大規模言語モデルを用いたソースコードからのドキュメント生成能力調査

杉山 咲 [1]　蒔苗 茉那 [2]　片山 歩希 [2]　坂井 優介 [2]　山口 賢一 [1]　渡辺 太郎 [2]
[1] 奈良工業高等専門学校　[2] 奈良先端科学技術大学院大学
AI1184@nara.kosen-ac.jp　yamaguti@info.nara-k.ac.jp
{makinae.mana.mh2, katayama.ayuki.kc1, sakai.yusuke.sr9, taro}@is.naist.jp

概要

ソフトウェアシステムの品質向上には，可読性が高いプログラムが必要不可欠である．特に，処理内容や要素の説明が記述されているコメントやドキュメントは可読性の向上に役立つ．しかし内容の一貫性や明確さ，正確さがの保証が困難であり，高品質なドキュメントの作成には多くの課題がある．またドキュメントが未記載のコード資源も大量に存在しており，コード資源の可用性向上にはドキュメントの整備が必要となる．そこで本稿では，大規模言語モデル (LLM) が可読性の高いドキュメントを生成する能力があるか，Python の関数データからドキュメンテーション文字列である Docstring を LLM で生成し調査する．複数の観点による分析の結果，LLM は概ね形式的であり可読性が高い Docstring が生成可能な一方で，可読性が低いコードに対しては意図しない書き換えなどを行うことが明らかとなった．

1　はじめに

大規模言語モデル (LLM) の発展や Copilot [1] などの開発支援ツールの登場によってソースコードの作成が容易になった．これによりソフトウェアの開発効率が大きく向上し，開発ハードルが大きく低下した [2]．しかしソフトウェアの開発が容易になった一方で，ソースコードの品質については保証されていない [3, 4, 5]．ソフトウェア品質の向上にはソフトウェアの保守性，明瞭性，一貫性が重要であり，これらを満たすにはソースコードの可読性向上が不可欠である [6, 7, 8]．また, ソースコード解読はシステム開発，保守において最も時間のかかる作業であるため，可読性の向上はシステム開発効率の向上にも繋がる [6, 9, 10, 8, 11]．ソースコードの可読性を向上させる手段の一つとして，概要が網羅的に記述されているドキュメントの作成が有用である [12]．

プログラム言語の Python では Docstring というドキュメンテーション文字列が用意されている [13]．各クラスや関数における概要，入出力要素や処理内容といった情報を Docstring 内に記述することで，開発者のコード理解を補助できる [6, 14]．しかし Docstring は自然言語で記述するため，開発者や開発環境によって記載する内容や量が異なり，可読性も変化する．また，Docstring が未整備のコード資源も多く存在している．Docstring やドキュメントが不十分な関数の処理内容を解読し完全に理解するには多くの時間を要し，開発効率の低下やプログラムの誤用につながる [6]．つまり，開発者に必要な情報を適切に伝えるためには，情報が適切かつ明確な Docstring を付与し，可読性を向上させる必要がある．そこで開発者間の記述内容の揺れを軽減し記述内容の明瞭性を向上させるために，Docstring には Google スタイルや reStructuredText といった記述スタイルが複数考案されている．さらに組織ごとに独自の記述スタイルを作成している場合もある．しかし，一見形式的に記述されている Docstring の中でも，記述ミスや開発者間で採用するスタイルや説明内容の粒度の違いなどにより，内容の一貫性が保証できない．また，Docstring 自体は自然言語であるため，記述や形式のミス検出が困難といった問題がある．

そこで本稿では，オープンソースプロジェクト内の Python の関数から LLM により Docstring の自動生成を行い，生成内容が関数内の要素と適切に対応しているか分析することで，LLM のドキュメント生成能力の調査を行う．ソースコードからドキュメントを生成する関連研究（付録 A で詳述）では正確性や簡潔性，類似性といった観点から評価を行うが，本稿では記述スタイルに着目し，生成される Docstring

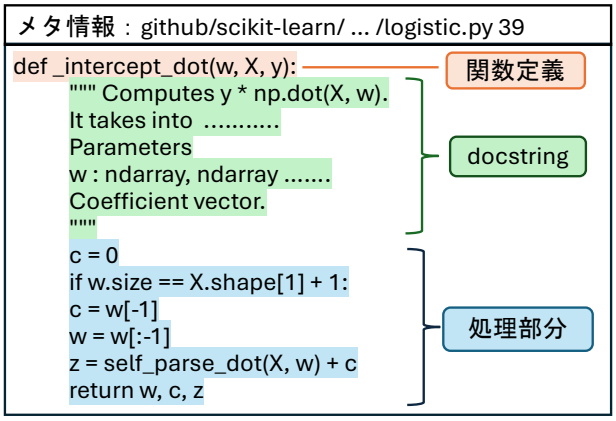

図 1: コーパス内のデータ構成

の一貫性や忠実性，頑健性に関する観点から調査を行う．はじめに自動で生成された場合の記述スタイルの偏りについての調査を行うことで LLM によって生成される Docstring の傾向について考察する．次にスタイルを指定して Docstring を生成した場合，指定した記述スタイルに即した Docstring が適切に生成されるか調査する．最後に，生成条件や対象の関数に対する頑健性を明らかにするため，関数名・変数名を内容と全く関係のない名前にした場合に生成される Docstring の内容に影響があるかについても分析した．検証の結果，LLM は概ね形式的であり可読性が高い Docstring が生成可能である一方で，可読性が低いコードに対しては意図しない書き換えなどを行うことが明らかとなった．

2 実験

本稿での実験として，LLM に Python の関数を入力として Docstring を生成させ，生成内容が関数の内容に対して適切かを，関数定義および内部処理に対する Docstring の生成（§ 2.2），スタイル指定を伴う Docstring 生成（§ 2.3），可読性が下がった状態での Docstring 生成（§ 2.4）の 3 つの観点から分析した．

2.1 検証に使用する LLM とコーパス

実験では，検証対象の LLM として GPT-4o mini（gpt-4o-mini-2024-07-18）を用いる．Python の function に関連する情報のコーパスとして，code-docstring-corpus[1] [15] を使用した．このコーパスは，code2doc（コードからのドキュメント生成）および doc2code（ドキュメントからのコード生成）のタスクを想定し，オープンソースリポジトリ内のコードを収集して作成された．コーパスのデータ構成を図 1 に示す．図 1 に示すように code-docstring-corpus は一つの関数に対して，宣言文とコード本体，docsting が付与されている．このコーパスは実際のソフトウェア内のソースコードを用いているため，関数外の情報といったノイズを含むデータが大多数を占めていた．そこで，前処理として訓練データから Docstring 内で関数の引数や返り値について言及している 260 個のデータを抽出し，検証に使用した．

表 1: 関数と生成した Docstring の引数の比較

結果	完全一致	不一致	解析失敗 (Docstring)	解析失敗 (関数)
引数の数	116	0	133	9

2.2 LLM の Docstring 生成能力の調査

実験内容 LLM に関数定義と内部処理を入力したとき，適切なドキュメントの生成能力をどの程度有しているか検証する．検証は 2 段階で行う．はじめに，任意の関数定義と内部処理を LLM へ入力し，対応する Docstring の生成を行う．LLM によって生成された Docstring の内容を解析し，記述内容が関数定義内の引数情報と一致するか確認することで評価を行う．次に，LLM が生成する Docstring の記述スタイルの傾向分析を行う．Docstring の記述スタイルの判断は，Docstring の構文解析ツールである docstring parser[2] を用いる．指定したスタイルに沿った構文解析を行い，解析ができたスタイルを生成された Docstring の記述スタイルとする．記述スタイルとして docstring parser が対応している Epydoc[3]，Google スタイル[4]，Numpydoc[5]，reStructuredText[6] の 4 スタイルを実験の対象とした．各スタイルの特徴は付録 B に示している．図 2 に本検証に使用するプロンプトを示す．内容は，docstring の生成指示と，考慮すべきコーパス内の記述ルールを記述している．図 2 に示すプロンプトを LLM へ入力すると，Docstring が付与された関数が生成されるため，Docstring の記載箇所のみ抽出し，分析を行う．

実験結果 260 個の関数データに対して不具合なく生成できた 258 個の Docstring を対象に分析を

1) https://github.com/EdinburghNLP/code-docstring-corpus

2) https://pypi.org/project/docstring_parser/
3) https://epydoc.sourceforge.net/manual-fields.html
4) https://google.github.io/styleguide/
5) https://numpydoc.readthedocs.io/en/latest/format.html
6) https://www.sphinx-doc.org/ja/master/usage/restructuredtext/index.html

< System > Task: As a code documentation assistant, your goal is to document the following code snippet at the function level. You should place comments directly under the 'def' statement of each function. Additionally, generate appropriate Docstring for the functions provided by the user.
Important Rules:

- The input functions will follow these rules:
  - "DCNL" represents a newline.
  - "DCSP" represents a space ( ).

The output should include the entire code with the added documentation.
The code is as follows: < 対象の関数 >

図 2: § 2.2 の実験で使用したプロンプト

表 2: 生成された Docstring の記述スタイルの比較

スタイル	Epydoc	Google	Numpy	RST	解析失敗
合致数	0	122	1	0	135

行った．表 1 に，入力した関数定義内の引数情報と，生成した Docstring に記載された引数の完全一致数を示す．表 1 の結果から，docstring parser で構文解析ができた Docstring については，引数情報の対応が適切に取れていることがわかる．次に，Docstring 生成データに対してスタイルの偏りを分析した．表 2 に構文解析の結果を示す．結果として，LLM に対してスタイルを指定せずに Docstring を生成した場合，Google スタイルに偏った出力がされる傾向があるものの，解析ができない Docstring が過半数だった．

解析が失敗した Docstring は，全て google スタイルがベースになっていた．しかしセクション名がスタイルのルールに沿っていないものや，他のスタイルが部分的に混じっているものが確認された．具体的な例は付録 C を参照．これらの揺れにより解析がうまくいかなかったものと考えられる．しかし全ての生成結果において，形式的に関数や要素に対する説明が記述されていた．

この検証から，LLM が生成したドキュメントは関数の内容を網羅的に記述していると言える．そして生成されるスタイルは基本的に Google スタイルに近く，可読性が高いことが示された．

2.3 スタイルの指定を伴う Docstring 生成

実験内容 プロンプト中で指定された Docstring の記述スタイルに沿って，適切に Docstring が生成可能か検証を行った．実験に使用するプロンプトを

< System > Task:As a code documentation assistant, your goal is to document the user's code snippet at the function level.
You should place comments directly under the 'def' statement of each function. Additionally, generate appropriate docstrings for the functions provided by the user.
Important Rules:

- Docstrings style is "< スタイル名 >".
- The input functions will follow these rules:
  - "DCNL" represents a newline.
  - "DCSP" represents a space ( ).

The output should include the entire code with the added documentation.

< User > The code snippet is < 対象の関数 >.

図 3: § 2.3 の実験で使用したプロンプト

表 3: 指定したスタイル通りに生成ができているか

指定スタイル	Epydoc	Google	Numpy	RST
沿っている	260	255	260	260
沿っていない	0	5	0	0

図 3 に示す．2.2 節の実験同様に，LLM によって生成された Docstring に対して，docstring parser によって構文解析を行い，解析可能な記述スタイルを当該スタイルとした．

実験結果 表 2 に指定した記述スタイルで Docstring が生成できていた個数を示す．表 2 から，Google スタイルを除いて，LLM はプロンプトで指定した記述スタイル通りに Docstring が生成できていることがわかった．一方，Google スタイルの場合のみ 5 つの事例について Docstring の解析が失敗した．解析が失敗した事例を図 4 に示す．Google スタイルでは Args, Raises, Returns それぞれに対して要素とその説明をコロンで接続することで記述する．一方，これら解析が失敗した例では図 4 のように各要素の説明途中で改行が含まれるため，解析が失敗していた．そのため，改行が含まれている不備を除けば，Google スタイルでもすべての事例で解析可能といえる．したがって LLM は指定されたスタイルに即した Docstring 生成が行えている．

2.4 可読性が低い関数への Docstring 生成

実験内容 LLM による Docstring 生成を行うにあたり，関数の可読性による影響を調査した．関数の可読性に影響されず内容に適した Docstring が生成

```
Args:
    parser: The template parser instance.
    token: The token representing the 'ssi' tag.
Raises:
    TemplateSyntaxError: If the number of arguments is not 2 or 3,
    or if the second argument is not 'parsed' when provided.
Returns:
    SsiNode: An instance of SsiNode with the compiled file path
    and parsed flag.
```

" or if the second argument is not 'parsed' when provided. " にコロン(：)が含まれていません.

図 4: Google スタイルで解析が失敗した例

```
def libvlc_video_get_spu_description(p_mi):
    f = (_Cfunctions.get('libvlc_video_get_spu_description', None)
or _Cfunction('libvlc_video_get_spu_description', ((1,),), None,
ctypes.POINTER(TrackDescription), MediaPlayer))
    return f(p_mi)
```

関数名，変数名置換

```
def aa(ab):
    ac = af.get('libvlc_video_get_spu_description', None) or
ad('libvlc_video_get_spu_description', ((1,),), None,
ah.POINTER(ag), ae)
    return ac(ab)
```

図 5: 変数名を無意味な文字列へ置換

できていれば，頑健性が高いと言える．

可読性が低い関数として，図 5 のように関数名と変数名全てを無意味な文字列に置き換えた関数を作成した．可読性が低下した関数から生成した Docstring 内の要素を置換前に戻し，元関数と引数要素が一致するかを確認することで評価を行った．実験に使用したプロンプトを図 6 に示す．スタイルは Google スタイルに統一した．

実験結果　表 4 に生成結果の分析を示す．表 4 より，変数名と関数名が処理内容に一切関係がない状態であっても過半数の事例ではほぼ正確に引数を取得することができている．一方，関数名に対する引数不一致を引き起こした 2 事例を分析すると，1 つ目は特殊変数である 'self' に対する説明の欠落によるものであり，置換前でも同様の問題が発生している事例だった．2 つ目の事例は関数内に関数が入れ子で定義されている場合であり，親関数の Docstring の引数と返り値に対して，子関数の情報のみが記述されていた．しかし，その他の記述は親関数の動作を適切に記述していた．このことから LLM は入れ子のような複雑な構造を有する関数への Docstring 生成に対して幻惑 [16, 17, 5] を誘発する場合があることが示唆される．また変数名の置換により発生した不一致では，表 5 に示すように，関数内に存在しない変数名が引数として Docstring に記述された．しかし返り値，例外処理，関数に対する説明は関数

< System > You are a great code reviewer.
Please generate docstring for the following function.
The docstrings should be in the **Google Style**.
You don't need to write the function itself, just the docstring.
please generate entire of the docstring as a json parameter.
The function is as follows:

< User > < 対象の関数 >

図 6: § 2.4 の実験で使用したプロンプト

表 4: 要素の置換に伴う生成結果

	置換前	関数名	変数名	関数&変数
引数完全一致	208	203	206	207
引数不一致	1	2	1	0
生成不備	49	53	51	50
スタイル通りでない	0	0	0	1

表 5: 生成時に生じた変数名の変化

	例 1	例 2
本来の名前	checkfile	writeLockCheck
置換後	aa	ab
生成された変数名	file_path	remove_lock

に対し適切であることから，関数名と内部の処理から LLM は関数の意図を汲み，無意味な変数名を常識に即した変数名に置き換えたものと考えられる．一方，変数名の書き換えが自動的に発生していることから，可読性の低いコードに関しては意図しない挙動を引き起こす可能性が示された．

3　おわりに

LLM による可読性および頑健性の高いドキュメントの自動生成を目的とし，Python の関数に対して適切な Docstring が生成可能かを調査した．実験結果から，関数の要素に対して高い精度でドキュメントが生成できることが明らかとなった一方，可読性が低い関数などに対しては意図しない変数名の書き換えが発生するなどの問題点も確認できた．

本稿では，主に Docstring のスタイル形式と引数の一致度に焦点を当てて分析を行った．そのため，Docstring 内に記述される例外処理や関数への説明など，自由度が高い他要素に対する定量的な分析が今後の課題となる．

参考文献

[1] Github copilot: Your ai pair programmer. https://github.com/features/copilot.

[2] Thomas Dohmke, Marco Iansiti, and Greg Richards. Sea change in software development: Economic and productivity analysis of the ai-powered developer lifecycle, 2023.

[3] Ruizhong Qiu, Weiliang Will Zeng, Hanghang Tong, James Ezick, and Christopher Lott. How efficient is llm-generated code? a rigorous high-standard benchmark, 2024.

[4] Yihong Dong, Xue Jiang, Zhi Jin, and Ge Li. Self-collaboration code generation via chatgpt. ACM Trans. Softw. Eng. Methodol., Vol. 33, No. 7, September 2024.

[5] Fang Liu, Yang Liu, Lin Shi, Houkun Huang, Ruifeng Wang, Zhen Yang, Li Zhang, Zhongqi Li, and Yuchi Ma. Exploring and evaluating hallucinations in llm-powered code generation, 2024.

[6] Xin Xia, Lingfeng Bao, David Lo, Zhenchang Xing, Ahmed E. Hassan, and Shanping Li. Measuring program comprehension: A large-scale field study with professionals. IEEE Transactions on Software Engineering, Vol. PP, pp. 1–1, 07 2017.

[7] K. K. Aggarwal, Yogendra Pratap Singh, and Jitender Kumar Chhabra. An integrated measure of software maintainability. Annual Reliability and Maintainability Symposium. 2002 Proceedings (Cat. No.02CH37318), pp. 235–241, 2002.

[8] Raymond P.L. Buse and Westley R. Weimer. Learning a metric for code readability. IEEE Transactions on Software Engineering, Vol. 36, No. 4, pp. 546–558, 2010.

[9] Lionel E. Deimel. The uses of program reading. SIGCSE Bull., Vol. 17, No. 2, p. 5–14, June 1985.

[10] Daniela Steidl, Benjamin Hummel, and Elmar Juergens. Quality analysis of source code comments. In 2013 21st International Conference on Program Comprehension (ICPC), pp. 83–92, 2013.

[11] Sarah Fakhoury, Yuzhan Ma, Venera Arnaoudova, and Olusola Adesope. The effect of poor source code lexicon and readability on developers' cognitive load. In Proceedings of the 26th Conference on Program Comprehension, ICPC '18, p. 286–296, New York, NY, USA, 2018. Association for Computing Machinery.

[12] Nuzhat J. Haneef. Software documentation and readability: a proposed process improvement. SIGSOFT Softw. Eng. Notes, Vol. 23, No. 3, p. 75–77, May 1998.

[13] David Goodger and Guido van Rossum. PEP 257 – Docstring Conventions. python.org, post-history:13-jun-2001 edition, May 2021. https://peps.python.org/pep-0257/.

[14] Bibek Poudel, Adam Cook, Sekou Traore, and Shelah Ameli. Documint: Docstring generation for python using small language models, 2024.

[15] Antonio Valerio Miceli Barone and Rico Sennrich. A parallel corpus of python functions and documentation strings for automated code documentation and code generation, 2017.

[16] Ziwei Ji, Nayeon Lee, Rita Frieske, Tiezheng Yu, Dan Su, Yan Xu, Etsuko Ishii, Ye Jin Bang, Andrea Madotto, and Pascale Fung. Survey of hallucination in natural language generation. ACM Comput. Surv., Vol. 55, No. 12, March 2023.

[17] Lei Huang, Weijiang Yu, Weitao Ma, Weihong Zhong, Zhangyin Feng, Haotian Wang, Qianglong Chen, Weihua Peng, Xiaocheng Feng, Bing Qin, and Ting Liu. A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions. ACM Trans. Inf. Syst., November 2024. Just Accepted.

[18] Vatsal Venkatkrishna, Durga Shree Nagabushanam, Emmanuel Iko-Ojo Simon, and Melina Vidoni. Docgen: Generating detailed parameter docstrings in python, 2023.

[19] Shubhang Shekhar Dvivedi, Vyshnav Vijay, Sai Leela Rahul Pujari, Shoumik Lodh, and Dhruv Kumar. A comparative analysis of large language models for code documentation generation, 2024.

[20] William Macke and Michael Doyle. Testing the effect of code documentation on large language model code understanding, 2024.

[21] Nicola Dainese, Alexander Ilin, and Pekka Marttinen. Can docstring reformulation with an llm improve code generation? In Conference of the European Chapter of the Association for Computational Linguistics, 2024.

A　関連研究

言語モデルによるソースコードから Docstring を自動生成に関する取り組みとして，Venkatkrishna ら [18] は，Python のソースコードと Docstring のペアを収集し，言語モデルの追加学習を行うことで，Docstring の自動生成を行った．品質を正確性・簡潔さ・明瞭さを基準として，参照文ありの自動評価を行っている．Shubhang ら [19] は，LLM ごとの Docstring の生成能力を比較を行っている．各 LLM が生成した Docstring に対して，生成結果の正確性，網羅性，コードの主題対する関連性，コード理解度の向上，ドキュメントの読みやすさ，生成時間といった要素を筆者ら自身による人手評価で比較している．これらの研究では，参照文として用意される Docstring との類似度比較など，自由生成される Docstring に対して人手評価を行っている一方，チーム開発現場で必要となる参照スタイルなどの基準に基づいた Docstring 生成の一貫性，忠実性などの観点からの検証は行われていない．

また Docstring からのソースコード生成に関する研究も行われており，主に Docstring の品質がコード生成結果へ与える影響について調査されている．Macke ら [20] は，一部誤りが含まれる Docstring から LLM によってソースコードを生成したときの生成傾向を分析しており，重大な誤りが含まれる場合，生成能力の低下が確認できる一方，記述内容の軽微な欠落のみであればコードの生成に大きな影響は与えないことを明らかにした．Dainese ら [21] は，LLM によるコード生成において，Docstring の内容が曖昧な場合，コードの生成能力の大幅な低下を引き起こす一方，Docstring 内に生成したいコードの具体的なヒントや正解コードの一部が含まれている場合，生成の精度が大幅に向上することを明らかにした．このように，ソースコード生成分野においては，Docstring など，LLM への入力文の品質や頑健性についての検証が行われている．

B　各記述スタイルごとの特徴

図 7 に Epydoc，図 8 に Google スタイル，図 9 に Numpydoc, 図 10 に reStructuredText における Docstring の記述スタイルの例をそれぞれ示す．それぞれの図中でハイライトされている箇所は上から順に，関数の概要（緑色），引数とその説明（赤色），返り値とその説明（青色）である．

```
def Epydoc(arg1, *args):
    """
    This is the place to describe the function and its description.
    It can contain multiple lines of text and can be written
    in markup language or other languages.

    @type  arg1: number
    @param arg1: This is a description of the parameter x to a function.
                 Note that the description is indented four spaces.
    @param *args: Other arguments.
    @rtype:  number
    @return: This is a description of the function's return value.
    """
```

図 7: Epydoc の記述例

```
def GoogleStyle(arg1, *args):
    """ Summarize the function in one line.

    This is an expanded description. Write a detailed description of
    the function.It may span multiple lines.

    Args:
        arg1(int) : Explanation of arg1. Type description is optional.
        *args : Other arguments.

    Returns:
        type: Explanation of anonymous return value of type ``type``.
    """
```

図 8: Google スタイルの記述例

```
def Numpydoc(arg1, *args):
    """ Summarize the function in one line.

    Several sentences providing an extended description. Refer to
    variables using back-ticks, e.g. `var`.

    Parameters
    ----------
    arg1 : int
        Explanation of arg1.
    *args : iterable
        Other arguments.

    Returns
    -------
    type
        Explanation of anonymous return value of type ``type``.
    """
```

図 9: Numpydoc の記述例

```
def reStructuredText (arg1, *args):
    """ Summarize the function in one line.

    This is an expanded description. Write a detailed description of
    the function.It may span multiple lines.

    :param arg1: This is a description of the parameter x to a function.
    :type arg1: int
    :param *args: Other arguments.
    :return: This is a description of the function's return value.
    :rtype: number
    """
```

図 10: reStructuredText の記述例

```
"""
Finds tracks that match the specified query criteria.

    Parameters:
    - tracks: List of track objects to search through.
    - query: A dictionary containing the search criteria. If None, an empty query is created.
    - limit: The maximum number of results to return. Defaults to 100.
    - offset: The starting index for the results. Defaults to 0.
    - uris: Optional parameter for specific URIs to filter by.

    Returns:
    - A SearchResult object containing the matching tracks.
"""
```

図 11: スタイル識別失敗の結果例

C　§ 2.2 における解析失敗例

図 11 に解析が失敗した関数の例を示す．赤いハイライトの箇所が Google スタイルに適していない．