# SelfBehave, Generating a Synthetic Behaviour-Driven Development Dataset Using SELF-INSTRUCT

Manon Galloy
*NADI, University of Namur*
Namur, Belgium

Martin Balfroid
*NADI, University of Namur*
Namur, Belgium
martin.balfroid@unamur.be

Benoît Vanderose
*NADI, University of Namur*
Namur, Belgium
benoit.vanderose@unamur.be

Xavier Devroey
*NADI, University of Namur*
Namur, Belgium
xavier.devroey@unamur.be

***Abstract*—While state-of-the-art large language models (LLMs) show great potential for automating various Behavioral-Driven Development (BDD) related tasks, such as test generation, smaller models depend on high-quality data, which are challenging to find in sufficient quantity. To address this challenge, we adapt the SELF-INSTRUCT method to generate a large synthetic dataset from a small set of human-written high-quality scenarios. We evaluate the impact of the initial *seeded* scenarios' quality on the generated scenarios by generating two synthetic datasets: one from 175 high-quality seeds and one from 175 seeds that did not meet all quality criteria. We performed a qualitative analysis using state-of-the-art quality criteria and found that the quality of seeds does not significantly influence the generation of complete and essential scenarios. However, it impacts the scenarios' capability to focus on a single action and outcome and their compliance with Gherkin syntactic rules. During our evaluation, we also found that while raters agreed on whether a scenario was of high quality or not, they often disagreed on individual criteria, indicating a need for quality criteria easier to apply in practice.**

***Index Terms*—Large Language Models (LLMs), behaviour-driven development (BDD), self-instruct**

## I. INTRODUCTION

Abstraction via storytelling is powerful for sharing ideas and thoughts. In software development, this is reflected in *user stories*, which state features from the user's point of view to ensure clear communication among team members. These stories are broken down into *scenarios*, which can serve various purposes, like requirements engineering, feature development prioritization, but also, acceptance testing with concrete examples of how an application should behave [1], [2]. Among the different formalisms, Behavior-Driven Development (BDD) uses the structured natural language *Gherkin* [3] to define scenarios, which include the initial state (*Given*), actions (*When*), and expected outcomes (*Then*). Listing 1 illustrates a scenario written in Gherkin, describing an account depositing operation. Such scenarios serve as input for implementing tests [4], [5], e.g., with Python's *behave* or Java's *Cucumber*. Despite the gain of popularity over the last decades, many tasks relying on BDDs remain manual.

Large Language Models (LLMs) can help perform various software engineering tasks [6]. For instance, Karpurapu et al. [7] evaluated the ability of LLMs to generate test code from BDD scenarios and found promising results for state-of-the-art LLMs. However, smaller models are not as good. Still, they are attractive as they can easily be deployed in-house for privacy concerns without demanding too much resources. Fine-tuning them on a BDD scenario dataset generated by a state-of-the-art LLM would put them on a par with the bigger models [8]. However, this requires a dataset with high-quality scenarios - i.e. that are self-contained (*complete*), that do not contain noise (*essential*), that focus on a single action and outcome (*singular*), and adhere to Gherkin rules (*integrous*)- which is scarce [9].

This paper focuses on the BDD dataset scarcity by adapting the SELF-INSTRUCT [8] method to generate a synthetic dataset from a smaller set of human-written scenarios. We evaluated our approach to answer the following research questions:

**RQ1** *To what extent does the quality of the seed scenarios influence the quality of the scenarios generated?*

**RQ2** *To what extent is the checklist for evaluating the Essential, Singular, Complete, and Integrous criteria applicable to automatically generated BDD scenarios?*

For that, we generated two synthetic datasets containing 1,000 scenarios each: one from 175 high-quality scenarios and one from 175 scenarios that did not necessarily meet all criteria—each. After a manual analysis of the quality of the generated scenarios, performed independently by the two first authors, we found that the quality of the seed scenarios has no significant influence on generating complete and essential scenarios. However, quality does matter when generating scenarios that focus on a single action and outcome and respect the Gherkin syntactic rules. We also found that while both raters agreed on whether a scenario is of high

This research was supported by the ARIAC project (No. 2010235) funded by the Service Public de Wallonie (SPW Recherche), and the SQUAL.AI project (No. 2025/658876) supported by Wallonie-Bruxelles International (WBI).

```
Scenario: Deposit money
  Given I have $100
  When I deposit $50
  Then I should have $150
```

Listing 1. Gherkin scenario for an account management system

TABLE I
EXCERPT FROM THE SCENARIO-LEVEL CRITERIA QUESTIONS IN THE CHECKLIST BY OLIVEIRA ET AL. [14]

| ID | Question | Attribute |
|---|---|---|
| 3 | Does the scenario carry all the information needed to understand it? | Complete |
| 4 | Does the scenario have steps that can be removed without affecting its understanding? | Essential |
| 6 | Can the scenario single action be identified on its title and match what the scenario is doing? | Singular |
| 7 | Can the scenario outcome or verifications be identified on its title and match what the scenario is doing? | Singular |
| 8 | Does the scenario respect the meaning of Gherkin keywords and their natural order? | Integrous |

quality, they often disagreed on individual criteria, which signals a need to refine the checklist for evaluating the criteria. Our modified implementation of SELF-INSTRUCT and our complete replication package are available on Zenodo https://zenodo.org/records/14054404 [10].

## II. BACKGROUND AND RELATED WORK

### A. Behavioral-Driven Development

BDDs [1] capture what is expected of the software in the form of *scenarios*. A feature (i.e., a *user story* in Agile) is a set of scenarios that describe its different aspects [11]. This work focuses on Gherkin [12], a language commonly used to represent scenarios (e.g., Listing 1). With the development of Natural Language Processing (NLP) and machine learning, BDD has gained popularity as it can be used as an intermediate representation in model-based testing [5], or as input for various automated software engineering tasks like test case generation [4], [7], [13]. For instance, Storer and Bob [13] automatically generate step implementation functions from BDD steps. They use NLP to parse and interpret Gherkin scenarios from GitHub, focusing on Python for step implementations. While their approach shows potential, it achieves a low success rate without human intervention, likely due to their reliance on NLP methods that do not fully capture natural language's complexity.

Among the eight criteria defined by Oliveira et al. [14], we selected those applicable at the scenario level, listed in Table I with their corresponding questions. The others apply either at a more granular level (step) or at a higher level (feature), which falls outside the scope of our study.

### B. Large Language Models

Recent NLP models, like Large Language Models (LLMs), dramatically improve performance for NLP-based tasks. For BDDs, state-of-the-art LLMs show great promise in generating nearly flawless BDD acceptance tests [7]. However, smaller models like `LLama2-13B` [15], while advantageous due to their open weights for privacy and lower resource requirements, still leave room for improvement [7]. García et al. [16] evaluated the extent to which ChatGPT is a helpful assistant in implementing end-to-end Android app test scenarios specified in Gherkin. They observed that even if it requires a manual step to fix generated tests manually, LLMs reduce total unit test writing time without compromising reliability.

### C. Generating synthetic dataset

SELF-INSTRUCT [8] is an iterative technique designed to generate a large and diverse dataset from a small set of manually written *task prompt*[1] such as "Generate a Python implementation of the quick sort algorithm". It does so by iteratively generating new *task prompts* and their corresponding input-output pairs using a large language model (LLM) to build upon both the initial human-written examples and previously generated prompts. To our knowledge, this method has not yet been adapted for generating BDD synthetic datasets. In a nutshell, the process includes a three-step loop (See Figure 1); the loop continues until the dataset reaches the desired size.

(1) *Task prompt generation:* Generate new *task prompts* in a few-shot manner using a pool of human-written examples (and, in later iterations, synthetic examples) with the following prompt:

```
Come up with a series of tasks:
Task 1: <human-written task 1>
Task 2: <human-written task 2>
...
Task 7: <LLM-generated task 1>
Task 8: <LLM-generated task 2>
Task 9:
```

Each prompt includes eight examples: six human-written and two LLM-generated, ensuring a balance between quality and diversity. In the first iteration, all tasks are human-written. It uses a *Simple colon* [17] prompt format, implicitly prompting the model to generate the next tasks sequentially after the colon. The model continues until reaching its token limit or producing up to seven new tasks (i.e., generating `Task 9` to `Task 15`).

(2) *Input-Output pairs generation:* The generation of input-output pairs follows two different strategies depending on whether the output is a class label. An *output-first* approach is used for classification tasks to prevent label bias, as models tend to generate inputs skewed toward a particular class. Conversely, non-classification tasks adopt an *input-first* approach, where outputs are conditioned on inputs.

A few-shot prompt template is used with 12 classifications and 19 non-classification *task prompts* as a demonstration. Below is a shortened version:

```
Can the following task be regarded as a classification task
     with finite output labels?

Task: Given my personality and the job, tell me if I would
    be suitable.
Is it classification? Yes
```

[1]The standard term is "instruction", but we use the term *task prompt* instead to avoid confusion with code instructions.

```
[...]

Task: Given a set of numbers, find all possible subsets
     that sum to a given number.
Is it classification? No

Task: <target task>
```

For classification tasks, the following template prompts the model to generate input-output pairs, starting with the outputs to guarantee that all the output distribution is covered. Below is a shortened version:

```
Given the classification task definition and the class
     labels, generate an input that corresponds to each of
     the class labels. If the task doesn't require input,
     just generate the correct class label.

Task: Classify the sentiment of the sentence into positive,
      negative, or mixed.
    Class label: Mixed
    Sentence: I enjoy [...] but their service is too slow.
    Class label: Positive
    Sentence: I had a great day today. [...]
    Class label: Negative
    Sentence: I was really disappointed [...]

[...]

Task: Detect if the Reddit thread contains hate speech.
    Class label: Hate Speech
    Thread: [...] stupid [...]
    Class label: Not Hate Speech
    Thread: The best way to cook a steak on the grill.

Task: <target task>
```

For non-classification tasks, we generate the input, if any, before the output. Below is a shortened version:

```
Come up with examples for the following tasks. Try to
     generate multiple examples when possible. If the task
     doesn't require additional input, you can generate the
     output directly.

Task: Which exercises are best for reducing belly fat at
     home?
Output:
- Lying Leg Raises
- Leg In And Out
- Plank
- Side Plank
- Sit-ups

[...]

Task: Sort the given list ascendingly.
Example 1
List: [10, 92, 2, 5, -4, 92, 5, 101]
Output: [-4, 2, 5, 5, 10, 92, 92, 101]

Example 2
Input 2 - List: [9.99, 10, -5, -1000, 5e6, 999]
Output: [-1000, -5, 9.99, 10, 999, 5e6]

Task: <target task>
```

(3) *Filtering and Postprocessing:* A multi-step filtering process is applied before integrating new *task prompts* into the pool to ensure high quality and diversity. This process consists of the following steps: (i) a new *task prompt* is added if it is at least 30% unique (ROUGE-L [18] score below 0.7); (ii) *Task prompts* containing specific keywords (e.g., *image, picture*) are excluded; (iii) exact duplicates of input-output pairs are removed; (iv) pairs where the same input but conflicting outputs are removed; (v) invalid generations are identified and removed based on heuristics, including *task prompts* that are excessively short or long and Outputs that are mere repetitions of the input.

This filtering process ensures that only diverse, high-quality, and logically consistent *task prompts* and input-output pairs are kept in the dataset.

## III. Approach

### A. Synthetic BDD Dataset Generation

We adapt the Self-Instruct method to generate a synthetic dataset of BDD acceptance test scenarios in Gherkin. We use the following template where the *task prompts* instruct the model to generate Python *steps* for a given *scenario* using the *behave* framework:

```
Task: Generate a Python implementation of the step
     functions for the following Gherkin scenario using the
     behave BDD testing framework: <SCENARIO>
Output: <STEPS>
```

As code generation is not a classification task, we use an *input-first* approach with the BDD scenario directly embedded in the task prompt, like so:

```
Come up with examples for the following tasks. Try to
     generate multiple examples when possible. If the task
     doesn't require additional input, you can generate the
     output directly.

Task 1: Generate a Python implementation of the step
     functions for the following Gherkin scenario using the
     behave BDD testing framework:
    Scenario: Add Product to Cart
        Given I am on the Blouse product page
        When I click add to cart
        Then the product should be added to the cart
Output 1:
    @given(u'I am on the Blouse product page')
    def step_impl(context):
        context.page_object = ProductPageActions(context.
             driver)
        assert context.page_object.
             is_product_page_displayed(), "Product page is
             not displayed"
        [...]

Task 2: Generate a Python implementation [...]
Output 2: [...]

[...]
```

The filtering mechanism in step (4) removes ill-formed and invalid instances before the next iteration.

### B. Implementation and Underlying LLM

We rely on the Self-Instruct implementation of Wang et al. [8]. We used the `Mixtral-8x7b-Instruct-v0.1` LLM for several reasons: it has open weights, which enforces open science and enhances the reproducibility of our research; it is also qualitatively similar to `GPT-3.5` [19], which was the best LLM at the time of performing this study; Finally, most companies, some also developing open weights LLMs like Meta, explicitly prohibit using their model outputs to train models from other companies. We did not find such a restriction in Mistral's Terms of Service, which is promising for our kind of research.

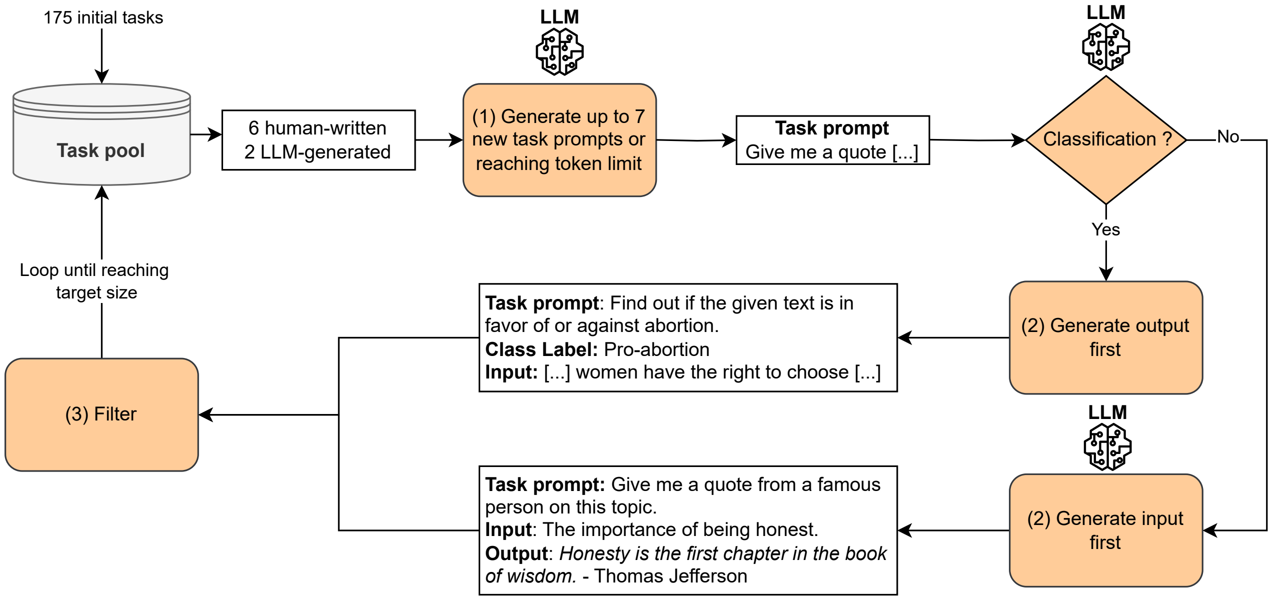

175 initial tasks
Task pool
6 human-written
2 LLM-generated
LLM
(1) Generate up to 7 new task prompts or reaching token limit
Task prompt
Give me a quote [...]
LLM
Classification ?
No
Yes
(2) Generate output first
Task prompt: Find out if the given text is in favor of or against abortion.
Class Label: Pro-abortion
Input: [...] women have the right to choose [...]
Loop until reaching target size
(3) Filter
LLM
(2) Generate input first
Task prompt: Give me a quote from a famous person on this topic.
Input: The importance of being honest.
Output: Honesty is the first chapter in the book of wisdom. - Thomas Jefferson

Fig. 1. SELF-INSTRUCT process [8]

## IV. EVALUATION SETUP

Our primary focus is on evaluating the quality of the generated scenarios using four of the scenario-level quality criteria from Oliveira et al. [14] (described in Table I). For that, we need to assess how the initial scenarios' quality could influence the generated scenarios' quality (**RQ1**). Collaterally, to the best of our knowledge, no other study has assessed the applicability of Oliveira's criteria to automatically generated BDD scenarios (**RQ2**). To focus clearly on the scenario scope, we have left analysis of the generated code, and analysis at the step and feature level for future work.

### A. Initial BDD Scenarios Seed Collection

We selected 39 repositories that use the *behave* framework, starting with those used by Storer et al. [13], covering diverse domains (Cucumber, Blender, ...). From each repository, the first author selected at least one scenario that either met all quality criteria of Table I or failed at least one of them, with an additional validation made by the second author. In Table II, we provide the final list of GitHub repositories curated for the seed datasets.

From the 39 identified repositories, we manually curated two seed datasets of 175 BDD scenarios each: (i) *High-Quality Seed Dataset* (HS), that consists exclusively of scenarios that meet all criteria in Table I. And (ii) *Mixed-Quality Seed Dataset* (MS), containing a mix of high-quality and lower-quality scenarios. It was created by randomly replacing 70 high-quality scenarios with lower-quality ones (i.e., not meeting at least one of the quality criteria), resulting in a 60-40 ratio. This balance provides a realistic mix of quality while ensuring that the lower-quality scenarios have sufficient influence without dominating. In short, the idea is to see whether a few low-quality scenarios can affect the overall quality.

For comparability, we chose the same size as prior work on SELF-INSTRUCT. This number is a good tradeoff between an adequate number of scenarios and the effort required for review. Future work should explore whether varying this number impacts the diversity and stability of the generated results.

### B. Synthetic Datasets Generation

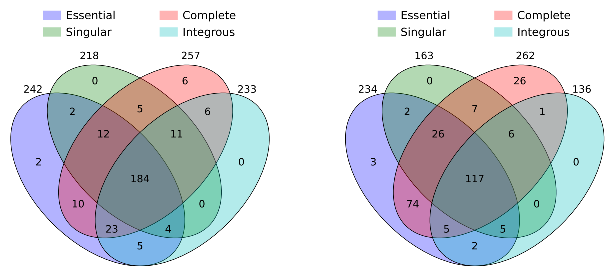
The SELF-INSTRUCT method was adapted to generate two new synthetic datasets: *High-Quality Seeded Generated Dataset* (HG) (generated from HS) and *Mixed-Quality Seeded Generated Dataset* (MG) (generated from MS). For each of the two seed datasets, twenty new scenarios are generated per iteration until the target of 1,000 new scenarios is reached. The filtering mechanism removed ill-formed and invalid instances.

### C. Data Analysis

Once the synthetic datasets were generated, the two first authors evaluated the quality of their BDD scenarios, based on Table I (**RQ1**). A scenario is considered of high quality if all four criteria are met. In case of disagreement, the two last authors evaluated the results, and a consensus was reached. We randomly selected a sample of 286 scenarios for each synthetic dataset for manual inspection. This sample size ($n = 286$) was calculated using Yamane's formula [20], with a total population of $N = 1,000$ and a margin of error of $e = 5\%$. We chose this margin of error as it is a standard trade-off between practicality and accuracy. We generate more scenarios than we evaluate to ensure greater diversity in the dataset and a more representative sample. This approach also allows for a statistically reliable evaluation based on random sampling rather than evaluating a

TABLE II
GITHUB REPOSITORIES CURATED FOR THE SEED DATASETS.

| Repository | Repository | Repository |
|---|---|---|
| mauriciochaves/python_workspace | pytroll/satpy | mezuro/kalibro_client_py |
| walterdolce/python-package-license-generator | nzabus/prj_bdd | kazuho/quic-perf-metrics |
| Grigorevskiy/RaceAutomationPython | jeanmichelem/tdd | PrateekVachher/WanderLust-TheTravelApp |
| bbvch/brightness-zoo | luke10x/malunas | kate777k/suit-test |
| florinn/veryfay-python | migibert/pynetlib | mc706/prog-strat-game |
| chrdavid/superlists | Winnetou/ManuTironis | timbortnik/behave_web2 |
| ivanori1/smoke_test_webstation | anaplian/tinylog | softwarefactory-project/rdopkg |
| Jyotiranjan767/59_effetctive | pcn/trtc | ilanasufrin/nyu-devops-homework-1 |
| peter-evolent/python-testing-examples | aloetesting/aloe | dev-lusaja/cucumber-python |
| vishalm/behave_parallel_demo | rbob96/Banking-BDD | behave-restful/behave-restful |
| spyountech/behave-webdriver | mmguzman/Behave | behave/behave.example |
| AndreasAugustin/Gherkin-Demos-python | radish-bdd/radish | machzqcq/Python_Page_Object |
| nikileshsa/hyperledger_pluggable_datastore | behave/behave | pcarney8/python-amq-multi-deploy-example |



(a) HG Synthetic Dataset: Scenarios generated from high quality scenarios

(b) MG Synthetic Dataset: Scenarios generated from mixed quality scenarios

Fig. 2. Classification of the synthetic scenarios according to the quality criteria

fixed set of generated scenarios. It also allows future work to be conducted without regenerating a new dataset.

To assess the level of agreement between the two first authors (**RQ2**), we employed Cohen's Kappa coefficient ($\kappa$) [21]. As $\kappa$ is an estimate derived from a sample, we calculate the confidence interval to understand its reliability. We opted for a confidence level of 95%, commonly used in research. Additionally, we interpreted the $\kappa$ values according to the widely accepted guidelines established by Landis and Koch [22]: $\kappa < 0$ as *poor*; $O \leq \kappa < 0.21$ as *slight*; $0.21 \leq \kappa < 0.41$ as *fair*; $0.41 \leq \kappa < 0.61$ as *moderate*; $0.61 \leq \kappa < 0.81$ as *substantial*; and $0.81 \leq \kappa$ as *almost perfect*. To determine whether a criterion is more frequently met in MG or HG, we use Fisher's exact test [23] (with $\alpha = 0.05$) with Odds Ratios (OR) (95% confidence interval, CI) for effect size.

## V. EVALUATION RESULTS

### A. Seed Quality Influence (RQ1)

In the MG synthetic dataset (Figure 2b), 117 out of 286 scenarios (41%) were of high quality. An additional 157 scenarios (55%) met at least one quality criterion. Among the criteria, *Integrous* was the least frequently met, with 136 scenarios (48%) satisfying it, followed by *Singular* with 163 scenarios (57%), *Essential* with 234 scenarios (82%), and *Complete*, which was the most frequently met criterion with 262 scenarios (92%). Thirteen scenarios (5%) did not meet any criteria.

TABLE III
COHEN'S KAPPA VALUES FOR HG

| Criterion | Lower Bound | Kappa | Upper Bound |
|---|---|---|---|
| Essential | 0.34 (*Fair*) | 0.46 (***Moderate***) | 0.59 (*Moderate*) |
| Singular | 0.35 (*Fair*) | 0.47 (***Moderate***) | 0.59 (*Moderate*) |
| Complete | 0.13 (*Slight*) | 0.37 (***Fair***) | 0.60 (*Fair*) |
| Integrous | -0.04 (*Poor*) | 0.15 (***Slight***) | 0.35 (*Slight*) |
| High-Quality | 0.58 (*Moderate*) | 0.67 (***Substantial***) | 0.76 (*Substantial*) |

In the HG synthetic dataset (Figure 2a), 184 out of 286 scenarios (64%) were of high quality. An additional 86 scenarios (30%) met at least one quality criterion. Among the criteria, *Singular* was the least frequently met, with 218 scenarios (76%) satisfying it, followed by *Integrous* with 233 scenarios (81%), *Essential* with 242 scenarios (85%), and *Complete*, which was the most frequently met criterion with 257 scenarios (90%). Sixteen scenarios (6%) did not meet any criteria. Using Fisher's exact test across the two datasets, no statistically significant associations were found for the *Essential* ($p = 0.434$) and *Complete* ($p = 0.564$) criteria. Scenarios generated from high-quality seed scenarios more frequently met the *Singular* criterion ($p < 0.001$; OR (95% CI) = [1.66, 3.53]) and the *Integrous* criterion ($p < 0.001$; OR (95% CI) = [3.27, 7.22]).

**Regarding the influence of the seed quality (RQ1),** including both the seeded and generated scenarios, we find that in the HS+HG dataset, 359 out of 461 scenarios (78%) are of high quality. For the MS+MG dataset, only 222 out of 461 scenarios (48%) are of high quality. Overall, the scenarios in HG are more likely to meet all criteria ($p < 0.001$; OR (95% CI) = [1.83, 3.71]) compared to those in MG. When combining both seeded and generated scenarios, the HS+HG dataset was even more likely to meet all criteria ($p < 0.001$; OR (95% CI) = [2.82, 5.1]) than the MS+MG dataset.

### B. Quality Criteria Applicability (RQ2)

**Regarding the applicability of the quality criteria (RQ2),** as can be seen from Tables III and IV, for both datasets, the raters agreed *substantially* if a scenario is of high quality or not. Interestingly, reviewers tend to be on the same page when a scenario is not of high quality, but usually not for the same

TABLE IV
COHEN'S KAPPA VALUES FOR MG

| Criterion | Lower Bound | Kappa | Upper Bound |
|---|---|---|---|
| Essential | 0.15 (*Slight*) | 0.27 (***Fair***) | 0.39 (*Fair*) |
| Singular | 0.42 (*Moderate*) | 0.52 (***Moderate***) | 0.63 (*Substantial*) |
| Complete | 0.21 (*Fair*) | 0.45 (***Moderate***) | 0.69 (*Substantial*) |
| Integrous | 0.43 (*Moderate*) | 0.53 (***Moderate***) | 0.63 (*Substantial*) |
| High-Quality | 0.61 (*Substantial*) | 0.69 (***Substantial***) | 0.78 (*Substantial*) |

reason as the inter-raters agreement only ranges from *slight* to *moderate* for criteria taken individually. This signals a need to refine the checklist from Oliveira et al. [14] for automatically generated BDDs, which we discuss in Section VI.

## VI. DISCUSSION

### A. Seed Quality Influence

The results indicate that the quality of the seeded scenarios has no significant influence on the *Essential* and *Complete* criteria. However, it has a significant influence on the *Integrous* and *Singular* criteria, with the impact being more pronounced for *Integrous*, as shown by the higher odds ratios for *Integrous* compared to *Singular*. Supporting this, in the MG dataset, the most frequent combination (74 scenarios - 26%) was scenarios that were *Essential* and *Complete* but neither *Singular* nor *Integrous*. In contrast, the HG dataset most frequently contained (23 scenarios - 8%) Essential, Complete, and Integrous but not Singular scenarios, while *Essential* and *Complete* but neither *Singular* nor *Integrous* scenario were observed 10 times (3%). These findings suggest that curating for seeded scenarios that focus on a single action and outcome (*Singular*) and, more significantly, that adhere to Gherkin rules (*Integrous*) substantially influences the quality of the generated scenarios. Note that we evaluated a sample of 286 out of the 1,000 generated scenarios, which results in a 5% margin of error based on the widely used Yamane's formula. This poses a threat to **conclusion validity**. Additionally, we only tested scenarios generated by a specific model and prompt, which may limit how well these results generalise (**external validity**). We have a mix of scenarios from diverse domains that might be incompatible, threatening **external validity** (generalisability). Similarly, we did not rigorously track the distribution of scenarios within each repository, which could impact the degree of representativeness of the dataset, introducing a **construct validity** threat. Thus, future work should include more rigorous tracking of scenario distribution to mitigate these problems. Note that selecting the initial scenarios manually could also have impacted the diversity and quality of the seeded scenarios, which poses a potential threat to **internal validity**

### B. Quality Criteria

The inter-rater agreement varied across quality criteria, with *Essential* showing only fair agreement in MG. This may be due to different interpretations between raters. For example, in HG43 (see Listing 2), one rater considered "seeing an edit option" redundant, while the other did not. On the one hand, *Complete* and *Integrous* had only fair and slight agreement,

```
Scenario:
   Given a user is logged in
   When they visit the profile page
   Then they should see their profile information
   And they should see an option to edit their profile
        information
   And
   When they click on the edit button
   Then they should be taken to the edit profile page
```

Listing 2. HG43

```
Scenario: a user can log in with their credentials
   Given I am a user with username "testuser" and password
        "testpassword"
   When I log in with my credentials
   Then I am logged in with my username "testuser" and
        password "testpassword"
```

Listing 3. HG214

respectively, in MG, highlighting the need for more consistent evaluation standards. On the other hand, *Singular* showed moderate agreement in both datasets, likely indicating that this criterion provides less room for interpretation. Clear guidelines are needed to reduce subjectivity, and implementing a Likert Scale rather than a binary choice might better capture nuanced differences in interpretation. This threatens both **construct** and **internal** validity, as criterion interpretation inconsistencies impact our quality assessments' reliability. We mitigated this threat by having the two last authors settle disagreements by consensus.

We observed issues in the generated scenarios that stem from a lack of common sense or awareness of security implications—issues not captured by our current checklist. For instance, LLM-generated scenarios sometimes expose personal data or compromise security in ways a human developer typically avoids. HS68 and HS67 reveal an actual email address, while HS214 (see Listing 3) and HS218 include assertions requiring exposed passwords, posing clear security risks. Additionally, some scenarios contained incorrect assertions (e.g., MG223 and HG127) or whimsical behaviours, such as MG243, where "hello world" disappears when a file is closed. Addressing these issues would require adapting evaluation criteria to focus on LLM biases.

Certain scenarios, like HG207, HG239, HG240, MG105, and MG126, meet all formal criteria but are essentially meaningless or degenerate. These scenarios highlight limitations in the criteria, as they formally comply without adding practical value. Another issue is overfitting specific seed examples, causing generated scenarios to mimic unusual phrasing from the seed scenarios. For instance, names like Edith and Francis, seen in MG137 and MG169, can be traced back to MS54 and MS58, suggesting that these scenarios were generated by closely following the seeded scenarios rather than producing genuinely varied examples.

## VII. CONCLUSION

In this study, we demonstrated the applicability of the SELF-INSTRUCT method to generate synthetic BDD scenarios.

We evaluated the influence of seed scenario quality on the generation using state-of-the-art quality criteria. Our findings indicate that, while *Essential* and *Complete* criteria were unaffected, *Integrous* and *Singular* criteria showed a significant influence, highlighting that carefully curated seed scenarios matter. We also showed that some criteria were ambiguous, affecting consistency in ratings. Additionally, the current checklist does not fully address specific issues found in LLM-generated scenarios, such as errors in basic logic or security risks. Some of these problems may also be due to the model itself.

Our future work includes refining the checklist to capture issues unique to LLMs better and be more resilient to subjectivity. One promising avenue is the so-called LLM as judge, which asks an LLM to do the evaluation. Although this might not be as good as humans, some large models could be an acceptable substitute, as they usually align with human preferences during training. We will also enhance the approach to generating the corresponding test code.

## References

[1] D. North, "Introducing BDD," *Better Software*, 2006.

[2] M. Utting and B. Legeard, *Practical Model-Based Testing - A Tools Approach*. Morgan Kaufmann, 2007.

[3] The Cucumber Open Source Project, "Cucumber," 2014.

[4] N. Li, A. Escalona, and T. Kamal, "Skyfire: Model-Based Testing with Cucumber," in *2016 IEEE International Conference on Software Testing, Verification and Validation (ICST)*, pp. 393–400, IEEE, Apr. 2016.

[5] F. C. Ferrari, V. H. S. Durelli, S. F. Andler, J. Offutt, M. Saadatmand, and N. Müllner, "On transforming model-based tests into code: A systematic literature review," *Softw. Test. Verification Reliab.*, vol. 33, no. 8, 2023.

[6] A. Fan, B. Gokkaya, M. Harman, M. Lyubarskiy, S. Sengupta, S. Yoo, and J. M. Zhang, "Large Language Models for Software Engineering: Survey and Open Problems," in *2023 IEEE/ACM International Conference on Software Engineering: Future of Software Engineering (ICSE-FoSE)*, pp. 31–53, May 2023.

[7] S. Karpurapu, S. Myneni, U. Nettur, L. S. Gajja, D. Burke, T. Stiehm, and J. Payne, "Comprehensive evaluation and insights into the use of large language models in the automation of behavior-driven development acceptance test formulation," *IEEE Access*, vol. 12, 2024.

[8] Y. Wang, Y. Kordi, S. Mishra, A. Liu, N. A. Smith, D. Khashabi, and H. Hajishirzi, "Self-instruct: Aligning language models with self-generated instructions," in *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 13484–13508, 2023.

[9] V. Cosentino, J. L. C. Izquierdo, and J. Cabot, "Findings from github: Methods, datasets and limitations," in *IEEE/ACM 13th Working Conference on Mining Software Repositories (MSR)*, pp. 137–141, May 2016.

[10] M. Galloy, "Selfbehave: Generating a behaviour-driven development dataset using the self-instruct method," Master's thesis, University of Namur, 2024.

[11] C. Solis and X. Wang, "A Study of the Characteristics of Behaviour Driven Development," in *2011 37th EUROMICRO Conference on Software Engineering and Advanced Applications*, (Oulu, Finland), pp. 383–387, IEEE, Aug. 2011.

[12] SmartBear Software, "Cucumber documentation." https://cucumber.io/docs/cucumber/. last accessed 08/11/2024.

[13] T. Storer and R. Bob, "Behave nicely! automatic generation of code for behaviour driven development test suites," in *19th International Working Conference on Source Code Analysis and Manipulation, SCAM 2019, Cleveland, OH, USA, September 30 - October 1, 2019*, pp. 228–237, IEEE, 2019.

[14] G. Oliveira, S. Marczak, and C. Moralles, "How to evaluate BDD scenarios' quality?," in *Proceedings of the XXXIII Brazilian Symposium on Software Engineering*, pp. 481–490, 2019.

[15] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale, D. Bikel, L. Blecher, C. Canton-Ferrer, M. Chen, G. Cucurull, D. Esiobu, J. Fernandes, J. Fu, W. Fu, B. Fuller, C. Gao, V. Goswami, N. Goyal, A. Hartshorn, S. Hosseini, R. Hou, H. Inan, M. Kardas, V. Kerkez, M. Khabsa, I. Kloumann, A. Korenev, P. S. Koura, M. Lachaux, T. Lavril, J. Lee, D. Liskovich, Y. Lu, Y. Mao, X. Martinet, T. Mihaylov, P. Mishra, I. Molybog, Y. Nie, A. Poulton, J. Reizenstein, R. Rungta, K. Saladi, A. Schelten, R. Silva, E. M. Smith, R. Subramanian, X. E. Tan, B. Tang, R. Taylor, A. Williams, J. X. Kuan, P. Xu, Z. Yan, I. Zarov, Y. Zhang, A. Fan, M. Kambadur, S. Narang, A. Rodriguez, R. Stojnic, S. Edunov, and T. Scialom, "Llama 2: Open foundation and fine-tuned chat models," *CoRR*, vol. abs/2307.09288, 2023.

[16] B. García, M. Leotta, F. Ricca, and J. Whitehead, "Use of chatgpt as an assistant in the end-to-end test script generation for android apps," in *Proceedings of the 15th ACM International Workshop on Automating Test Case Design, Selection and Evaluation*, pp. 5–11, 2024.

[17] L. Reynolds and K. McDonell, "Prompt programming for large language models: Beyond the few-shot paradigm," in *Extended Abstracts of the 2021 CHI Conference on Human Factors in Computing Systems*, pp. 1–7, 2021.

[18] C.-Y. Lin and F. J. Och, "Automatic evaluation of machine translation quality using longest common subsequence and skip-bigram statistics," in *Proceedings of the 42nd annual meeting of the association for computational linguistics (ACL-04)*, pp. 605–612, 2004.

[19] W.-L. Chiang, L. Zheng, Y. Sheng, A. N. Angelopoulos, T. Li, D. Li, H. Zhang, B. Zhu, M. Jordan, J. E. Gonzalez, and I. Stoica, "Chatbot arena: An open platform for evaluating llms by human preference," 2024.

[20] C. Uakarn, K. Chaokromthong, and N. Sintao, "Sample size estimation using yamane and cochran and krejcie and morgan and green formulas and cohen statistical power analysis by g*power and comparisons," *Apheit International Journal*, vol. 10, December 2021.

[21] M. L. McHugh, "Interrater reliability: the kappa statistic," *Biochemia medica*, vol. 22, no. 3, pp. 276–282, 2012.

[22] J. Landis, "The measurement of observer agreement for categorical data," *Biometrics*, 1977.

[23] R. A. Fisher, "On the Interpretation of $\chi^2$ from Contingency Tables, and the Calculation of P," 1922.