Simple and Effective Baselines for Code Summarisation Evaluation

Jade Robinson and Jonathan K. Kummerfeld
The University of Sydney
jonathan.kummerfeld@sydney.edu.au

Abstract

Code documentation is useful, but writing it is time-consuming. Different techniques for generating code summaries have emerged, but comparing them is difficult because human evaluation is expensive and automatic metrics are unreliable. In this paper, we introduce a simple new baseline in which we ask an LLM to give an overall score to a summary. Unlike n-gram and embedding-based baselines, our approach is able to consider the code when giving a score. This allows us to also make a variant that does not consider the reference summary at all, which could be used for other tasks, e.g., to evaluate the quality of documentation in code bases. We find that our method is as good or better than prior metrics, though we recommend using it in conjunction with embedding-based methods to avoid the risk of LLM-specific bias.

1 Introduction

Relevant and up-to-date documentation is useful for software maintenance (Stapleton et al., 2020; Misra et al., 2020; de Souza et al., 2006). To support one important form of documentation, researchers have developed models that generate one-line summaries of functions (Hu et al., 2018a; LeClair et al., 2020; Nguyen et al., 2024a, inter alia). However, evaluating these models is difficult. Expert human evaluations are expensive, slow to collect, and hard to consistently reproduce. Automatic metrics are cheap and consistent, but they have weak-to-moderate correlation with human scores (Roy et al., 2021; Haque et al., 2022; Mastropaolo et al., 2024). Methods in text summarisation evaluation can be applied to code, but the difference in modality (code vs. text) means metrics that compare a summary with its source are unlikely to be adaptable.

In this paper, we introduce a simple baseline: directly querying an LLM to get an overall rating of a generated summary. This approach considers the code when judging the summary, which most current metrics do not. We also propose a reference-free variant, which has not previously been done for this task. Not needing a reference summary enables new uses of these metrics, such as to flag low quality summaries in a code base or as part of the summary generation process.

We compare with all of the standard n-gram based metrics, a model-based metric (Mastropaolo et al., 2024), and embedding-based metrics. We evaluate by measuring correlation with two datasets of human judgements (Haque et al., 2022; Roy et al., 2021). In appendices, we also provide results on two datasets that consider specific aspects of summary quality.

Our approach is the best at predicting an overall score. For similarity with a reference, there is no significant difference between our approach and alternatives. We do find a risk that our method prefers output if it comes from the same LLM as the metric, and so we recommend using our method alongside an embedding-based metric.

While evaluation by querying an LLM has been done in other tasks with natural language outputs, our results differ from work in other areas. For example, unlike in machine translation, our method remains just as effective without a reference, and it improves over a metric using a supervised model, and unlike in QA, our method does not favour longer (or shorter) summaries. These differences highlight the distinctiveness of code summarisation and therefore, the value of research in this space. Our work provides novel baselines that are simple and effective, forming a solid foundation for further exploration.

2 Related Work

Code Summarisation Evaluation N-gram metrics, such as BLEU, METEOR, and ROUGE-L, were the first approach for evaluation, but have low correlation with human evaluation Roy et al. (2021).

Proceedings of The 23rd Annual Workshop of the Australasian Language Technology Association, pages 112–131
November 26-28, 2025

Embedding-based approaches, such as Sentence-BERT, improve on n-gram metrics, but still have a weak-to-moderate correlation (Haque et al., 2022; Mastropaolo et al., 2024). One trained metric exists, SIDE, and improves slightly over embedding methods (Mastropaolo et al., 2024).

Despite these findings, research still relies on n-gram metrics for evaluation. Of ten new code summarisation papers in 2024 (Nguyen et al., 2024b; Su and McMillan, 2024; Su et al., 2024; Zhao et al., 2024; Li et al., 2024; Pan et al., 2024; Sun et al., 2024; Ahmed et al., 2024; Cai et al., 2024; Mao et al., 2024), six used only n-gram metrics, three used n-gram metrics and embedding-based metrics, and one only used human evaluation.

Human Evaluation Datasets We focus on two datasets that were collected specifically for code summarisation metric evaluation (Roy et al., 2021; Haque et al., 2022). We also draw data from papers that proposed new code summarisation methods and asked people to evaluate specific aspects of quality (Gao et al., 2023; Su et al., 2024). Those results are mentioned in analysis and included in Appendix A due to space constraints.

LLM-prompting based NLG Evaluation Prompting has been successfully used to evaluate other forms of Natural Language Generation, e.g., for text summarisation and dialogue generation (Liu et al., 2023), and machine translation (Kocmi and Federmann, 2023). We observe some key differences between our results and other NLG work. We achieve equally strong results without a reference, but Qian et al. (2024) and Huang et al. (2024) investigate different prompting techniques and find that the reference summary is very beneficial. We also find that our approach consistently improves over a trained method, while trained models are still the most effective for MT (Anugraha et al., 2024; Freitag et al., 2024), probably because of the larger and higher quality datasets for metric development in MT.

There has also been considerable work evaluating the potential biases of LLM evaluators (Wu and Aji, 2023; Zheng et al., 2024; Koo et al., 2024), finding evidence that LLMs tend to evaluate their own outputs more highly and favour longer responses. We investigate this issue in Section 6.1.

Reference-Free Metrics We introduce the first reference-free approach for code summarisation evaluation, but there is significant prior work for other tasks (Rei et al., 2021; Scialom et al., 2021). These often have better correlations with human evaluations than equivalent reference-based metrics. However, Deutsch et al. (2022) argue that reference-free metrics are essentially creating their own pseudo-references, and so are constrained by their own generation ability. We agree that reference-free metrics are not a complete substitute, but for code summarisation they have the additional benefit that they could be used to flag low quality summaries within an existing code base.

3 Task

Code summarisation is the task of generating a summary of a code snippet. We are proposing new metrics for this task. The aim of the metric is to output a score that captures the overall quality of the summary, so that it can provide a broad indicator of the model's performance. These metrics have access to the code, the generated summary, and a human-written reference summary. However, we will also consider a variant of our approach that does not use the reference. We measure the quality of the metric by looking at how well it correlates with human ratings of overall score and similarity.

4 New Metric: Ask LLM Directly

Our metric is simple: ask an LLM to give the summary a rating, just like asking a human. One benefit is that this approach can consider the relevant *code* as well as the reference summary. In contrast, n-gram and embedding based metrics only measure the similarity between the generated summary and a reference summary. Our metric can also work without a reference. We include this variant in our results and note that (1) it is useful when high-quality references are not available, and (2) it could be used outside of model evaluation, for example to identify low quality human-written documentation.

To develop this metric we created a prompt by systematically testing different variations using established techniques such as chain-of-thought reasoning, role-based prompting and varying the problem description with Claude. Figure 4 shows the final prompt used, selected based on correlation with human ratings on a subset of the data. In the reference free case, the "Reference Summary" line is left out.

We also considered question-answering based prompts, where we focused on whether the LLM was able to answer questions about the reference

```
You are a professional software engineer.
Evaluate the statement by responding 'Strongly
agree', 'Somewhat agree', 'Somewhat disagree'
or 'Strongly disagree'. Independent of other
factors, I feel the new summary is accurate.

Reference summary: {Reference Summary}
Function:{Original Function}
Generated summary: {Generated Summary}
What are the steps you would take to evaluate
this statement? Show your steps and then
provide an evaluation (Strongly agree, Somewhat
agree, Somewhat disagree or Strongly disagree).
```

Figure 1: Ask LLM Directly Final Prompt

using information from the generated summary. For further details, see Appendix E.

5 Experiments

5.1 Datasets

We use two datasets that were created for metric evaluation. We aim to produce a single score, and so the most relevant data is Roy et al. (2021)'s Overall Score, a direct assessment of the overall quality of the summary. We also consider Haque et al. (2022)'s Similarity, which measures the similarity with the reference, but that does not account for a high quality but different summary. To avoid overfitting, during development we used a subset of the data. For the final results we used all of the data with 10-fold cross-validation. Note, while these are public datasets, the human evaluations are stored separately from the text being judged, so it is unlikely that LLM pretraining is causing data contamination.

In analysis, we also consider human evaluations of Adequacy that were collected in the process of evaluating a code summarisation system (Gao et al., 2023). Additional details are in Appendix D.2 and results comparing with specific aspects of quality are in Appendix A.

We release a version of all the datasets reformatted to be consistent, and with all of the same information.[1] This was somewhat involved as some datasets did not directly include the code. Fortunately, they did indicate their code and documentation source, and so we could go back to that source and match the summary to find the code.

5.2 Measuring Correlation

As in previous papers which evaluate code summarisation metrics (Roy et al., 2021; Haque et al., 2022; Mastropaolo et al., 2024), we aim to maximise correlation with human evaluation scores. We follow Haque et al. (2022)'s methodology: (1) when there are multiple human scores for a sample, we compare with the mean to reduce the impact of noise from disagreement, and (2) we use Spearman's Rank correlation for each metric because, unlike Pearson's correlation, it does not assume a normal distribution. We use a permutation test for significance testing, see Appendix B for details.

[1] Link to github repository

	Metrics	Overall Score (Roy et al.)	Similarity (Haque et al.)
n-gram	BLEU-A	0.28	0.55
	METEOR	0.31	0.75
	ROUGE-L	0.21	0.47
trained	SIDE	0.38	0.32
embedding	SentenceBERT	0.36	0.76
	gte-base-en	0.38	0.80
	voyage-code-3	0.43	**0.81**
ask-LLM	*ask-OLMo*	0.35	0.49
	ask-OLMo-no-ref	0.36	0.61
	ask-gpt	0.42	0.48
	ask-claude	**0.47**	0.57
	ask-claude-no-ref	0.46	0.61

Table 1: Spearman's Correlation with Human Ratings for Overall Score and Similarity

5.3 Metrics

We consider the most commonly used metrics (BLEU, METEOR and ROUGE-L), the best metrics according to prior work (SIDE and SentenceBERT), two new embeddings (gite-base-en, and coyage-code-3), and our own metric (ask-LLM and ask-LLM-no-ref), where *LLM* is the name of the model that is queried, and *no-ref* indicates the variant in which no reference summary is provided in the prompt. When measuring results, we only run each approach once. It is likely that there would be some variation each time these systems are run, but in our experience varying the prompt (see Appendix E) results were consistent and so they are likely to be consistent across runs with a single prompt as well. For further details, see Appendix C. Metrics that are evaluated here for the first time are in italics in Table 1.

6 Results

Table 1 shows correlations with Overall Score and Similarity to the reference summary. Below, we note several key results.

N-gram metrics are not as effective. For Overall Score, the trained method (SIDE), the best

embedding-based approach (voyage-code-3) and the best ask-LLM approach (ask-claude) outperform the best n-gram metric (BLEU-A). All of these improvements are statistically significant according to a permutation test at below the 0.01 level[2]. For Similarity, we find a different pattern, with SIDE performing worst, and the other three types of metrics in similar ranges. We find no statistically significant difference between the best n-gram based metric (METEOR) and either the best embedding-based metric (voyage-code-3) or the best ask-LLM metric (ask-claude-no-ref).

Embedding metrics are comparable to ask-LLM metrics. On Overall Score, the best embedding-based approach (voyage-code-3) and the best ask-LLM approach (ask-claude) are not statistically significantly different. For Similarity they are, with the embeddings being better, but we would expect embeddings to be better suited to that task. Note in particular that a summary may be good even though it isn't similar to the reference, and so a metric that focuses on similarity will sometimes struggle. There is also the issue that Similarity is only a measure of quality if the reference is high quality. In code summarisation datasets, nearly all reference summaries are stripped from Github with limited manual oversight. This introduces many quality issues.

Newer embeddings are better. For both Overall Score and Similarity, the newest embedding based metric, using voyage-code-3, improves on the previous state-of-the-art embeddings-based metric SentenceBERT. This is good news, since it indicates that continued progress on embedding methods is likely to continue to provide improvements here. One key difference between these approaches is cost, which will be discussed below.

ask-LLM-no-ref is just as effective. The performance of the Ask-LLM-Directly style metrics is stable regardless of whether the reference summary is provided, with no statistically significant difference between the two.

Different LLMs may perform differently. The choice of model (e.g. OLMo vs Claude) does lead to a significant difference. However, we used Claude when fine-tuning our prompt, making it an unfair comparison.

[2]Specific p-values are included in Appendix B

Quality Dimensions	Gao et al. - Java (Train) Adequate	Concise	Fluent
Accuracy: "Independent of other factors, I feel the new summary is accurate"	0.59	**0.38**	**0.44**
Adequacy: "The new summary contains all of the important information required for understanding the method"	**0.60**	0.35	0.37
Conciseness: "The new summary only contains necessary information."	0.59	0.35	0.41

Table 2: Test of Different Quality Dimensions

6.1 Analysis

To understand the strengths and weaknesses of our approach, we conducted several additional experiments.

Ask-LLM method can't easily be adapted to different quality dimensions Table 2 shows the results of our attempts to get the LLM to focus on specific aspects of quality. We see very little variation, with the scores continuing to mainly reflect Adequacy. Looking at specific examples, we found two issues. First, mentioning unrelated issues, e.g., for conciseness it produced: "The generated summary contains *incorrect information* and does *not accurately* summarize the function.". Second, inconsistency, e.g., for conciseness it produced "...the *lack of specificity* makes the generated summary less informative ...". We did not explore this further since our focus is on a single metric that aligns with overall quality. However, note that full results in Appendix A show that despite these issues, correlation with specific aspects was better than prior methods.

Ask-LLM is Not Sensitive to Length Many studies suggest that LLM evaluators are biased towards longer outputs (Wu and Aji, 2023; Zheng et al., 2024; Koo et al., 2024). However, for our metric, looking at the scores assigned by different metrics and the number of characters in the generated summary, in most cases we find the correlation is close to zero. For full results, see Appendix G.

Model Sensitivity There is a risk that an LLM will prefer its own output. We considered the relative ranking of each model according to each metric. Surprisingly, ask-gpt rates the models that use GPT-4 as the worst overall. None of the data we had used Claude and so we generated our own summaries

Metrics	Cost per Query (USD)
voyage-code-3	0.000002
ask-OLMo/ask-OLMo-no-ref	0.011
ask-gpt	0.012
ask-claude/ask-claude-no-ref	0.024

Table 3: Metric Costs

with Claude and valuated them. While Claude did find some issues within summaries it had generated, in 92.7% of cases it gives its summaries the highest possible rating. For full results, see Appendices F and H. Based on this, we recommend using these metrics in combination with embedding based methods.

Costs Table 3 shows the cost per summary of each of the metrics. These are API costs for commercial tools and compute costs open source model OLMo-2 (we used an A100). These results show that these approaches are clearly much cheaper than running human evaluations, but still more expensive than metrics which can be run locally, e.g. gte-base-en, Sentence-BERT and n-gram methods.

7 Conclusion

We introduce a simple LLM-based evaluation metric and evaluate it on the standard code summarisation datasets. Our approach is consistently better than prior metrics. We describe a reference-free variant of our approach, which also performs well, and could be used in a variety of ways. We recommend that future work uses a combination of embedding and Ask-LLM metrics for development, and turn to human raters for final evaluation. That will enabler faster development while maintaining reliable evaluation.

Limitations

Languages Evaluated Against Due to the lack of available data, it was not possible to get any human evaluation data from programming languages apart from Python and Java. Of those, overall scores are only available for the Java-based datasets, though in Appendix A we do include results for other scores on the Python datasets. Metrics that work for one language may not be as effective for another. This means that our results may not apply as well across different languages that we weren't able to evaluate against. In particular, our method may be less effective on languages that are less widely used and so less well understood by LLMs. However, by definition, rare langauges are not as common as common ones, and so our method will be useful for the vast majority of developers.

Reliance on Intrinsic Human Assessment All of the studies we use ask raters to assess the quality of the summary directly, rather than assess the impact of the different summaries on downstream tasks. As such, the ratings may not be optimising for summaries that aid development but rather developer perception, which while likely a good proxy, will never be perfect. The reference summaries are also human-written and vary considerably in quality. For reference-based methods, that could be misleading, as being similar to the reference may not indicate a summary is good.

Prompting Approaches Tested We were not able to run every prompt variation for every model, GPT and OLMo generally seem to perform worse here but this could be because the prompts were decided using Claude, but due to budget restraints we only tested the best performing prompt from initial testing on Claude.

AI Assistance Statement

ChatGPT was used to ask LaTeX questions to assist with the formatting of this paper.

Acknowledgments

This material is partially funded by an unrestricted gift from Google, and by the Australian Research Council through a Discovery Early Career Researcher Award.

References

Wasi Ahmad, Saikat Chakraborty, Baishakhi Ray, and Kai-Wei Chang. 2020. A transformer-based approach for source code summarization. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4998–5007, Online. Association for Computational Linguistics.

Toufique Ahmed, Kunal Suresh Pai, Premkumar Devanbu, and Earl Barr. 2024. Automatic semantic augmentation of language model prompts (for code summarization). In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, ICSE '24, New York, NY, USA. Association for Computing Machinery.

Uri Alon, Shaked Brody, Omer Levy, and Eran Yahav. 2019. code2seq: Generating sequences from structured representations of code. In *International Conference on Learning Representations*.

David Anugraha, Garry Kuwanto, Lucky Susanto, Derry Tanti Wijaya, and Genta Winata. 2024. MetaMetrics-MT: Tuning meta-metrics for machine translation via human preference calibration. In *Proceedings of the Ninth Conference on Machine Translation*, pages 459–469, Miami, Florida, USA. Association for Computational Linguistics.

Satanjeev Banerjee and Alon Lavie. 2005. 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*, pages 65–72, Ann Arbor, Michigan. Association for Computational Linguistics.

Aakash Bansal, Robert Wallace, Zachary Karas, Ningzhi Tang, Yu Huang, Toby Jia-Jun Li, and Collin McMillan. 2024. Programmer visual attention during context-aware code summarization. *Preprint*, arXiv:2405.18573.

Yufan Cai, Yun Lin, Chenyan Liu, Jinglian Wu, Yifan Zhang, Yiming Liu, Yeyun Gong, and Jin Song Dong. 2024. On-the-fly adapting code summarization on trainable cost-effective language models. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23, Red Hook, NY, USA. Curran Associates Inc.

Sergio Cozzetti B. de Souza, Nicolas Anquetil, and Káthia M. de Oliveira. 2006. Which documentation for software maintenance? *Journal of the Brazilian Computer Society*, 12(3):31–44.

Daniel Deutsch, Tania Bedrax-Weiss, and Dan Roth. 2021a. Towards question-answering as an automatic metric for evaluating the content quality of a summary. *Transactions of the Association for Computational Linguistics*, 9:774–789.

Daniel Deutsch, Rotem Dror, and Dan Roth. 2021b. A statistical analysis of summarization evaluation metrics using resampling methods. *Transactions of the Association for Computational Linguistics*, 9:1132–1146.

Daniel Deutsch, Rotem Dror, and Dan Roth. 2022. On the limitations of reference-free evaluations of generated text. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 10960–10977, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Alexander Fabbri, Chien-Sheng Wu, Wenhao Liu, and Caiming Xiong. 2022. QAFactEval: Improved QA-based factual consistency evaluation for summarization. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2587–2601, Seattle, United States. Association for Computational Linguistics.

Markus Freitag, Nitika Mathur, Daniel Deutsch, Chi-Kiu Lo, Eleftherios Avramidis, Ricardo Rei, Brian Thompson, Frederic Blain, Tom Kocmi, Jiayi Wang, David Ifeoluwa Adelani, Marianna Buchicchio, Chrysoula Zerva, and Alon Lavie. 2024. Are LLMs breaking MT metrics? Results of the WMT24 metrics shared task. In *Proceedings of the Ninth Conference on Machine Translation*, pages 47–81, Miami, Florida, USA. Association for Computational Linguistics.

Shuzheng Gao, Cuiyun Gao, Yulan He, Jichuan Zeng, Lunyiu Nie, Xin Xia, and Michael Lyu. 2023. Code structure–guided transformer for source code summarization. *ACM Trans. Softw. Eng. Methodol.*, 32(1):Article 23.

Sakib Haque, Zachary Eberhart, Aakash Bansal, and Collin McMillan. 2022. Semantic similarity metrics for evaluating source code summarization. In *Proceedings of the 30th IEEE/ACM International Conference on Program Comprehension*, page 36–47. Association for Computing Machinery.

Sakib Haque, Alexander LeClair, Lingfei Wu, and Collin McMillan. 2020. Improved automatic summarization of subroutines via attention to file context. In *Proceedings of the 17th International Conference on Mining Software Repositories*, MSR '20, page 300–310, New York, NY, USA. Association for Computing Machinery.

Vincent J. Hellendoorn, Charles Sutton, Rishabh Singh, Petros Maniatis, and David Bieber. 2020. Global relational models of source code. In *International Conference on Learning Representations*.

X. Hu, G. Li, X. Xia, D. Lo, and Z. Jin. 2018a. Deep code comment generation. In *2018 IEEE/ACM 26th International Conference on Program Comprehension (ICPC)*, pages 200–20010.

Xing Hu, Ge Li, Xin Xia, David Lo, Shuai Lu, and Zhi Jin. 2018b. Summarizing source code with transferred api knowledge. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*, pages 2269–2275. International Joint Conferences on Artificial Intelligence Organization.

Xu Huang, Zhirui Zhang, Xiang Geng, Yichao Du, Jiajun Chen, and Shujian Huang. 2024. Lost in the source language: How large language models evaluate the quality of machine translation. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 3546–3562, Bangkok, Thailand. Association for Computational Linguistics.

Tom Kocmi and Christian Federmann. 2023. Large language models are state-of-the-art evaluators of translation quality. In *Proceedings of the 24th Annual Conference of the European Association for Machine Translation*, pages 193–203, Tampere, Finland. European Association for Machine Translation.

Ryan Koo, Minhwa Lee, Vipul Raheja, Jong Inn Park, Zae Myung Kim, and Dongyeop Kang. 2024. Benchmarking cognitive biases in large language models as

evaluators. In *Findings of the Association for Computational Linguistics ACL 2024*, pages 517–545, Bangkok, Thailand and virtual meeting. Association for Computational Linguistics.

Alexander LeClair, Sakib Haque, Lingfei Wu, and Collin McMillan. 2020. Improved code summarization via a graph neural network. In *Proceedings of the 28th International Conference on Program Comprehension*, page 184–195. Association for Computing Machinery.

Alexander LeClair, Siyuan Jiang, and Collin McMillan. 2019. A neural model for generating natural language summaries of program subroutines. In *Proceedings of the 41st International Conference on Software Engineering*, page 795–806. IEEE Press.

Zongjie Li, Chaozheng Wang, Pingchuan Ma, Chaowei Liu, Shuai Wang, Daoyuan Wu, Cuiyun Gao, and Yang Liu. 2024. On extracting specialized code abilities from large language models: A feasibility study. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, ICSE '24, New York, NY, USA. Association for Computing Machinery.

Chin-Yew Lin. 2004. ROUGE: A package for automatic evaluation of summaries. In *Text Summarization Branches Out*, pages 74–81, Barcelona, Spain. Association for Computational Linguistics.

Yang Liu, Dan Iter, Yichong Xu, Shuohang Wang, Ruochen Xu, and Chenguang Zhu. 2023. G-Eval: NLG evaluation using GPT-4 with better human alignment. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 2511–2522, Singapore. Association for Computational Linguistics.

Yingjie Mao, Xiao Li, Zongwei Li, and Wenkai Li. 2024. Automated smart contract summarization via LLMs. *ArXiv*, abs/2402.04863.

A. Mastropaolo, M. Ciniselli, M. Di Penta, and G. Bavota. 2024. Evaluating code summarization techniques: A new metric and an empirical characterization. In *2024 IEEE/ACM 46th International Conference on Software Engineering (ICSE)*, pages 1002–1002.

Vishal Misra, Jakku Sai Krupa Reddy, and Sridhar Chimalakonda. 2020. Is there a correlation between code comments and issues? An exploratory study. In *SAC '20: Proceedings of the 35th Annual ACM Symposium on Applied Computing*.

Niklas Muennighoff, Nouamane Tazi, Loic Magne, and Nils Reimers. 2023. MTEB: Massive text embedding benchmark. In *Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics*, pages 2014–2037, Dubrovnik, Croatia. Association for Computational Linguistics.

Minh Nguyen, Nghi Bui, Truong Son Hy, Long Tran-Thanh, and Tien Nguyen. 2024a. HierarchyNet: Learning to summarize source code with heterogeneous representations. In *Findings of the Association for Computational Linguistics: EACL 2024*, pages 2355–2367, St. Julian's, Malta. Association for Computational Linguistics.

Minh Nguyen, Nghi Bui, Truong Son Hy, Long Tran-Thanh, and Tien Nguyen. 2024b. HierarchyNet: Learning to summarize source code with heterogeneous representations. In *Findings of the Association for Computational Linguistics: EACL 2024*, pages 2355–2367, St. Julian's, Malta. Association for Computational Linguistics.

Xinglu Pan, Chenxiao Liu, Yanzhen Zou, Xianlin Zhao, and Bing Xie. 2024. Context-focused prompt tuning pre-trained code models to improve code summarization. In *2024 IEEE 48th Annual Computers, Software, and Applications Conference (COMPSAC)*, pages 1344–1349.

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 of the Association for Computational Linguistics*, pages 311–318, Philadelphia, Pennsylvania, USA. Association for Computational Linguistics.

Shenbin Qian, Archchana Sindhujan, Minnie Kabra, Diptesh Kanojia, Constantin Orasan, Tharindu Ranasinghe, and Fred Blain. 2024. What do large language models need for machine translation evaluation? In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 3660–3674, Miami, Florida, USA. Association for Computational Linguistics.

Ricardo Rei, Ana C Farinha, Chrysoula Zerva, Daan van Stigt, Craig Stewart, Pedro Ramos, Taisiya Glushkova, André F. T. Martins, and Alon Lavie. 2021. Are references really needed? unbabel-IST 2021 submission for the metrics shared task. In *Proceedings of the Sixth Conference on Machine Translation*, pages 1030–1040, Online. Association for Computational Linguistics.

Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence embeddings using Siamese BERT-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3982–3992, Hong Kong, China. Association for Computational Linguistics.

Devjeet Roy, Sarah Fakhoury, and Venera Arnaoudova. 2021. Reassessing automatic evaluation metrics for code summarization tasks. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, page 1105–1116. Association for Computing Machinery.

Thomas Scialom, Paul-Alexis Dray, Sylvain Lamprier, Benjamin Piwowarski, Jacopo Staiano, Alex Wang,

and Patrick Gallinari. 2021. QuestEval: Summarization asks for fact-based evaluation. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 6594–6604, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

S. Stapleton, Y. Gambhir, A. LeClair, Z. Eberhart, W. Weimer, K. Leach, and Y. Huang. 2020. A human study of comprehension and code summarization. In *2020 IEEE/ACM 28th International Conference on Program Comprehension (ICPC)*, pages 01–12.

Chia-Yi Su, Aakash Bansal, Yu Huang, Toby Jia-Jun Li, and Collin McMillan. 2024. Context-aware code summary generation. *Preprint*, arXiv:2408.09006.

Chia-Yi Su, Aakash Bansal, Vijayanta Jain, Sepideh Ghanavati, and Collin McMillan. 2023. A language model of Java methods with train/test deduplication. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2023, page 2152–2156, New York, NY, USA. Association for Computing Machinery.

Chia-Yi Su and Collin McMillan. 2024. Distilled GPT for source code summarization. *Automated Software Engineering*, 31(1):22.

Weisong Sun, Chunrong Fang, Yuchen Chen, Quanjun Zhang, Guanhong Tao, Yudu You, Tingxu Han, Yifei Ge, Yuling Hu, Bin Luo, and Zhenyu Chen. 2024. An extractive-and-abstractive framework for source code summarization. *ACM Trans. Softw. Eng. Methodol.*, 33(3).

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 6000–6010, Red Hook, NY, USA. Curran Associates Inc.

VoyageAI. 2024. voyage-code-3: more accurate code retrieval with lower dimensional, quantized embeddings. https://blog.voyageai.com/2024/12/04/voyage-code-3/#:~:text=voyage%2Dcode%2D3%20supports%20much,Matryoshka%20embeddings. Accessed: 2024-12-10.

Yao Wan, Zhou Zhao, Min Yang, Guandong Xu, Haochao Ying, Jian Wu, and Philip S. Yu. 2018. Improving automatic source code summarization via deep reinforcement learning. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, ASE '18, page 397–407, New York, NY, USA. Association for Computing Machinery.

Alex Wang, Kyunghyun Cho, and Mike Lewis. 2020. Asking and answering questions to evaluate the factual consistency of summaries. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 5008–5020, Online. Association for Computational Linguistics.

Minghao Wu and Alham Fikri Aji. 2023. Style over substance: Evaluation biases for large language models. *Preprint*, arXiv:2307.03025.

Kun Xu, Lingfei Wu, Zhiguo Wang, Yansong Feng, Michael Witbrock, and Vadim Sheinin. 2018. Graph2seq: Graph to sequence learning with attention-based neural networks. *Preprint*, arXiv:1804.00823.

Xin Zhang, Yanzhao Zhang, Dingkun Long, Wen Xie, Ziqi Dai, Jialong Tang, Huan Lin, Baosong Yang, Pengjun Xie, Fei Huang, Meishan Zhang, Wenjie Li, and Min Zhang. 2024. mGTE: Generalized long-context text representation and reranking models for multilingual text retrieval. *CoRR*, abs/2407.19669.

Junjie Zhao, Xiang Chen, Guang Yang, and Yiheng Shen. 2024. Automatic smart contract comment generation via large language models and in-context learning. *Information and Software Technology*, 168:107405.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. 2024. Judging LLM-as-a-judge with MT-bench and Chatbot arena. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23, Red Hook, NY, USA. Curran Associates Inc.

Daniel Zügner, Tobias Kirschstein, Michele Catasta, Jure Leskovec, and Stephan Günnemann. 2021. Language-agnostic representation learning of source code from structure and context. In *International Conference on Learning Representations*.

A Other Aspects of Quality

While we focus on aligning with overall quality, we also look at how well these metrics align with ratings of the summary content and style. We find that generally, all metrics, including the new baselines we introduce, tend to align better with the content factors of accuracy and adequacy than the style factors of conciseness and fluency. In this section, we did not conduct significance tests since this is not our primary objective, but rather intended to just provide supplementary analysis.

Content Ratings The content (how accurate and how adequate is the information provided), is one of the most important factors in determining overall summary quality according to human raters. In the Haque et al. (2022) dataset, Content Adequacy had the highest correlation with Overall Score (Spearman's correlation of 0.83). In contrast, style factors like conciseness and fluency had correlations of 0.60 and 0.54 respectively. In Table 4, we can see the overall results for these content based factors across three different datasets.

Here, the Ask-LLM-Directly approaches with Claude consistently perform better than existing metrics. One possible cause of this difference is issues with reference summary quality. Figure 2 shows how metric performance varies based on the quality of the reference summary in the Haque et al. (2022) dataset. While all of the metrics perform similarly when the references are medium to high quality, performance drops off significantly for all but the ask-LLM metrics when reference quality is low. One explanation could be that our LLM metrics have access to the code, providing a signal that is independent of the poor quality reference summaries. Another explanation could be that the n-gram and embedding metrics are focused on similarity, and in this case, high similarity is not an indicator of quality.

Summary Style Ratings Table 5 shows the correlation with Fluency and Conciseness, which are in general much lower than the correlations with both the overall ratings and the content ratings by around 0.15, which suggests that all of the metrics are prioritising summary content over style when rating. The overall rankings of the metrics remains stable, which means the new improvements in correlation with overall ratings and correlation with adequacy and accuracy are likely not to be coming at the expense of evaluating conciseness and fluency.

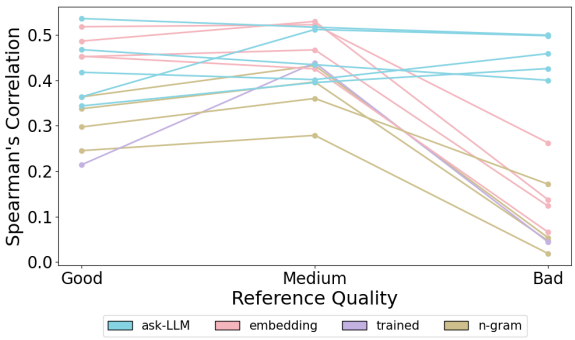


Figure 2: Correlation with Adequacy by Reference Quality on the Haque et al. dataset

As metrics continue to improve into the future, it may become useful to develop metrics for the quality factors individually. We attempted to do this by changing the prompt with the Ask-LLM-Directly technique, but we found it was surprisingly difficult to override the LLM's internal representation of overall summary quality by prompting.

Our Metrics Are Effective Across Programming Languages Gao et al. (2023)'s data includes both Java and Python. Comparing them in Tables 4 and 5, we find that our new metrics are fairly stable across languages, while BLEU-A and SIDE are worse on Python. For SIDE, this is probably because it was trained only on Java. This flexibility is a strength of the baselines we define.

Correlation with Informativeness The results for the final dataset we included, the Su et al. (2024) dataset, are in Table 6. It differs from the other datasets we used in a few ways: it includes evaluations of LLM-generated summaries, it evaluates only on the 'informativeness' of the summary, and all of the references were individually written and validated as part of another study. Here we see that SIDE actually outperforms all other metrics, but the other commonly used metrics such as the n-gram metrics and the SentenceBERT embeddings-based metric perform particularly poorly, with METEOR even giving a negative correlation. The new embeddings metrics and the LLM-based metrics we introduce both perform similarly, but the correlation is weak-to-moderate overall. It is clear that this data and/or the human evaluations are measuring something quite different from the other aspects we consider above. We believe this is because the reference summaries were written with the intent to explain the role of a function in a larger project, rather than explaining what it does.

	Metrics	Content Adequacy (Roy et al.)	Adequacy (Haque et al.)	Accuracy (Haque et al.)	Adequacy (Java) (Gao et al.)	Adequacy (Python) (Gao et al.)
n-gram	BLEU-A	0.27	0.37	0.33	0.48	0.27
	METEOR	0.31	0.45	0.47	0.49	0.44
	ROUGE-L	0.20	0.33	0.29	0.29	0.32
trained	SIDE	0.40	0.36	0.37	0.26	0.10
embedding	SentenceBERT	0.36	0.47	0.52	0.56	0.41
	gte-base-en	0.39	0.52	0.55	0.57	0.46
	voyage-code-3	0.44	0.58	0.62	0.59	0.49
ask-LLM	*ask-OLMo*	0.37	0.50	0.58	0.50	0.49
	ask-OLMo-no-ref	0.38	0.55	0.59	0.45	0.50
	ask-gpt	0.41	0.55	0.60	0.52	0.49
	ask-claude	0.47	0.54	0.62	**0.61**	**0.62**
	ask-claude-no-ref	**0.48**	**0.60**	**0.69**	0.55	0.58

Table 4: Spearman's Correlation with Human Ratings for Adequacy and Accuracy

		Conciseness				Fluency		
	Metrics	(Roy et al.)	(Haque et al.)	(Java) (Gao et al.)	(Python) (Gao et al.)	(Roy et al.)	(Java) (Gao et al.)	(Python) (Gao et al.)
n-gram	BLEU-A	0.15	0.16	0.20	0.10	0.13	0.32	0.12
	METEOR	0.17	0.27	0.19	0.23	0.15	0.29	0.23
	ROUGE-L	0.11	0.13	0.13	0.20	0.06	0.22	0.21
trained	SIDE	0.30	0.22	0.11	0.00	0.21	0.13	-0.01
embedding	SentenceBERT	0.22	0.26	0.16	0.20	0.17	0.26	0.18
	gte-base-en	0.25	0.33	0.21	0.25	0.18	0.30	0.27
	voyage-code-3	0.29	0.37	0.16	0.25	0.22	0.27	0.26
ask-LLM	*ask-OLMo*	0.25	0.40	**0.33**	0.32	0.17	0.37	0.32
	ask-OLMo-no-ref	0.28	0.41	0.31	0.33	0.22	0.31	0.33
	ask-gpt	0.29	0.42	0.32	0.28	0.24	0.34	0.31
	ask-claude	0.33	0.42	0.28	**0.35**	**0.26**	**0.38**	**0.38**
	ask-claude-no-ref	**0.36**	**0.46**	0.25	0.33	0.25	0.32	0.36

Table 5: Spearman's Correlation with Human Ratings for Conciseness and Fluency

	Metrics	Informativeness (Su et al.)
n-gram	BLEU-A	0.07
	METEOR	-0.09
	ROUGE-L	0.22
trained	SIDE	**0.45**
embedding	SentenceBERT	0.14
	gte-base-en	0.35
	voyage-code-3	0.34
ask-LLM	*ask-OLMo*	0.30
	ask-OLMo-no-ref	0.29
	ask-gpt	0.26
	ask-claude	0.23
	ask-claude-no-ref	0.28

Table 6: Spearman's Correlation with Human Ratings for Informativeness

A.1 Confidence Intervals

We measured confidence intervals and statistical significance by applying the methods from Deutsch et al. (2021b). Statistical significance has already been included in discussion above. For confidence intervals, their approach only applies when there are multiple systems, and so we can only use it for two of the datasets (Gao et al., 2023; Su et al., 2024). Due to the small size of these two datasets, the intervals were large. Note that even with broad confidence intervals, results can be statistically significantly different[3].

B Significance Testing

Confidence Intervals We follow the BOOTBOTH[4] method from Deutsch et al. (2021b) to calculate confidence intervals, which has been developed specifically for text summarisation metric evaluation. It accounts for human rating data not falling into the normal distribution as well as the fact that we have summaries generated by different models for the same document (code snippet). This approach requires the dataset to have generated summaries from different systems, which meant

[3]Also note that we only considered significance on the datasets with Overall Score and Similarity, and those datasets are not amenable to this CI calculation method.

[4]Using the implementation from nlpstats

that it was not possible to calculate confidence intervals for the Haque et al. (2022) dataset (they only test one system) and the Roy et al. (2021) dataset (which system generated each summary is not included in their publicly available data).

Permutation Tests While Deutsch et al. (2021b) also present code to run p-tests, we did not use their implementation due to the limitation of requiring generated summaries from different systems which we need for two of our datasets. We implement the test ourselves, sampling 10,000 times to approximate the distribution. We apply the Bonferroni correction, with 9 p-tests performed on each of the Roy et al. (2021) and Haque et al. (2022) datasets.

P-values for the tests we ran are shown in Table 7. P-values in bold are significant with $p < 0.05$ after being adjusted with Bonferroni correction.

C Metrics Tested

BLEU (Papineni et al., 2002) We use the BLEU-A variant, which is the average for the BLEU score of 1-, 2-, 3- and 4-grams individually. Calculated using HuggingFace's `evaluate` package (https://huggingface.co/spaces/evaluate-metric/bleu).

METEOR (Banerjee and Lavie, 2005) is also an n-gram based metric, but gives credit to synonyms and is more highly weighted towards recall. Calculated using HuggingFace's `evaluate` package (https://huggingface.co/spaces/evaluate-metric/meteor).

ROUGE-L (Lin, 2004) returns a score based on the longest common subsequence of words in the two summaries. Calculated using HuggingFace's `evaluate` package (https://huggingface.co/spaces/evaluate-metric/rouge) - the 'rougeL' statistic.

SIDE (Mastropaolo et al., 2024) uses contrastive learning to train an evaluator model. We use the example code provided in (Mastropaolo et al., 2024). We used the `models_with_hard_negatives` version of the model.

SentenceBERT (Reimers and Gurevych, 2019) is a text embedding method. We apply it to the generated summary and the reference, then calculate cosine similarity. Computed cosine similarity with `stsb-roberta-large` available from HuggingFace `sentence-transformers` (https://huggingface.co/sentence-transformers/stsb-roberta-large).

Generated: `returns the label text for the given element`

Original Code with Reference Summary:

```
// removes namespace prefix from label text
public String getLabelText(String xpath, String
     siblingPath, String indexId) {
  if (siblingPath != null && indexId != null) {
    String nodeName = NamespaceRegistry.
         stripNamespacePrefix(XPathUtils.getNodeName
         (siblingPath));
    return (nodeName + "_${" + indexId + "+1}");
  } else {
     String nodeName = NamespaceRegistry.
          stripNamespacePrefix(XPathUtils.
          getNodeName(normalizedXPath));
     return nodeName;
  }
}
```

Human Ratings:

Similarity:	[2, 2, 1]
Accuracy:	[3, 4, 4]
Adequacy:	[3, 4, 2]
Conciseness:	[4, 4, 3]

Figure 3: Example from Roy et al. (Note: this is a particularly short example)

Claude-3-Opus We used Claude-3-Opus-20240229, currently available from https://www.anthropic.com/api.

GPT-4o We used GPT-4o-2024-05-13, currently available from https://platform.openai.com/.

OLMo-2 We used OLMo-2-1124-13B-Instruct, available from HuggingFace (https://huggingface.co/allenai/OLMo-2-1124-13B-Instruct).

gte-base-en (Zhang et al., 2024) is an open-source model which performs well on the Massive Text Embedding Benchmark (MTEB) (Muennighoff et al., 2023) and is small enough to run without a GPU. on a standard personal computer. Score calculated with cosine similarity, using `gte-base-en-v1.5` available on HuggingFace (https://huggingface.co/Alibaba-NLP/gte-base-en-v1.5).

voyage-code-3 (VoyageAI, 2024) is a commercial embedding model trained specifically for code. Computed cosine similarity with VoyageAI's `voyage-code-3` embedding model as of December 2024.

D Dataset Statistics

Figure 3 shows an example of code, a generated summary, and a reference summary. At the bottom

	Overall Score (Roy et al., 2021)	Similarity (Haque et al., 2022)
SIDE / best n-gram	SIDE / BLEU **0.0002**	SIDE / METEOR **0.0001**
SIDE / best embedding	SIDE / voyage-code-3 0.0445	SIDE / voyage-code-3 **0.0000**
SIDE / best ask-LLM	SIDE / ask-claude 0.0030	SIDE / ask-claude-no-ref 0.0034
best embedding / worst embedding	voyage-code-3 / SentenceBERT **0.0000**	voyage-code-3 / SentenceBERT **0.011**
best embedding / best n-gram	voyage-code-3 / BLEU **0.0000**	voyage-code-3 / METEOR 0.0593
best embedding / best ask-LLM	voyage-code-3 / ask-claude 0.1035	ask-claude-no-ref / voyage-code-3 **0.0001**
best ask-LLM / worst ask-LLM	ask-claude / ask-OLMo **0.0000**	ask-claude-no-ref / ask-OLMo 0.1344
best ask-LLM / best n-gram	ask-claude / BLEU **0.0000**	ask-claude-no-ref / METEOR 0.0321
ask-claude / ask-claude-no-ref	ask-claude / ask-claude-no-ref 0.6779	ask-claude / ask-claude-no-ref 0.3585

Table 7: P-values for Overall Quality Scores

are human annotation of four aspects of quality for the summary.

D.1 License Information

The Roy et al. dataset has been released under an MIT license. The Github repositories for the Gao et al. and Haque et al. datasets do not contain any license information. The Su et al. datasets has not yet been publicly released so they do not have any license yet. Our use is compatible with the intended use when it was provided.

D.2 Dataset Measures

Roy et al. rate from Strongly Agree to Strongly Disagree:

- **Content Adequacy**: The extent to which the summary lacks information needed to understand the code.
- **Conciseness**: The degree to which the summary contains unnecessary information.
- **Fluency**: The continuity or smoothness rate in the generated summary.
- **Overall Score**: a Direct Assessment (DA) score from 1-100 of the overall quality of the summary.

Gao et al. also rate adequacy, conciseness and fluency, with slightly different definitions from 1 ('very dissatisfied') to 5 ('very satisfied'). These are defined as:

- **Adequacy**: How much the functional meaning of the code is preserved after summarisation.
- **Conciseness**: The ability to express the function of the code snippet without unnecessary words.
- **Fluency**: The quality of the generated language such as the correctness of grammar.

Haque et al. first show each rater either the reference or generated summary. They rate that summary on accuracy, adequacy and conciseness from Strongly Disagree to Strongly Agree and then rate similarity after seeing the other summary.

- **Accuracy**: Independent of other factors, I feel the summary is accurate.
- **Adequacy**: The summary is missing important information, and that can hinder the understanding of the method.
- **Conciseness**: The summary contains a lot of unnecessary information.[5]
- **Similarity**: These two comments are similar.

Su et al. focus instead on how the method fits within the entire project, rating the following prompt from Strongly Agree to Strongly Disagree:

[5]Note that Adequacy and Conciseness are phrased negatively, such that a Strongly Disagree rating is the most positive response. For readability and consistency throughout this paper these have been flipped so high agreement is positive.

- **Informativeness**: The summary contains information that helps to understand why the method exists in the project.

D.3 Models Tested

Each dataset also includes ratings of summaries generated by different models, which have different characteristics based on the method of generation.

Roy et al. (2021) sample methods from the Funcom dataset (LeClair et al., 2019). They collect ratings of the human-written reference summaries as well as five other summarisation models, listed below in order of human rater preference:

- CODE2SEQ (Alon et al., 2019) Encoder-Decoder RNN which represents code as compositional paths over its AST.
- GRAPH2SEQ (Xu et al., 2018) Encoder-Decoder RNN which represents code as a graph.
- AST-ATTENDGRU-FC (Haque et al., 2020) Encoder-Decoder RNN with three encoders, one for the textual code data, one for the AST, and one for the 'file context' - textual code data from other methods in the same file.
- AST-ATTENDGRU (LeClair et al., 2019) Same as AST-ATTENDGRU-FC but without the additional file context information.
- TRANSFORMER (Vaswani et al., 2017) The original Transformer model with no modifications for code summarisation.

Unfortunately, the data available online for this dataset does not include annotations which specify which summary was produced by which model so we are unable to analyse metric performance by model on this particular dataset.

Gao et al. (2023) sample methods instead from the TL-CodeSum dataset (Hu et al., 2018b) for the Java data and from the Wan et al. (2018) dataset for the Python data. They also select five different code summarisation models, ordered below by human rater preference on 'adequacy', but they do not collect ratings of the reference summaries.

- SG-TRANS (Gao et al., 2023) Transformer enhanced with structural information of the input, a graph created based on both local structures, e.g. if the tokens belong to the same statement, and global structures, e.g. if there a data flow between the tokens.
- GREAT (Hellendoorn et al., 2020) Transformer-based model enhanced with graph representations which encode control flow and data flow relations.
- NEURALCODESUM (Ahmad et al., 2020) Transformer with small modifications to attention process for the code summarisation task, with no AST or additional code structure information.
- TRANSFORMER (Vaswani et al., 2017) The original Transformer model with no modifications for code summarisation.
- CODETRANSFORMER (Zügner et al., 2021) Transformer which makes both the code and the AST available. It does this in a programming language agnostic way (i.e. it does not require any language-dependent pipelines such as generating a control flow graph).

Haque et al. (2022), like Roy et al. (2021) also sample data from the Funcom dataset (LeClair et al., 2019), but only ask raters to rate the reference summaries and summaries generated by a single baseline, ATTENDGRU (LeClair et al., 2019), an encoder-decoder RNN which takes only the textual code data as input.

Su et al. (2024), instead of drawing from a large dataset of open-source code, uses human-written reference summaries collected by Bansal et al. (2024) where programmers were asked to summarise the purpose of the method in the project. They evaluate the human references, as well as five different methods for generating code summaries:

- GPT4-BASE (Su et al., 2024) Summaries obtained by prompting GPT-4.
- GPT4-CONTEXT (Su et al., 2024) Summaries obtained by prompting GPT-4 given summaries of all the functions that call it in the code (these summaries were also automatically generated by the model).
- GEMINI-BASE (Su et al., 2024) Same as GPT4-BASE, but Gemini is prompted instead.
- GEMINI-CONTEXT (Su et al., 2024) Same as GPT4-CONTEXT, but Gemini is prompted instead.
- JAM-FT (Su et al., 2024) Fine-tuned version of JAM (Su et al., 2023) based on the outputs of GEMINI-CONTEXT.

Table 8: Human Evaluation Datasets

Source	**Language**	**Size**	**Methodology**	**Evaluator Background**	**Link**
(Haque et al., 2022)	Java	6,300: 210 summary pairs evaluated 30 times each	Rate similarity between generated and reference summary and accuracy, content adequacy, conciseness (4-point scale: strongly agree to strongly disagree).	Professional Java developers with an average of 9.3 years experience x 30	github
(Roy et al., 2021)	Java	6,894: 2,298 summaries evaluated 3 times each	Rate conciseness, fluency, adequacy (1-5), rate overall (0-100).	Professional developers x 48, academics x 61, graduate students x 87, undergraduate students x 17, others x 13	github[a]
(Gao et al., 2023)	Java, Python	3,000: 500 Java and 500 Python summaries evaluated 3 times each	Rate conciseness, fluency and adequacy (1-5)	Professional developers with >4 years experience x 10	github
(Su et al., 2024)	Java	2400: 240 summary pairs evaluated 10 times each	Rate informativeness: whether 'the summary contains information that helps to understand why the method exists in the project'. (4-point scale: strongly agree to strongly disagree).	UK/US residents with a Computer Science degree, and at least 1 year of Java experience x 60	Direct contact with authors[b]

[a] A more complete version of the dataset including the original methods and summaries is made available by (Mastropaolo et al., 2024) at https://github.com/antonio-mastropaolo/code-summarisation-metric/tree/main

[b] They have indicated that they will release the dataset publicly soon.

```
You are a professional software engineer.
Evaluate the statement by responding ‘Strongly
agree’, ‘Somewhat agree’, ‘Somewhat disagree’
or ‘Strongly disagree’. Independent of other
factors, I feel the new summary is accurate.

Reference summary: {Reference Summary}
Function:{Original Function}
Generated summary: {Generated Summary}
What are the steps you would take to evaluate
this statement? Show your steps and then
provide an evaluation (Strongly agree, Somewhat
agree, Somewhat disagree or Strongly disagree).
```

Figure 4: Ask LLM Directly Final Prompt

E Other Variations

E.1 Prompt Variations

Figure 4 shows the final prompt used. In the reference free case, the "Reference Summary" line is left out.

We varied the Ask Claude Directly prompts in six different ways: the quality dimension definition the summary was to be rated on, the role definition for role-based prompting, the format of the expected response, whether chain-of-thought prompting was used, whether the reference summary was included and whether the reference code was included.

E.1.1 Quality Dimensions

Consistent-1 Rate how consistent the following summary is with the corresponding function and reference summary. Note that consistency means that all the information in the new summary is supported by the code [or the reference summary, when provided].

Consistent-2 The following summary is consistent. Note that consistency means that all the information in the new summary is supported by the code [or the reference summary, when provided].

Accurate-POS Independent of other factors, I feel the new summary is accurate.

Accurate-NEG Independent of other factors, I feel the new summary is inaccurate.

Adequate-POS The new summary contains all of the important information required for understanding the method.

Adequate-NEG The new summary is missing important information, and that can hinder the understanding of the method.

Concise-POS The new summary only contains necessary information.

Concise-NEG The new summary contains a lot of unnecessary information.

Informative-1 The summary contains information that helps to understand why the method exists in the project

Informative-2 Independent of other factors, I feel that the new summary contains relevant information that helps to understand why the method exists in the project

E.1.2 Role Definitions

Software Engineer You are a professional software engineer.

Professor You are a Professor of Computer Science at a reputable university.

E.1.3 Response Options

We often mentioned the response options multiple times in the prompt. In italics is the location of the that particular piece of text which can be cross-referenced with the prompt scaffolds in subsubsection E.1.5.

1-5 *After Data* Rating (1 to 5):

0-100 *Before Data* Give a rating from 0 to 100 where 0 means completely inconsistent and 100 means the summary is fully consistent with the code or the reference summary.
After Data Rating (0 to 100):

Agree-Disagree *Before Quality Dimension* Evaluate the statement by responding ‘Strongly agree’, ‘Somewhat agree’, ‘Somewhat disagree’ or ‘Strongly disagree’.
After Data Evaluation (Strongly agree, Somewhat agree, Somewhat disagree or Strongly disagree):

Agree-Disagree + Chain of Thought *Before Quality Dimension* Evaluate the statement by responding ‘Strongly agree’, ‘Somewhat agree’, ‘Somewhat disagree’ or ‘Strongly disagree’.
After Data What are the steps you would take to evaluate this statement? Show your steps and then provide an evaluation (Strongly agree, Somewhat agree, Somewhat disagree or Strongly disagree):

Agree-Neutral-Disagree + Chain of Thought *Before Quality Dimension* Evaluate the statement by responding (Strongly agree, Somewhat agree,

	Quality Dimension	Role Definition	Response Options	Chain of Thought	Reference Summary	Reference Code
consistency-no-ref	Consistent-1	✗	1-5	✗	✗	✓
consistency-no-ref-code	Consistent-1	✗	1-5	✗	✓	✗
consistency-1-5	Consistent-1	✗	1-5	✗	✓	✓
consistency-0-100	Consistent-1	✗	0-100	✗	✓	✓
consistency-agree-disagree	Consistent-2	✗	Agree/Disagree	✗	✓	✓
accuracy	Accurate-POS	✗	Agree/Disagree	✗	✓	✓
adequacy-neg	Adequate-NEG	✗	Agree/Disagree	✗	✓	✓
conciseness-neg	Concise-NEG	✗	Agree/Disagree	✗	✓	✓
adequacy	Adequate-POS	✗	Agree/Disagree	✗	✓	✓
conciseness	Concise-POS	✗	Agree/Disagree	✗	✓	✓
accuracy-sftw-eng	Accurate-POS	Soft. Eng.	Agree/Disagree	✗	✓	✓
accuracy-professor	Accurate-POS	Professor	Agree/Disagree	✗	✓	✓
Final Method	Accurate-POS	Soft. Eng.	Agree/Disagree	✓	✓	✓
accuracy-neg	Accurate-NEG	✗	Agree/Disagree	✗	✓	✓
accuracy-sftw-eng-cot-neutral	Accurate-POS	Soft. Eng.	Agree/Neutral/Disagree	✓	✓	✓
accuracy-sftw-eng-cot-no-ref	Accurate-POS	Soft. Eng.	Agree/Disagree	✓	✗	✓
informative-sftw-eng-cot	Informative-1	Soft. Eng.	Agree/Disagree	✓	✓	✓
informative2-sftw-eng-cot	Informative-2	Soft. Eng.	Agree/Disagree	✓	✓	✓

Table 9: Variants tested for Ask-Claude

	Adequacy	Conciseness	Fluency
consistency-no-ref	0.59	0.32	0.40
consistency-no-ref-code	0.55	0.31	0.36
consistency-1-5	0.58	0.35	0.40
consistency-0-100	0.59	0.30	0.37
consistency-agree-disagree	0.58	0.41	0.42
accuracy	0.59	0.38	0.43
adequacy-neg	0.46	0.27	0.31
conciseness-neg	-0.37	-0.32	-0.40
adequacy-pos	0.60	0.33	0.37
conciseness	0.59	0.35	0.41
accuracy-sftw-eng	0.60	0.37	0.43
accuracy-professor	0.58	0.37	0.43
Final Method	0.64	0.32	0.43
accuracy-neg	0.16	-0.01	0.03
accuracy-sftw-eng-cot-neutral	0.60	0.34	0.40
accuracy-sftw-eng-cot-no-ref	0.60	0.31	0.34

Table 10: Variants tested for Ask-Claude: Spearman's Correlation with Adequacy, Conciseness and Fluency on Gao et al. training dataset

	Informativeness
Final Method	0.30
informative-sftw-eng-cot	0.29
informative2-sftw-eng-cot	0.23

Table 11: Quality Dimension Variants tested for Ask-Claude on Su et al. dataset training dataset

Neutral, Somewhat disagree or Strongly disagree). *After Data* What are the steps you would take to evaluate this statement? Show your steps and then provide an evaluation (Strongly agree, Somewhat agree, Neutral, Somewhat disagree or Strongly disagree):

E.1.4 Data Provided

Reference Summary and Reference Code
Reference summary:{reference summary}
Function:
{reference method}
Generated summary: {generated summary}

Reference Summary Only
Reference summary: {reference summary}
Generated summary: {generated summary}

Reference Code Only
Function:
{reference method}
Generated summary: {generated summary}

E.1.5 Prompt Scaffold

`[Role Definition] [Response Options: Before Quality Dimension] [Quality Dimension] [Response Options: Before Data]`

```
[Data Provided]
[Response Options: After Data]
```

E.2 Question-Answering Variation

We also tried to use Claude as part of a Question-Answering-style metric, inspired by previous work in text summarisation (e.g. QAGS (Wang et al., 2020), QAEval (Deutsch et al., 2021a), QuestEval (Scialom et al., 2021) and QAFactEval (Fabbri et al., 2022)), but this approach did not end up providing any improvements compared to the n-gram based metrics. The main idea is that after reading a good generated summary you should be able to answer questions about the subject similarly to if you had read the reference summary instead. The details of our approach are as follows:

1. Find all noun phrases in the reference summary using spaCy (en_core_web_sm).
2. Generate questions by replacing each noun phrase with a gap. For example, for the summary "`returns the label text for the given element`", one question would be "`returns the _ _ _ for the given element`".
3. Given only the generated summary, prompt an LLM to try to fill in the blank for each question generated in Step 2. We used Claude 3 Opus.
4. Compare the correct answers with the responses generated by the model by converting each answer to an embedding (we used gte-base-en), and calculating cosine similarity.
5. Return the mean of the cosine similarity scores for each question as the final score.

The main difference between our approach and the standard approach for text summarisation is Step 2, question generation, as we programmatically generate the questions as a fill-in-the-blank rather than ask an LLM to generate the questions given the reference code and summary. The reason for this change was because we found that the questions generated were too specific (e.g., 'What does the variable `i` do?'), whose answers don't appear in a code summary.

E.2.1 QA Prompt

```
Based on the following code summary, fill in
the blanks for the other code summary based on
the same function.

For example:
Code Summary: 'get the list of the user'
Question:    'returns   ___   of   collaborate
collections for the given user id'
Answer: 'the list'

Code Summary: {Generated_summary}
Question: {Question}
Answer:
```

Figure 5: Question Answering Prompt for Question Generation Step

E.2.2 Variants

We tested many different variants on the Gao et al. dataset, varying seven different aspects of the process. The combinations tested are provided in Table 12, and the results are in Table 13. We outline each of the variations below:

+ verb phrases In the answer selection step, include all tokens whose part of speech is 'VERB' as well as all of the 'noun chunks', as outputted by `SPaCY en_core_web_sm`.

- verb phrases In the answer selection step, only include the 'noun chunks' from the summary.

+ few shot Include an example(s) of the expected output in the prompt. The exact wording depends on 1) the return format and 2) if n.a. is an option if there is not enough information.

JSON prompt (with n.a.):

```
Based on the following code summary,
fill in the blanks for the other
code summary based on the same function.

For example:
Code Summary: 'get the list of the user'
Question: returns ___ of collaborate
collections for the given user id
Answer:{
    "answers": [
        "the list"
    ]
}
```

Return as word only prompt (without n.a.):

	+verb phrases	+few shot	+return as word only	+n.a. if not enough information	+use whole result sentence for similarity	+use gte-base-en for embeddings	+handle n.a. differently
Final QA Method	✗	✓	✓	✗	✗	✓	n.a.
NA-counts-as-0	✗	✓	✓	✓	✗	✓	✓(n.a. = 0)
NPs-Only	✗	✓	✓	✓	✗	✓	✗
NPs-Only-JSON	✗	✓	✗	✓	✗	✓	✗
NA-counts-as-0.5	✗	✓	✓	✓	✗	✓	✓(n.a. = 0.5)
NA-not-counted	✗	✓	✓	✓	✗	✓	✓(exclude n.a.)
NPs+VPs	✓	✓	✓	✓	✗	✓	✗
NPs+VPs-SBERT	✓	✓	✓	✓	✗	✗	✗
NPs-Only-zero-shot	✗	✗	✓	✓	✗	✓	✗
full-sent-SBERT	✓	✓	✓	✓	✓	✗	✗
full-sent-GTE	✓	✓	✓	✓	✓	✓	✗

Table 12: Variants tested for Question Answering (Ordered from best to worst on Accuracy on Gao et al. (Java) train dataset)

	Adequacy	Conciseness	Fluency
Final QA Method	**0.55**	0.11	0.20
NA-counts-as-0	0.54	0.17	0.25
NPs-Only	0.54	0.16	0.22
NPs-Only-JSON	0.53	0.17	0.25
NA-counts-as-0.5	0.53	0.14	0.20
NA-not-counted	0.53	0.14	0.20
NPs+VPs	0.53	**0.21**	0.28
NPs+VPs-SBERT	0.49	0.20	**0.29**
NPs-Only-zero-shot	0.46	0.13	0.22
full-sent-SBERT	0.45	0.11	0.22
full-sent-GTE	0.44	0.07	0.16

Table 13: Variants tested for Question Answering: Spearman's Correlation with Adequacy, Conciseness and Fluency on Gao et al. training dataset

```
Based on the following code summary,
fill in the blanks for the other
code summary based on the same function.

For example:
Code Summary: `get the list of the user`
Question: `returns ___ of collaborate
collections for the given user id`
Answer: `the list`
```

- few shot In the no few shot scenario, the prompt just includes an explanation of the task, e.g.

```
Use only the information from the code
summary to fill in the blank on the
following question. If there is not enough
information to give an answer, write 'n.a.'.
```

+ return as word only Prompt the LLM to just return the answer to the question.

```
Only provide your answer in the response.
... {the summary and the question} ...
Answer:
```

- return as word only Prompt the LLM to return the answer to the question in JSON format.

```
Return your answer in json format,
for example
{
    "answers": [
        "answer"
    ]
}
```

+ n.a. if not enough information Prompt the LLM to return n.a. if it thinks there is not enough information in the summary to answer the question.

- n.a. if not enough information Do not specify how to handle situations where there is not enough information in the summary to answer the question.

+ use whole result sentence for similarity The response from the LLM was re-inserted back into the blank, and then the embedding of this full sentence was compared against the embedding of the original summary.

- use whole result sentence for similarity Only the single word that was generated and the original word that should have filled the blank were compared.

+ use gte-base-en for embeddings Embeddings for each of the generated and reference answers were calculated using `gte-base-en-v1.5`.

- use gte-base-en for embeddings Embeddings were instead calculated with `SentenceBERT`.

+ handle n.a. differently When the model responded with n.a., these questions were automatically assigned a predetermined similarity score (0 or 0.5) or excluded from the final calculation completely.

- handle n.a. differently For cases where the model responded with n.a., the contribution of n.a. to the final score was just the cosine similarity of the expected answer and the string 'n.a.', i.e. handled the same as all other answers.

F Claude Evaluating Itself

We asked Claude to rate its own output. The summaries were generated with the prompt in Figure 6. They were evaluated using the Ask-Claude consistency-agree-disagree prompt on the Gao et al. Java training dataset. The results are in Figure 7.

```
Write a comment that summarises the following
code. Ensure that it is fully consistent, so
all information in the comment is supported by
the code.
Function: {Function}
Comment:
```

Figure 6: Prompt given to Claude for Summary Generation

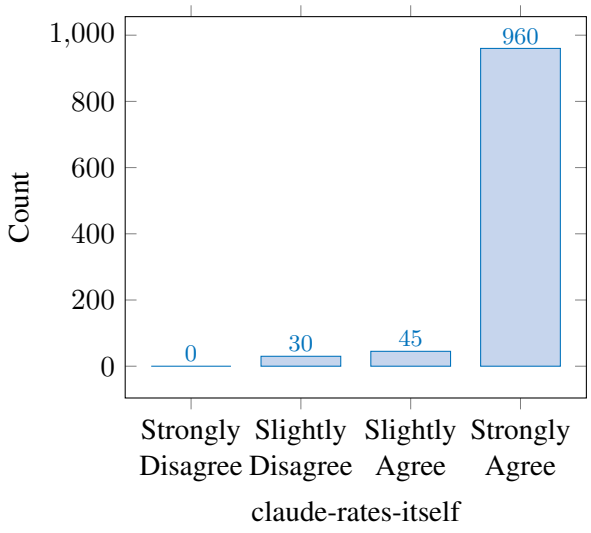


Figure 7: Ask-Claude scores for summaries generated by Claude 3.5 (for functions from the Gao et al. dataset)

G Correlation with Comment Length

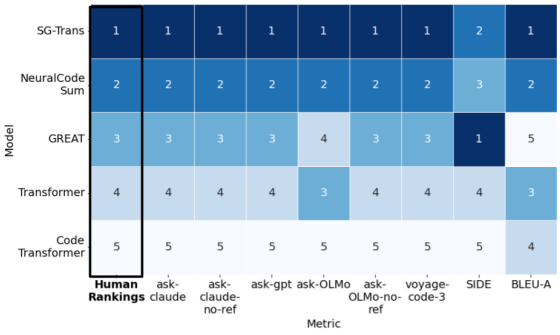
See Table 14.

H Relative Rankings of Each Model by Metric

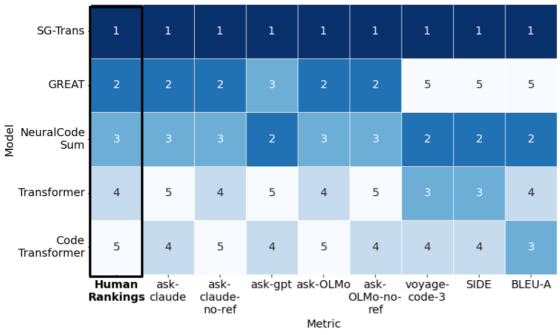
See Figure 8.

	Roy et al.	Haque et al.	Gao et al. (Java)	Gao et al. (Python)	Su et al.
adequacy	0.00	0.15	-0.04	0.00	n.a.
conciseness	-0.08	-0.08	-0.36	-0.02	n.a.
fluency	-0.02	n.a.	-0.15	0.06	n.a.
accurate	n.a.	0.06	n.a.	n.a.	n.a.
similarity	n.a.	0.27	n.a.	n.a.	n.a.
overall score	-0.01	n.a.	n.a.	n.a.	n.a.
informativeness	n.a.	n.a.	n.a.	n.a.	-0.41
BLEU-A	0.20	0.43	-0.02	0.05	-0.07
METEOR	0.16	0.32	-0.08	0.02	0.06
ROUGE-L	0.03	0.25	-0.20	-0.05	-0.35
SIDE	0.02	0.16	-0.21	-0.01	-0.49
SentenceBERT	0.09	0.35	-0.13	0.02	0.17
gte-base-en	0.07	0.29	-0.12	0.01	-0.03
voyage-code-3	0.06	0.33	-0.10	0.00	-0.01
ask-OLMo	-0.08	0.14	-0.15	-0.05	0.25
ask-OLMo-no-ref	-0.13	0.14	-0.18	-0.01	0.20
ask-gpt	-0.03	0.15	-0.24	0.06	0.22
ask-claude	-0.01	0.13	-0.12	-0.01	0.26
ask-claude-no-ref	-0.11	0.13	-0.19	-0.03	0.25

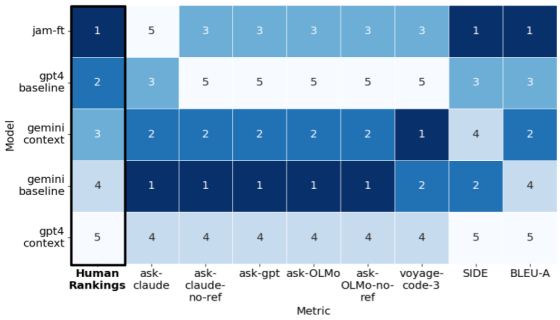
Table 14: Correlation with Comment Length



(a) Model Rankings by Metric (Gao et al. Java, Adequacy)



(b) Model Rankings by Metric (Gao et al. Python, Adequacy)



(c) Model Rankings by Metric (Su et al., Informativeness)

Figure 8: Relative Rankings of Each Model by Metric