.80
trezor/upysize	0.62 / 0.63 0.71 / 0.67	0.52 / 0.56 0.63 / 0.59	0.70 / 0.90 0.90 / 0.94	0.76 / 0.79 0.85 / 0.84	0.45 / 0.47 0.55 / 0.54