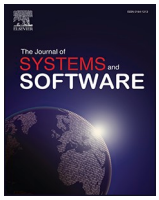
The Journal of Systems and Software 222 (2025) 112330

Contents lists available at ScienceDirect

The Journal of Systems & Software

journal homepage: www.elsevier.com/locate/jss



Effectiveness of symmetric metamorphic relations on validating the stability of code generation LLM

Pak Yuen Patrick Chan [a], Jacky Keung [a], Zhen Yang [b,*]

[a] Department of Computer Science, City University of Hong Kong, Kowloon, Hong Kong, China
[b] School of Computer Science and Technology, Shandong University, Qingdao, China

ARTICLE INFO

Keywords:
Metamorphic testing
Metamorphic relation
True satisfaction
Large language model
Code generation

ABSTRACT

Pre-trained large language models (LLMs) are increasingly used in software development for code generation, with a preference for private LLMs over public ones to avoid the risk of exposing corporate secrets. Validating the stability of these LLMs' outputs is crucial, and our study proposes using symmetric Metamorphic Relations (MRs) from Metamorphic Testing (MT) for this purpose. Our study involved an empirical experiment with ten LLMs (eight private and two public) and two publicly available datasets. We defined seven symmetric MRs to generate "Follow-up" datasets from "Source" datasets for testing. Our evaluation aimed to detect violations (inconsistent predictions) between "Source" and "Follow-up" datasets and assess the effectiveness of MRs in identifying correct and incorrect non-violated predictions from ground truths. Results showed that one public and four private LLMs did not violate "Case transformation of prompts" MR. Furthermore, effectiveness and performance results indicated that proposed MRs are effective tools for explaining the instability of LLM's outputs by "Case transformation of prompts", "Duplication of prompts", and "Paraphrasing of prompts". The study underscored the importance of enhancing LLMs' semantic understanding of prompts for better stability and highlighted potential future research directions, including exploring different MRs, enhancing semantic understanding, and applying symmetry to prompt engineering.

1. Introduction

Code generation is a popular research topic in software engineering (Li et al., 2023). This task generates code by the given natural language description with advanced technologies and techniques, in order to advance the software development process (Asare et al., 2023; Zeng et al., 2022). Pre-trained Large language model (LLM) technology has been utilized for code generation (Cassano et al., 2023; Dong et al., 2023; Hussain et al., 2023), and it is essential to evaluate their performance. However, evaluating code generation LLMs poses a test oracle problem since it is difficult to distinguish correct code generation behavior from potentially incorrect behavior. Metamorphic Testing (MT) was selected for this challenge as it is a widely used technique to alleviate the test oracle problem (Sun et al., 2022; Xiao et al., 2022; Xie et al., 2020; Ying et al., 2021; Zhang et al., 2021; Z. Q. Zhou et al., 2020). MT addresses the test oracle problem by examining the expected relationships between input-output pairs in consecutive executions of the systems (Duque-Torres et al., 2023). These expected relationships are expressed as metamorphic relations (MRs), and if the output results in various software executions violate an MR, then a fault is revealed (Segura et al., 2019; Z. Q. Zhou et al., 2020). We can avoid the test oracle problem and evaluate the models through this approach.

In this study, we adopted the effective MT approach to validate the stability of code generation LLMs' outputs. We derived three symmetric MRs inspired by Z. Q. Zhou et al. (2020), and these MRs assume that code generations should not be affected as the semantic meanings of the prompts are preserved. We experimented with ten different LLMs on two distinct datasets to simulate software houses using LLM to generate source codes for seen and unseen projects. One dataset contains training and testing data to simulate a seen project, whereas the other dataset only provides testing data to simulate an unseen project.

We used the proposed MRs on the "Source" testing datasets to produce fourteen "Follow-up" testing datasets. We trained eight LLMs with seen project training datasets to create private code generation LLMs, which are referred to as private LLMs. Then, we used eight private LLMs and two public code generation LLMs, which is referred to as public LLMs, to generate source codes based on the prompts from the "Source" testing datasets, resulting in "Source" outputs for seen and unseen

* Corresponding author.
E-mail address: zhenyang@sdu.edu.cn (Z. Yang).

https://doi.org/10.1016/j.jss.2024.112330
Received 8 June 2024; Received in revised form 4 December 2024; Accepted 24 December 2024
Available online 25 December 2024

projects. We repeated this process with the "Follow-up" testing datasets to generate "Follow-up" outputs for seen and unseen projects. We then compared the two sets of "Source" and "Follow-up" outputs and analyzed their differences. A failure is revealed if the relation is violated, which means differences occur between the outputs. Furthermore, by examining the true violations and accuracies of outputs, we could determine the effectiveness of MRs.

The results indicate that the proposed MRs can effectively validate the stability of code generation LLMs' outputs. The main contributions of this study include:

(1) We effectively validate the stability of ten code generation LLMs' outputs using our proposed MRs with violation measurements, and explain the instabilities with effectiveness and performance measurements. To the best of our knowledge, this study is the first to introduce this approach for validating the stability of code generation LLMs' outputs.
(2) We have introduced symmetric MRs, which are semantic-preserving, to validate the stability of ten code generation LLMs' outputs effectively. One MR can effectively validate one public and four private LLMs in both seen and unseen projects. On the other hand, the MRs can explain the instability of LLMs' outputs. These results demonstrate that the proposed MRs can be extended to validate and identify the causes of the instabilities with other domains using different languages and different LLMs.
(3) We have developed a systematic validation approach using symmetric MRs with violation, effectiveness and performance measurements for researchers and practitioners to follow and apply.
(4) We have outlined several research challenges and opportunities that researchers may encounter in extending the proposed approach for LLMs in other areas.

The rest of this paper is organized as follows. Section two introduces the underlying concepts of code generation using large language model and MT. It also discusses the challenges of evaluating LLMs, which is test oracle problem, and briefly discusses the recent related works. Section three describes the methodology of this study to determine the effectiveness of the proposed MRs in validating the stability of code generation LLMs' outputs. Section four discusses and analyses the results. Section five covers the threats to the validity of this study. Finally, section six concludes this paper with the limitations of this study and future directions.

2. Background

This section briefly introduces code generation using large language models, evaluation, and the preliminary knowledge of MT and related works.

2.1. Code generation

Code generation, also known as code suggestion or text-to-code, is a widely used feature that provides suggested source code based on known source code contexts, traditionally (Hussain et al., 2023; Zhang et al., 2023). This task generates code using the given natural language description with advanced technologies and techniques (Zeng et al., 2022). Code generation is a valuable topic in the software development research domain (Li et al., 2023). Many studies related to code generation have been conducted aiming to increase the productivity of software development and maintain the quality of the source codes (Asare et al., 2023). Code generation methodologies include statistical approach, neural symbolic approach (Alur et al., 2013; Chaudhuri et al., 2021; Dong and Lapata, 2018; Gulwani et al., 2017; Yin and Neubig, 2017), and pre-trained large language model approach (Cassano et al., 2023; Dong et al., 2023; Hussain et al., 2023). The employment of large language models, in particular, has demonstrated tremendous promise in completing code and synthesizing code from natural language descriptions (Cassano et al., 2023; Xu et al., 2022).

2.2. Large language model (LLM) for code generation

With the advancements in artificial intelligence, researchers and software developers have shifted their focus towards leveraging artificial intelligence in software engineering tasks, and advanced code generation have been developed because of applying language models (Asare et al., 2023; Mouselinos et al., 2023). The core of this innovation lies in pre-trained large language models (LLMs), which have shown remarkable effectiveness in both natural and programming language tasks (Cassano et al., 2023; Xu et al., 2022; Zeng et al., 2022). These models are widely used for solving challenging code processing tasks, such as code generation or code summarization, by providing a flexible interface that enables the synthesis of programs from natural language specifications (Poesia et al., 2022; Troshin and Chirkova, 2022).

LLM-based code completion frameworks such as GitHub Copilot, CodeX, CodeTran, CodeGen, CodeLSTM, RoBERTaCode, GPT and SYNCHROMESH are trained using deep learning over vast quantities of unstructured text and open source code (Hussain et al., 2023; Li et al., 2023; Poesia et al., 2022; Xu et al., 2022; Zeng et al., 2022). Furthermore, it is suggested that self-collaboration among LLMs could potentially enable LLMs to efficiently handle complex real-world tasks that are not readily addressed through direct code generation (Dong et al., 2023).

Although there are many advantages of using LLMs, companies that use third-party generative AI tools or LLMs, such as ChatGPT, may risk exposing corporate secrets (Post, 2023). As language models grow, they also raise safety concerns including but not limited to data contamination, toxic or biased generation, personal information leak, and hallucinations (Zhang et al., 2023).

2.3. Evaluation

Recently, there has been extensive development of code generation systems designed to generate source code based on natural language instructions. However, despite their advancements, these systems still face robustness issues where even slightly different instructions can result in significantly different code semantics (Yan et al., 2023). Robustness is critical for code generation LLMs, as it can significantly impact trust in the generated codes (Yan et al., 2023). Therefore, evaluation is essential for ensuring the performance of code generation LLMs. Conala, HumanEval and MBPP datasets are widely used execution-based code generation benchmarks for evaluating code generation LLMs (Austin et al., 2021; Chen et al., 2021; Mouselinos et al., 2023; Xu et al., 2022).

The robustness of code generation LLMs can be evaluated by combining with code similarity scoring, such as bleu scoring (Li et al., 2023; Yan et al., 2023; Zeng et al., 2022), Levenshtein edit similarity (Li et al., 2023), execution match accuracy, edit distance (Poesia et al., 2022), pass@K (Athiwaratkun et al., 2022; Wang et al., 2022). Other evaluation methods for pre-trained code generation LLMs include intrinsic evaluation by using unseen project (Xu et al., 2022), evaluating robustness via adversarial attacks (Zeng et al., 2022), automated intervention mechanism reminiscent of adversarial testing to expose biases (Mouselinos et al., 2023), black-box testing with mutating the seed prompts (Li et al., 2023), testing the robustness of code generation systems by exploiting their characteristics (Yan et al., 2023), evaluating accuracy rate of LLMs in classification task for predicting the next code token (Hussain et al., 2023), evaluating LLMs through a comprehensive robustness evaluation benchmark (Wang et al., 2022).

Most evaluation approaches focus on the performance of code generation rather than validating the stability of LLMs' outputs. Stable code generation is essential as if LLM generated codes differently every time,

this can introduce risks in the coverage of existing testing procedures. Without validating the stability of the model's outputs, the performance of code-generation LLMs can be unpredictable and lead to unstable systems; however, validating the stability of code generation LLMs' outputs poses a test oracle problem since it is difficult to distinguish correct code-generation behavior from potentially incorrect behavior. Thus, MT was chosen to alleviate the test oracle problem.

2.4. Metamorphic testing (MT)

Metamorphic Testing (MT) was introduced by Chen et al. (2020) as a novel approach to alleviate the problem of test oracles by verifying the relations between the inputs and outputs across multiple test executions of the programs or systems under examination. These relationships, known as Metamorphic Relations (MRs), embody the essential attributes of the target function or algorithm concerning the multitude of inputs and their anticipated outputs (Chen et al., 2020; 2018). The methodology of MT involves the initial generation of specific program inputs, referred to as "Source" inputs, to serve as the foundational test cases. Subsequently, leveraging these "Source" test cases, MRs are employed to generate new inputs, thus constituting the "Follow-up" test cases. Unlike the conventional testing methodology, MT comprehensively verifies the "Source" and "Follow-up" test cases and their corresponding outputs against the specified Metamorphic Relation (Chen et al., 2020, 2018).

The idea of MR can be illustrated by a mathematical property of the *sine* function, which is "$sin(x) = sin(\pi - x)$". For example, if the corresponding MR has two inputs, x1 and x2, that satisfy "$x1 + x2 = \pi$", then two function outputs should be equal, i.e., sin(x1) = sin(x2). Based on MR, x2 is constructed based on $x1$ and sin($x1$). Therefore, x1 is the "Source" input, and x2 is the "Follow-up" input. Such MR can be used to test the program as if the outputs of sin(x1) and sin(x2) are different, then the implementation of function "sin()" must have faults as it violates the above MR (Zhang et al., 2021).

MT has been used for oracle problem in different domains, such as bioinformatic software (Stacy et al., 2022), machine learning classification (Ellis et al., 2021; Riccio et al., 2020; Saha and Kanewala, 2019; Xie et al., 2011), cryptographic software (Pugh et al., 2019), online search engines (Segura et al., 2019). MT can also be applied in validating autonomous vehicles systems (Deng et al., 2023; Luu et al., 2024), fake news detection systems (Miao et al., 2023), automatic image classification systems (Xu et al., 2021), and multiple linear regression systems (Luu et al., 2021). It is also plausible to define metamorphic relation patterns (MRPs) for deriving multiple MRs (Segura et al., 2019). Furthermore, it is plausible to improve the capacity of MT by introducing simulation testing concept (Ying et al., 2022) and by introducing effectiveness measurement of MT, which refers to measuring how well the evaluation results from MT can reflect the real performance or target system (Jiang et al., 2022).

Motivated by the necessity of more advanced evaluation techniques for code generation LLMs that suffer from the test oracle problem and lack of evaluation approaches focusing on validating the LLM itself instead of solely focusing on the performance of code generation, thus, this study aims to demonstrate MT as effective tools for validating the stability of code generation LLMs' outputs and for explaining the instability through a comprehensive scaled experiment.

3. Methodology

This section describes the setup of the experiment. It defines the research objective and research questions of our experiment. Then, it describes the subject systems and data used in the experiment. Thereafter, it discusses the detailed experimental design, including environment configuration, measurement, experimental procedure, dataset preparation, and parameter setting.

3.1. Research objective and questions

The main objective of the experiment is to demonstrate the feasibility of the proposed MRs for validating the stability of code generation LLMs' outputs and studying the effectiveness of proposed MRs in explaining the instabilities. This experiment is to attempt to answer three research questions.

RQ1: Can MRs validate the stability of code generation Large Language Models' outputs? MT is usually conducted when the correct results of individual test cases are unavailable, and the validation capacity of the proposed MRs, which refers to their ability to validate, may vary depending on the outputs of different LLMs. Our goal is to assess the validation capacity of proposed MRs by counting the occurrence of related violations as the measurements of stability under the assumption that there is no ground truth available.

RQ2: How effective are MRs in explaining the instability of code generation Large Language Models' outputs?

We further evaluate the effectiveness of proposed MRs in explaining the instability of LLMs' outputs by identifying the inconsistencies between performance, "Source" predictions, "Follow-up" predictions, and ground truths related to MRs. These numbers can indicate the effectiveness of the proposed MRs, which refers to measuring how well the evaluation results from MRs can reflect the ground truths related to MRs.

RQ3: What are the causes that make MRs less effective in explaining the instability of code generation Large Language Models' outputs? When ineffectiveness occurs, we analyze the results with the characteristics of MRs and LLMs to seek the causes that lead to the ineffectiveness.

3.2. Proposed metamorphic relations (MRs)

In this study, we adopted the MT approach to validate LLMs and derived three symmetric MRs, which are semantic-preserving, inspired by Z. Q. Zhou et al. (2020). Z. Q. Zhou et al. (2020) introduce innovative uses of MRs beyond their traditional role in MT. They proposed the "symmetry" MR pattern and the "change direction" MR input pattern. These patterns help derive multiple concrete MRs, which can reveal previously unknown failures in popular applications. This study focuses on the "symmetry" MR pattern, and our MRs are centered on the concept of symmetry and operate under the assumption that predictions should remain unaffected when the semantic meanings of prompts are preserved (Table 1), in order to validate the stability of the LLMs' outputs. In other words, systems should generate similar codes with "Source" and "Follow-up" prompts as long as the semantic meaning of prompts is preserved. If MRs are not upheld, it indicates a violation, and the associated MR can reveal the instability of the LLM's outputs.

MR1 - Case conversion of prompts. "Source" and "Follow-up" prompts should mean the same to the code generation LLMs, regardless of whether uppercase or lowercase is used. As only the case format of the prompts is altered, there is no change in the semantic meaning of prompts; thus, code generation LLMs should produce identical outputs with the "Source" and "Follow-up" prompts. For example, we assume

Table 1
Descriptions of MRs.

MR	Descriptions
MR1 Case transformation of prompts	MR1.1 - Convert 100% prompt texts to uppercase MR1.2 - Convert 50% prompt texts to uppercase
MR2 Duplication of prompts	MR2.1 - Duplicate the prompt texts two times MR2.2 - Duplicate the prompt texts three times MR2.3 - Duplicate the prompt texts four times
MR3 Paraphrasing of prompts	MR3.1 - Paraphrase prompt texts by online paraphrasing tool MR3.2 - Paraphrase prompt texts by "Mistral-Medium" chatbot from the AI chat platform "Poe"

two prompts, "loop over files in directory" and "LOOP OVER FILES IN DIRECTORY", are semantic equivalent to the LLMs, and LLMs should generate similar program statements.

MR2 - Duplication of prompts. "Source" and "Follow-up" prompts should mean the same to the code generation LLMs, regardless of the duplication of the content of prompts. As the contents of prompts are duplicated but not modified, there is no change in the semantic meaning of prompts; thus, code generation LLMs should give the same outputs with the "Source" and "Follow-up" prompts. For example, we assume two prompts, "loop over files in directory" and "loop over files in directory loop over files in directory", are semantic equivalent to the LLMs, and LLMs should generate similar program statements.

MR3 - Paraphrasing of prompts. "Source" and "Follow-up" prompts should mean the same to the code generation LLMs, regardless of paraphrasing. In other words, LLMs should consider the contents of prompts have similar semantic meanings; thus, code generation LLMs should give the same outputs with the "Source" and "Follow-up" prompts. For example, we assume two prompts, "loop over files in directory" and "loop through files in directory", are semantic equivalent to the LLMs, and LLMs will generate similar program statements. In addition, we use one online paraphrasing tool[1] for MR3.1 and the official "Mistral-Medium" chatbot (https://poe.com/Mistral-Medium) from the AI chat platform "Poe"[2] is chosen for MR3.2.

3.3. Test scenario

Inspired by Post (2023) mentioned that the widespread use of AI chatbots and writing tools could leave companies vulnerable to data leaks and lawsuits. AI chatbots could be a potential area that is exploited by AI companies or hackers to access sensitive corporate information, which has been fed into the chatbots, or perform actions against the company in the future (Post, 2023). This study hypothesized that software development houses use private code generation LLMs to mitigate the risk of exposing corporate secrets, such as source code leaks, by restricting access to authorized personnel as the test scenario. Software development houses must safeguard the company's software assets while using LLMs to generate source codes for software enhancements and maintenance; however, using private LLMs might not have the same level of performance as public LLMs. Thus, eight private code generation LLMs and two public code generation LLMs are chosen in this study. We selected two separate datasets to simulate the seen and unseen project situation. One dataset includes both training and testing data to simulate a seen project. Whereas the other dataset only provides testing data to simulate an unseen project.

3.4. Subject under test (SUTs)

3.4.1. Private pre-trained large language model

Eight pre-trained LLMs from the machine learning community "Hugging Face"[3]are chosen as private code generation LLMs. We simulated private software houses that downloaded those LLMs from "Hugging Face" and set them up as private LLMs, which are trained privately and only accessed by authorised users. They are referred to as private LLMs throughout our study. Due to the limit of computer resources, this study employed the private LLMs with a small number of parameters.

bert2bert is a text-to-text generation LLM with an encoder-to-decoder framework. The encoder is used to understand the input texts, while the decoder is used to pick up inputs from the encoder and generate text outputs to perform different language tasks, such as code generation. It uses an uncased version of the BERT base language representation model as an encoder and decoder (HuggingFace, 2023k). BERT base language representation model operates as a bidirectional pre-trained language model that processes the raw text inputs by jointly conditioning on both left and right context in all layers (Devlin et al., 2018). bert2bert is pre-trained and fine-tuned on the "CNN" and "Dailymail" summarisation datasets in the English language (HuggingFace, 2023k).

bert2gpt2 is another text-to-text generation LLM with an encoder-to-decoder framework that uses an uncased BERT base language representation model as an encoder and GPT2 base language representation model as a decoder to perform different language tasks, such as code generation (HuggingFace, 2023f). GPT2 is an uni-directional and generative pre-training transformer-based language model created and released by the technology company OpenAI, and it uses language modelling on a large corpus with the ability to process long-range dependencies (Radford et al., 2018; 2019). GPT2 optimizes the layer normalization, expands the vocabulary size to 50,257, increases the context size from 512 to 1024 tokens, and optimizes with a larger batch size of 512 tokens (Radford et al., 2019). In addition, GPT2 is pre-trained on "WebText", created from scraping web pages, and the dataset roughly contains eight million documents (Radford et al., 2019). bert2gpt2 model is further pre-trained and finetuned with the "id_liputan6" dataset in the Indonesian language (HuggingFace, 2023f).

marianCG is a code generation transformer model with an encoder-to-decoder framework developed using the Marian neural machine translation (marianMT) model as an encoder and decoder to perform different language tasks, such as code generation (HuggingFace, 2023e). marianMT is the core model of the Microsoft Translator, which is the basis for natural language to code translation engine, and uses a sinusoidal positional embedding technique to represent the position of each token in the text, as well as no layer normalization embedding (Soliman et al., 2022). marianCG is pre-trained with "Conala" and "DJANGO" datasets (HuggingFace, 2023e; Soliman et al., 2022).

distilbert2mar is a text-to-text generation LLM with an encoder-to-decoder framework that uses distilbert, which is an uncased distilled version of the BERT base language representation model, as encoder and language representation model marianMT as a decoder to perform different language tasks, such as code generation (HuggingFace, 2023a). distilbert is a pre-trained language representation transformer model trained with distillation to create a smaller and faster version of BERT (Sanh et al., 2019). distilbert2mar is pre-trained with "BookCorpus" dataset (HuggingFace, 2023a, 2023 g; Soliman et al., 2022).

distilroberta2mar is a text-to-text generation LLM with an encoder-to-decoder framework that uses distilroberta, which is a language representation model distilled version of the RoBERTa-base model, as an encoder and language representation model marianMT as a decoder to perform different language tasks, such as code generation (HuggingFace, 2023b). Roberta is a transformer model similar to BERT, and it is enhanced with dynamic masking, sentences without next sentence prediction loss, a larger batch size, and a more extensive vocabulary size, which gives it access to more knowledge than other pre-trained language models (Liu et al., 2019). distilroberta is the distilled version of Roberta, which follows the same training procedure as distilbert (HuggingFace, 2023h). distilroberta2mar is pre-trained with the "OpenWebTextCorpus" dataset (HuggingFace, 2023b, 2023h; Soliman et al., 2022).

electra2mar is a text-to-text generation LLM with an encoder-to-decoder framework that uses Electra base language representation model as an encoder and language representation model marianMT as a decoder to perform different language tasks, such as code generation (HuggingFace, 2023c). Unlike BERT, Electra uses a more sample-efficient pre-training task called replaced token detection to replace some tokens with plausible alternatives sampled from a small generator network. Then, instead of training a model that predicts the original identities of the corrupted tokens, the model was trained as a discriminative model that predicts whether each token in the corrupted input was replaced by a generator sample (Clark et al., 2020).

[1] Online paraphrasing tool website is https://paraphrasing-tool.com/.
[2] The Poe website is https://poe.com/.
[3] Hugging Face website is https://huggingface.co/.

electra2mar is pre-trained with different datasets, such as "Wikipedia" and "BooksCorpus" (Clark et al., 2020; HuggingFace, 2023c, 2023i; Soliman et al., 2022).

luke2mar is a text-to-text generation LLM with an encoder-to-decoder framework that uses LUKE base language representation model as an encoder and language representation model marianMT as a decoder to perform different language tasks, such as code generation (HuggingFace, 2023d). LUKE is a pre-trained model designed to create contextualized representations of both words and entities, and is based on the bidirectional transformer architecture and incorporates an entity-aware self-attention mechanism (Yamada et al., 2020). This mechanism allows LUKE to treat words and entities as independent tokens, enhancing its ability to understand and process text involving entities (Yamada et al., 2020). luke2mar is pre-trained with many entity-annotated corpus obtained from "Wikipedia" (HuggingFace, 2023d, 2023l; Soliman et al., 2022; Yamada et al., 2020).

pegasus_x is a code generation transformer model that is developed using the PEGASUS-X base language representation model, which is an extension of the PEGASUS model, with additional long input pre-training to handle large size tokens inputs (HuggingFace, 2023j; Phang et al., 2023). It is an encoder-to-decoder framework that uses the PEGASUS-X base language representation model as an encoder and decoder to perform different language tasks, such as code generation (HuggingFace, 2023j). pegasus-x is pre-trained and fine-tuned on a variety of datasets, such as "C4", "arXiv", "GovReport", "XSUM", "CNN" and "DailyMail" (Phang et al., 2023).

3.4.2. Public pre-trained large language model

Two publicly available pre-trained code generation LLMs are chosen for comparison purposes. They are referred to as public LLMs throughout our study

codeLlama is the first chosen publicly available pre-trained code generation LLM, and it is provided by "poe.com"[2]. "poe.com" is an online platform that provides a wide variety of AI-powered chatbots for use after authentication (POE.COM, 2023). codeLlama is an AI-powered chatbot developed by Meta (POE.COM, 2024) and is based on LLAMA 2 LLM for code generation (Roziere et al., 2023). LLAMA 2 was trained on a diverse set of publicly available datasets, including "CommonCrawl", "C4", "GitHub" repositories, "Wikipedia", "Books3 and Gutenberg", "arXiv", and "Stack Exchange". This extensive corpus helps LLAMA 2 understand and generate human-like text across a wide range of topics and styles (Roziere et al., 2023; Touvron, Lavril, et al., 2023, 2023).

copilot is the second chosen publicly available pre-trained code generation LLM provided by Microsoft. It is based on Microsoft Copilot AI-powered chatbots, and users can access this chatbot through the sidebar of the Microsoft Edge browser (Microsoft, 2024a, 2024b). Based on the user's input prompt and their consent to share data with Microsoft, Microsoft Edge may send relevant data to Microsoft Copilot for response (Microsoft, 2024a, 2024b). Microsoft Copilot provides access to powerful AI chatbot that is built on multimodal large language models and text-to-image models, including GPT-4o, which has been pre-trained with a large variety of text sources (Achiam et al., 2023; Microsoft, 2024b). It is grounded in the Bing search service to provide responses with the most current information from the web, with verifiable citations for transparency (Microsoft, 2024a, 2024b, 2024c). Thus, copilot combines the power of generative AI with up-to-date information from the vast amount of public web information to perform different language tasks, such as code generation (Microsoft, 2024b).

3.5. Dataset

This study utilizes two distinct publicly available datasets to simulate software houses using LLM to generate source codes for seen and unseen projects: Conala[4] and MBPP[5]. These datasets contain data pairs of natural language intents and their corresponding Python snippets, which are the prompts and their corresponding source codes, matching this study's intent.

Conala is selected as the seen project dataset, meaning subject LLMs are trained with the Conala dataset to simulate software houses using existing projects' source codes to train the private LLM. The Conala dataset is created by the Carnegie Mellon University NeuLab and Strudel labs for the purpose of testing the generation of code snippets from natural language (Wang et al., 2023). This study utilizes the Conala training dataset, which contains 24,687 rows of data with intents and snippets, and the Conala validation dataset, which contains 1237 rows of data with intents and snippets, to train the eight private LLMs. We have selected the first 100 rows of data from Conala testing datasets to create "Source" and "Follow-up" testing datasets for this study.

MBPP is selected as the unseen project dataset, meaning software houses' LLMs have no prior information about the source codes of these projects. MBPP is a widely used benchmark of Python problems with the prompt for code generation, and the generated code is then tested with the same set of assertions (Cassano et al., 2023). We select the first 100 rows of data from MBPP testing datasets to create "Source" and "Follow-up" testing datasets for this study.

The Conala and MBPP testing datasets each consist of 100 data pairs, with intents and corresponding Python snippets. These are referred to as the "Source" testing datasets. We applied MRs with "Source" testing datasets to create seven "Follow-up" testing datasets for comparison and analysis.

3.6. Environment

This experiment environment included using "Intel Core" i7 CPU with 64 GB RAM, "Windows 10″ operating systems, "Jupyter Notebook" development platform and "Python" programming language and related libraries, such as "transformers", "pytorch", "scikit_learn", for machine learning algorithms. Standard parameters are applied to all LLMs, and the rest of the options are set to default values.

3.7. Measurement

This experiment utilized similarity scores to measure the stability of code generation LLMs' outputs. It also employed execution success rate and passing rate to measure the performance of code generation LLMs' outputs.

3.7.1. Similarity

For measuring the similarity between "Source" and "Follow-up" outputs, this study utilizes three scoring metrics: "bleu", "rouge-l", and "sacrebleu". "bleu" is a "corpus-level metric based on the modified n-gram precision measure" to measure the similarity between sentences (Evtikhiev et al., 2023). "rouge-l" is a metric from the ROUGE family of metrics and is a "recall-oriented" metric that measures similarity of two sentences by searching for the longest common subsequence between them (Evtikhiev et al., 2023). "sacrebleu" is python scoring tool which is also using bleu metric and improve the metric by treating bleu "with a bit more reverence" (Post, 2018). We employ the average values of these three different metrics to mitigate any bias effects in the scoring process.

[4] The Conala dataset is downloaded from the Python library "datasets" with the function "load_dataset("data_conala");" on 24/12/2023.

[5] The MBPP dataset is downloaded from the Python library "datasets" with the function "load_dataset("mbpp", "sanitized");" on 24/12/2023.

All three similarity scores range from zero to one hundred. If the average of the three similarity scores is over twenty, the two sentences will be considered similar. We have chosen a cut-off score of twenty because it is based on the minimum bleu score of GoogleCloud translation that the translation is considered understandable (GoogleCloud, 2023).

3.7.2. Violation of MR

For measuring the inconsistencies between "Source" and "Follow-up" outputs, we employ the **Violation Rate (VR)** inspired by (Xie et al., 2020). VR measures the occurrences of inconsistent results between "Source" and "Follow-up" outputs, which are cases with a similarity score below twenty in this study. If there is no inconsistency between "Source" and "Follow-up" outputs, VR is zero, and LLM is insensitive to the associated MR, which means the MR could be used for validation. Let the total number of test trials be T_{total} and the violated cases be $T_{violated}$. VR is calculated as Eq. (1).

$$VR = T_{violated}/T_{total} \tag{1}$$

3.7.3. Effectiveness of MR

MT is usually conducted without oracles; thus, the truth results of individual test cases are unavailable. Consequently, the test result reported by MT may not be consistent with the ground truth (Jiang et al., 2022). In this study, "Source" testing datasets contain truth test results, and the "Follow-up" testing datasets, which were created by the proposed symmetric MRs and supposed their results are not changed, also contain the same truth test result as "Source". Thus, we can use the truth test results to assess the effectiveness of MRs. For measuring the effectiveness of MR, we employ the following measurements, which are inspired by Jiang et al. (2022) (Table 2).

Ground Truth (Truth): These are the ground truth or truth test results of testing datasets.

True Satisfaction (TS): MR has no violation, and the "Source" prediction matches the ground truth. This rate reflects the effectiveness of MR. TS is calculated by counting the occurrences of non-violated cases with consistent results between "Source" outputs and ground truth, which are the cases with similarity scores above twenty between "Source" and "Follow-up" outputs, and between "Source" and ground truth outputs in this study. Let the total number of test trials be T_{total}, and the non-violated and truth-matched cases be $T_{non\text{-}violated\&\&truth}$. TS is calculated as Eq. (2).

$$TS = T_{non-violated\&\&truth}/T_{total} \tag{2}$$

False Satisfaction (FS): MR has no violation, and the "Source" prediction does not match the ground truth. This rate indicates the inconsistency between the system's behaviour and MR. FS is calculated by counting the occurrences of non-violated cases with inconsistent results between "Source" outputs and ground truth, which are cases with similarity scores above twenty between "Source" and "Follow-up" outputs, and with similarity scores below twenty between "Source" and ground truth outputs in this study. Let the total number of test trials be T_{total} and the non-violated and truth-unmatched cases be $T_{non\text{-}violated\&\&not\text{-}truth}$. FS is calculated as Eq. (3).

$$FS = T_{non-violated\&\&not-truth}/T_{total} \tag{3}$$

True Violation (TV): MR has a violation, and the "Source" prediction matches the ground truth. This indicates that the violation is consistent with the match of the "Source" prediction and the ground truth. TV is calculated by counting the occurrences of violated cases with consistent results between "Source" outputs and ground truth, which are the cases with the similarity scores below twenty between "Source" and "Follow-up" outputs, and with the similarity scores above twenty between "Source" and ground truth outputs in this study. Let the total number of test trials be T_{total}, and the violated and truth-matched cases be $T_{violated\&\&truth}$. TV is calculated as Eq. (4).

$$TV = T_{violated\&\&truth}/T_{total} \tag{4}$$

False Violation (FV): MR has a violation, and the "Source" prediction does not match the ground truth. This indicates that the violation is inconsistent with the mismatch of "Source" prediction and the ground truth. VR is the total of TV and FV.

FV is calculated by counting the occurrences of violated cases with inconsistent results between "Source" outputs and ground truth, which are the cases with similarity scores below twenty between "Source" and "Follow-up" outputs, and with similarity scores below twenty between "Source" and ground truth outputs in this study. Let the total number of test trials be T_{total}, and the violated and truth-unmatched cases be $T_{violated\&\&not\text{-}truth}$. FV is calculated as Eq. (5).

$$FV = T_{violated\&\&not-truth}/T_{total} \tag{5}$$

Genuine Satisfaction Ratio (GSR): This ratio indicates the proportion of non-violated cases with consistent results between "Source" outputs and ground truth over total violations and non-violated cases with inconsistent results between "Source" outputs and ground truth.

GSR is calculated by dividing TS by the sum of FS, TV, and FV. If the ratio is more than 100%, this indicates that over 50% of cases are TS. On the other hand, if the ratio is less than 100%, this indicates there are less than 50% cases are TS, which shows the MR introduces true violations (when TV is high) or there are many inconsistencies between "Source" outputs and ground truth outputs (when FV and FS are high). GSR is calculated as Eq. (6).

$$GSR = TS/(FS + TV + FV) \tag{6}$$

Table 2 explains that VR is the sum of FV and TV, and the sum of TS and FS is the total of consistent cases between "Source" and "Follow-up". Table 3 illustrates a six cases example of how to calculate VR and GSR. As there are two violated cases out of a total of six cases under the "Violated" column in Table 3, VR is 33.33%. On the other hand, two TS cases and four cases belonged to FV, FS and TV under the "Effective measure" column in Table 3. Thus, GSR is 50%.

Although the outputs between "Source" and "Follow-up" are stable, GSR is lower than 100%, which indicates that non-violated cases with consistent results between "Source" outputs and ground truth are less than 50% of the total of FS, TV and FV. The related MR is low in GSR due to a high number of FS and FV, which means there are many inconsistencies between "Source" outputs and ground truth outputs, or a high number of TV, which means MR introduces more inconsistencies (Table 3). Therefore, effective MR explains the instabilities of LLMs by introducing TV.

Table 2
Illustration of effective measurements.

	Source = Follow-up	Source != Follow-up (VR)
Source = Truth	TS	TV
Source != Truth	FS	FV

Table 3
Illustration of the similarity and effective calculations.

Case	Ground Truth	Source prediction	Follow-up prediction	Violated	Effective measure
1	true	true	true	N	**TS**
2	false	false	false	N	**TS**
3	true	false	true	**Y**	FV
4	false	true	true	N	FS
5	true	false	false	N	FS
6	false	false	true	**Y**	**TV**

$VR = T_{violated}/T_{total} = 2/6 = 33.33\%$
$GSR = TS/(FS+TV+FV) = 2/4 = 50\%$

3.7.4. Performance of code generation

We also occupied two performance measurements of LLMs to examine how the MRs affect the performance of LLMs.

pass@1: We utilize the pass@K metric (Athiwaratkun et al., 2022; Wang et al., 2022) to assess the performance of our subject LLMs. We only run one sample-generated code from the "Source" and "Follow-up" prompts, which is a pass@1 test (Athiwaratkun et al., 2022). Additionally, we use the publicly available test cases of MBPP from the CodeT project[6] (Chen et al., 2022) for calculating this measurement. We executed the codes generated by subject LLMs with "Source" and "Follow-up" prompts of the selected 100 MBPP datasets and tested them with the associated MBPP test cases from CodeT. Then, we counted the number of passed test cases out of 100 test cases as the pass@1 rate of LLMs. This measurement indicates the correctness of LLMs' generated codes to reflect their performance. Let the total number of test cases be T_{total}, and the passed test cases be T_{passed}. pass@1 is calculated as Eq. (7).

$$pass@1 = T_{passed} \,/\, T_{total} \tag{7}$$

Execution Success Rate (ESR): As the CodeT project does not provide test cases of Conala, we measure the performance of LLMs by simply executing the codes generated by subject LLMs with "Source" and "Follow-up" prompts of the selected 100 Conala datasets. Then, we count the number of successful executions out of 100 cases as the ESR of LLMs. This measurement indicates the ability of LLMs to generate executable codes. As there is no test case to check against the results of generated codes, ESR will be used with similarity scores to reflect the performance of LLMs with Conala datasets. Let the total number of test trails be T_{total} and the successfully executed cases be $T_{executable}$. ESR is calculated as Eq. (8).

$$ESR = T_{executable} \,/\, T_{total} \tag{8}$$

Suppose the codes generated by subject LLMs with "Follow-up" prompts give lower ESR or pass@1 than those generated by "Source" prompts. In that case, this can be interpreted as the associated MR with more inconsistencies, which is considered an effective MR to explain the instabilities of LLMs' outputs.

3.8. Experiment design

Our experiment includes five steps as follows:

Step 1. We applied the proposed MRs to two "Source" testing datasets and generated fourteen "Follow-up" testing datasets (Fig. 1). Step 2. Private LLMs train with the training dataset. Step 3. Private and public LLMs make predictions on the "Source" testing dataset to create "Source" outputs. Step 4. Private and public LLMs make predictions on "Follow-up" testing datasets to create "Follow-up" outputs. Step 5. We compare two separate sets of "Source" and "Follow-up" outputs and analyse their differences in measurement. A failure is revealed if the relation is violated, which means differences occur between the outputs (Fig. 2). In addition, based on the violations and performance, we can reveal the effectiveness of MRs. Eight trained private LLMs and two public LLMs were tested with two "Source" testing datasets and fourteen "Follow-up" testing datasets. Thus, there were twenty sets of "Source" outputs and one hundred and forty sets of "Follow-up" outputs for comparison in this study.

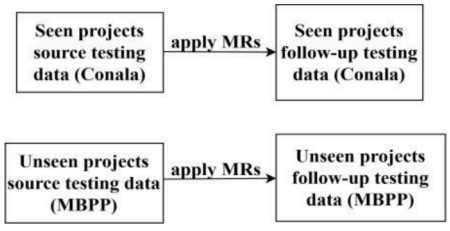


Fig. 1. Illustration of Step 1.

4. Discussion

In our study, we presented our experimental results and provided analysis and interpretation for our research questions.

4.1. RQ1: can MRs validate the stability of code generation large language models?

To address this question, we examined the results of "Zero violation with all MRs" under the simulation that there is no ground truth available.

4.1.1. Zero violation with all MRs

Table 4 displays VR values between "Source" and "Follow-up" outputs of the seen project with all subject LLMs and proposed MRs. Table 5 displays VR values between the "Source" and "Follow-up" outputs of the unseen project with all subject LLMs and proposed MRs. Tables 4 and 5 indicate that private LLMs (bert2bert, bert2gpt2, distilbert2mar, and electra2mar) did not violate MR1.1 and MR1.2 in both seen and unseen projects. The results show that the stability of these private LLMs' outputs is not affected by the case transformation of prompts as the "Source" outputs and the "Follow-up" outputs are similar. bert2bert, bert2gpt2, and distilbert2mar are using uncased version BERT as encoder, so zero violation with case transformation is reasonable for them even though the decoders of bert2gpt2 and distilbert2mar are not uncased version. electra2mar is not an uncased version, but Electra, as the encoder, was trained as a discriminative model that can replace tokens with plausible alternatives. Thus, this technique is able to guard against the influence of case transformation of prompts and keep electra2mar's outputs stable. Tables 4 and 5 also show that public LLMs copilot did not violate MR1.1 in the seen project and MR1, MR2 and MR3.2 in the unseen project. The results indicate that case transformation of the prompt will not affect the stability of the copilot's outputs. Additionally, the copilot performed better in the unseen project, which can be explained by public LLM, which is trained with vast

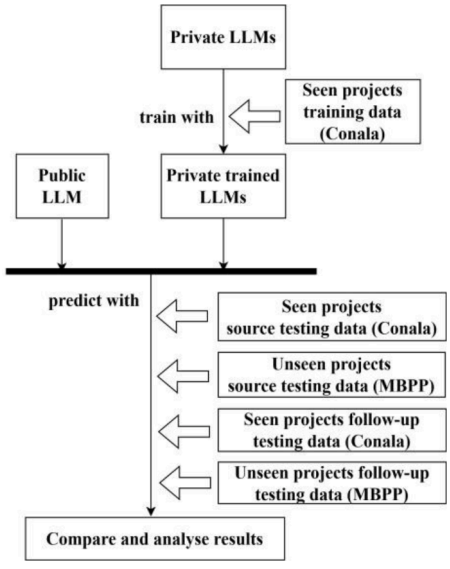


Fig. 2. Illustration of Step 2 to Step 5.

[6] The MBPP test cases are downloaded on 03/09/2024 from https://github.com/microsoft/CodeT/blob/main/CodeT/data/dataset/mbpp_sanitized_for_test_case_generation.jsonl.

Table 4
VR values of all LLMs with all MRs in seen project.

LLM	MR1.1	MR1.2	MR2.1	MR2.2	MR2.3	MR3.1	MR3.2
bert2bert	**0%**	**0%**	6%	13%	11%	35%	10%
bert2gpt2	**0%**	**0%**	12%	18%	15%	46%	21%
distilbert2mar	**0%**	**0%**	19%	19%	23%	50%	21%
distilroberta2mar	67%	44%	10%	10%	12%	41%	18%
electra2mar	**0%**	**0%**	20%	30%	33%	47%	29%
pegasus_x	68%	37%	11%	16%	20%	45%	21%
luke2mar	64%	39%	12%	24%	25%	47%	21%
marianCG	67%	37%	4%	9%	11%	47%	16%
copilot	**0%**	1%	1%	1%	2%	1%	3%
codeLlama	17%	5%	11%	1%	4%	13%	7%

Table 5
VR values of all LLMs with all MRs in unseen project.

LLM	MR1.1	MR1.2	MR2.1	MR2.2	MR2.3	MR3.1	MR3.2
bert2bert	**0%**	**0%**	33%	43%	54%	68%	17%
bert2gpt2	**0%**	**0%**	35%	49%	50%	81%	31%
distilbert2mar	**0%**	**0%**	34%	40%	41%	61%	24%
distilroberta2mar	90%	69%	34%	41%	44%	59%	29%
electra2mar	**0%**	**0%**	36%	58%	65%	52%	16%
pegasus_x	98%	51%	32%	41%	46%	66%	17%
luke2mar	85%	56%	34%	40%	40%	57%	26%
marianCG	88%	66%	24%	31%	31%	64%	22%
copilot	**0%**	0%	0%	0%	0%	3%	0%
codeLlama	7%	4%	2%	3%	9%	16%	5%

Table 6
TS, FS, TV, and FV values of all LLMs with all MRs in unseen project.

LLM		MR1.1	MR1.2	MR2.1	MR2.2	MR2.3	MR3.1	MR3.2
bert2bert	TS	2%	2%	1%	1%	1%	1%	1%
	FS	98%	98%	66%	56%	45%	31%	82%
	TV	0%	0%	1%	1%	1%	1%	1%
	FV	0%	0%	32%	42%	53%	67%	16%
bert2gpt2	TS	3%	3%	2%	2%	2%	1%	2%
	FS	97%	97%	63%	49%	48%	18%	67%
	TV	0%	0%	1%	1%	1%	2%	1%
	FV	0%	0%	34%	48%	49%	79%	30%
distilbert2mar	TS	4%	4%	3%	3%	3%	3%	3%
	FS	96%	96%	63%	57%	56%	36%	73%
	TV	0%	0%	1%	1%	1%	1%	1%
	FV	0%	0%	33%	39%	40%	60%	23%
distilroberta2mar	TS	0%	1%	0%	0%	0%	1%	2%
	FS	10%	30%	66%	59%	56%	40%	69%
	TV	2%	1%	2%	2%	2%	1%	0%
	FV	88%	68%	32%	39%	42%	58%	29%
electra2mar	TS	6%	6%	4%	4%	4%	2%	6%
	FS	94%	94%	60%	38%	31%	46%	78%
	TV	0%	0%	2%	2%	2%	4%	0%
	FV	0%	0%	34%	56%	63%	48%	16%
pegasus_x	TS	0%	3%	5%	4%	5%	4%	6%
	FS	2%	46%	63%	55%	49%	30%	77%
	TV	6%	3%	1%	2%	1%	2%	0%
	FV	92%	48%	31%	39%	45%	64%	17%
luke2mar	TS	0%	2%	3%	5%	5%	2%	4%
	FS	15%	42%	63%	55%	55%	41%	70%
	TV	5%	3%	2%	0%	0%	3%	1%
	FV	80%	53%	32%	40%	40%	54%	25%
marianCG	TS	1%	2%	4%	4%	3%	2%	4%
	FS	11%	32%	72%	65%	66%	34%	74%
	TV	3%	2%	0%	0%	1%	2%	0%
	FV	85%	64%	24%	31%	30%	62%	22%
copilot	TS	59%	59%	59%	59%	59%	58%	59%
	FS	41%	41%	41%	41%	41%	39%	41%
	TV	0%	0%	0%	0%	0%	1%	0%
	FV	0%	0%	0%	0%	0%	2%	0%
codeLlama	TS	52%	52%	55%	53%	51%	48%	53%
	FS	41%	44%	43%	44%	40%	36%	42%
	TV	3%	3%	0%	2%	4%	7%	2%
	FV	4%	1%	2%	1%	5%	9%	3%

datasets. copilot might have trained with datasets similar to MBPP, so it is reasonable that copilot performed well with the unseen projects. However, codeLlama did not achieve 0% VR with any MR.

Answering RQ1: Four private LLMs did not violate MR1.1 and MR1.2 in both projects, and public LLM copilot did not violate MR1.1 in both projects. These results confirm that the proposed MRs can serve as a validation tool for the stability of code generation LLMs' outputs under the simulation that there is no ground truth available. In other words, if the outputs of these five LLMs are different between the "Source" and MR1.1, it highlights areas in them that may require further investigation to address the causes of unstable outputs.

4.2. RQ2: how effective are MRs in explaining the instability of code generation large language models' outputs?

To address this question, we examined the outcomes of "Effectiveness and Performance measurements".

4.2.1. Effectiveness and performance measurements

The effectiveness and performance measurements have given a different perspective than VR in Tables 4 and 5. Tables 6 to 11 display the effectiveness and performance measurements between "Source" and "Follow-up" outputs of unseen and seen projects with all subject LLMs and proposed MRs. Public LLMs have performed better than private LLMs; even some private LLMs achieved zero VR (Tables 4 and 5), most of them contain significant proportions of FS, TV and FV (Tables 6 and 8) and the generated codes from private LLMs are not as executable as the codes generated by public LLMs (Tables 10 and 11).

In unseen project, copilot achieved over 130% GSR and 0% TV and FV with six out of seven MRs (Table 6 and 7), 84% pass@1 with MR1, MR2.2 and MR2.3, however below 84% pass@1 with MR2.1 and MR3, which are lower than "Source" outputs in the unseen project (Table 11). The results indicate that the copilot is insensitive to MR1, MR2.2 and MR2.3 and capable of generating executable codes consistent with ground truth without being affected by MR1, MR2.2 and MR2.3 in the unseen project. On the other hand, the copilot's unstable outputs, which are the true violations, can be explained by MR2.1 and MR3, which means MR2.1 and MR3 can lead to copilot's unstable code generations in unseen project (Tables 4, 5, 7 and 9). codeLlama achieved over 100% GSR with all MRs except 92.31% with MR3.1 (Table 7), and below 61% pass@1 values with all MRs, which are lower than "Source" outputs in the unseen project (Table 11). Therefore, all the MRs can contribute to codeLlama's unstable code generations in the unseen project.

Comparatively, all the private LLMs performed worse than public LLMs regarding effectiveness and performance measurements. All private LLMs have zero pass@1 with all MRs (Table 11) in the unseen project because all the generated codes, including the "Source" outputs, are non-executable or unrelated to the prompts. Even bert2bert, bert2gpt2, distilbert2mar, and electra2mar have zero VR with MR1.1 and MR1.2 in unseen project (Table 5), all private LLMs' GSR are below 7%, and pegasus_x, distilroberta2mar, and luke2mar had zero GSR means they have no TS with MR1.1 and MR2 (Table 7).

Table 6 shows that most private LLMs have high proportions of FS and FV, indicating that "Source" outputs are primarily inconsistent with ground truth in the unseen project (Table 12). Despite the low GSR, table 6 shows MR1 did not introduce TV in bert2bert, bert2gpt2, distilbert2mar, and elecrtra2mar, MR2 did not introduce TV in marianCG and luke2mar, and MR3.2 did not introduce TV in distilrobert2mar, electra2mar and marianCG. Although effectiveness results in this study show the proposed MRs did not introduce TV in private LLMs' outputs in the unseen project, the pass@1 values of private LLMs are all zero, which suggests all the proposed MRs can contribute to the private LLMs' unstable outputs in the unseen project.

In seen project, public LLMs achieved over 100% GSR in the seen project with all MRs (Tables 8 and 9), especially copilot achieved a similar GSR with all MRs as in the unseen project (Tables 7 and 9). Additionally, copilot achieved over 99% ESR with all MRs except MR1.1 and MR2.2 (Table 10). The results show that copilot is capable of generating executable codes that are consistent with ground truth without being affected by all MRs, except MR2.3, which introduced 1% TV in the seen project (Table 8) and MR1.1 and MR2.2, which have lower ESR than the "Source" output. Thus, MR1.1, MR2.2, and MR2.3 can contribute to the copilot's unstable outputs in the seen project. codeLlama achieved better GSR in the seen project, and all GSR values are over 150% with all MRs, except 117.39% with MR1.1 (Tables 7 and 9). codeLlama also achieved over 94% ESR with all MRs, except MR1.1, MR2.1 and MR2.3 (Table 10) in the seen project. However, table 8 shows all MRs introduced TV in codeLlama's outputs. Therefore, all the proposed MRs can also contribute to the codeLlama's unstable outputs in the seen project.

Most private LLMs performed better in GSR in seen projects, as bert2gpt2, distilbert2mar, electra_marian, pegasus_x, and luke2mar achieved over 100% GSR with MR1 and MR2 (Table 9). However, Tables 8 and 10 show that MR2 and MR3 introduce TV or reduce ESR to lower than "Source" outputs in all private LLMs in the seen project. Tables 6 to 9 also show that MR1 introduces TV in private LLMs distilroberta2mar, pegasus_x, luke2mar, and marianCG. The results reflect that these four private LLMs are case-sensitive. Thus, all the proposed MRs can lead to private LLMs' unstable outputs in the seen project.

Answering RQ2: Results indicated that the subject LLMs' unstable outputs can be explained by "case transformation of prompt", "duplication of prompt", and "tokens in prompt changed by paraphrasing", which introduce true violations to LLMs and affect LLMs to generate executable codes. Therefore, all the proposed MRs can be effective tools for explaining the instability of LLMs' outputs in both projects.

4.3. RQ3: what are the causes that make MR less effective in explaining the instability of code generation large language models' outputs?

To address this research question, we examined with different proposed MRs.

4.3.1. MR1

Tables 8 to 10 show that MR1 did not introduce TV or reduce ESR to

Table 7
GSR values of all LLMs with all MRs in unseen project.

LLM	MR1.1	MR1.2	MR2.1	MR2.2	MR2.3	MR3.1	MR3.2
bert2bert	2.04%	2.04%	1.01%	1.01%	1.01%	1.01%	1.01%
bert2gpt2	3.09%	3.09%	2.04%	2.04%	2.04%	1.01%	2.04%
distilbert2mar	4.17%	4.17%	3.09%	3.09%	3.09%	3.09%	3.09%
distilroberta2mar	0.00%	1.01%	0.00%	0.00%	0.00%	1.01%	2.04%
electra2mar	6.38%	6.38%	4.17%	4.17%	4.17%	2.04%	6.38%
pegasus_x	0.00%	3.09%	5.26%	4.17%	5.26%	4.17%	6.38%
luke2mar	0.00%	2.04%	3.09%	5.26%	5.26%	2.04%	4.17%
marianCG	1.01%	2.04%	4.17%	4.17%	3.09%	2.04%	4.17%
copilot	143.90%	143.90%	143.90%	143.90%	143.90%	138.10%	143.90%
codeLlama	108.33%	108.33%	122.22%	112.77%	104.08%	92.31%	112.77%

Table 8
TS, FS, TV, and FV values of all LLMs with all MRs in seen project.

LLM		MR1.1	MR1.2	MR2.1	MR2.2	MR2.3	MR3.1	MR3.2
bert2bert	TS	45%	45%	44%	43%	41%	33%	43%
	FS	55%	55%	50%	44%	48%	32%	47%
	TV	0%	0%	1%	2%	4%	12%	2%
	FV	0%	0%	5%	11%	7%	23%	8%
bert2gpt2	TS	62%	62%	59%	55%	55%	33%	51%
	FS	38%	38%	29%	27%	30%	21%	28%
	TV	0%	0%	3%	7%	7%	29%	11%
	FV	0%	0%	9%	11%	8%	17%	10%
distilbert2mar	TS	66%	66%	58%	57%	55%	34%	54%
	FS	34%	34%	23%	24%	22%	16%	25%
	TV	0%	0%	8%	9%	11%	32%	12%
	FV	0%	0%	11%	10%	12%	18%	9%
distilroberta2mar	TS	23%	40%	61%	61%	61%	38%	57%
	FS	10%	16%	29%	29%	27%	21%	25%
	TV	43%	26%	5%	5%	5%	28%	9%
	FV	24%	18%	5%	5%	7%	13%	9%
electra2mar	TS	59%	59%	52%	46%	42%	36%	46%
	FS	41%	41%	28%	24%	25%	17%	25%
	TV	0%	0%	7%	13%	17%	23%	13%
	FV	0%	0%	13%	17%	16%	24%	16%
pegasus_x	TS	23%	46%	65%	63%	60%	42%	58%
	FS	9%	17%	24%	21%	20%	13%	21%
	TV	45%	22%	3%	5%	8%	26%	10%
	FV	23%	15%	8%	11%	12%	19%	11%
luke2mar	TS	28%	45%	58%	52%	51%	40%	53%
	FS	8%	16%	30%	24%	24%	13%	26%
	TV	35%	18%	5%	11%	12%	23%	10%
	FV	29%	21%	7%	13%	13%	24%	11%
marianCG	TS	23%	47%	72%	70%	70%	42%	66%
	FS	10%	16%	24%	21%	19%	11%	18%
	TV	49%	25%	0%	2%	2%	30%	6%
	FV	18%	12%	4%	7%	9%	17%	10%
copilot	TS	59%	59%	59%	59%	58%	59%	59%
	FS	41%	40%	40%	40%	40%	40%	38%
	TV	0%	0%	0%	0%	1%	0%	0%
	FV	0%	1%	1%	1%	1%	1%	3%
codeLlama	TS	54%	60%	61%	63%	63%	60%	62%
	FS	29%	35%	28%	36%	33%	27%	31%
	TV	10%	4%	3%	1%	1%	4%	2%
	FV	7%	1%	8%	0%	3%	9%	5%

Table 9
GSR values of all LLMs with all MRs in seen project.

LLM	MR1.1	MR1.2	MR2.1	MR2.2	MR2.3	MR3.1	MR3.2
bert2bert	81.82%	81.82%	78.57%	75.44%	69.49%	49.25%	75.44%
bert2gpt2	163.16%	163.16%	143.90%	122.22%	122.22%	49.25%	104.08%
distilbert2mar	194.12%	194.12%	138.10%	132.56%	122.22%	51.52%	117.39%
distilroberta2mar	29.87%	66.67%	156.41%	156.41%	156.41%	61.29%	132.56%
electra2mar	143.90%	143.90%	108.33%	85.19%	72.41%	56.25%	85.19%
pegasus_x	29.87%	85.19%	185.71%	170.27%	150.00%	72.41%	138.10%
luke2mar	38.89%	81.82%	138.10%	108.33%	104.08%	66.67%	112.77%
marianCG	29.87%	88.68%	257.14%	233.33%	233.33%	72.41%	194.12%
copilot	143.90%	143.90%	143.90%	143.90%	138.10%	143.90%	143.90%
codeLlama	117.39%	150.00%	156.41%	170.27%	170.27%	150.00%	163.16%

Table 10
ESR values of all LLMs with “Source” and all MRs in seen project.

LLM	SRC	MR 1.1	MR 1.2	MR 2.1	MR 2.2	MR 2.3	MR 3.1	MR 3.2
bert2bert	0%	0%	0%	0%	0%	0%	0%	0%
bert2gpt2	1%	1%	1%	0%	0%	0%	0%	1%
distilbert2mar	3%	3%	3%	4%	3%	3%	4%	5%
distilroberta2mar	11%	7%	9%	6%	7%	6%	8%	11%
electra2mar	7%	6%	6%	8%	6%	5%	7%	12%
pegasus_x	6%	7%	11%	3%	6%	3%	14%	7%
luke2mar	4%	3%	6%	4%	5%	3%	10%	3%
marianCG	4%	6%	3%	4%	3%	3%	8%	5%
copilot	99%	81%	99%	99%	98%	99%	99%	99%
codeLlama	94%	84%	96%	93%	95%	88%	97%	94%

lower than "Source" outputs in bert2gpt2 and distilbert2mar, which means MR1 is ineffective in explaining their instabilities in the seen project. Although they use uncased BERT as an encoder, which gives them an edge in handling case transformation of prompts, bert2bert, which uses uncased BERT as encoder and decoder, has zero ESR with all MRs. So, using uncased BERT as an encoder and mixing with other technologies like GPT2 or marianCG as a decoder could resist the influence of MR1 in the seen project.

Results also show that copilot is insensitive with MR1 in the unseen project (Tables 6, 7 and 11). copilot uses the latest GPT-4o generative AI technology and is trained with up-to-date information from the vast amount of public web information and datasets. copilot may have been trained with datasets similar to MBPP (unseen project). Thus, it is reasonable that copilot is insensitive to MR1.

4.3.2. MR2

Tables 6, 7 and 11 show that MR2 did not introduce TV in copilot in the unseen project, and MR2.2 and MR2.3 did not reduce pass@1 to lower than "Source" outputs in copilot in the unseen project. These results show that copilot is insensitive to MR2 in the unseen project as copilot uses the multimodal large language models, including latest GPT-4o generative AI technology, and is trained with up-to-date information from the vast amount of public web information and datasets. It is possible that copilot has been trained with datasets similar to MBPP (unseen project), and the use of multimodal large language models can handle the duplication of prompts. Thus, it is reasonable that copilot is insensitive to MR2.

4.3.3. MR3

Tables 8, 9 and 10 show that MR3 did not introduce TV or reduce ESR to lower than "Source" outputs in copilot in the seen project only. These results show that copilot can resist the influence of MR3 in the seen project. copilot may use the multimodal large language models to handle prompt paraphrasing. Thus, it is reasonable that the copilot is insensitive to MR3.

Answering RQ3: Results show that using uncased BERT as encoder and mixing with other technologies as decoder can resist the influence of MR1, and copilot can resist the influences of all MRs due to it being pre-trained with a vast number of datasets and using multimodal large language models. Thus, using a mix of LLM technologies and pre-trained with a vast number of datasets can cause the MRs under study to be ineffective. In addition, results imply that all LLMs, except copilot, have low semantic understanding and rely on statistical patterns to infer the prompts (Browning and LeCun, 2023; X. Zhou et al., 2020) because all the prompt transformations are semantic preserving and only copilot can resist MRs' influences.

4.4. Further analysis

4.4.1. Effectiveness and performance measurements of MT

The VR results indicate that the validation capacity of MRs is at zero violation, meaning that predictions between "Source" and "Follow-up" are consistent. GSR, TV, TS, FV, and FS results demonstrate the effectiveness of MRs as effectiveness measurements introduce ground truth into the measurements; if TV is not zero, this can reflect MRs introduce instability of LLMs' outputs, which can explain the instability by MRs. Especially with the unseen project, effectiveness measurements, GSR, reveal that all private LLMs performed worse than public LLMs in TS, which means private LLMs have many inconsistencies between "Source" outputs and ground truth, despite some private LLMs having zero VR. Thus, effectiveness measurements are essential for understanding the instability of LLMs, which is supported by Jiang et al. (2022).

Effectiveness measurements can also provide insights into MRs and LLMs. Despite not achieving zero violation results, private LLM marianCG with MR2.1 has the highest GSR and zero TV in the seen project (Tables 8 and 9). However, performance measurements show a different perspective. In the seen project, public LLMs' ESRs are better than private LLMs' ESRs (Table 10) because public LLMs generate codes that include "import library" statements, which makes the codes executable by "copy and paste" to the testing environment (which is "Jupyter Notebook" development platform in this study). We further reviewed the reference codes (ground truth) of the seen project and found that most of them are code snippets and are not executable by themselves. Therefore, even though marianCG has high GSRs with MR2, the reference truths are not executable, so it has low ESR with MR2 because marianCG's outputs are highly consistent with non-executable "Source" outputs and non-executable ground truths. When the reference code snippets are not executable, it can lead to inaccurate performance measurements. Therefore, it is important to include effectiveness measurements to reflect the instability of the LLMs' outputs.

4.4.2. MR creation

This study applied Z. Q. Zhou et al.'s (2020) "symmetry" MR pattern and created symmetric MRs that can be used to transform prompts and preserve the semantic meanings of prompts. The proposed MRs have been chosen as they are easy to understand and apply to other domains or languages. The results not only show that the proposed MRs are effective tools for validating the stability of LLMs' outputs, they also show several other insights.

Table 12
The percentage of total cases that have consistent results between "Source" outputs and ground truth.

LLM	Source = Truth	
	Seen project	Unseen project
bert2bert	45%	2%
bert2gpt2	62%	3%
distilbert2mar	66%	4%
distilroberta2mar	66%	2%
electra2mar	59%	6%
pegasus_x	68%	6%
luke2mar	63%	5%
marianCG	72%	4%
copilot	59%	59%
codeLlama	64%	55%

Table 11
pass@1 values of all LLMs with "Source" and all MRs in unseen project.

LLM	SRC	MR 1.1	MR 1.2	MR 2.1	MR 2.2	MR 2.3	MR 3.1	MR 3.2
bert2bert	0%	0%	0%	0%	0%	0%	0%	0%
bert2gpt2	0%	0%	0%	0%	0%	0%	0%	0%
distilbert2mar	0%	0%	0%	0%	0%	0%	0%	0%
distilroberta2mar	0%	0%	0%	0%	0%	0%	0%	0%
electra2mar	0%	0%	0%	0%	0%	0%	0%	0%
pegasus_x	0%	0%	0%	0%	0%	0%	0%	0%
luke2mar	0%	0%	0%	0%	0%	0%	0%	0%
marianCG	0%	0%	0%	0%	0%	0%	0%	0%
copilot	84%	84%	84%	81%	84%	84%	76%	79%
codeLlama	61%	57%	56%	58%	54%	56%	45%	58%

Firstly, results show that all subject LLMs, except copilot, have fewer violations with MR3.2 than MR3.1 (Tables 4 to 9). This result means LLMs are more sensitive to prompts that were paraphrased by online paraphrasing tools than by AI paraphrasing tools. It can be attributed to online paraphrasing tools are swapping tokens with thesaurus of tokens, such as, "loop over files in directory' source'" to "circle over documents in registry' source'". On the other hand, the AI paraphrasing tool (Mistral) paraphrased the prompt with fewer token changes, such as "loop over files in directory' source'" to "loop through files in directory' source'". Additionally, the stability of six out of seven LLMs' outputs is affected by MR3.1 in both projects (Table 6 to 11). MR3.1 uses an online paraphrasing tool to swap tokens with a thesaurus, which can impact the stability of highly-trained LLMs. Thus, paraphrasing the prompt by swapping tokens with thesaurus is a valid MR choice because this can explain the instability of six out of seven LLMs' outputs, and the results advised keeping the critical terms used in prompts constant and consistent for the stability of all private LLMs' and codeLlama's outputs.

Secondly, private LLMs bert2bert, bert2gpt2, distilbert2mar, and electra2mar, which used uncased BERT, have zero VR with MR1 as expected (Table 4 and 5). Although uncased BERT gives them an edge in handling case transformation of prompts, they all have high FS in the unseen project (Tables 8 and 9). Especially bert2bert, effective measurements show bert2bert cannot achieve over 100% GSR with all MRs in both projects (Tables 7 and 9) even if it uses uncased BERT in both encoder and decoder. Thus, results justify the selection of MR1; even if the uncased BERT is used, effectiveness and performance measurements can still reveal that bert2bert's "Source" outputs are not executable and inconsistent with ground truth.

4.4.3. Private and public code generation LLMs

The results indicate that public LLMs perform better in seen and unseen projects, and there are several reasons behind this. Firstly, copilot and codeLlama have been trained with vast number of datasets; therefore, the distinction of unseen and seen datasets in this study might not apply to the public LLMs. However, marianCG has been trained with Conala, and it only performed well in GSR with MR2 in the seen project (Table 9). Therefore, public LLMs are more capable of handling the different semantic preserving transformations of prompts by their continuous vast training data and their technologies. As a result, public LLMs have a higher level of semantic understanding than private LLMs.

Secondly, Table 10 shows that even though the ground truth of seen projects cannot be executed, public LLMs are powerful enough to add associated "import library" statements and make public LLMs' generated codes executable. Adding more statements to make the generated codes executable is the reason why public LLMs generated codes that were less similar to the ground truth with the seen project than private LLMs. Private LLMs are trained using the seen dataset, which means they are likely to produce code that resembles the ground truth from the seen dataset. Consequently, if the ground truths are not executable, we can expect that the code generated by private LLMs will also be non-executable. On the other hand, public LLMs' outputs have included more statements to make their outputs executable despite being less consistent with ground truth than the private LLMs' outputs. Additionally, Table 11 shows that only public LLMs can generate codes that can pass the test cases from the CodeT project, while private LLMs can generate consistent outputs with "Source" outputs. However, private LLMs' generated codes are non-executable as they all have zero pass@1 in the unseen project. The results show that public LLMs can generate codes that are more executable than private LLMs, indicating a higher level of semantic understanding.

Thus, if software houses attempt to employ private code generation LLMs to avoid risk of exposing corporate secrets and enhance software development's productivity at the public code generation LLM level, software houses need to look for solutions to improve the semantic understanding of private code generation LLMs.

4.4.4. Training data for code generation LLMs

The results indicate that all LLMs perform similarly in VR and GSR in the seen project (Tables 4 and 9). However, public LLMs perform significantly better in the unseen project. The critical distinction between seen and unseen projects is the availability of training data. Without adequate training, it is reasonable that private LLMs perform worse than public LLMs when presented with prompts for the unseen project, as public LLMs have been trained with a continuous vast number of datasets, some of which might be similar to the unseen project. The performance difference shows that providing more different types of training data for code generation could be one of the approaches to enhance the semantic understanding of private LLMs. Although using private LLMs can avoid risk of exposing corporate secrets, vast data training or training targeted at semantic understanding enhancement is needed for private LLMs, as the results in the unseen project show that private LLMs generated incorrect codes.

One potential approach for private LLMs is to use data from existing projects to train them. Cross-project training, which involves using data from other projects, has been successfully employed to enhance the performance of software defect predictions (Steenhoek et al., 2023; Tabassum et al., 2020; Yamamoto et al., 2023). To further enhance the code generation capabilities of private LLMs for future unseen projects, it may be beneficial to consider using diverse code generation related training data from various existing projects from different sources. Additionally, specific projects within one's own company could be leveraged to train private LLMs to be more specialized in a particular method of code generation.

Public LLMs codeLlama and copilot optimized its training by using various types of data from different sources, such as source codes, discussions about source codes, code snippets, and natural language questions and answers; and various approaches, including code infilling and long context fine-tuning (Achiam et al., 2023; Roziere et al., 2023; Touvron, Lavril, et al., 2023, 2023). As a result, public LLMs are capable of handling the unseen project. Future studies can apply the approaches of codeLlama (Roziere et al., 2023) to systematically collect and convert different sources into training data, which can enhance the semantic understanding of private LLMs.

Although marianCG was trained with Conala, it only had better results than public LLMs in the seen project with MR2. Thus, public LLMs show a higher level of semantic understanding of prompts, and training that can improve the semantic understanding of LLMs can be helpful. As the results have demonstrated that the proposed symmetric MRs are effective and semantic preserving, they can also create different semantic preserving training datasets based on "Source" training datasets for further training, like data augmentation, to enhance the semantic understanding even in the situation of testing data scarcity.

Thus, using multiple sources to create more training datasets for private LLMs is a potential approach to improve the semantic understanding of private code generation LLMs.

4.4.5. Prompt engineering

The generation results of the prompt must be justifiable, explainable, and verifiable (Shah, 2024). It is important to explore how the prompts are formulated to guide LLMs to replicate the code generation because most LLMs are challenging to update or modify. Therefore, prompt engineering plays a critical role in ensuring stable and reliable code generation.

Tables 4 to 11 demonstrate that, except for MR3.1, the proposed MRs can serve as effective validation tools for the stability of LLMs' outputs. These results also suggest that the concept of symmetry can be applied to prompt engineering. Symmetric MRs are expected to maintain stable code generation as long as the semantic meanings of the prompts remain consistent. Therefore, prompts created by symmetric MRs can generate source codes that are justifiable, explainable, and verifiable.

The results of MR3.1 show that it is essential to identify the key terms of prompt for improving the stability. Thus, paraphrasing with the

thesaurus swapping of tokens in prompts and testing with our approach can assist in identifying the critical tokens of prompts that cannot be changed to improve stability.

Lastly, more descriptions and more structures in the prompt might be needed for private LLMs as the pass@1 results show they provide inexecutable codes or codes that are not related to the prompts (Tables 10 and 11), which reflects that private LLMs do not understand the prompts in the unseen project.

5. Threats to validity

This section discusses the threats to validity that are faced by this study.

5.1. Internal validity

One internal threat to the experiment is related to parameter configuration. Standard parameters were applied to all LLMs, and we built eight private LLMs with the pre-trained models provided by the machine learning community "Hugging Face" and the libraries of "Python". The public LLMs are unconfigurable and available on "Poe.com" website and "Microsoft Edge" browser. Additionally, we utilized Microsoft Excel to create the "Follow-up" testing datasets. These tools and technologies are reliable and widely used. White-box testing has also been conducted to ensure error-free source codes. Another concern is the measurements of this study. We specifically focused on the relationship between MRs and measurements, and the prediction outcomes of all LLMs were listed and checked to ensure measurements were calculated accurately. At last, the transformations of the prompt might lead to incorrect code generations. All the proposed MRs are inspired by Z. Q. Zhou et al.'s (2020) "symmetry" MR pattern, and results show LLMs can generate "Follow-up" executable outputs that are not violated with "Source" outputs and ground truths, which indicates that the choice of the proposed MRs is valid as they are capable of preserving the semantic meaning of prompts and are easy to understand and apply. We believe the chance of MRs creating invalidity is low.

5.2. External validity

One external threat is the generality of the proposed MRs. This study uses symmetric MRs, which are semantic-preserving, generic and applicable to any experiment using the English language. This study has chosen two publicly available datasets with over 20,000 data pairs, and nine different LLMs were employed to run trials with seven MRs. The results should be generalized, and the effect of datasets on the effectiveness of the proposed MRs should be small. Lastly, the measurements used in this study are rather general and replicable; thus, the potential threat to the external validity of this study is low.

5.3. Construct validity

The independent variables of this study are prediction outcomes of the "Source" and "Follow-up" testing datasets of ten subject code generation LLMs, of which eight were trained by the same training dataset as private LLMs, and two are publicly available LLMs. The dependent variable is the differences between the independent variables, which are the differences between the prediction outcomes of "Source" and "Follow-up" testing datasets of LLMs, which is the VR. In addition, this study uses the accuracy of "Source" predictions and dependent variables to assess the effectiveness of MRs: the GSR, TV, FV, TS and FS. At last, this study uses the prediction outcomes of "Source" and "Follow-up", which are the generated codes, to assess the performance of MRs: the ESR and pass@1.

This study employed a score of twenty as the cut-off score to classify the "Source" and "Follow-up" outputs as similar. The chosen score was based on the minimum bleu score of GoogleCloud translation, indicating that the translation is understandable (GoogleCloud, 2023). However, it is essential to note that the cut-off score of twenty was a heuristic choice. Despite this, this cut-off score is applied to all LLMs with all the proposed MRs, ensuring that the experimental result contains no bias. This study aims to validate the stability of code generation of LLMs' outputs. Thus, we believe using this cut-off score is appropriate.

As this study focuses on validating the stability of LLMs' outputs and measuring the effectiveness of MRs, VR could show the violations. In contrast, GSR, TV, FV, TS and FS could show the effectiveness, and ESR and pass@1 could show how the MRs affect the performance of LLMs. Suppose there is no violation between "Source" and "Follow-up" prediction outcomes; it means there is no inconsistent prediction between them. This result indicates that the associated LLM is insensitive to the related MR, and the MR could be used for validation. A violation with the MR could reveal an error in LLM because the related LLM should be insensitive to the MR. If VR is not zero and MR introduces TV, meaning "Source" predictions are consistent with ground truth and inconsistent with "Follow-up" predictions, this is interpreted as the related MR is effective as it can explain the instability of LLMs' output.

Furthermore, the performance measurements of "Follow-up" outputs are lower than the "Source" outputs; this also can be interpreted as the associated MR is effective as it can explain the instability of LLMs' output. Those can correctly measure the intent of this study. To sum up, the construct validity of the independent variables and dependent variables of this article should be acceptable.

6. Conclusion and future direction

As using public LLMs could leave companies vulnerable to data leaks and lawsuits (Post, 2023), the use of private LLMs could avoid risk of exposing corporate secrets and advance software development by improving productivity. However, an effective evaluation approach is needed to validate these LLMs. This study proposes using symmetric MRs of MT to validate code generation LLMs and to study the effectiveness of MT in validating the stability of LLMs' outputs. Seven symmetric MRs, based on the concept of symmetry and semantic preserving, were defined. We conducted experiments using the proposed MRs in the test scenario where the software development house uses private LLMs to generate source codes for software enhancements and maintenance while avoid exposing the company's software assets. Eight private LLMs and two public LLMs are employed in this study.

In conclusion, this study demonstrated that the proposed MRs could serve as a validation tool with violation measurements for the stability of code generation LLMs' outputs under the simulation that there is no ground truth available. One public LLM and four private LLMs did not violate MR1.1 (Case transformation of prompts) in the seen or unseen project, suggesting that if the outputs of these five LLMs violate MR1.1, there are faults in those LLMs systems needed to be investigated that cause the unstable outputs.

The effectiveness and performance results, which include identifying the inconsistencies between performance, "Source" predictions, "Follow-up" predictions, and ground truths related to MRs, indicated that proposed MRs are indeed effective tools for explaining the instability of LLM's outputs by "Case transformation of prompts", "Duplication of prompts", and "Paraphrasing of prompts". As MRs are semantic preserving, results also show that enhancing the semantic understanding of prompts is required by all private LLMs and codeLlama for better code generation. The main factors limiting the effectiveness of MRs in identifying the causes of instability are LLMs use a mix of LLM technologies and LLMs are pre-trained with the vast number of datasets. On the other hand, the factors imply that LLMs have low semantic understanding of prompts because all the prompt transformations are semantic preserving and only the copilot can resist MRs' influences.

Additionally, results show that, firstly, effectiveness measurements are essential for understanding the instability of LLMs, especially when performance measurements cannot give an accurate reflection.

Secondly, the proposed MR choices are valid. Thirdly, it is necessary to look for solutions to improve the semantic understanding of private code generation LLMs, in order to enhance their performance. Using multiple sources, such as existing projects, source codes, discussions about source codes, code snippets, and natural language questions and answers, to create more training datasets for private LLMs is a potential approach. Fourthly, proposed symmetric MRs can be used for prompt engineering and identifying key tokens of prompts. Lastly, private LLMs require prompts that include more detailed descriptions and structures, in order to better understand the prompts.

6.1. Future direction

We provide a few concrete future work recommendations for researchers informed by our results. Firstly, future studies can extend our MT model with effectiveness measurements for validating different domains, such as extending this experiment to class-level or line-level code generation. Secondly, more studies are needed to explore more different MRs with symmetry, which is helpful as different symmetric MRs can generate more testing datasets to measure the effectiveness of validations and further understand the LLMs. Thirdly, another focus is exploring approaches to turn different sources, such as existing projects, into training data systematically for LLMs to improve the semantic understanding of prompts, especially for the prompts of future unseen projects.

Fourthly, this study only focused on "symmetry" MR pattern and future studies can explore "change direction" MR pattern (Z. Q. Zhou et al., 2020) to validate the stability of the LLMs' outputs and understand LLMs further. Fifthly, apart from increasing training data, future studies can focus on other approaches, such as using a mix of different LLMs' technologies, for improving the semantic understanding of private LLMs to enhance their performance.

Sixthly, exploring the application of the concept of symmetry to prompt engineering for developing a verifiable, validated, and replicable code generation process could be a potential future research focus. Lastly, as the public LLMs have been trained with a vast number of datasets, the distinction between unseen and seen datasets in this study does not apply to the public LLMs. Future research could involve creating a private dataset that is not publicly accessible and using it as an unseen dataset to experiment with public LLMs.

6.2. Limitations

Due to difficulties in accessing OpenAI's ChatGPT-4 and Google's Gemini technology in Hong Kong, this study did not consider those latest AI technologies.

Declaration of generative AI in scientific writing

The authors declare that they have not used any generative artificial intelligence (AI) and AI-assisted technologies in the writing process.

CRediT authorship contribution statement

Pak Yuen Patrick Chan: Conceptualization, Methodology, Software, Formal analysis, Investigation, Data curation, Validation, Writing – original draft, Writing – review & editing. **Jacky Keung:** Supervision, Validation, Writing – review & editing. **Zhen Yang:** Validation, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

This work is supported in part by the General Research Fund of the Research Grants Council of Hong Kong and the research funds of the City University of Hong Kong (6000796, 9229109, 9229098, 9220103, 9229029), and the Natural Science Foundation of Shandong Province under Grant ZR2024QF093.

References

Achiam, J., Adler, S., Agarwal, S., Ahmad, L., Akkaya, I., Aleman, F.L., Almeida, D., Altenschmidt, J., Altman, S., & Anadkat, S. (2023). Gpt-4 technical report. arXiv preprint.

Alur, R., Bodik, R., Juniwal, G., Martin, M.M., Raghothaman, M., Seshia, S.A., Singh, R., Solar-Lezama, A., Torlak, E., Udupa, A., 2013. Syntax-guided synthesis. 2013 Formal Methods Comput. Aided Design.

Asare, O., Nagappan, M., Asokan, N., 2023. Is github's copilot as bad as humans at introducing vulnerabilities in code? Empir. Softw. Eng. 28 (6), 129.

Athiwaratkun, B., Gouda, S.K., Wang, Z., Li, X., Tian, Y., Tan, M., Ahmad, W.U., Wang, S., Sun, Q., & Shang, M. (2022). Multi-lingual evaluation of code generation models. arXiv preprint.

Austin, J., Odena, A., Nye, M., Bosma, M., Michalewski, H., Dohan, D., Jiang, E., Cai, C., Terry, M., & Le, Q. (2021). Program synthesis with large language models. arXiv preprint.

Browning, J., LeCun, Y., 2023. Language, common sense, and the Winograd schema challenge. Artif. Intell., 104031

Cassano, F., Gouwar, J., Nguyen, D., Nguyen, S., Phipps-Costin, L., Pinckney, D., Yee, M.-H., Zi, Y., Anderson, C.J., Feldman, M.Q., 2023. MultiPL-E: a scalable and polyglot approach to benchmarking neural code generation. IEEE Trans. Softw. Eng. 49 (7), 3675–3691.

Chaudhuri, S., Ellis, K., Polozov, O., Singh, R., Solar-Lezama, A., & Yue, Y. (2021). Neurosymbolic programming. Foundations and Trends® in Programming Languages, 7(3), 158–243.

Chen, B., Zhang, F., Nguyen, A., Zan, D., Lin, Z., Lou, J.-G., & Chen, W. (2022). Codet: code generation with generated tests. arXiv preprint.

Chen, M., Tworek, J., Jun, H., Yuan, Q., Pinto, H.P.d.O., Kaplan, J., Edwards, H., Burda, Y., Joseph, N., & Brockman, G. (2021). Evaluating large language models trained on code. arXiv preprint.

Chen, T.Y., Cheung, S.C., & Yiu, S.M. (2020). Metamorphic Testing: a New Approach for Generating Next Test Cases. arXiv.org. https://doi.org/10.48550/arxiv.2002.12543.

Chen, T.Y., Kuo, F.-C., Liu, H., Poon, P.-L., Towey, D., Tse, T.H., Zhou, Z.Q., 2018. Metamorphic testing: a review of challenges and opportunities. ACM Comput. Surv. 51 (1), 4. https://doi.org/10.1145/3143561.

Clark, K., Luong, M.-T., Le, Q.V., & Manning, C.D. (2020). ELECTRA: pre-training text encoders as discriminators rather than generators. arXiv e-prints, arXiv: 2003.10555.

Deng, Y., Zheng, X., Zhang, T., Liu, H., Lou, G., Kim, M., Chen, T.Y., 2023. A declarative metamorphic testing framework for autonomous driving. IEEE Trans. Softw. Eng. 49 (4), 1964–1982.

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2018). Bert: pre-training of deep bidirectional transformers for language understanding. arXiv preprint.

Dong, L., Lapata, M., 2018. Coarse-to-Fine decoding for neural semantic parsing. In: Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics, 1. Long Papers.

Dong, Y., Jiang, X., Jin, Z., & Li, G. (2023). Self-collaboration code generation via chatgpt. arXiv preprint.

Duque-Torres, A., Pfahl, D., Klammer, C., Fischer, S., 2023. Bug or not bug? Analysing the reasons behind metamorphic relation violations. In: 2023 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER).

Ellis, J.D., Iqbal, R., Yoshimatsu, K., 2021. Verification of the neural network training process for spectrum-based chemical substructure prediction using metamorphic testing. J. Comput. Sci. 55, 101456.

Evtikhiev, M., Bogomolov, E., Sokolov, Y., Bryksin, T., 2023. Out of the bleu: how should we assess quality of the code generation models? J. Syst. Softw. 203, 111741.

GoogleCloud, 2023. Evaluating Models. Google. Retrieved 24/12/2023 from. https://cloud.google.com/translate/automl/docs/evaluate.

Gulwani, S., Polozov, O., & Singh, R. (2017). Program synthesis. Foundations and Trends® in Programming Languages, 4(1–2), 1–119.

HuggingFace, 2023a. AhmedSSoliman/DistilBERT-Marian-Model-on-CoNaLa. HuggingFace. Retrieved 24/12/2023 from. https://huggingface.co/AhmedSSoliman/DistilBERT-Marian-Model-on-CoNaLa.

HuggingFace, 2023b. AhmedSSoliman/DistilRoBERTa-Marian-Model-on-CoNaLa. HuggingFace. Retrieved 24/12/2023 from. https://huggingface.co/AhmedSSoliman/DistilRoBERTa-Marian-Model-on-CoNaLa.

HuggingFace, 2023c. AhmedSSoliman/ELECTRA-Marian-Model-on-CoNaLa. HuggingFace. Retrieved 24/12/2023 from. https://huggingface.co/AhmedSSoliman/ELECTRA-Marian-Model-on-CoNaLa.

HuggingFace, 2023d. AhmedSSoliman/LUKE-Marian-Model-on-CoNaLa. HuggingFace. Retrieved 24/12/2023 from. https://huggingface.co/AhmedSSoliman/LUKE-Marian-Model-on-CoNaLa.

HuggingFace, HuggingFace, 2023e. AhmedSSoliman/MarianCG-CoNaLa-Large. Retrieved 24/12/2023 from. https://huggingface.co/AhmedSSoliman/MarianCG-CoNaLa-Large.

HuggingFace, 2023f. Cahya/Bert2gpt-Indonesian-Summarization. HuggingFace. Retrieved 24/12/2023 from. https://huggingface.co/cahya/bert2gpt-indonesian-summarization.
HuggingFace, 2023g. Distilbert/Distilbert-Base-Uncased. HuggingFace. Retrieved 24/12/2023 from. https://huggingface.co/distilbert/distilbert-base-uncased.
HuggingFace, 2023h. Distilbert/Distilroberta-Base. HuggingFace. Retrieved 24/12/2023 from. https://huggingface.co/distilbert/distilroberta-base.
HuggingFace, 2023i. Google/Electra-Base-Discriminator. HuggingFace. Retrieved 24/12/2023 from. https://huggingface.co/google/electra-base-discriminator.
HuggingFace, 2023j. Google/Pegasus-X-Base. HuggingFace. Retrieved 24/12/2023 from. https://huggingface.co/google/pegasus-x-base.
HuggingFace, 2023k. Mrm8488/Bert-Small2bert-Small-Finetuned-Cnn_Daily_Mail-Summarization. HuggingFace. Retrieved 24/12/2023 from. https://huggingface.co/mrm8488/bert-small2bert-small-finetuned-cnn_daily_mail-summarization.
HuggingFace, 2023l. Studio-Ousia/Luke-Base. HuggingFace. Retrieved 24/12/2023 from. https://huggingface.co/studio-ousia/luke-base.
Hussain, Y., Huang, Z., Zhou, Y., Wang, S., 2023. Boosting source code suggestion with self-supervised transformer gated highway. J. Syst. Softw. 196, 111553.
Jiang, M., Chen, T.Y., Wang, S., 2022. On the effectiveness of testing sentiment analysis systems with metamorphic testing. Inf. Softw. Technol. 150, 106966.
Li, Z., Wang, C., Liu, Z., Wang, H., Chen, D., Wang, S., Gao, C., 2023. Cctest: testing and repairing code completion systems. In: 2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE).
Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., & Stoyanov, V. (2019). RoBERTa: a robustly optimized BERT pretraining approach. Retrieved July 01, 2019, from https://ui.adsabs.harvard.edu/abs/2019arXiv190711692L.
Luu, Q.-H., Lau, M.F., Ng, S.P., Chen, T.Y., 2021. Testing multiple linear regression systems with metamorphic testing. J. Syst. Softw. 182, 111062.
Luu, Q.-H., Liu, H., Chen, T.Y., Vu, H.L., 2024. A sequential metamorphic testing framework for understanding autonomous vehicle's decisions. IEEE Trans. Intell. Veh.
Miao, L., Towey, D., Ma, Y., Chen, T.Y., Zhou, Z.Q., 2023. Exploring metamorphic testing for fake-news detection software: a case study. In: 2023 IEEE 47th Annual Computers, Software, and Applications Conference (COMPSAC).
Microsoft, 2024a. Copilot. Microsoft.com. Retrieved 19/08/2024 from. https://www.microsoft.com/en-us/edge/features/copilot?form=MA13FJ.
Microsoft, 2024b. Overview of Microsoft Copilot. Microsoft.com. Retrieved 19/08/2024 from. https://learn.microsoft.com/en-us/copilot/overview?ns-enrollment-type=Collection&ns-enrollment-id=7rmiy0yrnw8d6.
Microsoft, 2024c. Tackle Any Challenge With Copilot. Microsoft.com. Retrieved 19/08/2024 from. https://www.microsoft.com/en-us/microsoft-copilot/personal-ai-assistant.
Mouselinos, S., Malinowski, M., Michalewski, H., 2023. A simple, yet effective approach to finding biases in code generation. In: Findings of the Association for Computational Linguistics: ACL 2023.
Phang, J., Zhao, Y., Liu, P.J., 2023. Investigating efficiently extending transformers for long input summarization. In: Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing.
POE.COM, 2023. About. Retrieved 2023-05-18 from. https://poe.com/about.
POE.COM, 2024. Code-Llama-34b. Retrieved 2024-01-15 from. https://poe.com/Code-Llama-34b.
Poesia, G., Polozov, O., Le, V., Tiwari, A., Soares, G., Meek, C., & Gulwani, S. (2022). Synchromesh: reliable code generation from pre-trained language models. arXiv preprint.
Post, M. (2018). A Call for Clarity in Reporting BLEU Scores. Proceedings of the Third Conference on Machine Translation: research Papers.
Post, S.C.M., 2023. ChatGPT Usage Could Open Companies to Leaks of Corporate Secrets and lawsuits, Cyber Firm Warns. South China Morning Post. Retrieved 24/12/2023 from. https://www.scmp.com/tech/big-tech/article/3217571/chatgpt-usage-could-open-companies-leaks-corporate-secrets-and-lawsuits-cyber-firm-warns.
Pugh, S., Raunak, M.S., Kuhn, D.R., Kacker, R., 2019. Systematic testing of post-quantum cryptographic implementations using metamorphic testing. In: 2019 IEEE/ACM 4th International Workshop on Metamorphic Testing (MET).
Radford, A., Narasimhan, K., Salimans, T., & Sutskever, I. (2018). Improving language understanding by generative pre-training.
Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I., 2019. Language models are unsupervised multitask learners. OpenAI blog 1 (8), 9.
Riccio, V., Jahangirova, G., Stocco, A., Humbatova, N., Weiss, M., Tonella, P., 2020. Testing machine learning based systems: a systematic mapping. Empir. Softw. Eng. 25 (6), 5193–5254.
Roziere, B., Gehring, J., Gloeckle, F., Sootla, S., Gat, I., Tan, X.E., Adi, Y., Liu, J., Remez, T., & Rapin, J. (2023). Code llama: open foundation models for code. arXiv preprint.
Saha, P., Kanewala, U., 2019. Fault detection effectiveness of metamorphic relations developed for testing supervised classifiers. In: 2019 IEEE International Conference On Artificial Intelligence Testing (AITest).
Sanh, V., Debut, L., Chaumond, J., & Wolf, T. (2019). DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. arXiv preprint.
Segura, S., Durán, A., Troya, J., Ruiz-Cortés, A., 2019. Metamorphic relation patterns for query-based systems. In: 2019 IEEE/ACM 4th International Workshop on Metamorphic Testing (MET).
Shah, C. (2024). From Prompt Engineering to Prompt Science With Human in the Loop. arXiv preprint.
Soliman, A.S., Hadhoud, M.M., Shaheen, S.I., 2022. MarianCG: a code generation transformer model inspired by machine translation. J. Eng. Appl. Sci. 69 (1), 104.
Stacy, B., Hauzel, J., Lindvall, M., Porter, A., Pop, M., 2022. Metamorphic testing in bioinformatics software: a case study on metagenomic assembly. In: 2022 IEEE/ACM 7th International Workshop on Metamorphic Testing (MET).
Steenhoek, B., Rahman, M.M., Jiles, R., Le, W., 2023. An empirical study of deep learning models for vulnerability detection. In: 2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE).
Sun, C.-a., Liu, B., Fu, A., Liu, Y., Liu, H., 2022. Path-directed source test case generation and prioritization in metamorphic testing. J. Syst. Softw. 183, 111091. https://doi.org/10.1016/j.jss.2021.111091.
Tabassum, S., Minku, L.L., Feng, D., Cabral, G.G., Song, L., 2020. An investigation of cross-project learning in online just-in-time software defect prediction. In: Proceedings of the ACM/IEEE 42nd international conference on software engineering.
Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M.-A., Lacroix, T., Rozière, B., Goyal, N., Hambro, E., & Azhar, F. (2023a). Llama: open and efficient foundation language models. arXiv preprint.
Touvron, H., Martin, L., Stone, K., Albert, P., Almahairi, A., Babaei, Y., Bashlykov, N., Batra, S., Bhargava, P., & Bhosale, S. (2023b). Llama 2: open foundation and fine-tuned chat models. arXiv preprint.
Troshin, S., Chirkova, N., 2022. Probing pretrained models of source codes. In: Proceedings of the Fifth BlackboxNLP Workshop on Analyzing and Interpreting Neural Networks for NLP.
Wang, J., Peng, Y., Le, X., Chen, C., Guan, X., 2023. Code generation from natural language using two-way pre-training. In: 2023 15th International Conference on Advanced Computational Intelligence (ICACI).
Wang, S., Li, Z., Qian, H., Yang, C., Wang, Z., Shang, M., Kumar, V., Tan, S., Ray, B., & Bhatia, P. (2022). Recode: robustness evaluation of code generation models. arXiv preprint.
Xiao, D., LIU, Z., Yuan, Y., Pang, Q., Wang, S., 2022. Metamorphic testing of deep learning compilers. Proc. ACM Meas. Anal. Comput. Syst. 6 (1), 15. https://doi.org/10.1145/3508035.
Xie, X., Ho, J.W.K., Murphy, C., Kaiser, G., Xu, B., Chen, T.Y., 2011. Testing and validating machine learning classifiers by metamorphic testing. J. Syst. Softw. 84 (4), 544–558. https://doi.org/10.1016/j.jss.2010.11.920.
Xie, X., Zhang, Z., Chen, T.Y., Liu, Y., Poon, P.-L., Xu, B., 2020. METTLE: a metamorphic testing approach to assessing and validating unsupervised machine learning systems. IEEe Trans. Reliab. 69 (4), 1293–1322.
Xu, F.F., Alon, U., Neubig, G., Hellendoorn, V.J., 2022. A systematic evaluation of large language models of code. In: Proceedings of the 6th ACM SIGPLAN International Symposium on Machine Programming.
Xu, L., Towey, D., French, A.P., Benford, S., Zhou, Z.Q., Chen, T.Y., 2021. Using metamorphic relations to verify and enhance artcode classification. J. Syst. Softw. 182, 111060.
Yamada, I., Asai, A., Shindo, H., Takeda, H., & Matsumoto, Y. (2020). LUKE: deep contextualized entity representations with entity-aware self-attention. arXiv preprint.
Yamamoto, H., Wang, D., Rajbahadur, G.K., Kondo, M., Kamei, Y., Ubayashi, N., 2023. Towards privacy preserving cross project defect prediction with federated learning. In: 2023 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER).
Yan, M., Chen, J., Zhang, J.M., Cao, X., Yang, C., & Harman, M. (2023). Coco: testing code generation systems via concretized instructions. arXiv preprint.
Yin, P., Neubig, G., 2017. A syntactic neural model for general-purpose code generation. In: Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, 1. Long Papers.
Ying, Z., Bellotti, A., Towey, D., Chen, T.Y., Zhou, Z.Q., 2022. Using metamorphic relation violation regions to support a simulation framework for the process of metamorphic testing. In: 2022 IEEE 46th Annual Computers, Software, and Applications Conference (COMPSAC).
Ying, Z., Towey, D., Bellotti, A., Zhou, Z.Q., Chen, T.Y., 2021. Preparing SQA professionals: metamorphic relation patterns, exploration, and testing for big data. In: Proceedings of the International Conference on Open and Innovation Education (ICOIE 2021), pp. 22–30.
Zeng, Z., Tan, H., Zhang, H., Li, J., Zhang, Y., Zhang, L., 2022. An extensive study on pre-trained models for program understanding and generation. In: Proceedings of the 31st ACM SIGSOFT international symposium on software testing and analysis.
Zhang, M., Keung, J.W., Chen, T.Y., Xiao, Y., 2021. Validating class integration test order generation systems with metamorphic testing. Inf. Softw. Technol. 132, 106507.
Zhang, Z., Chen, C., Liu, B., Liao, C., Gong, Z., Yu, H., Li, J., & Wang, R. (2023). A survey on language models for code. arXiv preprint.
Zhou, X., Zhang, Y., Cui, L., Huang, D., 2020a. Evaluating commonsense in pre-trained language models. In: Proceedings of the AAAI conference on artificial intelligence.
Zhou, Z.Q., Sun, L., Chen, T.Y., Towey, D., 2020b. Metamorphic relations for enhancing system understanding and use. IEEE Trans. Softw. Eng. 46 (10), 1120–1154.