A LIGHTWEIGHT HEURISTIC FOR DETECTING UNFAIR TESTS IN SOFTWARE ENGINEERING BENCHMARKS

Anonymous authors
Paper under double-blind review

ABSTRACT

As existing software engineering benchmarks age, the danger of model contamination is driving interest in automated curation pipelines. Unfortunately, ensuring high-quality task instances without manual assessment is a significant challenge. An obvious choice is to use large language models (LLMs) in place of human annotators, but this comes with the usual drawbacks: complex scaffolding, prompt sensitivity, hyperparameter dependence, lack of reproducibility, and substantial environmental cost. We buck this trend by proposing a lightweight, deterministic algorithm for detecting overly-specific software tests, and in doing so we support the selection of high-quality benchmark tasks. We evaluate our heuristic against the human annotations used to develop SWE-bench Verified and we compare the resulting accuracy to the accuracies of two LLM-based alternatives. We find that the accuracy of our heuristic is slightly higher than the reported accuracy of all non-fine-tuned LLM configurations across both alternatives, and slightly lower than the reported accuracy of a fine-tuned model. Given the additional effort, complexity and environmental impact associated with fine-tuning, we consider this to be a positive result. We further propose a version of our heuristic that is less precise, but more sensitive, and we use it to highlight the importance of balancing precision and recall. Our work demonstrates that straightforward, analytical techniques remain a viable alternative to both manual, and LLM-based, benchmark curation methods.

1 INTRODUCTION

Software engineering benchmarks are an important tool for evaluating code generation techniques. Typically, such benchmarks provide datasets of programming tasks with model solutions. SWE-bench Verified (SWE-V) (Chowdhury et al., 2024), a manually curated subset of SWE-bench (Jimenez et al., 2024), has become something of a standard for evaluating the software engineering capabilities of large language models (LLMs). Unlike HumanEval (Chen et al., 2021) and MBPP (Austin et al., 2021), which are based on carefully constructed, standalone programming exercises, both SWE-V and SWE-bench are derived from issue reports in real, open-source software projects. Chowdhury et al. developed SWE-V to filter out benchmark tasks that are effectively unsolvable due to ambiguous framing. Without such curation, SWE-bench risks systematically underestimating the software engineering capabilities of LLMs by including tasks that no LLM, or even human, could reasonably be expected to solve given the information provided.

Chowdhury et al. highlight two specific problems that SWE-V tries to mitigate:

- **Underspecified issue descriptions**: If an issue description is ambiguous or omits important details about the problem, no LLM can reasonably be expected to generate a solution.
- **Unfair tests**: If a test condition specifies information missing from the issue description (e.g. specific error messages, function names, or timestamps), and such information cannot be inferred, no LLM can reasonably be expected to generate a solution that passes the test.

We focus here on the second of these ("unfair tests"), although it overlaps somewhat with the first; if a test includes requirements omitted from the issue description, that issue description is, by definition, underspecified with respect to the test.

The manual curation process for SWE-V was quite onerous: OpenAI recruited 93 software developers (“annotators") using a substantial testing and onboarding process; each annotator was provided with a detailed rubric and each instance was labelled by three different annotators. Such investment is impractical for most research groups, underscoring the need for automated alternatives. It is also noted that, at the time of writing, SWE-bench is approximately two years old, pre-dating the knowledge cut-offs of recent LLMs; whilst SWE-V was developed more recently, it uses the same task instances as SWE-bench. In order to mitigate the danger of contamination from these ‘static’ benchmarks, and to quickly generate large numbers of high-quality task instances for both training and evaluation, a number of automated curation pipelines have recently been introduced (Zhang et al., 2025; Badertdinov et al., 2025; Adamenko et al., 2025; Guo et al., 2025; Vergopoulos et al., 2025). Large language models have been heavily deployed in these pipelines, a trend that seems likely to continue in future. However, the scaffolding required for effective LLM use can be complex and the results are sensitive to both hyperparameter selection and the precise wording of input prompts. Additionally, LLM inference is probabilistic, meaning that LLM-generated benchmarks are unlikely to be reproducible. And finally, LLMs come with ethical and environmental implications that may not be applicable to other approaches.

As a response to some of these concerns, we propose a lightweight, fully-deterministic, heuristic for the detection of unfair tests in software engineering benchmarks. The basic premise is that any novel lexical elements appearing in the original (‘gold’) solution patch, and required by the corresponding test patch, but not mentioned in the issue description, will render an instance unsolvable because the test acts as a hidden specification that is unavailable to candidate LLMs. For example, in Astropy Issue 13477 [1], the test patch checks for the following error message: `"Can only compare SkyCoord to Frame with data"`, and this error message is raised in the gold solution; however, the issue description does not contain this text. Since they are unlikely to use precisely the same error message, we would not expect a programmer, human or machine, to be able to write a solution that passes the test. As such, this is an ‘unfair’ test in the context of a software engineering benchmark. In order to detect such instances, our heuristic essentially takes the intersection of the string, number, and Python identifier tokens in both the gold and test patches and compares them to the corresponding issue descriptions. If the intersecting tokens are missing from the issue description, that instance is considered to be unsolvable and is flagged for removal. A detailed description of our heuristic mechanism is provided in Section 2.1.

To contextualise the accuracy of our approach, we draw on two recent projects that use LLMs to evaluate test fairness and issue clarity in SWE-bench instances. In the first of these, Oliva et al. (2025) introduces SPICE, an automated labelling tool for SWE-bench-style datasets. SPICE includes two distinct pipelines for issue clarity assessment (ICA) and test fairness (known as test coverage assessment or TCA). The ICA pipeline uses an LLM-engineered prompt based on recurring patterns of rationales provided by the SWE-V annotators (extended with instructions to detect whether the issue description includes solution code, hints or outlines); the TCA pipeline uses Aider (a terminal-based programming assistant) to provide repository context to the underlying LLM and an auxiliary LLM to parse Aider’s output and retrieve the test coverage label. For both tasks, a consensus-based system is used where issues are labelled three times and the majority label is applied. Performance is evaluated by comparing a stratified sample of SPICE-generated labels to the human annotations from SWE-V with the majority-vote principle applied to both.

In the second project, Badertdinov et al. (2025) introduces SWE-Rebench, an automated pipeline for the construction of software engineering benchmarks. Similar to other continuously-updated benchmarks, such as SWE-bench Live (Zhang et al., 2025), the primary benefit appears to be the mitigation of data contamination from long-lived, static benchmarks such as SWE-bench. SWE-Rebench also includes a standardised evaluation framework, with a fixed scaffolding, to help ensure a fair comparison of models. The full pipeline has four stages, the last of which, “quality assessment”, is relevant to the work presented here. In this stage, SWE-Rebench generates a set of issue clarity and test fairness (“test patch correctness") labels similar to SPICE and SWE-V. Unlike Oliva et al., Badertdinov et al. fine tune an LLM specifically for the purposes of assessing issue clarity and test fairness. The generated labels are evaluated against human (SWE-V) annotations. We will compare our approach in detail with these alternatives.

[1] `https://github.com/astropy/astropy/pull/13477`

In the following section, we introduce the details of our approach, the data sets used for evaluation, and the methodology of the experiments conducted.

2 METHODS

Terminology An *instance* in SWE-bench comprises: the issue title and body (sometimes known as the "problem statement"), the solution patch from the pull request that originally resolved the issue (sometimes known as the "gold" patch), and a test patch from the same pull request.

2.1 HEURISTIC MECHANISM

We use the following synthetic example to illustrate the salient points of our approach. It comprises a solution patch (left) and test patch (right):

```
  def select_method(dat, method):
+   if (method == "ten"):
+      scale_ten(dat)
...
+ def scale_ten(dat):
+   return [x*10 for x in dat]
```

Example gold solution patch

```
+ def test_ten():
+   inp = [1,2,3]
+   exp = [10,20,30]
+   out = select_method(inp, "ten")
+   assert (out == exp)
+   assert (scale_ten(inp) == exp)
```

Example test patch

The issue description is simply: *Add new functionality to* `select_method` *for scaling by 10*. Note, this does not specify the function name `scale_ten`, nor the string `"ten"` used to select the method.

For each instance in SWE-bench, or similar benchmarks, our heuristic works as follows:

1. From the gold patch, extract the set S_p of all its string literals, the set N_p of all its numeric literals, and the set I_p of its relevant *declared* identifiers. Constructing I_p requires static analysis of the source files in the patch to extract those bound identifiers which are publicly available.
   For the example, this yields $S_p = \{\texttt{"ten"}\}$, $N_p = \{10\}$, $I_p = \{\texttt{scale\_ten}, \texttt{dat}\}$.
2. From the test patch, extract the set S_t of all its string literals, the set N_t of all its numeric literals, and the set I_t of its relevant *used* identifiers, i.e., not already bound in the test patch. Constructing I_t requires analysing the binding structure of the source code modified by the test patch.
   For the example, $S_t = \{\texttt{"ten"}\}, N_t = \{1, 2, 3, 10, 20, 30\}, I_t = \{\texttt{select\_method}, \texttt{scale\_ten}\}$.
3. Take the intersections of the corresponding sets from the gold patch (1) and test patch (2), i.e., $S = S_p \cap S_t$ and $N = N_p \cap N_t$ and $I = I_p \cap I_t$.
   For the example, $S = \{\texttt{"ten"}\}, N = \{10\}, I = \{\texttt{scale\_ten}\}$.
4. Remove any tokens from 3 that appear in the issue description. Any remaining tokens are considered to be 'unspecified'. That is, for issue tokens T, compute $S_u = S \setminus T$ and $N_u = N \setminus T$ and $I_u = I \setminus T$, witnessing underspecification.
   For the example, $S_u = \{\texttt{"ten"}\}, N_u = \emptyset, I_u = \{\texttt{scale\_ten}\}$.
5. Flag the instance for removal (positive result) if there are any unspecified tokens, that is, if $(S_u \neq \emptyset) \vee (N_u \neq \emptyset) \vee (I_u \neq \emptyset)$, otherwise retain the instance (negative result).
   For the example, the condition is true, doubly so due to the underspecification of both the string argument and the function name; thus the instance is flagged as being unfair.

For the *declared* identifiers in Step 1, 'relevant' means any declarations or assignments that could feasibly be imported and used by tests contained in a separate module. For Python, this includes: non-nested function names and parameter variable names (which get exported in Python), class names, instance methods and attributes, and global (module-level) variables. Since local variables can be renamed without affecting external imports, these are considered irrelevant (e.g., `inp`, `exp`, and `out` in the example). For the *used* identifiers in Step 2, 'relevant' means any uses of variables outside of local scopes (i.e., variables that could feasibly have been imported from the solution). We exclude standard idiomatic Python names from this, e.g., `self` or `cls`, and magic method names which are standard across Python objects, e.g. `__init__` as these would not be unusual as declared and used identifiers (but would not need specifying in an issue description).

Whilst this describes the primary, 'Semantic', configuration of our heuristic, we also provide a simpler, 'Tokens-Only' configuration which incorporates *all* the identifier tokens regardless of their type, scope, or visibility. This aligns with the treatment of string and numeric literals for which all relevant tokens are always considered (as long as they appear in both the gold and test patches). This obviates the need for analysing syntax trees. Whilst the approach may appear simplistic, a good deal of 'selection' work is handled by the intersection of the gold and test patches. For example, docstrings are generally omitted because they are unlikely to be duplicated in both the solution and the test. For the Tokens-Only configuration, in addition to the idiomatic Python names mentioned previously, we exclude all Python keywords, soft keywords and built-in Python identifiers.

In terms of implementation, the string and numeric literals (and Python identifiers in the Tokens-Only configuration) are extracted using the Python tokenizer (the lexer). In the Semantic configuration, Python identifiers (declared in the gold patch and used in the test patch) are extracted using a recursive walk of the abstract syntax tree (as output from the parser). In both cases, the gold and test patches are first applied to the base commit of the relevant repository and the modified files and line numbers are passed to the tokenizer and/or parser as required (such that only tokens in the patched parts of the files are considered). The final output is a single, binary label for each instance, where true means the instance is flagged for removal. Where processing errors occur that prevent an instance from being assessed (e.g. if patches cannot be successfully applied), the instance is not flagged for removal and the final label is false. Whilst our heuristic is not specific to any one programming language, our implementation is specific to Python as the language of SWE-bench tasks. An anonymised version of the implementation and associated documentation is included in the supplementary materials.

2.2 DATASETS

The following datasets have been used to evaluate our heuristic:

- The standard SWE-bench dataset on Hugging Face (SWE-bench, 2023).
- The raw, and ensembled, SWE-V annotation datasets (published as supplementary materials in Chowdhury et al. (2024)).
- The SWE-bench instance identifiers used by SPICE (Oliva et al., 2025) and SWE-Rebench (Badertdinov et al., 2025) in their respective evaluations (provided via email).

The SWE-bench dataset contains one instance for each GitHub issue in the benchmark (issue description, solution patch, test patch). For each instance, a "base commit", representing the state of the repository prior to the pull request is also provided, along with various items that are not required for the work presented here. Full details of the SWE-bench dataset structure can be found on Hugging Face (SWE-bench, 2023).

The raw SWE-V dataset contains three (human) annotator records for each SWE-bench instance. Each record contains: A unique annotator ID, the SWE-bench instance ID, and integer scores for issue clarity (known as "underspecified") and test fairness (known as "false negative"). Each score has an integer value between zero and three inclusive with higher scores resulting in instance exclusions (essentially higher scores are 'worse'). The dataset also includes binary true/false labels, which we understand to be derived from the four-point scores using the following logic: Scores of zero or one resolve to false, scores of two or three resolve to true. An overall annotator confidence score, ranging from one to five inclusive, is also provided, presumably reflecting the overall subjective confidence of each annotator in their assessment. We use these confidence scores to align our evaluation methods with Badertdinov et al. (2025) (see Section 2.3). Various other scores, labels and notes are also provided that are not required for the work presented here.

An ensembled dataset is provided by Chowdhury et al. based on the raw SWE-V dataset. For each instance in the raw dataset, the ensembled dataset takes the highest-severity[2] score for each of issue clarity and test fairness across the three corresponding annotator records. An overall binary label, which determines whether an instance should be excluded, is derived from a combination of these four-point scores. To the best of our understanding, this label is set to true when the *maximum* of

[2]Since higher scores are considered worse, "highest-severity" simply means the highest available score.

the (ensembled) issue clarity and test fairness scores is at least two, or if the relevant annotator has flagged the issue for removal using the "other major issues" flag (which is not considered here).

In order to facilitate a meaningful comparison of results, Oliva et al. and Badertdinov et al. have kindly provided us with the unique identifiers of the SWE-bench instances used in their respective evaluations. To the best of our understanding, [3] SPICE (Oliva et al., 2025) uses a stratified sample of ten instances, that have corresponding SWE-V annotations, from each repository (excepting those repositories with less than ten instances, where all instances were used). SWE-Rebench (Badertdinov et al., 2025) uses a subset of five *repositories* including all instances with corresponding SWE-V annotations and an annotator confidence of four or five (the highest confidence values); instances with missing (four-point) annotator scores for issue clarity or test fairness appear to have been excluded. For the reference accuracies, we take the highest accuracy values reported by each of Oliva et al. and Badertdinov et al.. For SWE-Rebench, we take the best reported accuracies of both their base, and fine-tuned, versions, and compare each of these to the accuracy of our heuristic.

The overall evaluation approaches used by Oliva et al. and Badertdinov et al. appear to be similar: Generated labels are compared to the human (SWE-V) annotations for a subset of SWE-bench instances. However, as discussed in Section 2.3, there are significant differences in terms of sample instances and pre-processing methods. For the SPICE tool, the highest reported accuracy across all underlying LLMs is 87.3% for issue clarity assessment and 68.5% for test coverage assessment. For SWE-Rebench, the highest reported accuracy is 80% for issue clarity and 67% for test patch correctness. Due to differences in the evaluation methods, these accuracy values cannot be directly compared between the two projects; however, it is worth noting that in both cases the highest reported accuracy for test fairness is substantially lower than the highest reported accuracy for issue clarity. Oliva et al. put this down to the former task requiring models to "process and reason over significantly more contextual information" than the latter.

2.3 Experiments

To calculate the accuracy of our heuristic, we applied it to a sample of SWE-bench instances and compared the resulting labels to the human (SWE-V) annotations for the same instances. To the greatest extent possible, we aligned our inputs and pre-processing methods to Oliva et al. (2025) and Badertdinov et al. (2025) so that our calculated accuracies could be compared with the reported accuracies of SPICE and SWE-Rebench respectively. Due to differences in the evaluation methods of the two reference projects, this resulted in two different experiments.

Experiment 1 used the sample instances provided by Oliva et al., whilst the human (SWE-V) annotations were ensembled by taking the majority[4] four-point score for each instance and assessment (e.g., the test fairness assessment), from which a binary label for each assessment was derived. This contrasts with the original method of Chowdhury et al., where the highest (highest-severity) four-point scores were ensembled and then resolved to a single, overall binary label for each instance. Experiment 2 used the instance identifiers provided by Badertdinov et al. and the raw SWE-bench annotations with an annotator confidence of 4 or 5 (and without missing issue clarity or test fairness scores). Finally, Experiment 3 used *all* benchmark instances that have corresponding SWE-V annotations, and ensembled the SWE-V annotations in the same way as Experiment 1 (taking the majority vote, or the median if none exists).

For all three experiments, we calculated the accuracy, recall, precision, F1, specificity and NPV (negative predictive values) for our generated labels in the usual way using the processed SWE-V annotations as target labels (i.e., 'ground truth'). For each experiment, both the Semantic and Tokens-Only heuristics were evaluated, resulting in a total of six experimental configurations. To the best of our knowledge, the reported accuracies of both SPICE and SWE-Rebench were calculated using the binary true/false labels. Our heuristic generates a binary label only. Finally, as our heuristic assumes no prior knowledge of the underlying class distribution of 'fair' and 'unfair' tests, we also include, for comparison, a uniform random baseline with no weighting and no ability to select the majority class. All results are presented as percentages with the same precision as the applicable reference projects (Experiment 3 uses one decimal place).

[3] Our understanding of the respective evaluation methods of Oliva et al. and Badertdinov et al. come from a combination of the applicable literature and email correspondence with the authors, for which we are grateful.

[4] or the median where no majority exists

Table 1: Experiment 1 results (Oliva et al. subset & comparison)

	Random	Heuristic		LLM Alternative	
		semantic	tokens-only	base model	fine-tuned
Accuracy	50.0%	70.9%	70.9%	68.5%	-
Precision	40.0%	71.4%	62.5%	-	-
Recall	50.0%	45.5%	68.2%	-	-
F1 Score	44.4%	55.6%	65.2%	-	-
Specificity	50.0%	87.9%	72.7%	-	-
NPV	60.0%	70.7%	77.4%	-	-

Table 2: Experiment 2 results (Badertdinov et al. subset & comparison)

	Random	Heuristic		LLM Alternative	
		semantic	tokens-only	base model	fine-tuned
Accuracy	50%	63%	63%	60%	67%
Precision	44%	66%	58%	-	-
Recall	50%	31%	56%	-	-
F1 Score	47%	42%	57%	-	-
Specificity	50%	88%	68%	-	-
NPV	56%	62%	67%	-	-

Table 3: Experiment 3 results (all instances with SWE-V annotations)

	Random	Heuristic	
		semantic	tokens-only
Accuracy	50.0%	69.0%	64.8%
Precision	37.9%	67.7%	53.1%
Recall	50.0%	35.1%	60.4%
F1 Score	43.1%	46.2%	56.5%
Specificity	50.0%	89.8%	67.5%
NPV	62.1%	69.4%	73.6%

3 RESULTS

Tables 1, 2 and 3 report the accuracy of our heuristic, along with various additional statistics, for Experiments 1, 2 and 3 respectively. For all three experiments, the results of both the Semantic and Tokens-Only configurations are presented alongside the applicable reference accuracies (SPICE for Experiment 1 and SWE-Rebench for Experiments 2). SWE-Rebench has two reference accuracies: one for its base model and one for its fine-tuned model. Oliva et al. do not report additional statistics besides accuracy for the SPICE tool; for SWE-Rebench, Badertdinov et al. report additional statistics for particular difficulty sub-groups that are not used in our evaluation.[5] The confusion matrices for all six experimental configurations are provided in Tables 4 to 9.

The headline result is that the accuracy of our heuristic is slightly higher than the highest-report accuracies of the non-fine-tuned LLMs used by SPICE and SWE-Rebench; however, it does not outperform the fine-tuned version of Qwen-2.5-72B-Instruct used by SWE-Rebench.

For Experiment 1, using the Semantic configuration, the accuracy of our heuristic is 2.4 percentage points higher than the best reported accuracy of SPICE (which uses DeepSeek Reasoner as the underlying LLM); this applies to both the Semantic and Tokens-Only configurations. For Experiment 2, the accuracy of our heuristic is 3 percentage points higher than the SWE-Rebench base model and 4 percentage points *lower* than the fine-tuned model (again for both the Semantic and Tokens-Only configurations). There are no reference values for Experiment 3; however, with respect to the human

[5]We are currently in the process of determining whether the reported aggregate statistics (aggregated across difficulty subgroups) in Badertdinov et al. can safely be compared to our results for the entire sample.

SWE-V annotations, the overall accuracy of our heuristic is 69% for the Semantic configuration, reducing to 64.8% for the Tokens-Only configuration. Relative to the random baselines, the highest accuracy of our heuristic, across all experimental configurations, is 70.9% for Experiment 1. This has a corresponding precision of 71.4%, putting both the accuracy and precision substantially above random (20.9 and 31.4 percentage points respectively).[6]

Recall is arguably the weakest area, particularly for the Semantic results. For the Semantic configuration of Experiment 1, the recall is 4.5 percentage points below random at 45.5%; and for Experiment 3, which uses the entire annotated dataset, this drops to just 35.1% (14.9 percentage points below random). The Tokens-Only configuration yields a much higher recall in both cases (18.2 and 10.4 percentage points above random for Experiments 1 and 3 respectively). Whilst the precision values drop, as one might expect given the simplified approach, these still sit comfortably above random at 62.5% and 53.1% for Experiments 1 and 3 respectively. Correspondingly, F1 scores for the Tokens-Only configuration exceed the corresponding Semantic scores for all three experiments; however, it is worth highlighting the very high specificity of the Semantic results, suggesting that very few 'fair' tests are incorrectly flagged when applying this configuration.

Table 4: Confusion matrix for Experiment 1 (Semantic)

	Predicted Positives	Predicted Negatives	Total
Actual Positives	20	24	44
Actual Negatives	8	58	66
Total	28	82	110
Curation Errors	0	5	5

Table 5: Confusion matrix for Experiment 1 (Tokens-Only)

	Predicted Positives	Predicted Negatives	Total
Actual Positives	30	14	44
Actual Negatives	18	48	66
Total	48	62	110
Curation Errors	0	1	1

Table 6: Confusion matrix for Experiment 2 (Semantic)

	Predicted Positives	Predicted Negatives	Total
Actual Positives	56	124	180
Actual Negatives	29	204	233
Total	85	328	413
Curation Errors	0	26	26

Table 7: Confusion matrix for Experiment 2 (Tokens-Only)

	Predicted Positives	Predicted Negatives	Total
Actual Positives	101	79	180
Actual Negatives	74	159	233
Total	175	238	413
Curation Errors	0	8	8

4 DISCUSSION

The purpose of our heuristic is to support automated curation pipelines by ensuring that instances with unfair tests are removed. As such, its recall, representing the ability to detect the unfair tests, is

[6]For precision, the random baseline is below 50% due to class imbalance in the ground-truth annotations.

Table 8: Confusion matrix for Experiment 3 (Semantic)

	Predicted Positives	Predicted Negatives	Total
Actual Positives	226	418	644
Actual Negatives	108	947	1055
Total	334	1365	1699
Curation Errors	0	52	52

Table 9: Confusion matrix for Experiment 3 (Tokens-Only)

	Predicted Positives	Predicted Negatives	Total
Actual Positives	389	255	644
Actual Negatives	343	712	1055
Total	732	967	1699
Curation Errors	0	12	12

arguably the main concern. On the other hand, since flagging every instance would result in a recall of 100% and no remaining benchmark instances, the precision is clearly also of importance. Based on the F1 scores alone, a good balance of precision and recall appears to favour the Tokens-Only configuration; however, the optimal choice of configuration will ultimately depend on the context in which our heuristic is deployed. If the initial benchmark dataset, prior to filtering, is very large (which could feasibly be the case for automated curation pipelines), and the final benchmark is allowed to be small (e.g., if it is intended for evaluation rather than training) then a high recall might be favoured. On the other hand, if the initial dataset is small, or a large number of training instances are required, then a high-precision configuration might be preferable. As such, we maintain that both configurations should be made available and deployed as appropriate in the relevant contexts.

Regardless of configuration, our heuristic is limited in a number of ways. First, we do not include any kind of dynamic analysis: a string that is generated by the gold solution at runtime, e.g., using Python f-strings, and which corresponds to a string literal in the test, will not be included in the intersection of the solution and test tokens. Similarly, if a test requires a specific value that is generated dynamically by the gold solution, but not mentioned in the corresponding issue description, the relevant instance would not be flagged for removal. Our *static* analysis could also be more sophisticated: Just because a function is declared in the gold patch, and a function of the *same name* is used in the test patch, that doesn't necessarily mean the latter is a usage of the former. Ideally, our heuristic would determine whether specific constructs declared in the gold patch are actually imported and used by the test. However, there is no guarantee this would substantially improve the performance of the heuristic, and its implementation would certainly be more complicated.

Our heuristic also assumes a precise definition of 'unfair test'. The rubric provided to the SWE-V annotators (by Chowdhury et al.) asks the question: "Are the tests well-scoped such that all reasonable solutions to the issue should pass the tests?". Whilst the preceding instructions include the problem of tests relying on specific function names, variable names and error messages that are not mentioned in the issue description, this does not preclude a broader interpretation of the question itself. The instruction attached to the highest-severity score category is particularly broad: "The tests are too narrow/broad or they look for something different than what the issue is about".

Finally, it is worth noting that our current implementation is designed for Python task instances only. For multi-language curation pipelines, the appropriate tokenization and static analysis tools would have to be implemented for each language. In this setting, the Tokens-Only configuration might be easier to deploy, requiring only a lexer for the relevant languages.

There is also room for improvement in the *evaluation* of our heuristic. Based on the accuracy results alone, we can assert that our heuristic is slightly better, relative to certain non-fine-tuned LLM-based tools, at categorising task instances that either contain, or do not contain, unfair tests. However, we cannot currently be sure if it is better, or worse, at *detecting* unfair tests, nor the relative degree of precision with which it can do so. The variation in results between experiments, particularly between Experiments 1 and 3 which use the same SWE-V pre-processing methods, is also of some

concern; for example, the recall goes down by more than 10 percentage points for the Semantic configuration. This provides some indication of the sensitivity of our heuristic to the contents of the sample instances. It is noted that the instances used in Experiment 3 (and Experiment 2) are not distributed equally across repositories; as such, if there are task-instance properties particular to over-represented repositories, this could potentially bias the final results.

It is also worth considering the quality of the ground-truth dataset. Whilst the curation process used by Chowdhury et al. is clearly thorough, the level of disagreement between human annotators raises some concerns. Oliva et al. calculate a Krippendorff's Alpha of just 0.41 for the test quality annotations, substantially lower than any standard minimum threshold. Whilst this may be mitigated, to some extent, by taking the majority score for each instance, it suggests that calling the SWE-V annotations 'correct' may be something of an assumption.

5 Conclusions

In order to support the development of automated curation pipelines, and the selection of high-quality task instances for software engineering benchmarks, we propose a lightweight, deterministic heuristic for the detection of unfair tests. Evaluated against the human annotations used for SWE-bench Verified, we show that our heuristic is at least as accurate as the best-performing, non-fine-tuned, configurations of two LLM-based alternatives. It is noted that SWE-bench Verified, and by extension the two alternatives (SPICE and SWE-Rebench), only address two specific weaknesses in SWE-bench, and that our heuristic reduces this to one. Many other weaknesses, all of which should, ideally, be addressed by automated curation pipelines, have been highlighted since the development of SWE-bench Verified (Wang et al., 2025; Liang et al., 2025; Chen et al., 2025; Aleithan, 2025).

Whilst we acknowledge that LLMs are likely to be more effective for certain curation tasks, such as the issue clarity assessment demonstrated by SPICE and SWE-Rebench (with a high degree of accuracy), that doesn't mean the overall use of LLMs, across the full span of a curation pipeline, cannot be substantially reduced. We suggest that our heuristic could eventually become one of many, ensuring high-quality task instances across multiple stages of future pipelines. In addition to improving the heuristic presented here, and the associated evaluation methods, identifying these additional heuristics could be a fruitful avenue for further work. Finally, it is worth bearing in mind that, for GPT-4o inference alone, Jegham et al. (2025) estimates an annual global energy consumption exceeding that of 35,000 US households, freshwater evaporation comparable to the drinking requirements of 1.2 million people, and a "Chicago-sized forest" worth of carbon emissions. Whilst our heuristic alone won't change these statistics, the general principle of using LLMs when, and only when, required is clearly something that is worth encouraging.

6 Reproducibility statement and LLM usage

An anonymised codebase is provided in the supplementary materials. This contains all the necessary code and instructions to reproduce our results (see `README.md`). With the exception of the sample ID lists, which are included in the repository itself (with the permission of the authors), the necessary input datasets are downloaded directly from their respective sources. Some pre-processed files are included in the repository, and these can be reproduced from the input datasets if needed. All the code can be executed using Python3 and Bash. Recent experiments have shown that the operation and/or evaluation of our heuristic may be sensitive to the execution environment, particularly the Python version used. The values presented here were calculated using Python 3.12.10; however, it should be noted that earlier versions of Python can yield slightly lower results.

With the exception of literature search, for which they have been highly beneficial, LLMs have not played any substantial role in the writing of this paper. The use of LLMs for general programming assistance is individual to the authors.

References

Pavel Adamenko, Mikhail Ivanov, Aidar Valeev, Rodion Levichev, Pavel Zadorozhny, Ivan Lopatin, Dmitry Babayev, Alena Fenogenova, and Valentin Malykh. SWE-MERA: A dynamic benchmark

for agenticly evaluating large language models on software engineering tasks, July 2025. URL http://arxiv.org/abs/2507.11059. arXiv:2507.11059 [cs].

Reem Aleithan. Revisiting SWE-Bench: On the Importance of Data Quality for LLM-Based Code Models. In *2025 IEEE/ACM 47th International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, pp. 235–236, April 2025. doi: 10.1109/ICSE-Companion66252.2025.00075. URL https://ieeexplore.ieee.org/abstract/document/11024333. ISSN: 2574-1934.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie J. Cai, Michael Terry, Quoc V. Le, and Charles Sutton. Program synthesis with large language models. *ArXiv*, abs/2108.07732, 2021. URL https://api.semanticscholar.org/CorpusID:237142385.

Ibragim Badertdinov, Alexander Golubev, Maksim Nekrashevich, Anton Shevtsov, Simon Karasik, Andrei Andriushchenko, Maria Trofimova, Daria Litvintseva, and Boris Yangel. SWE-rebench: An automated pipeline for task collection and decontaminated evaluation of software engineering agents. https://arxiv.org/abs/2505.20411, May 2025. arXiv:2505.20411 [cs.SE].

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harrison Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Joshua Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code. *CoRR*, abs/2107.03374, 2021. URL https://arxiv.org/abs/2107.03374.

Zhi Chen, Wei Ma, and Lingxiao Jiang. Unveiling Pitfalls: Understanding Why AI-driven Code Agents Fail at GitHub Issue Resolution, March 2025. URL http://arxiv.org/abs/2503.12374. arXiv:2503.12374 [cs].

Neil Chowdhury, James Aung, Chan Jun Shern, Oliver Jaffe, Dane Sherburn, Giulio Starace, Evan Mays, Rachel Dias, Marwan Aljubeh, Mia Glaese, Carlos E. Jimenez, John Yang, Leyton Ho, Tejal Patwardhan, Kevin Liu, and Aleksander Madry. Introducing SWE-bench verified, August 2024. URL https://openai.com/index/introducing-swe-bench-verified/.

Lianghong Guo, Yanlin Wang, Caihua Li, Pengyu Yang, Jiachi Chen, Wei Tao, Yingtian Zou, Duyu Tang, and Zibin Zheng. SWE-Factory: Your Automated Factory for Issue Resolution Training Data and Evaluation Benchmarks, June 2025. URL http://arxiv.org/abs/2506.10954. arXiv:2506.10954 [cs].

Nidhal Jegham, Marwan Abdelatti, Lassad Elmoubarki, and Abdeltawab Hendawi. How Hungry is AI? Benchmarking Energy, Water, and Carbon Footprint of LLM Inference, July 2025. URL http://arxiv.org/abs/2505.09598. arXiv:2505.09598 [cs].

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R Narasimhan. SWE-bench: Can language models resolve real-world github issues? In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=VTF8yNQM66.

Shanchao Liang, Spandan Garg, and Roshanak Zilouchian Moghaddam. The SWE-Bench Illusion: When State-of-the-Art LLMs Remember Instead of Reason, June 2025. URL http://arxiv.org/abs/2506.12286. arXiv:2506.12286 [cs].

Gustavo A. Oliva, Gopi Krishnan Rajbahadur, Aaditya Bhatia, Haoxiang Zhang, Yihao Chen, Zhilong Chen, Arthur Leung, Dayi Lin, Boyuan Chen, and Ahmed E. Hassan. SPICE: An automated SWE-Bench labeling pipeline for issue clarity, test coverage, and effort estimation. https://arxiv.org/abs/2507.09108v4, 2025. arXiv:2507.09108v4 [cs.SE].

SWE-bench. princeton-nlp/swe-bench (hugging face), October 2023. URL https://huggingface.co/datasets/princeton-nlp/SWE-bench.

Konstantinos Vergopoulos, Mark Niklas Mueller, and Martin Vechev. Automated benchmark generation for repository-level coding tasks. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=qnE2m3pIAb.

You Wang, Michael Pradel, and Zhongxin Liu. Are "Solved Issues" in SWE-bench Really Solved Correctly? An Empirical Study, March 2025. URL http://arxiv.org/abs/2503.15223. arXiv:2503.15223 [cs].

Linghao Zhang, Shilin He, Chaoyun Zhang, Yu Kang, Bowen Li, Chengxing Xie, Junhao Wang, Maoquan Wang, Yufan Huang, Shengyu Fu, Elsie Nallipogu, Qingwei Lin, Yingnong Dang, Saravan Rajmohan, and Dongmei Zhang. SWE-bench Goes Live!, June 2025. URL http://arxiv.org/abs/2505.23419. arXiv:2505.23419 [cs].