S3D Software and Societal Systems Department | Carnegie Mellon Portugal | fct Fundação para a Ciência e a Tecnologia

Are Large Language Models Memorizing Bug Benchmarks?






Daniel Ramos

Claudia Mamede

Kush Jain

Paulo Canelas

Catarina Gamboa

Claire Le Goues

Large Language Models are everywhere yet there is still a big gap between benchmark performance and reality

Large Language Models are everywhere yet there is still a big gap between benchmark performance and reality

Codegen-6B for bug repair

Prompt

```
int start = 0;
// get a good guess on the size of the result buffer so
// it doesnt have to double if it goes over a bit
int increase = 0;
// count the replacement text elements that are larger
// than their corresponding text being replaced
```

Defects4J, Bug 39 (in Lang project)

Large Language Models are everywhere yet there is still a big gap between benchmark performance and reality

Codegen-6B for bug repair

Prompt

```
int start = 0;
// get a good guess on the size of the result buffer so
// it doesnt have to double if it goes over a bit
int increase = 0;
// count the replacement text elements that are larger
// than their corresponding text being replaced
```

Generated

```
for (int i = 0; i < searchList.length; i++) {
    if (searchList[i] == null || replacementList[i] == null) {
        continue;
    }
    int greater = replacementList[i].length() - searchList[i].length();
    if (greater > 0) {
        increase += 3 * greater; // assume 3 matches
    }
}
// have upper-bound at 20% increase, then let Java take ...
```

Defects4J, Bug 39 (in Lang project)

Large Language Models are everywhere yet there is still a big gap between benchmark performance and reality

Codegen-6B for bug repair

Prompt

```
int start = 0;
// get a good guess on the size of the result buffer so
// it doesnt have to double if it goes over a bit
int increase = 0;
// count the replacement text elements that are larger
// than their corresponding text being replaced
```

Generated

```
for (int i = 0; i < searchList.length; i++) {
    if (searchList[i] == null || replacementList[i] == null) {
        continue;
    }
    int greater = replacementList[i].length() - searchList[i].length();
    if (greater > 0) {
        increase += 3 * greater; // assume 3 matches
    }
}
// have upper-bound at 20% increase, then let Java take ...
```

Original fix file

```
int start = 0;
// get a good guess on the size of the result buffer so
// it doesnt have to double if it goes over a bit
int increase = 0;
// count the replacement text elements that are larger
// than their corresponding text being replaced

for (int i = 0; i < searchList.length; i++) {
    if (searchList[i] == null || replacementList[i] == null) {
        continue;
    }
    int greater = replacementList[i].length() - searchList[i].length();
    if (greater > 0) {
        increase += 3 * greater; // assume 3 matches
    }
}
// have upper-bound at 20% increase, then let Java take ...
```

Defects4J, Bug 39 (in Lang project)

Large Language Models are everywhere yet there is still a big gap between benchmark performance and reality

Codegen-6B for bug repair

Prompt

```
int start = 0;
// get a good guess on the size of the result buffer so
// it doesnt have to double if it goes over a bit
int increase = 0;
// count the replacement text elements that are larger
// than their corresponding text being replaced
```

Generated

```
for (int i = 0; i < searchList.length; i++) {
    if (searchList[i] == null || replacementList[i] == null) {
        continue;
    }
    int greater = replacementList[i].length() - searchList[i].length();
    if (greater > 0) {
        increase += 3 * greater; // assume 3 matches
    }
}
// have upper-bound at 20% increase, then let Java take ...
```

Original fix file

```
int start = 0;
// get a good guess on the size of the result buffer so
// it doesnt have to double if it goes over a bit
int increase = 0;
// count the replacement text elements that are larger
// than their corresponding text being replaced

for (int i = 0; i < searchList.length; i++) {
    if (searchList[i] == null || replacementList[i] == null) {
        continue;
    }
    int greater = replacementList[i].length() - searchList[i].length();
    if (greater > 0) {
        increase += 3 * greater; // assume 3 matches
    }
}
// have upper-bound at 20% increase, then let Java take ...
```

Defects4J, Bug 39 (in Lang project)

Large Language Models are everywhere yet there is still a big gap between benchmark performance and reality

Codegen-6B for bug repair

Prompt

```
int start = 0;
// get a good guess on the size of the result buffer so
// it doesnt have to double if it goes over a bit
int increase = 0;
// count the replacement text elements that are larger
// than their corresponding text being replaced
```

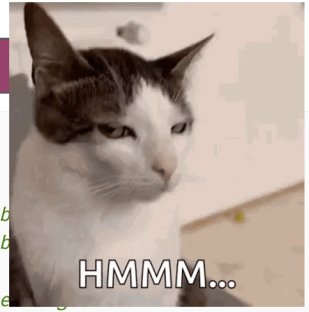
Generated

```
for (int i = 0; i < searchList.length; i++) {
    if (searchList[i] == null || replacementList[i] == null) {
        continue;
    }
    int greater = replacementList[i].length() - searchList[i].length();
    if (greater > 0) {
        increase += 3 * greater; // assume 3 matches
    }
}
// have upper-bound at 20% increase, then let Java take ...
```

Original fix file

```
int start = 0;
// get a good guess on the size of the result b
// it doesnt have to double if it goes over a b
int increase = 0;
// count the replacement text elements that are
// than their corresponding text being replaced

for (int i = 0; i < searchList.length; i++) {
    if (searchList[i] == null || replacementList[i] == null) {
        continue;
    }
    int greater = replacementList[i].length() - searchList[i].length();
    if (greater > 0) {
        increase += 3 * greater; // assume 3 matches
    }
}
// have upper-bound at 20% increase, then let Java take ...
```



Defects4J, Bug 39 (in Lang project)

Memorization often happens due to data leakage which is hard to detect and quantify 🥺

Xu, Ruijie, et al. "Benchmarking benchmark leakage in large language models." arXiv preprint arXiv:2404.18824 (2024).

Memorization often happens due to data leakage which is hard to detect and quantify 🥺

We don’t know if the model was trained on the test set

Xu, Ruijie, et al. "Benchmarking benchmark leakage in large language models." arXiv preprint arXiv:2404.18824 (2024).

Memorization often happens due to data leakage which is hard to detect and quantify 🥺

We don't know if the model was trained on the test set

Model weights are often unavailable

Xu, Ruijie, et al. "Benchmarking benchmark leakage in large language models." arXiv preprint arXiv:2404.18824 (2024).

Memorization often happens due to data leakage which is hard to detect and quantify 🥺

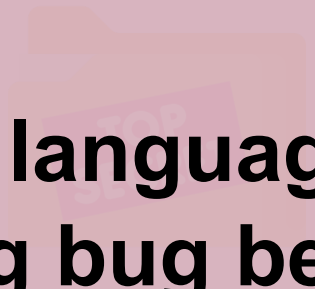
We don't know if the model was trained on the test set

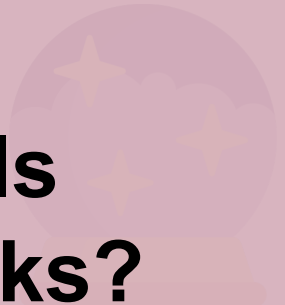
Model weights are often unavailable

Hard to define a metric to detect leakage

Xu, Ruijie, et al. "Benchmarking benchmark leakage in large language models." arXiv preprint arXiv:2404.18824 (2024).

Memorization often happens due to data leakage which is hard to detect and quantify 🥺





Are large language models memorizing bug benchmarks?

We don’t know if the model was trained on the test set

Model weights are often unavailable

Hard to define a metric to detect leakage

We designed an experimental setup to detect memorization across models and bug benchmarks

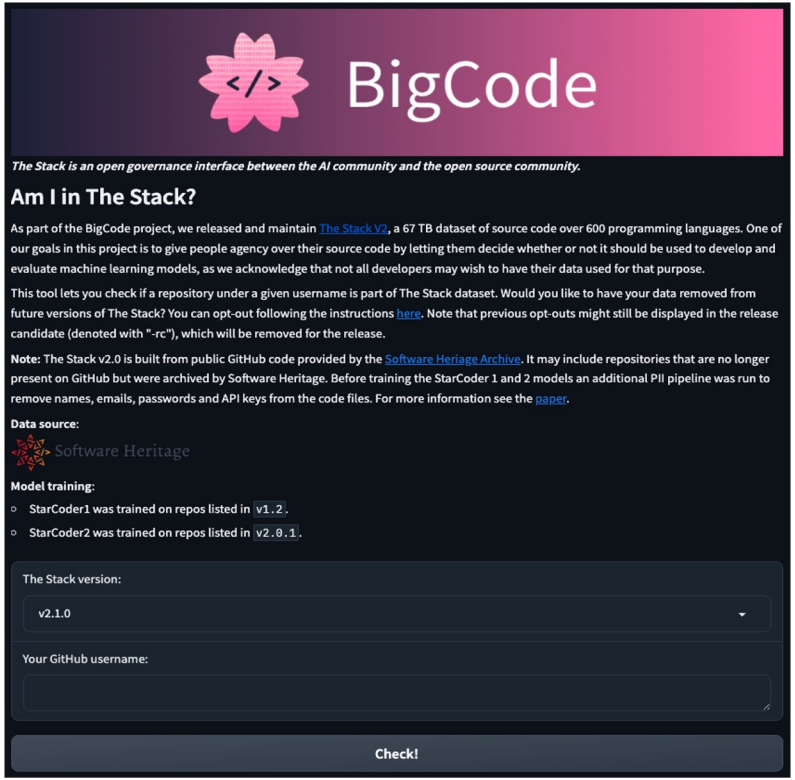
LEAKAGE DETECTION

We designed an experimental setup to detect memorization across models and bug benchmarks

LEAKAGE DETECTION

MEMBERSHIP

~ is my data part of The Stack ?



We designed an experimental setup to detect memorization across models and bug benchmarks

LEAKAGE DETECTION

MEMBERSHIP

~ is my data part of The Stack ?

NEG. LOG LIKELIHOOD

~ how surprised is the model by my input ?

We designed an experimental setup to detect memorization across models and bug benchmarks

LEAKAGE DETECTION

MEMBERSHIP

~ is my data part of The Stack ?

NEG. LOG LIKELIHOOD

~ how surprised is the model by my input ?

5-GRAM ACCURACY

~ how close is the model output to the ground truth ?

We designed an experimental setup to detect memorization across models and bug benchmarks

LEAKAGE DETECTION

MEMBERSHIP

~ is my data part of The Stack ?

NEG. LOG LIKELIHOOD

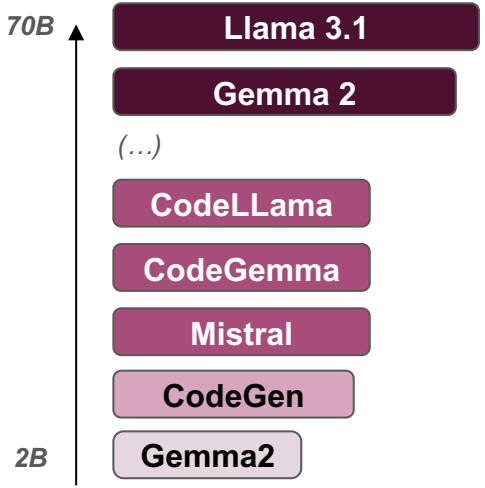
~ how surprised is the model by my input ?

5-GRAM ACCURACY

~ how close is the model output to the ground truth ?

MODEL SELECTION

9 open-source models



We designed an experimental setup to detect memorization across models and bug benchmarks

LEAKAGE DETECTION

MEMBERSHIP

~ is my data part of The Stack ?

NEG. LOG LIKELIHOOD

~ how surprised is the model by my input ?

5-GRAM ACCURACY

~ how close is the model output to the ground truth ?

MODEL SELECTION

9 open-source models

70B

- Llama 3.1
- Gemma 2
- (...)
- CodeLLama
- CodeGemma
- Mistral
- CodeGen
- Gemma2

2B

BENCHMARKS

5 Bug Benchmarks

- BugsC++
- GitBug-Java
- BugsInPy
- Defects4J
- Swebench-Lite

+

New, likely unseen data

Github repos in Java and Python

Our findings show that…

MEMBERSHIP **All benchmarks were part of The Stack to some degree.**

Dataset	Year	Membership (%)		
		v1.0	v2.0	v2.1
Defects4J	**2019**	80.0	80.0	80.0
BugsInPy	2020	94.1	64.7	64.7
BugsC++	2021	60.9	60.9	65.2
Gitbug-Java	2023	61.1	42.6	38.9
SweBench-Lite	**2024**	83.3	91.7	83.3

Our findings show that…

MEMBERSHIP **All benchmarks were part of The Stack to some degree.**

👎 D4J, SWEBench Lite

Dataset	Year	Membership (%)		
		v1.0	v2.0	v2.1
Defects4J	**2019**	80.0	80.0	80.0
BugsInPy	2020	94.1	64.7	64.7
BugsC++	2021	60.9	60.9	65.2
Gitbug-Java	2023	61.1	42.6	38.9
SweBench-Lite	**2024**	83.3	91.7	83.3

Our findings show that…

MEMBERSHIP **All benchmarks were part of The Stack to some degree.**

👎 D4J, SWEBench Lite

👍 Gitbug-Java

Dataset	Year	Membership (%)		
		v1.0	v2.0	v2.1
Defects4J	2019	80.0	80.0	80.0
BugsInPy	2020	94.1	64.7	64.7
BugsC++	2021	60.9	60.9	65.2
Gitbug-Java	2023	61.1	42.6	38.9
SweBench-Lite	2024	83.3	91.7	83.3

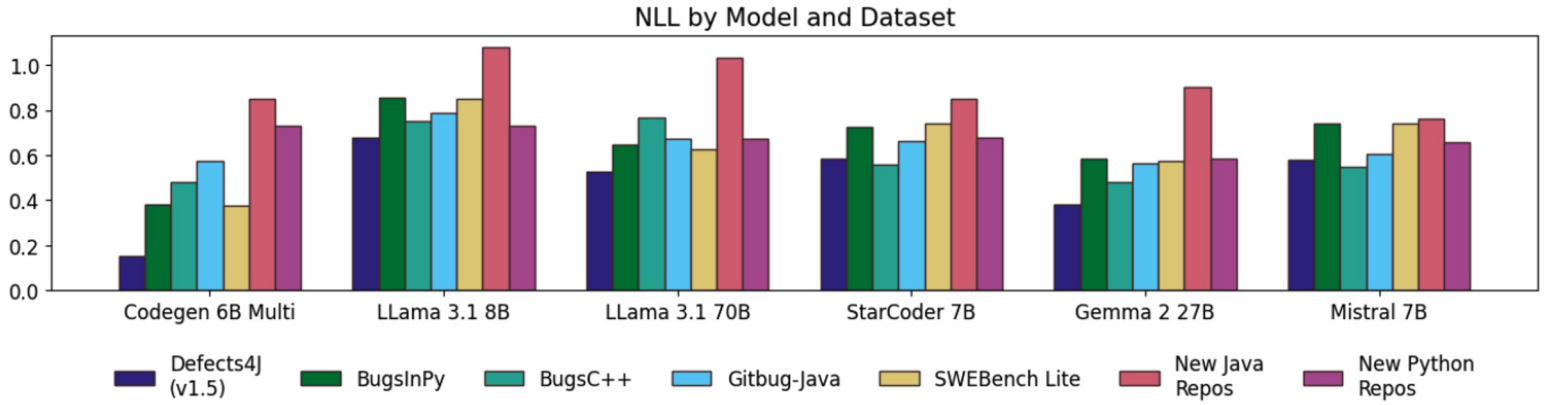
Our findings show that…

MEMBERSHIP

All benchmarks were part of The Stack to some degree.

NEG LOG LIKELIHOOD

Prominent benchmarks elicit lower NLLs across all models.

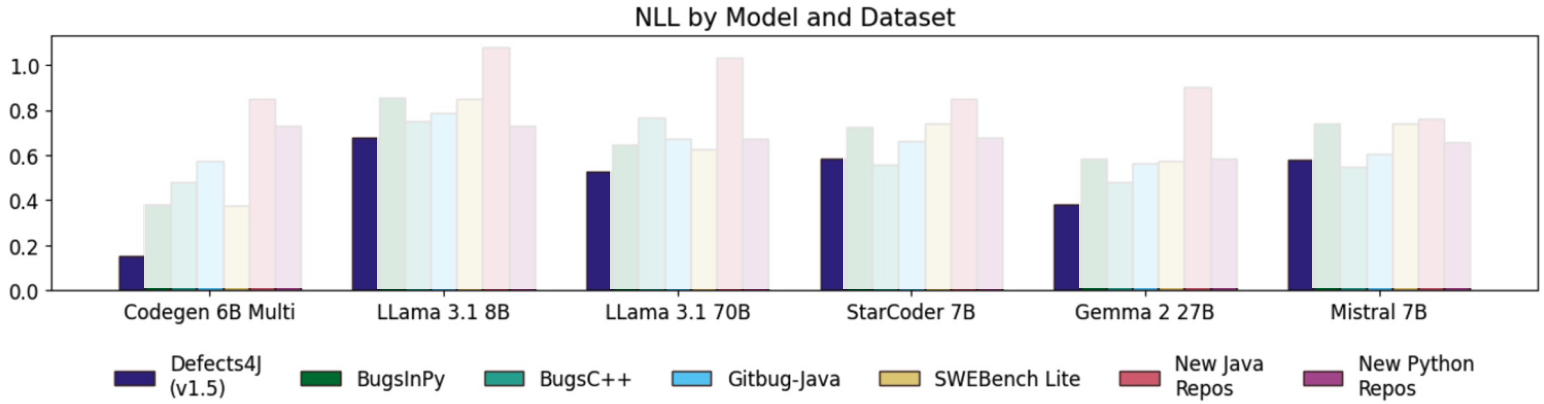


Our findings show that…

MEMBERSHIP All benchmarks were part of The Stack to some degree.

NEG LOG LIKELIHOOD **Prominent benchmarks elicit lower NLLs across all models.**

👎 D4J, SWEBench Lite



Our findings show that…

MEMBERSHIP

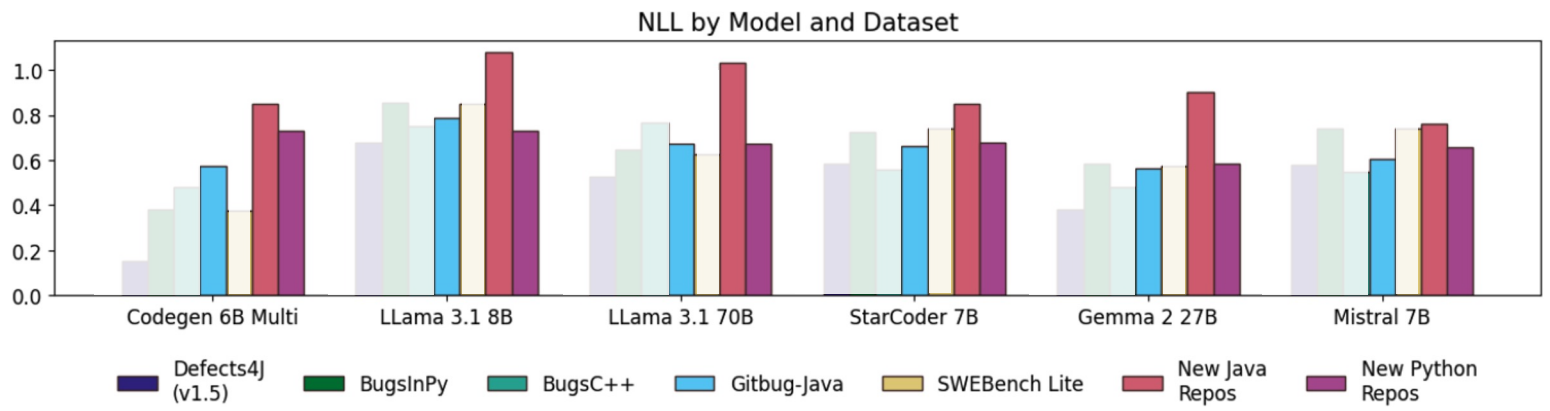
All benchmarks were part of The Stack to some degree.

NEG LOG LIKELIHOOD

Prominent benchmarks elicit lower NLLs across all models.

👎 D4J, SWEBench Lite

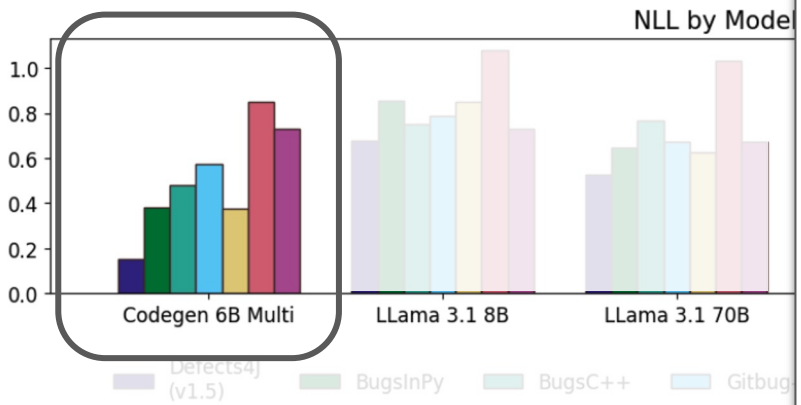
👍 New and unseen data, Gitbug-Java



Our findings show that…

MEMBERSHIP All benchmarks were part

NEG LOG LIKELIHOOD Prominent bench



NLL Ratios for Codegen 6B Multi

	Defects4J (v1.5)	BugsInPy	Bugs C++	Gitbug Java	SWEBench Lite	New Java	New Python
Defects4J (v1.5)	1.00	2.55	3.18	3.83	2.51	5.63	4.86
BugsInPy	0.39	1.00	1.25	1.50	0.98	2.21	1.91
Bugs C++	0.31	0.80	1.00	1.20	0.79	1.77	1.53
Gitbug Java	0.26	0.67	0.83	1.00	0.65	1.47	1.27
SWEBench Lite	0.40	1.02	1.27	1.53	1.00	2.25	1.94

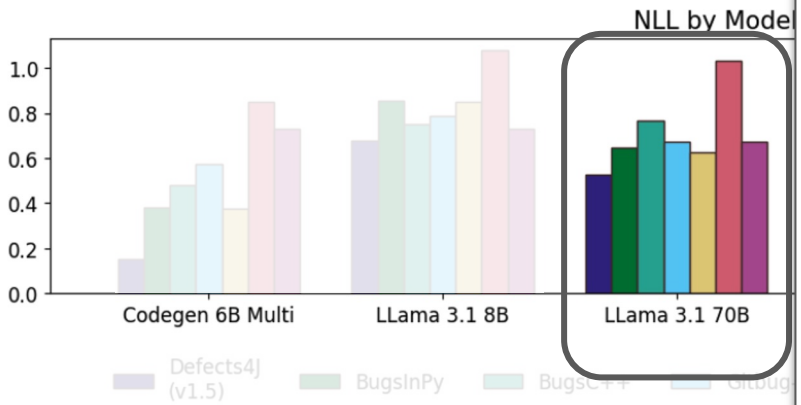
Potential memorization signals.

It is ~5x more familiar with D4J than newer data.

Our findings show that…

MEMBERSHIP All benchmarks were part

NEG LOG LIKELIHOOD Prominent bench



NLL Ratios for LLama 3.1 70B

	Defects4J (v1.5)	BugsInPy	Bugs C++	Gitbug Java	SWEBench Lite	New Java	New Python
Defects4J (v1.5)	1.00	1.22	1.45	1.27	1.18	1.95	1.28
BugsInPy	0.82	1.00	1.19	1.04	0.97	1.59	1.04
Bugs C++	0.69	0.84	1.00	0.88	0.82	1.34	0.88
Gitbug Java	0.79	0.96	1.14	1.00	0.93	1.54	1.01
SWEBench Lite	0.84	1.03	1.23	1.07	1.00	1.65	1.08

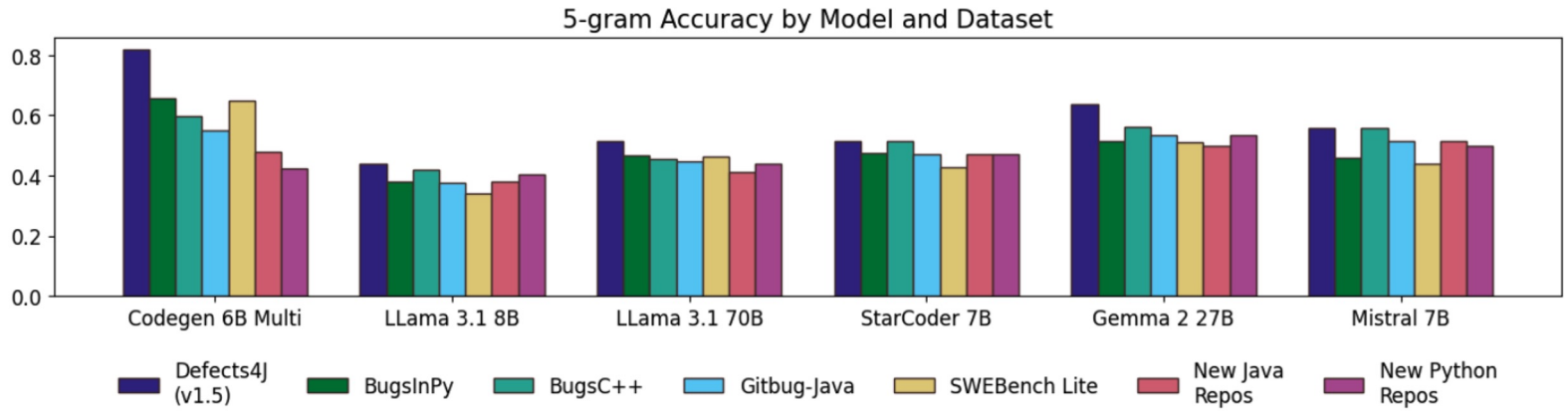
✨ **More consistent behavior across benchmarks.**

Our findings show that…

MEMBERSHIP — All benchmarks were part of The Stack to some degree.

NEG LOG LIKELIHOOD — Prominent benchmarks elicit lower NLLs across all models.

5-GRAM ACCURACY — **Prominent benchmarks elicit higher 5-gram matches across model families.**



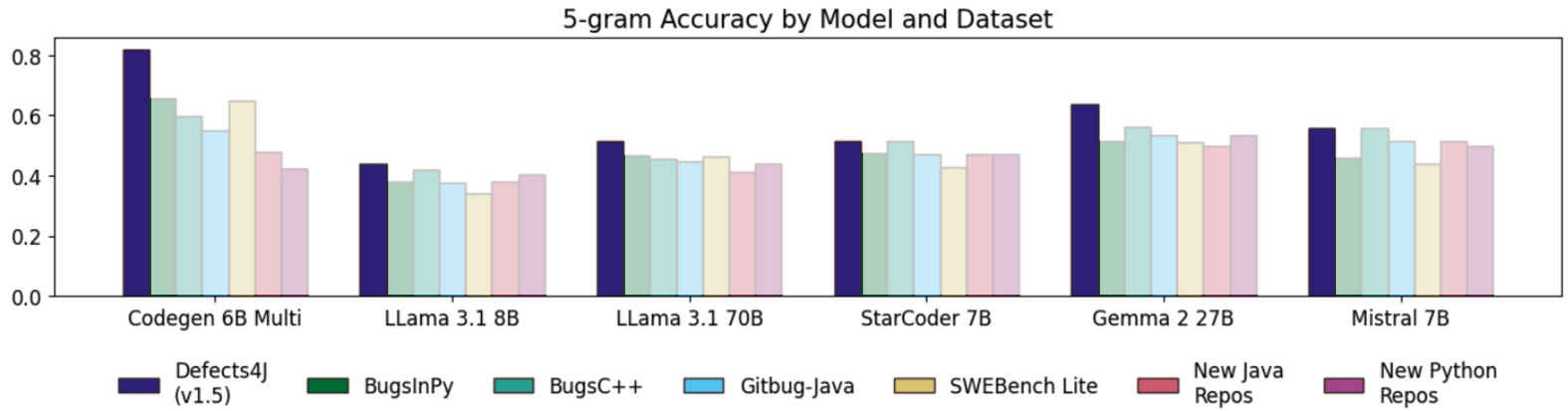
Our findings show that…

MEMBERSHIP — All benchmarks were part of The Stack to some degree.

NEG LOG LIKELIHOOD — Prominent benchmarks elicit lower NLLs across all models.

5-GRAM ACCURACY — **Prominent benchmarks elicit higher 5-gram matches across model families.**

👎 D4J



Our findings show that…

MEMBERSHIP

All benchmarks were part of The Stack to some degree.

NEG LOG LIKELIHOOD

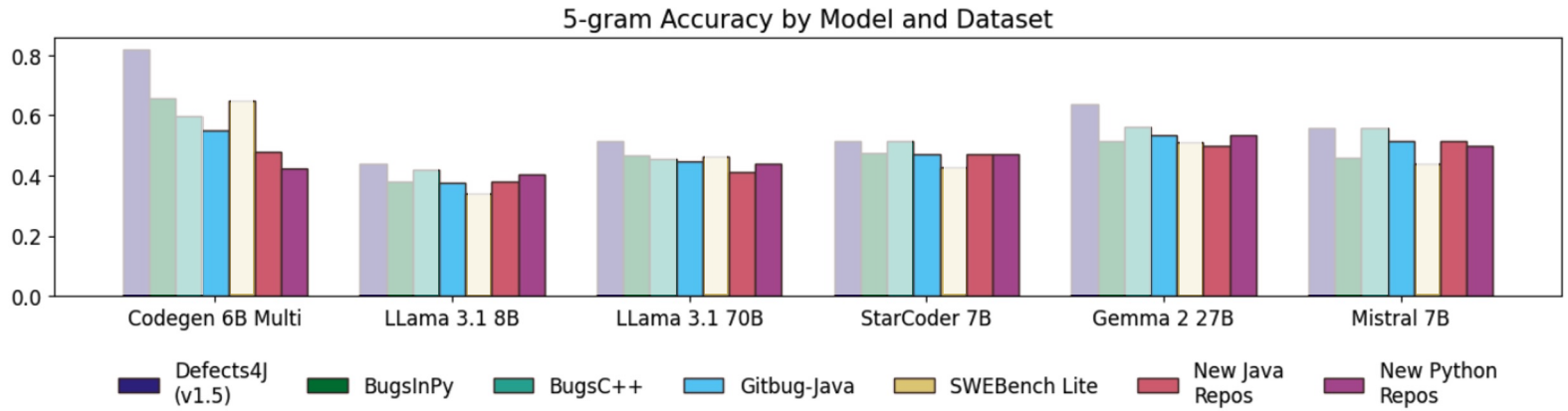
Prominent benchmarks elicit lower NLLs across all models

5-GRAM ACCURACY

Prominent benchmarks elicit higher 5-gram matches across model families.

👎 D4J

👍 New and unseen data, Gitbug-Java



Our findings show that…

🙁 **Data leakage is an especially significant issue for Defects4J (V1.5)**

Our findings show that…

🙁 **Data leakage is an especially significant issue for Defects4J (V1.5)**

Smaller models, trained on far less data, seem more prone to memorization

Our findings show that…

🙁 **Data leakage is an especially significant issue for Defects4J (V1.5)**

Smaller models, trained on far less data, seem more prone to memorization

Bigger and more recent models seem to exhibit lower memorization of benchmark solutions

If we are not careful with benchmark selection, we risk reporting inflated model performance due to data leakage

So, we suggest…

Evaluate on benchmarks with new data

If we are not careful with benchmark selection, we risk reporting inflated model performance due to data leakage

So, we suggest…

Evaluate on benchmarks with new data

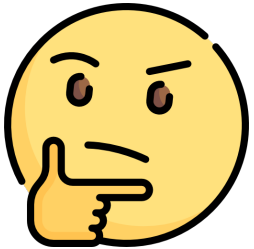
Monitor leakage risk over time by computing membership, NLL and 5-gram matches

If we are not careful with benchmark selection, we risk reporting inflated model performance due to data leakage

So, we suggest…

Evaluate on benchmarks with new data

Monitor leakage risk over time by computing membership, NLL and 5-gram matches



Carefully interpret benchmarks numbers

Memorization often happens due to data leakage which is hard to detect and quantify 🥹

We don't know if the model was trained on the test set

Model weights are often unavailable

Hard to define a metric to detect leakage

We designed an experimental setup to detect memorization across models and bug benchmarks

LEAKAGE DETECTION

MEMBERSHIP
~ is my data part of The Stack ?

NEG. LOG LIKELIHOOD
~ how surprised is the model by my input ?

5-GRAM ACCURACY
~ how close is the model output to the ground truth ?

MODEL SELECTION

9 open-source models

70B
- Llama 3.1
- Gemma 2
- (...)
- CodeLLama
- CodeGemma
- Mistral
- CodeGen
- Gemma2

2B

BENCHMARKS

5 Bug Benchmarks

- BugsC++
- GitBug-Java
- BugsInPy
- Defects4J
- Swebench-Lite

+

New, likely unseen data

- Github repos in Java and Python

Check out the paper! 😜

Our findings show that…

🙁 **Data leakage is an especially significant issue for Defects4J (V1.5)**

Smaller models, trained on far less data, seem more prone to memorization

Bigger and more recent models seem to exhibit lower memorization of benchmark solutions

If we are not careful with benchmark selection, we risk reporting inflated model performance due to data leakage

So, we suggest…

Evaluate on benchmarks with new data

Monitor leakage risk over time by computing membership, NLL and 5-gram matches

Carefully interpret benchmarks numbers

S3D Software and Societal Systems Department

Carnegie Mellon Portugal

fct Fundação para a Ciência e a Tecnologia