SelfCheckGPT はコード生成における ハルシネーションを検知できるか

伊東 和香 [1]　小原 有以 [1]　佐藤 美唯 [1]　秋信 有花 [3]　倉林 利行 [3]　倉光 君郎 [2]
[1] 日本女子大学大学 理学研究科 [2] 日本女子大学 理学部
[3]NTT ソフトウェアイノベーションセンタ
m2016013iw@ug.jwu.ac.jp kuramitsuk@fc.jwu.ac.jp

概要

大規模言語モデル (Large Language Model, LLM) によるコード生成は，ソフトウェア開発の効率化に寄与する技術として注目されている．しかし LLM の出力にはハルシネーションを含むことが問題となる．ハルシネーションを含む出力コードはエラーを引き起こす可能性があるため，事前にハルシネーションを検知することが重要である．

本研究では，自然言語におけるハルシネーション検知手法である SelfCheckGPT をコード生成に適用する．一般的なコード生成の評価手法である実行ベース評価と，SelfCheckGPT によるコードの評価を比較することで，両者の関連性を確認する．

実行ベース評価と比較した結果，特に BLEU, ROUGE-L，EditSim を利用した SelfCheckGPT による評価において，実行ベース評価との関連性が見られた．

1　はじめに

生成 AI の進展に伴い，モデルの出力の信頼性を検証する手法が注目を集めている．その中で，SelfCheckGPT[1] は，大規模言語モデル（LLM）が生成するテキスト内のハルシネーション (事実に基づかない情報）を検出するための手法である．外部データベースを必要とせず，同じ入力に対して複数の出力をサンプリングし，それらの一貫性から事実性を評価できる点が特徴である．

本研究の目的は，SelfCheckGPT によるハルシネーション検出をコード生成タスクに適用し，コード生成の正確さと比較することである．コード生成タスクでは，HumanEval 等のベンチマーク [2, 3, 4] で導入された実行ベース評価，すなわち生成されたコードを実行し，テストケースの通過率を測定する手法が正確さの評価手法として用いられてきた．我々の関心は，実行ベース評価の代わりに，SelfCheckGPT のハルシネーション検知はどの程度，有効であるか調査することである．

過去，SelfCheckGPT は，自然言語タスクに適用され，有用性 [5, 6, 7] が検証されてきたが，コード生成タスクにおいては検証されていない．もし SelfCheckGPT でコード生成のエラーを予測できるのであれば，ソフトウェア開発現場に対し，テストケースの作成コスト [8, 9, 10] の軽減などの貢献が期待できる．他方，もし SelfCheckGPT の有効性が限定的である場合，自然言語とプログラミング言語の構造的な差異に対して有益な示唆が期待できる．

2　コード生成とハルシネーション

ハルシネーションの分類は，ハルシネーションの原因や解決策を探るために重要であり，コード生成においてもハルシネーションの分類が行われている．

Liu ら [11] は，3,084 件のハルシネーションを含むコードを調査し，その種類を体系的に分類した．表 1 にその結果を示す．これらのハルシネーションを含むコードのうち，全テストケースに通過するコードの割合は 10%以下に止まることが示された．また，特に発生割合の高いハルシネーションである Intent Conflicting と Inconsistency では全テストケースを通過する割合が 2%を下回ることが確認された．これらの結果より，ハルシネーションを含むコードの多くが，機能的に正しく実行できないコードであることが結論づけられている．

以上より，我々はコード生成におけるハルシネーションとは，コードが正しく実行できるかどうかに類似するという立場を採用する．

表 1 コード生成におけるハルシネーションの分類

分類		発生割合 (%)	説明
Intent Conflicting		32.1	プロンプトの要求と生成コードの動作が異なる
Context Deviation	Inconsistency	31.8	プロンプトの要求に準じてはいるが，要求の実現を妨げるコードが存在する
	Repetition	17.3	プロンプトの内容を繰り返している，または生成コード内に繰り返しが生成している
	Dead Code	3.2	冗長なコードや決して実行されないコードが含まれている
Knowledge Conflicting		15.1	間違った変数の利用，関数指定時の因数の欠落，インポートされていない API の呼び出し等

3 SelfCheckGPT

SelfCheckGPT は，ハルシネーション検知手法として注目を集めている．我々はこの手法をコード生成に適用した．本節では，SelfCheckGPT を概説する．

3.1 概要

SelfCheckGPT は，「LLM が与えられた入力に対して知識を保持している場合，サンプリングされた回答 (同一のプロンプトに対する複数の回答) は高い類似性を持ち，一貫した事実を含む」というアイデアに基づいている．

SelfCheckGPT の特徴の一つに，外部リソースを必要としないゼロリソースの検知手法である点が挙げられる．他の検知手法である Collu-Bench 等 [12, 6, 13] と比較して，トークン単位の対数確率 (log probability) の利用など，モデルによっては外部 API を必要とする要素が SelfCheckGPT には含まれていない．

3.2 ハルシネーションスコアの算出

SelfCheckGPT は，サンプリングされた回答間の類似度から算出されるハルシネーションスコアを用いてハルシネーション検知を行う．ハルシネーションスコアは，検知対象 R と同一のプロンプトより得られる N 個のサンプルとの類似度から算出される．

検知対象 R の i 番目の文章を r_i，n 個目のサンプルの k 番目の文章を s_k^n とした際に，r_i のハルシネーションスコア $H_{Sim}(i)$ は以下のような式で表される．また，$Sim(r_i, s_k^n)$ は r_i と s_k^n の類似度を表す．

$$H_{Sim}(i) = 1 - \frac{1}{N}\sum_{n=1}^{N} \max_k (Sim(r_i, s_k^n))$$

さらに，検知対象 R の文章全体のハルシネーションスコアは，以下の式のように検知対象 R の各文におけるハルシネーションスコア $H_{Sim}(i)$ の平均をとることで求められる．

$$H_{passage} = \frac{1}{|R|}\sum_i H_{Sim}(i)$$

$H_{Sim}(i), H_{passage}$ は 0 から 1 の値を取り，1 に近いほどハルシネーションの可能性が高いことを意味する．

4 実験

我々は，図 1 に示すように，出力コード (検知対象 R) に対する，SelfCheckGPT のハルシネーションスコアによるコードの評価と，実行ベースの評価を比較し，双方の関連性を調査した．

本節では，実験設定・手順と結果を示す．

4.1 検知対象データセット

本実験では，HumanEval[2] データセットを利用する．HumanEval は，LLM のコード生成の能力を評価するための標準的なベンチマークである．このベンチマークでは，LLM に関数定義と英文のドキュメンテーションをプロンプトとして与える．LLM は，それに続くコードを生成し，関数定義を完成させることが求められる．

4.2 実行ベース評価

実行ベース評価では，LLM の生成したコードの正しさを，用意されたテストケースに全て通過するか否かで評価する [14].

本実験では，実行ベース評価の評価指標として pass@1 を採用した．これは，PASS(1) と FAIL(0) を意味する二値分類と等しくなる．

4.3 ハルシネーションスコア算出手順

SelfCheckGPT のハルシネーションスコアを以下の手順で算出し，スコアに基づきハルシネーションの有無 (PASS か FAIL か) を検知する．本実験では，ハルシネーションスコアの二値分類に ROC-AUC を用いた．

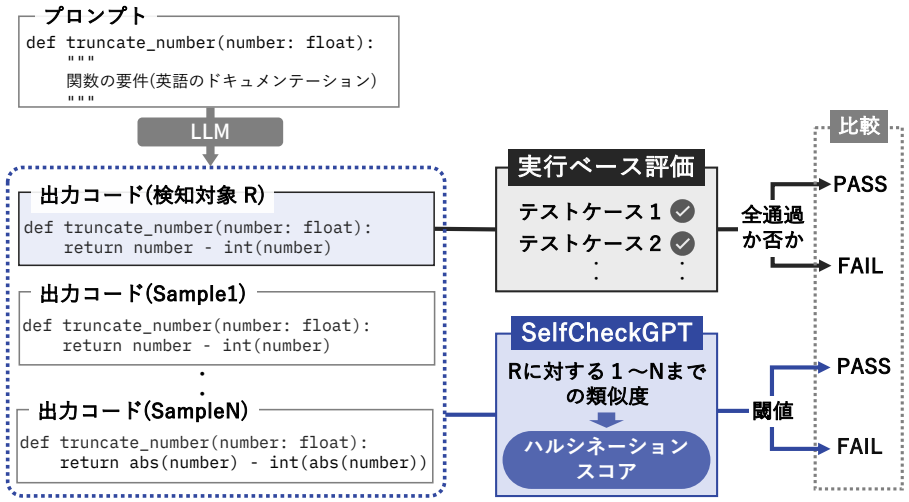

図 1　実験概要

表 2　各モデルのコード生成能力の評価結果

モデル	pass@1	モデル	pass@1
GPT-4o-mini[1]	0.433	CodeLlama[2]	0.367
Llama3[3]	0.433	OpenCoder[4]	0.800
Gemma2[5]	0.400	LLM-jp-3[6]	0.133
Phi-3.5[7]	0.667	Llama-3-Swallow[8]	0.433
Qwen2.5-Coder[9]	0.600	Chico[10]	0.000

4.3.1　回答のサンプリング

SelfCheckGPT は，同一のプロンプトに対する複数の回答からハルシネーションを検知する手法である．本研究では，ハルシネーションスコアと実行ベース評価の関連性を見つけるため，広範囲なコード生成能力をもった LLM を 10 個選定した．それらのモデルを用いて，以下の手順で回答のサンプリングを行った．

- **検知対象 R の設定**: temperature を 0 に設定し，モデルが出力したコードを検知対象 R とした．
- **サンプルコードの生成**: temperature を 1 に設定し，検知対象 R と同じプロンプトを用いて 6 つのサンプルコードを生成した．

表 2 は，対象としたモデルとそのコード生成能力を pass@1 の値でまとめたものである．

4.3.2　コードの類似度算出

検知対象 R と 6 つのサンプルコードの類似度を算出する．本実験では，コード生成において有効な類似度指標を確認することを目的として，コード生成の評価指標として採用されている 7 種類の類似度指標（表 3）を用いた．これらの類似度指標と 3.2 節で示した式に基づき，検知対象 R のハルシネーションスコア $H_{passage}$ を算出する．

4.3.3　ROC-AUC による閾値評価

我々は，選定した HumanEval30 件と 10 個のモデルから得られた，計 300 件のハルシネーションスコアを基に，コードがテストケースを PASS するか否かを予測する．

本実験では，ハルシネーションスコアに基づく二値分類の評価指標として，ROC-AUC を採用した．ROC-AUC(Receiver Operating Characteristic - Area Under the Curve) は，分類モデルの性能を評価する際に広く用いられる指標の一つであり，ROC 曲線によって描かれる面積である．ROC 曲線は，モデルの予測スコアに基づく様々な閾値における偽陽性率 (False Positive Rate) と真陽性率 (True Positive Rate) を可視化したものであり，ROC 曲線の下の面積 (AUC) により，分類性能を定量的に評価する．ROC-AUC が 1 に近いほど分類精度が高く，0.5 に近い場合はほぼランダムに分類されていることを示す．

本実験では，ROC 曲線上の点のうち，(0,1) と距離が最小となる点を閾値とし，PASS と FAIL に分類した．また，ハルシネーションスコアに基づく pass@1 の予測精度を評価するため，本実験では正解率 (Accuracy) を算出した．

1) https://platform.openai.com/docs/models/gpt-4o-mini
2) https://huggingface.co/meta-llama/CodeLlama-7b-Instruct-hf
3) https://huggingface.co/meta-llama/Meta-Llama-3-8B-Instruct
4) https://huggingface.co/infly/OpenCoder-8B-Instruct
5) https://huggingface.co/google/gemma-2-2b-it
6) https://huggingface.co/llm-jp/llm-jp-3-1.8b-instruct
7) https://huggingface.co/microsoft/Phi-3.5-mini-instruct
8) https://huggingface.co/tokyotech-llm/Llama-3-Swallow-8B-Instruct-v0.1
9) https://huggingface.co/Qwen/Qwen2.5-1.5B-Instruct
10) https://huggingface.co/NaoS2/tinycodellama-jp-0.6b-20k-2

表 3　使用した類似度指標

類似度指標	類似度指標の説明
編集距離類似度 (EditSim[15])	Levenshtein 距離 [16] に基づく類似度
Jaccard 係数	二つの集合の共通部分の割合に基づく類似度.
BLEU[17]	検知対象とサンプルの n-gram 一致度と，文章全体の単語数の適切性に基づく類似度.
ROUGE-L[18]	最長共通部分文字列に基づく類似度.
BERTScore[19]	BERT[20] から得られるベクトル表現に基づく類似度.
CodeBERTScore[21]	CodeBERT[22] から得られるベクトル表現に基づく類似度.
EmbSim[23]	検知対象とサンプルの埋め込み表現のコサイン類似度.

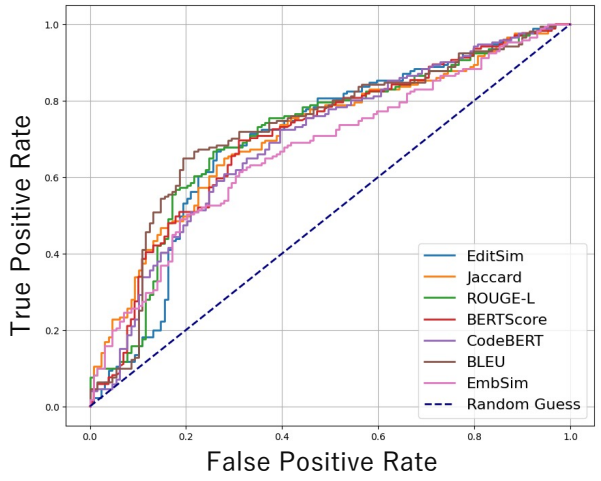

図 2　ROC 曲線

表 4　各類似度の ROC-AUC, 正解率 (Accuracy)

類似度指標	ROC-AUC	Accuracy
EditSim	0.702	0.697
Jaccard 係数	0.712	0.680
BLEU	0.724	0.713
ROUGE-L	0.712	0.700
BERTScore	0.705	0.693
CodeBERTScore	0.700	0.657
EmbSim	0.671	0.647

4.4　評価結果

実験結果より描画された ROC 曲線を図 2 に示す．また，表 4 に各類似度における ROC-AUC の値と閾値により求められた Accuracy の結果を示す．各モデルと全類似度における pass@1 の予測精度の全結果を，付録 A に記載する．

図 2 と表 4 の ROC-AUC の結果より，どの類似度手法においても ROC-AUC は約 0.7 と，ハルシネーションスコアに基づく二値分類が正しく行われている傾向が見られた．また，表 4 の Accuracy の結果では，最も高い Accuracy を示したのは，BLEU, ROUGE-L, EditSim であり，特に pass@1 との関連性を持つことが確認された．これらの結果が得られた理由として，BLEU, ROUGE-L, EditSim は，他の類似度指標と比較して，トークンレベルでの一致や順序を重視している点が挙げられる．これらの特徴が，コード特有の構造やキーワードを踏まえた評価を可能にしたと考えられる．

5　おわりに

本研究では，自然言語処理のハルシネーション検知手法である SelfCheckGPT[1] をコード生成に適用した．また，コードの評価と実行ベース評価を比較し，両者の関連性を調査した．その結果，ROC-AUC の最大値が 0.724，実行ベース評価との Accuracy が最大 0.713 と，両者に関連性があることが確認された．我々の調査から，SelfCheckGPT を用いたコード生成評価は，実行ベース評価の代替手法として活用可能であることが示唆された．

今後は，検知対象の文章全体のハルシネーションスコアに加え，各文におけるスコアの評価を進める．また，自然言語とプログラミング言語の構造的な差異を調査し，コード生成に最適な検知手法の実現を目指す．

参考文献

[1] Potsawee Manakul, Adian Liusie, and Mark JF Gales. Selfcheckgpt: Zero-resource black-box hallucination detection for generative large language models. **arXiv preprint arXiv:2303.08896**, 2023.

[2] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. **arXiv preprint arXiv:2107.03374**, 2021.

[3] Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. Program synthesis with large language models. **arXiv preprint arXiv:2108.07732**, 2021.

[4] Terry Yue Zhuo, Minh Chien Vu, Jenny Chim, Han Hu, Wenhao Yu, Ratnadira Widyasari, Imam Nur Bani Yusuf, Haolan Zhan, Junda He, Indraneil Paul, et al. Bigcodebench: Benchmarking code generation with diverse function calls and complex instructions. **arXiv preprint arXiv:2406.15877**, 2024.

[5] Thanet Markchom, Subin Jung, and Huizhi Liang. Nu-ru at semeval-2024 task 6: Hallucination and related observable overgeneration mistake detection using hypothesis-target similarity and selfcheckgpt. In **Proceedings of the 18th International Workshop on Semantic Evaluation (SemEval-2024)**, pp. 253–260, 2024.

[6] Bairu Hou, Yang Zhang, Jacob Andreas, and Shiyu Chang. A probabilistic framework for llm hallucination detection via belief tree propagation. **arXiv preprint arXiv:2406.06950**, 2024.

[7] Xiangkun Hu, Dongyu Ru, Lin Qiu, Qipeng Guo, Tianhang Zhang, Yang Xu, Yun Luo, Pengfei Liu, Yue Zhang, and Zheng Zhang. Knowledge-centric hallucination detection. In **Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing**, pp. 6953–6975, 2024.

[8] Moritz Beller, Georgios Gousios, Annibale Panichella, Sebastian Proksch, Sven Amann, and Andy Zaidman. Developer testing in the ide: Patterns, beliefs, and behavior. **IEEE Transactions on Software Engineering**, Vol. 45, No. 3, pp. 261–284, 2017.

[9] Mikhail Evtikhiev, Egor Bogomolov, Yaroslav Sokolov, and Timofey Bryksin. Out of the bleu: how should we assess quality of the code generation models? **Journal of Systems and Software**, Vol. 203, p. 111741, 2023.

[10] Weixi Tong and Tianyi Zhang. Codejudge: Evaluating code generation with large language models. **arXiv preprint arXiv:2410.02184**, 2024.

[11] 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. **arXiv preprint arXiv:2404.00971**, 2024.

[12] Nan Jiang, Qi Li, Lin Tan, and Tianyi Zhang. Collu-bench: A benchmark for predicting language model hallucinations in code. **arXiv preprint arXiv:2410.09997**, 2024.

[13] Neeraj Varshney, Wenlin Yao, Hongming Zhang, Jianshu Chen, and Dong Yu. A stitch in time saves nine: Detecting and mitigating hallucinations of llms by validating low-confidence generation. **arXiv preprint arXiv:2307.03987**, 2023.

[14] Sumith Kulal, Panupong Pasupat, Kartik Chandra, Mina Lee, Oded Padon, Alex Aiken, and Percy S Liang. Spoc: Search-based pseudocode to code. **Advances in Neural Information Processing Systems**, Vol. 32, , 2019.

[15] Alexey Svyatkovskiy, Shao Kun Deng, Shengyu Fu, and Neel Sundaresan. Intellicode compose: Code generation using transformer. In **Proceedings of the 28th ACM joint meeting on European software engineering conference and symposium on the foundations of software engineering**, pp. 1433–1443, 2020.

[16] Li Yujian and Liu Bo. A normalized levenshtein distance metric. **IEEE transactions on pattern analysis and machine intelligence**, Vol. 29, No. 6, pp. 1091–1095, 2007.

[17] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 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.

[18] Chin-Yew Lin. Rouge: A package for automatic evaluation of summaries. In **Text summarization branches out**, pp. 74–81, 2004.

[19] Tianyi Zhang, Varsha Kishore, Felix Wu, Kilian Q Weinberger, and Yoav Artzi. Bertscore: Evaluating text generation with bert. **arXiv preprint arXiv:1904.09675**, 2019.

[20] Jacob Devlin. Bert: Pre-training of deep bidirectional transformers for language understanding. **arXiv preprint arXiv:1810.04805**, 2018.

[21] Shuyan Zhou, Uri Alon, Sumit Agarwal, and Graham Neubig. Codebertscore: Evaluating code generation with pretrained models of code. **arXiv preprint arXiv:2302.05527**, 2023.

[22] Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, et al. Codebert: A pre-trained model for programming and natural languages. **arXiv preprint arXiv:2002.08155**, 2020.

[23] Yaoming Zhu, Sidi Lu, Lei Zheng, Jiaxian Guo, Weinan Zhang, Jun Wang, and Yong Yu. Texygen: A benchmarking platform for text generation models. In **The 41st international ACM SIGIR conference on research & development in information retrieval**, pp. 1097–1100, 2018.

A　各モデルにおけるハルシネーションスコアの予測精度

各モデルの出力に対して算出された SelfCheckGPT のハルシネーションスコアより予測された pass@1 を，実行ベース評価と比較した際の Accuracy

表 5　各モデルと類似度指標における Accuracy

	EditSim	Jaccard 係数	BLEU	ROUGE-L	BERTScore	CodeBERTScore	EmbSim	合計
GPT-4o-mini	0.567	0.567	0.567	0.567	0.567	0.600	0.433	0.552
Llama3	0.600	0.567	0.600	0.600	0.600	0.600	0.633	0.600
Gemma2	0.467	0.500	0.500	0.500	0.433	0.433	0.500	0.476
Phi-3.5	0.733	0.667	0.700	0.767	0.567	0.533	0.533	0.643
Qwen2.5-Coder	0.567	0.567	0.600	0.600	0.667	0.600	0.500	0.586
CodeLlama	0.767	0.767	0.833	0.800	0.700	0.700	0.733	0.757
OpenCoder	0.733	0.767	0.733	0.767	0.767	0.733	0.667	0.738
LLM-jp	0.833	0.867	0.833	0.900	0.867	0.833	0.867	0.857
Llama-3-Swallow	0.700	0.533	0.633	0.633	0.767	0.567	0.600	0.633
Chico	1.00	1.00	1.00	1.00	1.00	0.967	1.00	0.995