NIST GenAI Pilot Code Challenge: Round-1 and Round-2 Overview, Results, and Lessons Learned

NIST GenAI Code Team: Peter Fontana, Sonika Sharma, Hari Iyer, Kay Peterson, Lukas Diduch, Bikesh Regmi, Yooyoung Lee

May 19, 2026

NIST INFORMATION TECHNOLOGY LABORATORY

Our Contributing Members: The NIST GenAI Code Team



Peter Fontana
(Computer Scientist,
Project Lead)



Sonika Sharma
(Computer Scientist)



Hari Iyer
(Statistician)



Kay Peterson
(Social Scientist)



Lukas Diduch
(Research Engineer)



Bikesh Regmi
(Computer Scientist, PREP)



Yooyoung Lee
(Group Leader)

Disclaimer

While participants may report their own results, participants may NOT make advertising claims about their standing in the evaluation, in regards to rank, winning the evaluation, or claim NIST endorsement of their system(s). The following language in the U.S. Code of Federal Regulations (15 C.F.R. § 200.113(d)) shall be respected: NIST does not approve, recommend, or endorse any proprietary product or proprietary material. No reference shall be made to NIST or to reports or results furnished by NIST in any advertising or sales promotion which would indicate or imply that NIST approves, recommends, or endorses any proprietary product or proprietary material or which has as its purpose an intent to cause directly or indirectly the advertised product to be used or purchased because of NIST test reports or results.

Disclaimer

Per the Sept. 29, 2026, Executive Order on Inaugurating the Era of Super Intelligence* NIST is working to update its communications to incorporate the term “super intelligence” as directed.

* https://www.whitehouse.gov/presidential-actions/2026/09/inaugurating-the-era-of-super-intelligence/

Disclaimer

Certain equipment, instruments, software, or materials, commercial or non-commercial, are identified in this paper in order to specify the experimental procedure adequately. Such identification does not imply recommendation or endorsement of any product or service by NIST, nor does it imply that the materials or equipment identified are necessarily the best available for the purpose.

Overview

NIST INFORMATION TECHNOLOGY LABORATORY

Purpose of NIST Code Challenge

Can Super Intelligence Write Code for Testing Simple Software Reliably?

Our Evaluation



Input
(what the participants will receive)

Participants
(the participant's task)

Output
(what we will receive)

Fixed Prompt Construction from Specification

```
We have python code that implements the following specification.

Specification:

<Specification_String>


Please write python pytest test code that comprehensively tests the code for method <primary_method_name> to determine
if the code correctly meets the specification or not. When writing tests:
* write a comprehensive test suite,
* test edge cases,
* only generate tests that you are confident are correct, and
* include tests for TypeError cases.

Please write '###|=-=-=beginning of tests=-=-=|' before the tests. Write '###|=-=-=end of tests=-=-=|' immediately
after the tests. Import any needed packages, including pytest. Additionally import the code being tested by adding the
line `from genai_code_file import <primary_method_name>` the line after '###|=-=-=beginning of tests=-=-=|'. Do not
provide an implementation of the method <primary_method_name> with the tests.
```

Two problem-specific inputs are added from each problem: the <specification_string>, the specification; and <primary_method_name>, the name of the primary method to implement

Task for Code Pilot Challenge

Our focus is on test code generation

Take a text specification that describes the software functionality

Output Python code that tests the correctness of the software

Teams for Round-1 & Round-2

Round-1

Total Registered: 12

Submitted: 4

Round-2

Total Registered: 18

Submitted: 5

* Every participant in round-1 was automatically registered for round-2
* 2 teams submitted in both round-1 and round-2

Submissions (Number of Runs)

Round-1

Fixed Prompt: 17

Custom Prompt: 24

Round-2

Fixed Prompt: 33

Custom Prompt: 33

Fixed prompt submissions used the provided fixed prompts

Custom prompt submissions accepted any prompt the participant wished to use

*We refer to a participant's complete set of test outputs across all problems as a *run*. Each participant submission had at least one fixed prompt run and at least one custom prompt run, with up to nine custom prompt runs permitted.

Terminology: Our Data

The Code Bank:

A collection of individual trials (problems), each **trial** consisting of a specification and 3 held-back implementations (1 correct, 2 incorrect) programs. Each trial is identified by a **trial_id**. We refer to the output set of unit tests for each trial as a **test output**.

Simple Trials

15 max lines of code

1 method

Extended Trials

30 max lines of code per method

1 - 2 methods

*must either have two methods or exceed 15 lines of code

Example of Simple Trial vs. Extended Trial Specification

Simple Trials

```
def add(x: int, y: int) -> int:
    """
    Given two integers x, and y,
return the sum of x and y. If
either x or y is not
    an integer, raise a
TypeError Exception.
    """
```

Extended Trials

```
def make_palindrome(st: str) ->
str:
    """
    Given a string st, return
the shortest palindrome that
begins with the supplied string
str. A palindrome is a string
    that reads the same when
read backwards or forwards. If
st is not a string, raise a
TypeError exception.
    If st is empty, return an
empty string.
    """
```

Example of Simple Trial vs. Extended Trial Correct Code

Simple Trials

```
def add(x: int, y: int) -> int:
    if (not isinstance(x, int)) or (not
isinstance(y, int)):
        raise TypeError('Inputs x and y
must both be integers.')
    return x + y
```

Extended Trials

```
def is_palindrome(string: str) -> bool:
    """ Test if given string is a palindrome """
    return string == string[::-1]

def make_palindrome(st: str) -> str:
    if not isinstance(st, str):
      raise TypeError('Input st must be a
string.')
    if not st:
        return ''

    beginning_of_suffix = 0

    while not
is_palindrome(st[beginning_of_suffix:]):
        beginning_of_suffix += 1

    return st + st[:beginning_of_suffix][::-1]
```

Example of Simple Trial Fixed prompt

Simple Trials

```
We have python code that implements the following specification.

Specification:

def add(x: int, y: int) -> int:
    """
    Given two integers x, and y, return the sum of x and y. If either x or y is not
    an integer, raise a TypeError Exception.
    """


Please write python pytest test code that comprehensively tests the code for method add to determine if
the code correctly meets the specification or not. When writing tests:
* write a comprehensive test suite,
* test edge cases,
* only generate tests that you are confident are correct, and
* include tests for TypeError cases.

Please write '###|=-=-=beginning of tests=-=-=|' before the tests. Write '###|=-=-=end of tests=-=-=|'
immediately after the tests. Import any needed packages, including pytest. Additionally import the code
being tested by adding the line `from genai_code_file import add` the line after '###|=-=-=beginning of
tests=-=-=|'. Do not provide an implementation of the method add with the tests.
```

Example of Extended Trial Fixed prompt

Extended Trials

```
We have python code that implements the following specification.

Specification:

def make_palindrome(st: str) -> str:
    """
    Given a string st, return the shortest palindrome that begins with the supplied string str. A palindrome is a string
    that reads the same when read backwards or forwards. If st is not a string, raise a TypeError exception.
    If st is empty, return an empty string.
    """


Please write python pytest test code that comprehensively tests the code for method make_palindrome to determine if the code
satisfies the specification or not. When writing tests:
* write a comprehensive test suite,
* test edge cases,
* only generate correct tests, and
* include tests for TypeError cases.

Please write '###|=-=-=beginning of tests=-=-=|' before the tests. Write '###|=-=-=end of tests=-=-=|' immediately after the
tests. Import any needed packages, including pytest. Import the code being tested by adding the line `from genai_code_file
import make_palindrome` the line after '###|=-=-=beginning of tests=-=-=|'. Do not provide an implementation of the method
make_palindrome with the tests.
```

Primary Scoring Over Runs

Correct Tests

If the test output finds no error in the held-back correct implementation

Functional (CI1) & Type (CIT) Errors

If the test output is correct and finds at least one error in both held-back incorrect implementations

Code Coverage (cov): The percentage of lines of code that the tests cover (or execute during the tests) in the correct implementation (cov is NA if the test output is not correct)

Round-1 Overview

NIST INFORMATION TECHNOLOGY LABORATORY

Overview of Round-1

Code Bank

40 simple trial_ids
9 extended trial_ids

Our Metrics

Mean Coverage
Correct Tests
Finds CI1 Error
Find CIT Error
Finds CI1 & CIT Errors
100% Coverage & Finds All Errors

Lessons Learned From Round-1

Some test outputs failed due to ambiguous specifications

Manual Review

every failing test **output** of each team's most recent submission

Outcome

some specifications did not pinpoint the outputs for the provided test cases

Example of an Ambiguous Specifications

```
def snake_to_camel(word: str) -> str:
    """
    Given a string word, return the string word converted to CamelCase
from snake_case. If word is not a string,
    raise a TypeError exception. If word is empty, raise a ValueError
exception.
    """
```

How to handle input not in snake case?

`snake_to_camel_case("hello___World")”`
`= helloWorld`

Should camelCase capitalize the first letter?

often “no”, but some sources have CamelCase and camelCase as two different cases

Lessons Learned from Round-1

Type Checking

changed boolean to be considered as integers

we used isinstance(x, int) as our type check

Empty List Input

raise a ValueError for empty list inputs in list-based programs

Negative Integers

specify what should happen when integer inputs are negative.

Code Bank Size Round-1

Phase	Simple	Extended	Total
Round-1	40	9	49
Round-1 (ds)	20	6	26

*Round-1 (ds) is Round-1 after excluding the problems with ambiguous specifications, where **ds** stands for **d**ropped **s**pecifications.

Round-2 Overview

NIST INFORMATION TECHNOLOGY LABORATORY

Overview of Round-2

Specifications Dropped

4 (3 simple) specifications were dropped in Round-2

Specifications Repaired

12 (10 simple) specifications were directly repaired

7 (7 simple) specifications were corrected when the general changed were addressed

Specifications Added

We added 1 (0 simple) additional specification

Round 2 has a total of 46 (37 simple) specifications

Lessons Learned from Round-2

Test correctness (correct_tests) is a valuable metric, as some test outputs are not correct even on simple problems

Precise specifications are crucial for properly measuring data

Pilot Round-2 submissions show progress

Code Bank Size Round-1 & Round-2

Phase	Simple	Extended	Total
Round-1	40	9	49
Round-1 (ds)	20	6	26
Round-2	37	9	46

*Round-1 (ds) is Round-1 after excluding the problems with ambiguous specifications.

Scoring Terminology

NIST INFORMATION TECHNOLOGY LABORATORY

Terminology: Run and Submission

- We refer to a participant's complete set of test outputs across all problems as a **run**. Each participant **submission** had at least one fixed prompt run and at least one custom prompt run, with up to nine custom prompt runs permitted.
- Each **submission** has a `submission_id`,
- Each **run** is identified by a `(submission_id, prompt_number)` pair, and
  - `prompt_number` of 0 is fixed prompt condition
  - `prompt_number` in 1–9 is a custom prompt condition
- Each **test output** is identified by a `(submission_id, prompt_number, trial_id)` tuple

Terminology: Scoring test outputs

- Each test output is run on the corresponding three held-back programs and gets a 0/1 score for each program
- **`correct_tests`** (0/1): 1 if test output finds no errors on the correct program
- **`finds_ci1_error`** (0/1): 1 if test output is `correct_tests` and finds an error in the incorrect program 1 (**ci1**), the code with the functional error
- **`finds_cit_error`** (0/1): 1 if test output is `correct_tests` and finds an error on the incorrect program t (**cit**), the code missing type error and value error checking
- **`finds_ci1_and_cit_errors`** (0/1): 1 if test output is `correct_tests` and finds an error in both incorrect programs
- **`cov`** (%): % of lines of code the tests covered in the correct code

ci1 (code incorrect 1) programs contain functional errors, **cit** (code incorrect t) programs miss type error checking

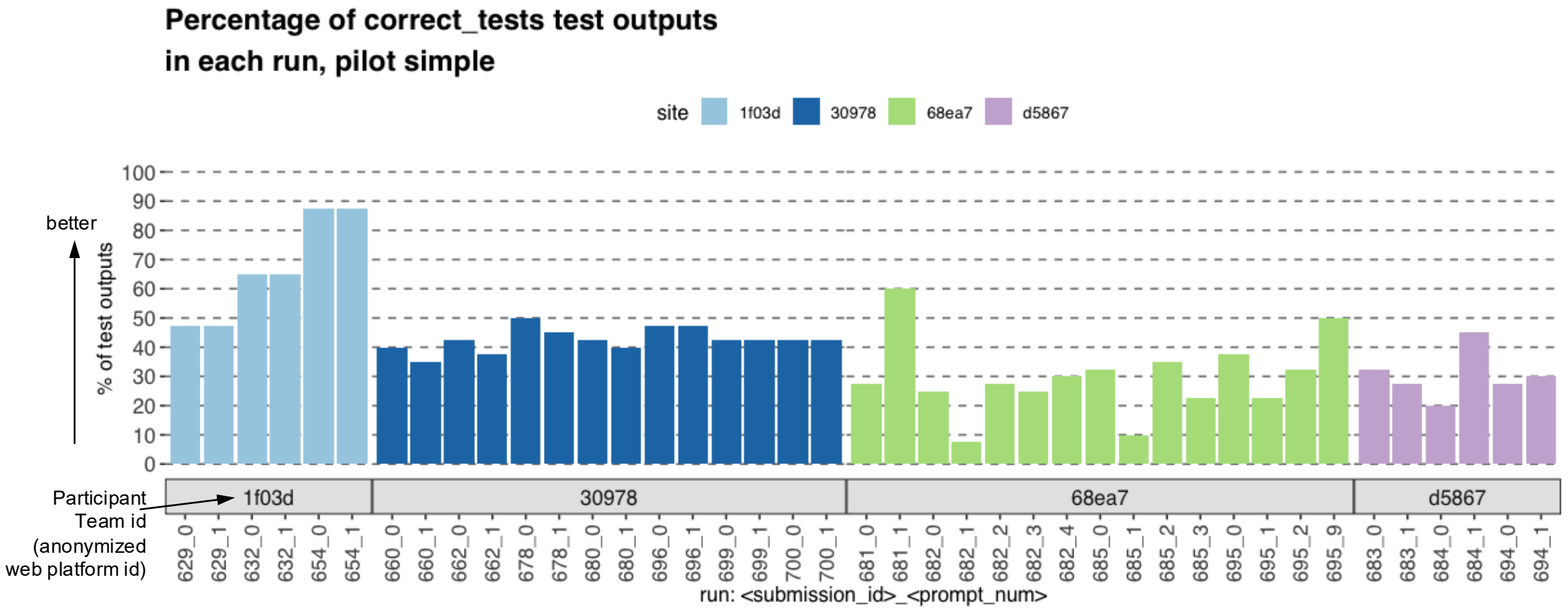
Terminology: We compute the % of test outputs to get a % score for each run

- `correct_tests` (%): **% of test outputs that** find no errors on the correct program
- `finds_ci1_error` (%): **% of test outputs that** are `correct_tests` and find an error in the incorrect program 1 (**ci1**), the code with the functional error
- `finds_cit_error` (%): **% of test outputs that** are `correct_tests` and find an error on the incorrect program t (**cit**), the code missing type error and value error checking
- `finds_ci1_and_cit_errors` (%): **% of test outputs** that are `correct_tests` and find an error in both incorrect programs
- `cov` (%): **mean % of coverage** over all test outputs

Round-1 Results

NIST INFORMATION TECHNOLOGY LABORATORY

Round-1 `correct_tests` result

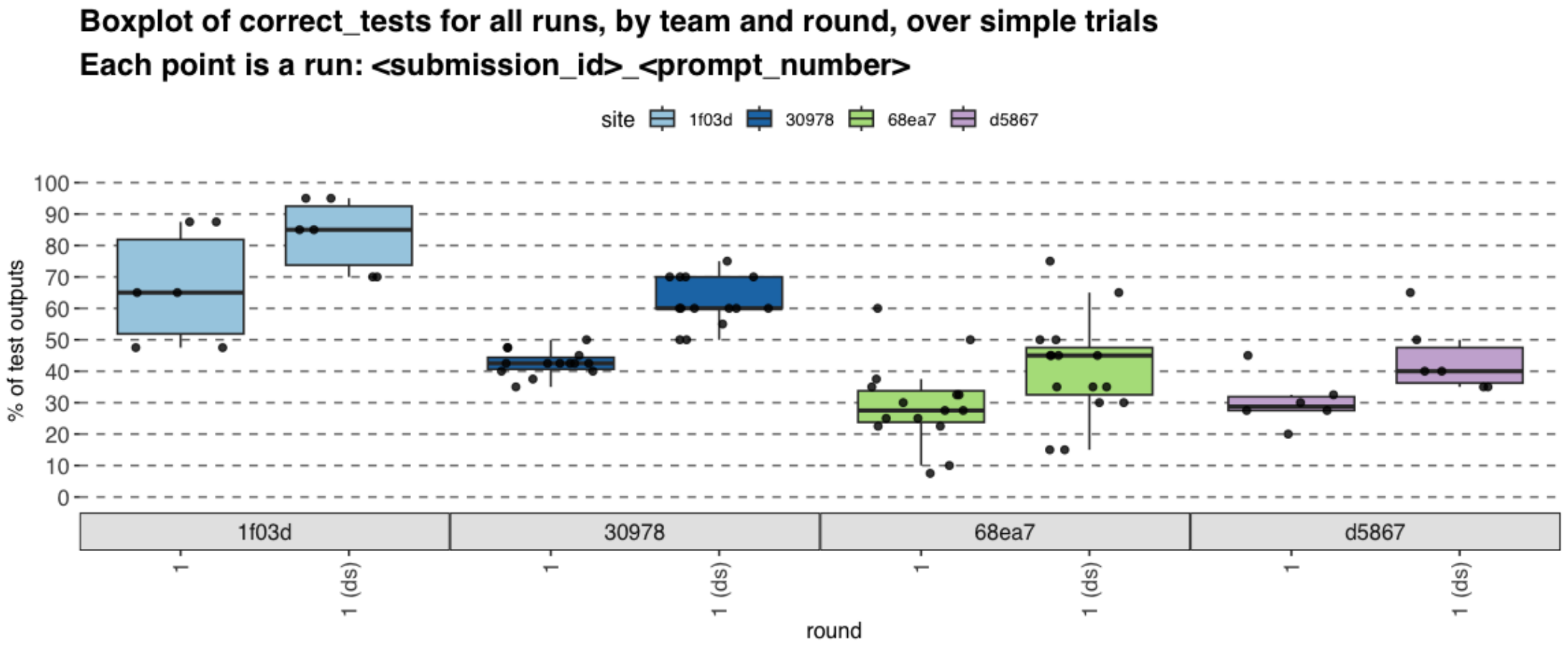


`correct_tests` (%): **% of test outputs that** find no errors on the correct program

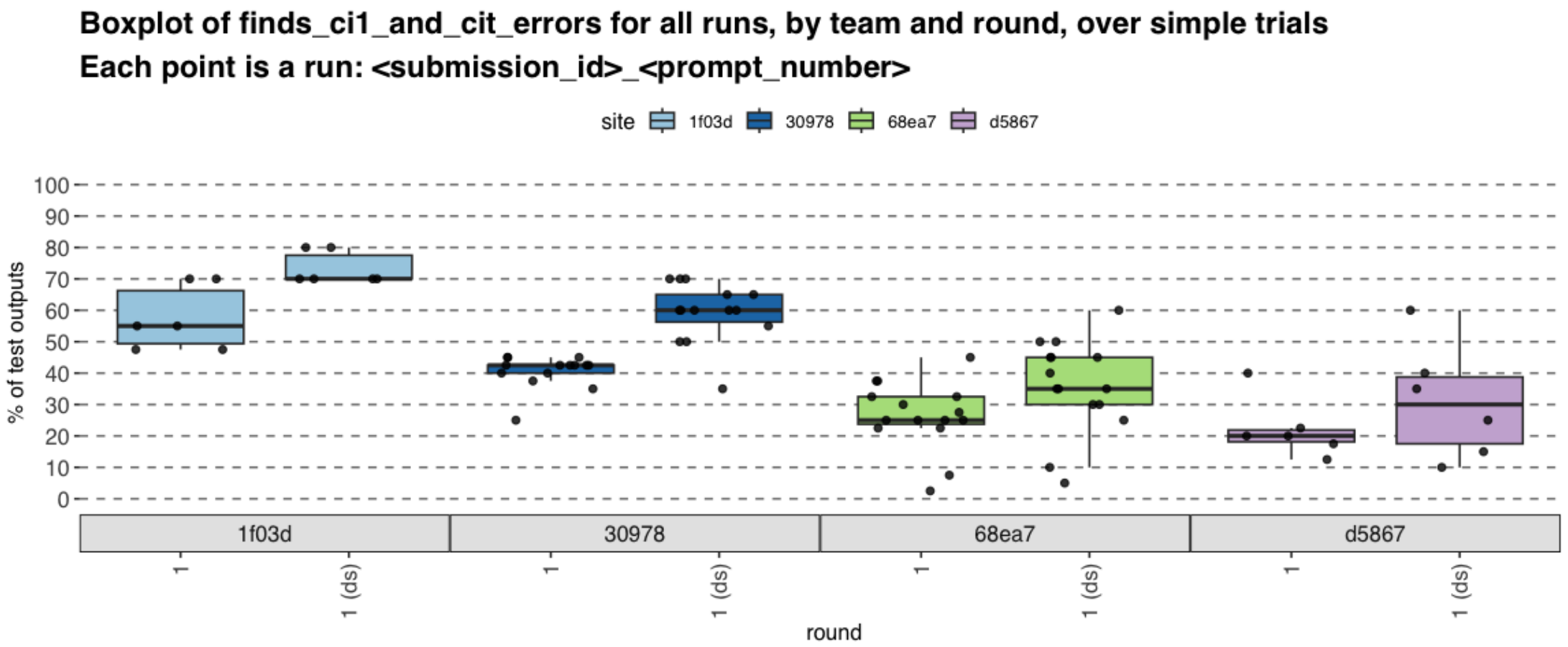
Insight: Some test outputs failed due to ambiguous specifications

- **Manual review** of every **failing test output** of each team's most recent submission
- **Outcome:** Manual comparison of test outputs to correct code and original specifications showed that **some specifications did not precisely specify outputs for certain inputs**.
- **Round-1 (ds): Round-1** after dropping ambiguous specifications and rescoring

Round-1 (ds) `correct_test` scores are higher when imprecise specifications are dropped



Round-1 (ds) `finds_ci1_and_cit_errors` scores are higher when imprecise specifications are dropped



Round-1 (ds) vs. Round-1 correct_tests: each median is at least 12.2% higher and each max is at least 7.5% higher

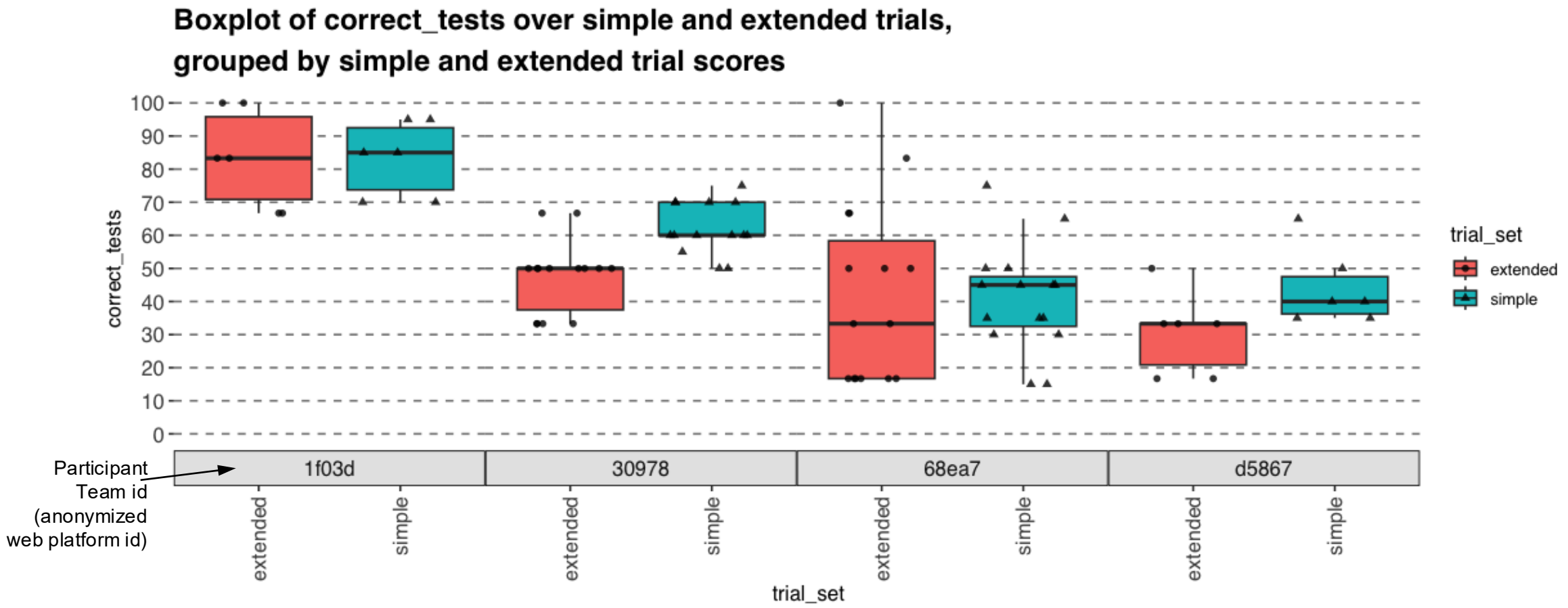
Team	1f03d		30978		68ea7		d5867	
Statistic	Round-1	Round-1 (ds)	Round 1	Round-1 (ds)	Round-1	Round-1 (ds)	Round-1	Round-1 (ds)
mean	66.7	83.3	42.7	62.1	29.7	41	30.4	44.2
sd	15.2	11.3	4.0	7.8	13.3	16.2	8.3	11.6
median	**65**	**85**	**42.5**	**60**	**27.5**	**45**	**28.8**	**40**
75th percentile	81.9	92.5	44.4	65	33.8	47.5	31.9	47.5
90th percentile	87.5	95	47.5	70	45	59	38.8	57.5
maximum	**87.5**	**95**	**50**	**70**	**60**	**75**	**45**	**65**

Round-1 (ds) is **Round-1** after dropping ambiguous specifications and rescoring

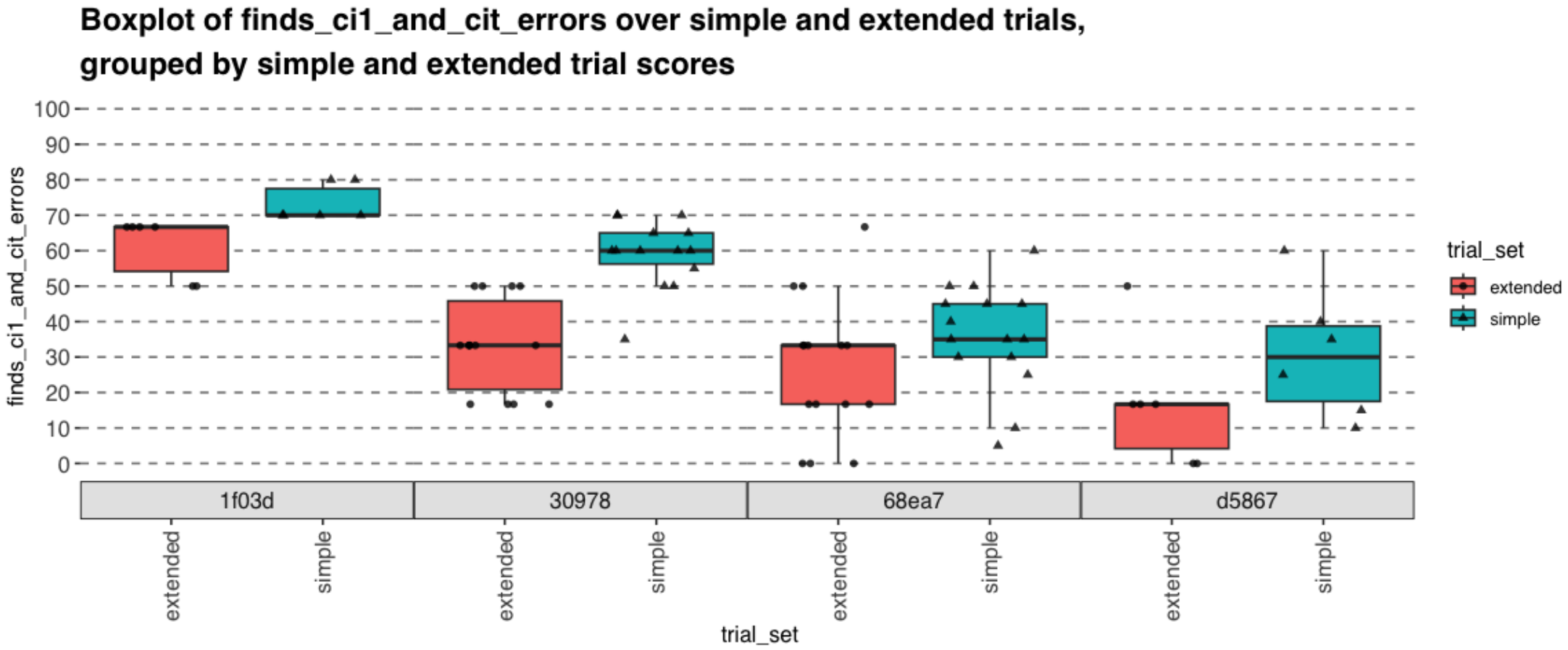
Round-1 (ds) Results

NIST INFORMATION TECHNOLOGY LABORATORY

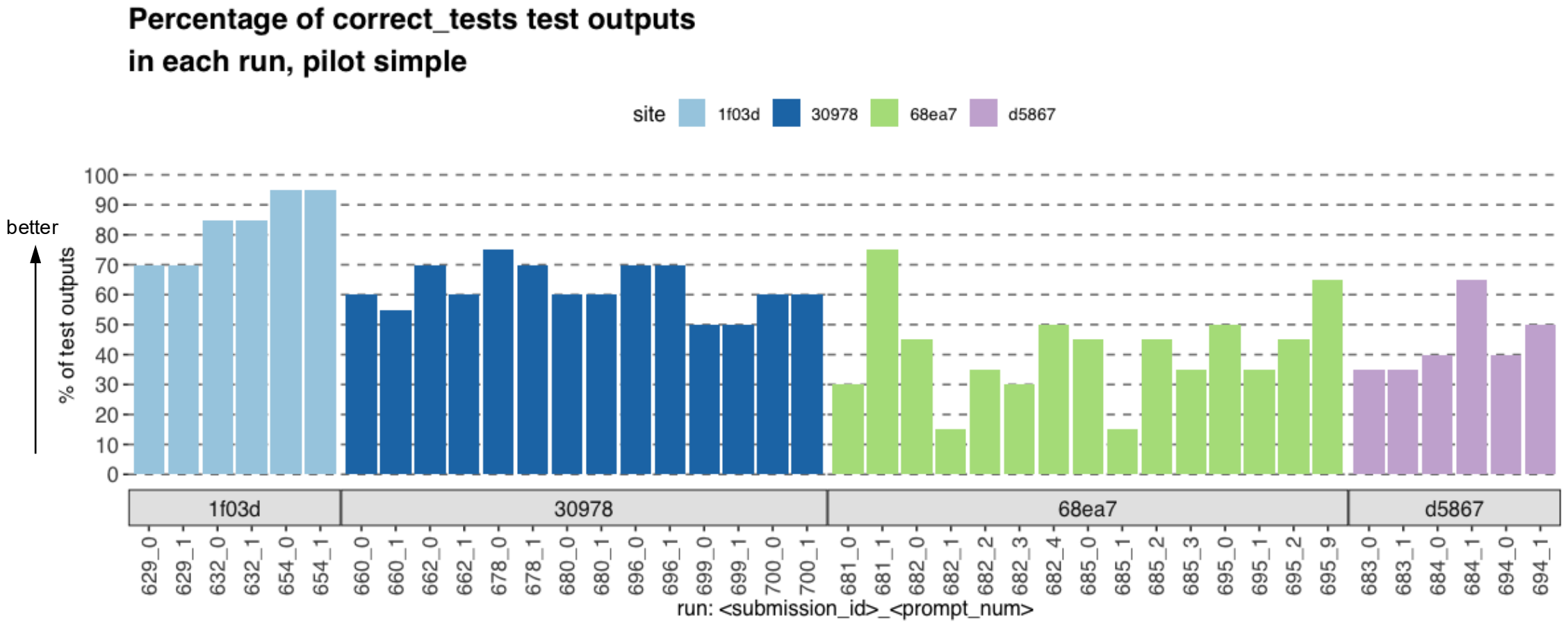
Round-1 (ds) results: `correct_tests`



Round-1 (ds) results: `finds_ci1_and_cit_errors`

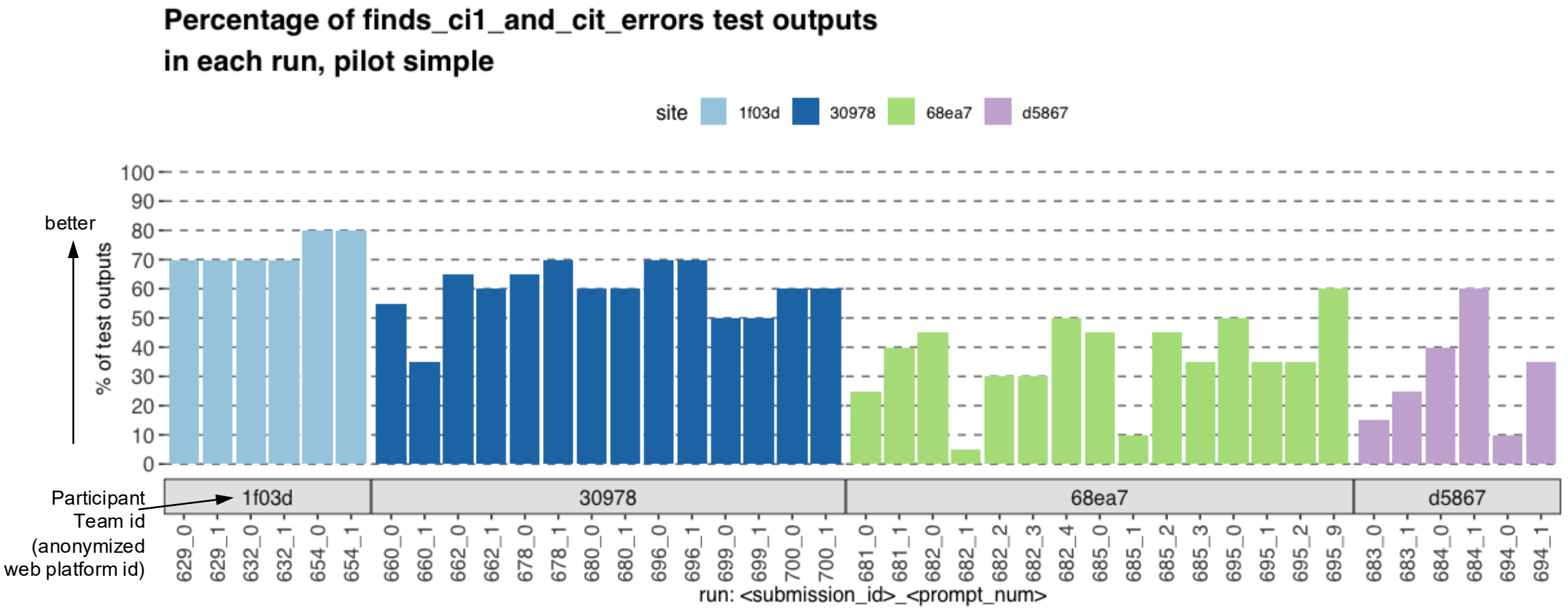


Round-1 (ds): `correct_tests` over simple trials



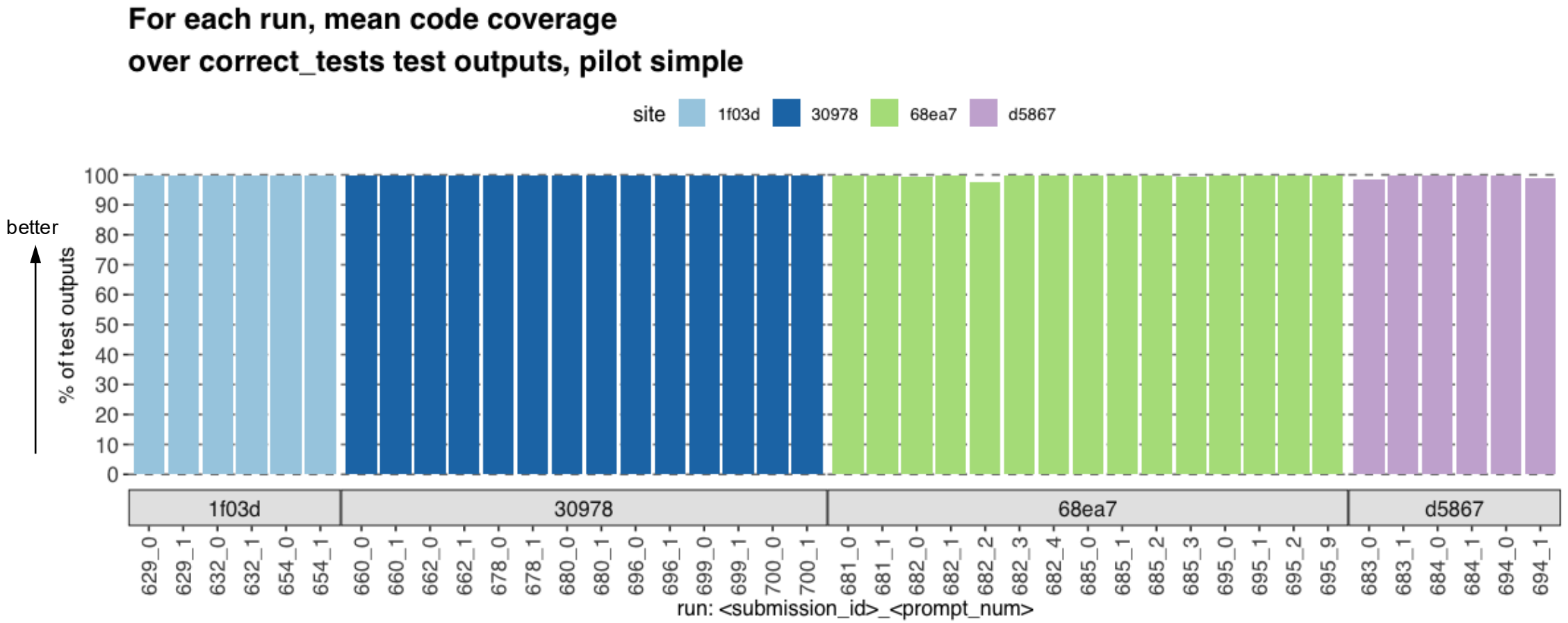
`correct_tests` (%): **% of test outputs that** find no errors on the correct program

Round-1 (ds): `finds_ci1_and_cit_errors` over simple trials



`finds_ci1_and_cit_errors` (%): **% of test outputs** that are `correct_tests` and find an error in both incorrect programs

Round-1 (ds): cov over simple trials

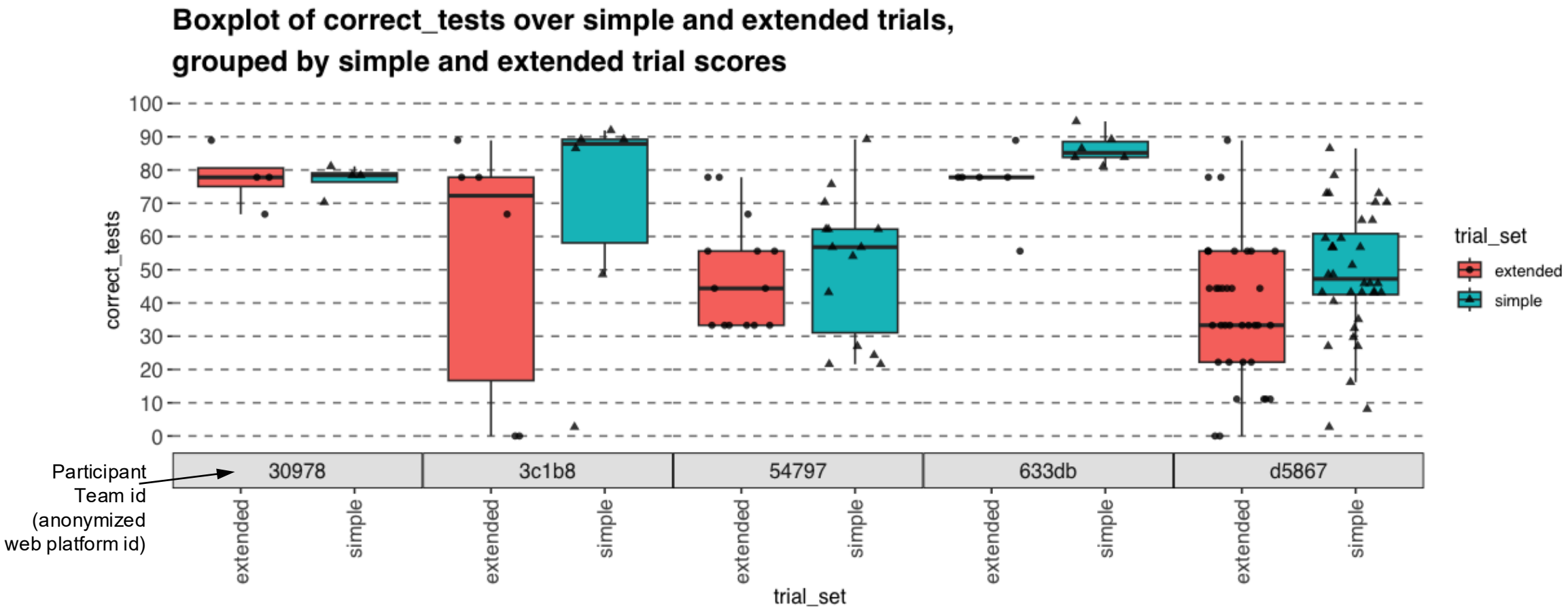


cov (%): **mean % of coverage** over all test outputs

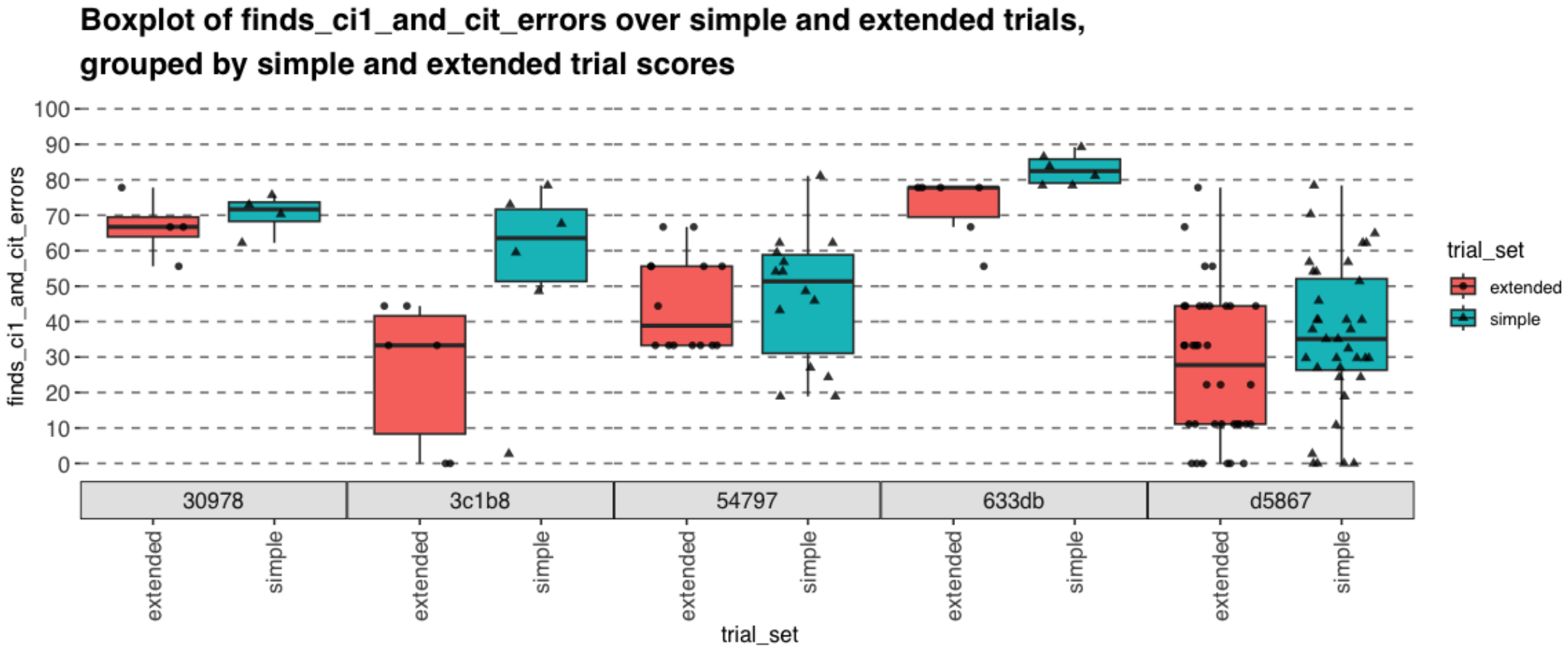
Round-2 Results

NIST INFORMATION TECHNOLOGY LABORATORY

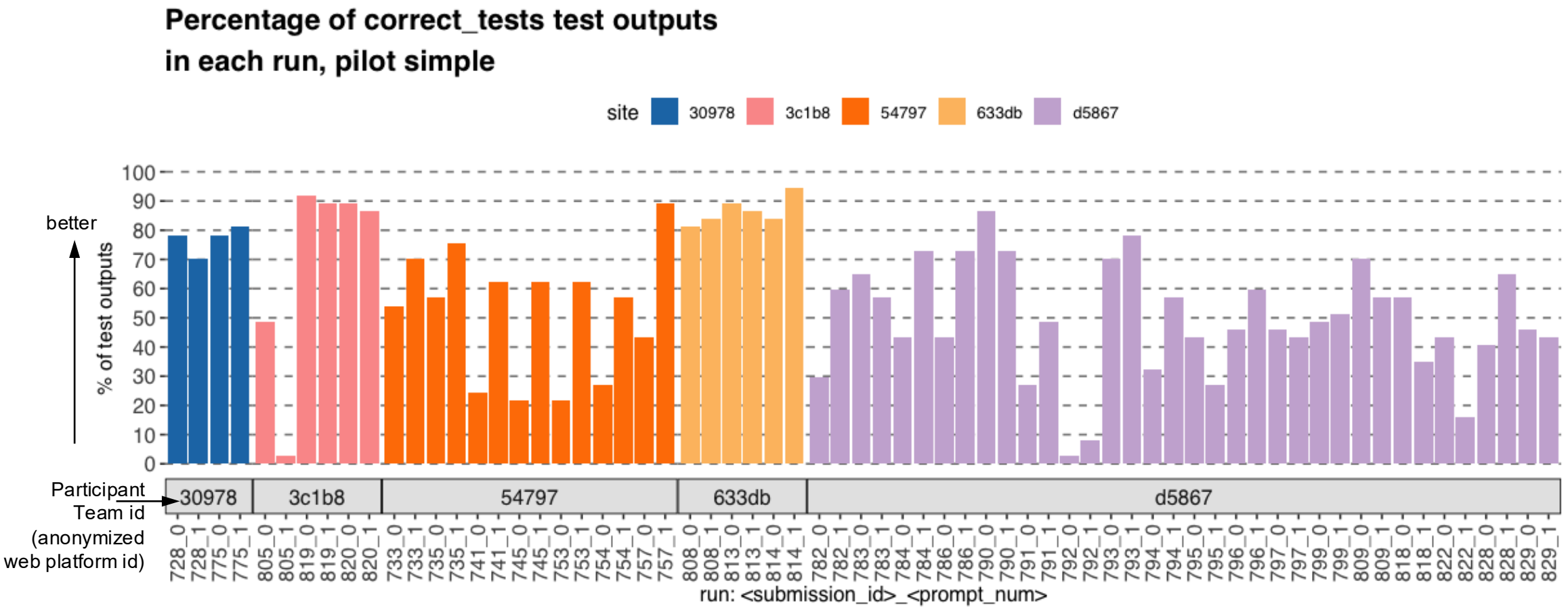
Round-2 Results: `correct_tests`



Round-2 Results: `finds_ci1_and_cit_errors`

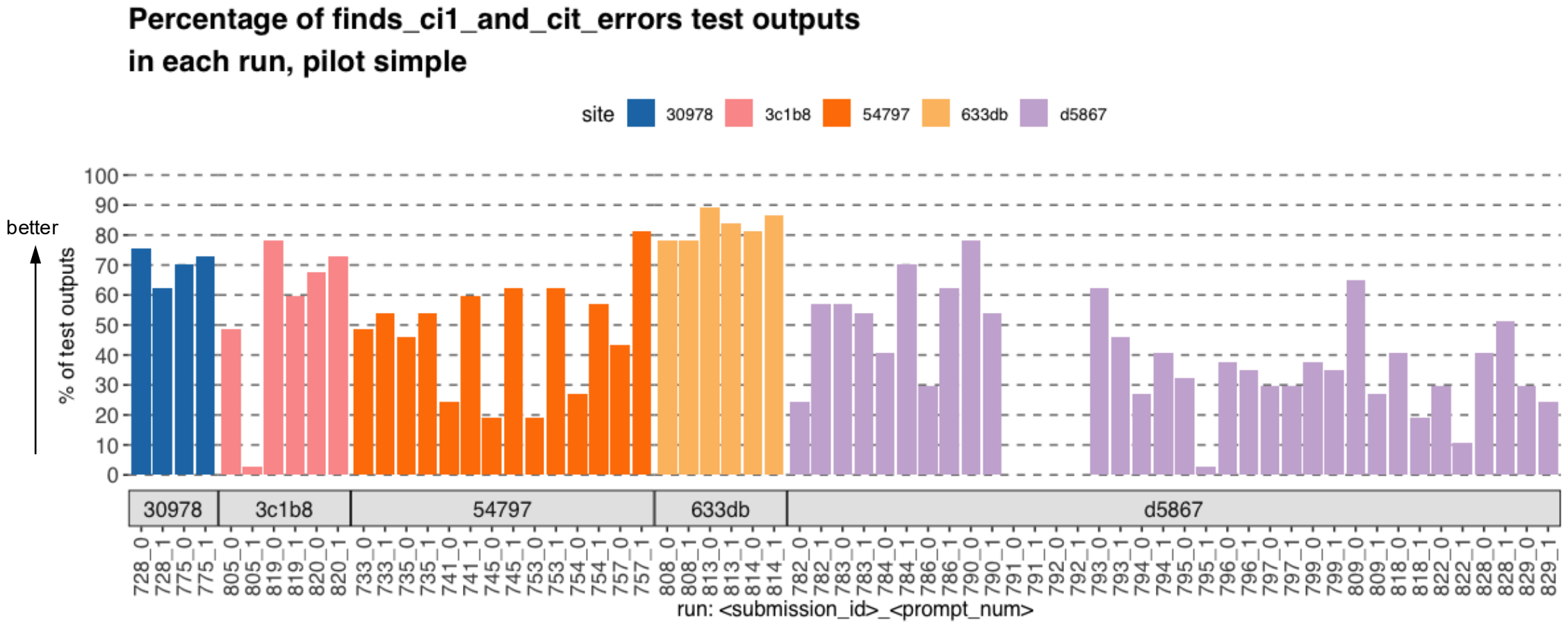


Round-2: `correct_tests` over simple trials



`correct_tests` (%): **% of test outputs that** find no errors on the correct program

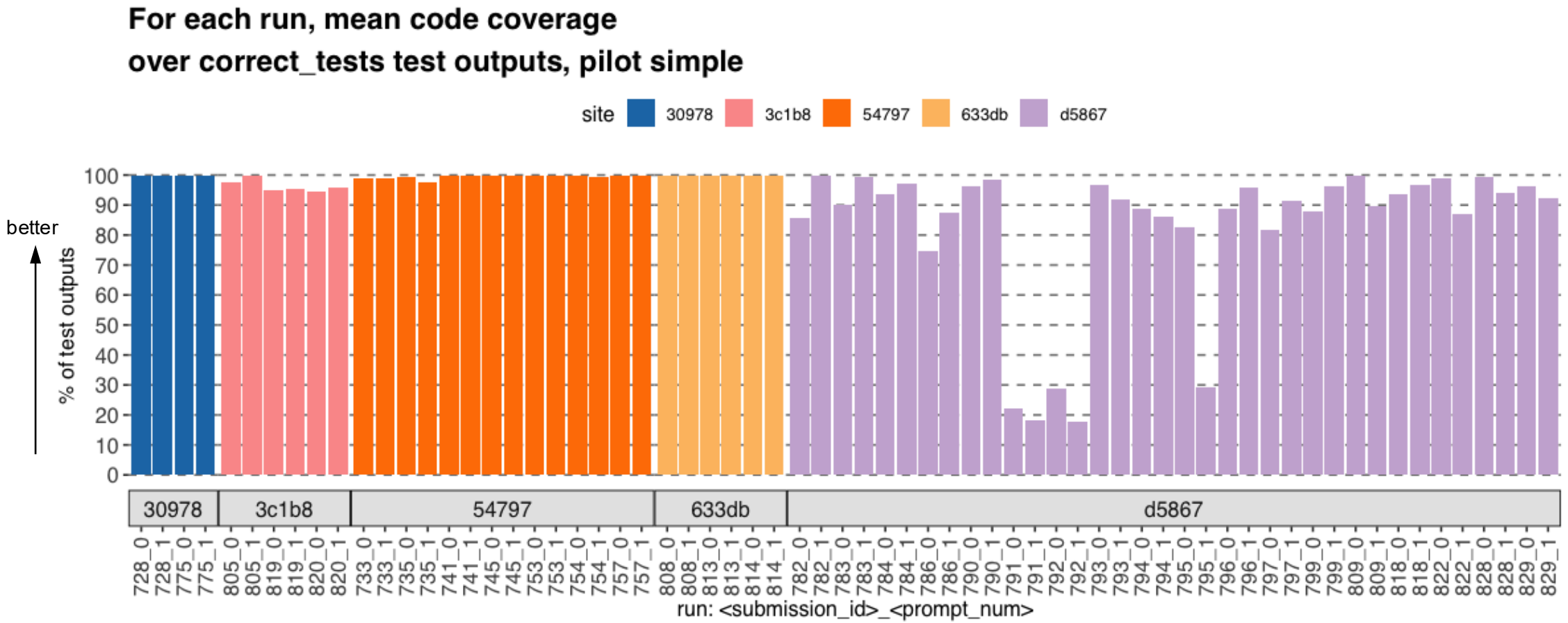
Round-2: `finds_ci1_and_cit_errors` over simple trials



`finds_ci1_and_cit_errors` (%): **% of test outputs** that are `correct_tests` and find an error in both incorrect programs

Round-2: cov result over simple trials

Median code coverage is 96.8% over simple trials

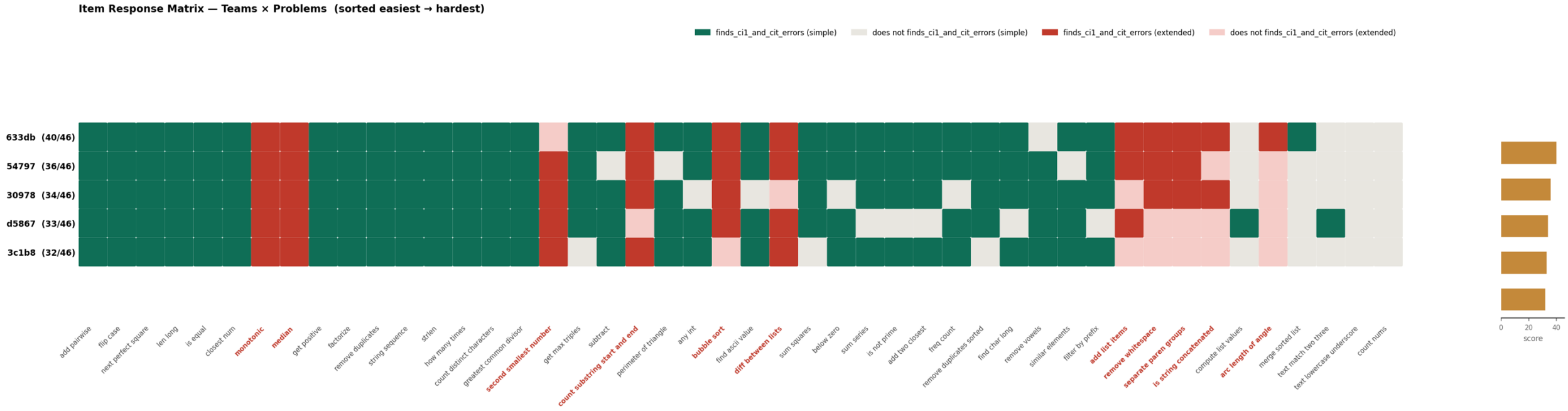


cov (%): **mean % of coverage** over all test outputs

Round-2 Summary by team, on correct_tests and finds_ci1_and_cit_errors over simple trials

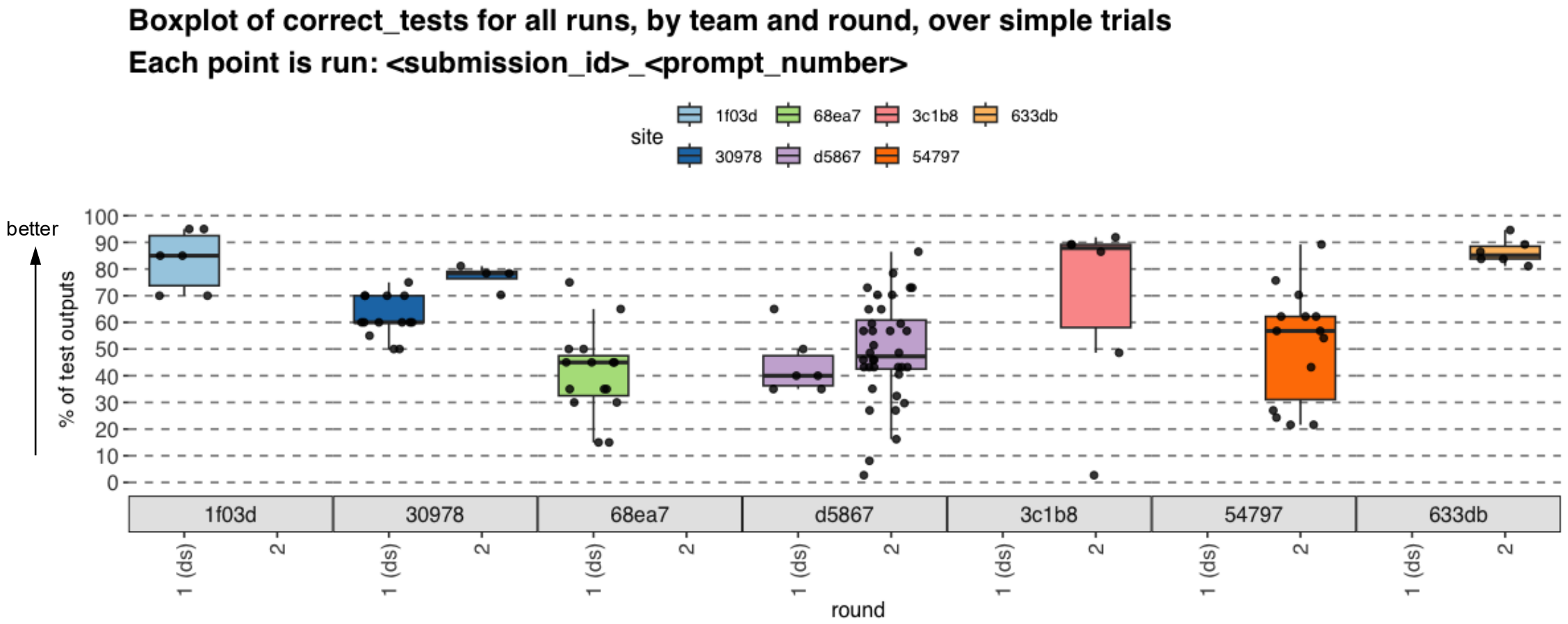
Team	30978		3c1b8		54797		633db		d5867	
Statistic	correct_tests	finds_ci1_and_cit_errors	correct_tests	finds_ci1_and_cit_errors	correct_tests	finds_ci1_and_cit_errors	correct_tests	finds_ci1_and_cit_errors	correct_tests	finds_ci1_and_cit_errors
mean	77.0	70.3	68	55	51.9	46.9	86.5	82.9	49	35.6
sd	4.7	5.83	35.9	27.7	21.4	18.6	4.83	4.41	19.2	20.7
median	78.4	71.6	87.8	63.6	56.8	51.4	85.2	82.4	47.2	35.1
75th percentile	79.1	73.7	89.2	71.6	62.2	58.8	88.5	85.8	60.8	52.1
90th percentile	80.3	74.9	90.6	75.7	74.1	66.2	91.9	87.8	73	62.2
maximum	**81.1**	**75.7**	**91.9**	**78.4**	**89.2**	**81.1**	**94.6**	**89.2**	**86.5**	**78.4**

Round-2 Results: Best submission per team, finds_ci1_and_cit_error score by trial

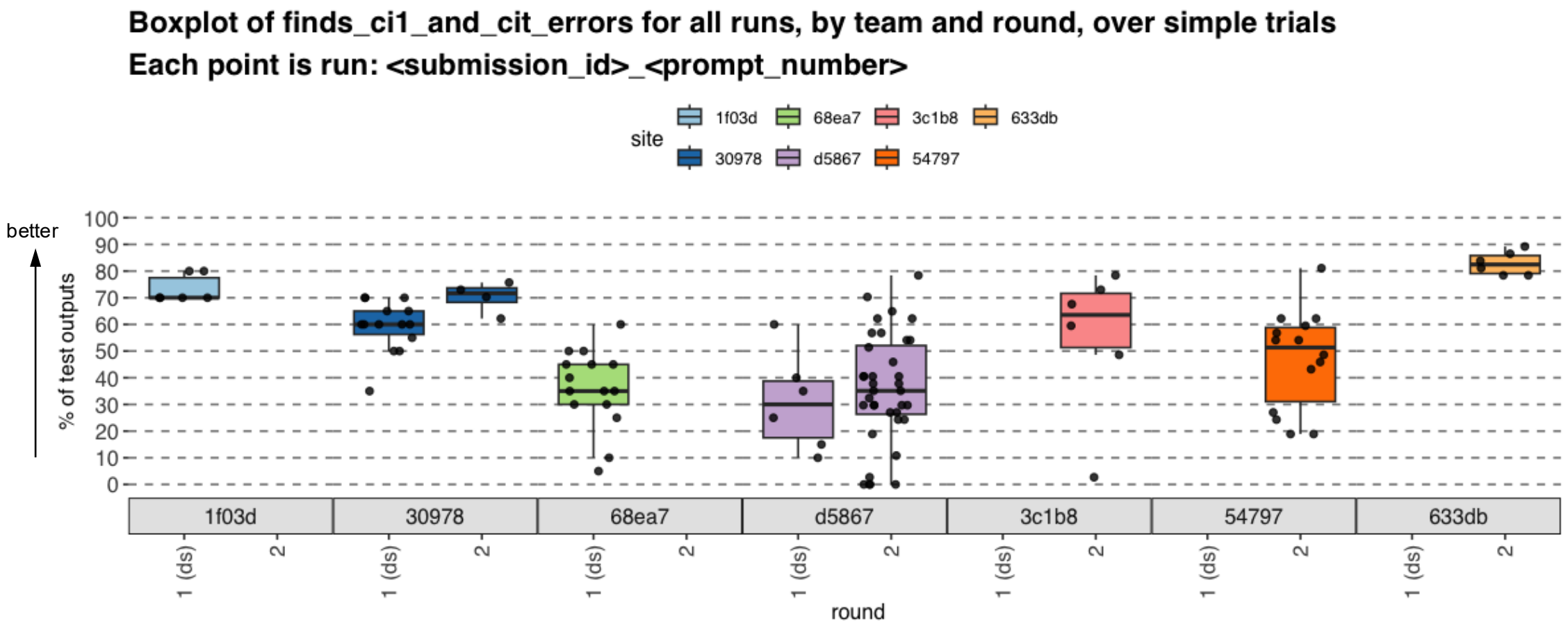


Comparing Round-1 (ds) and Round-2

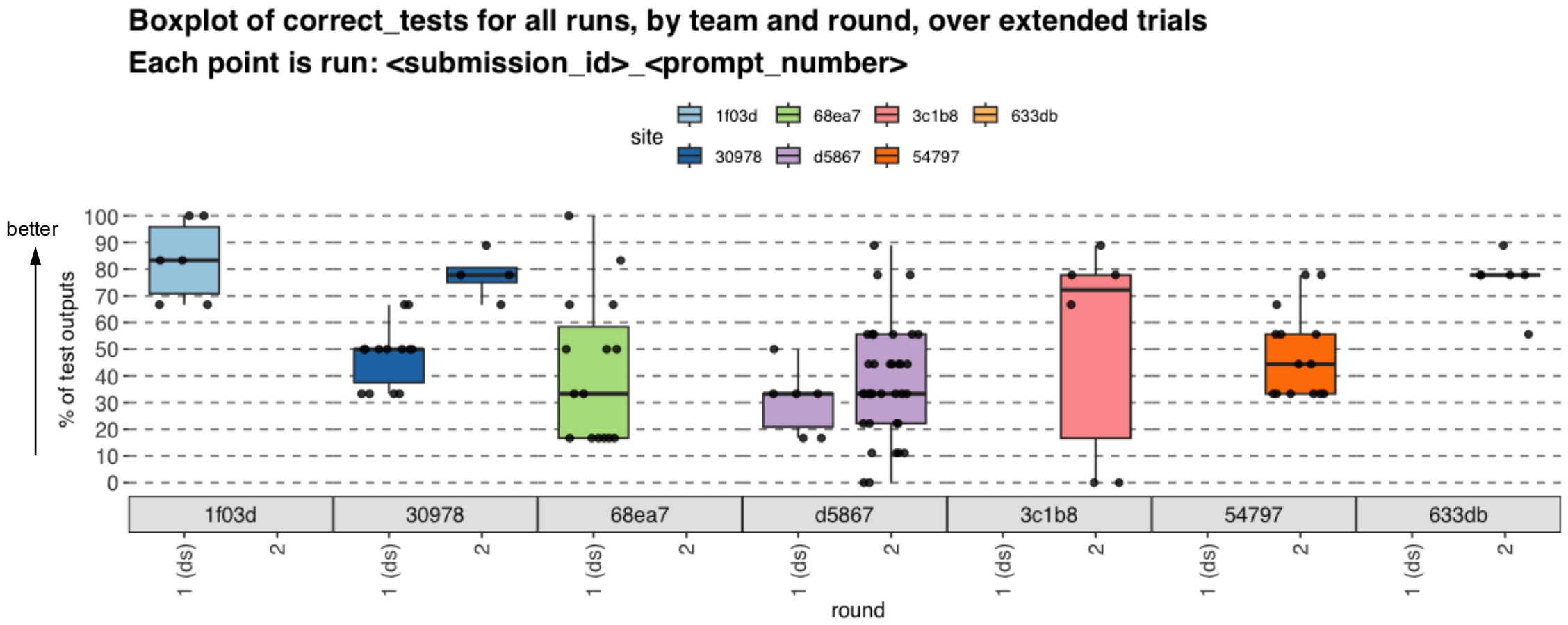
Comparing all teams over Round-1 (ds) and Round-2: correct_tests over simple trials



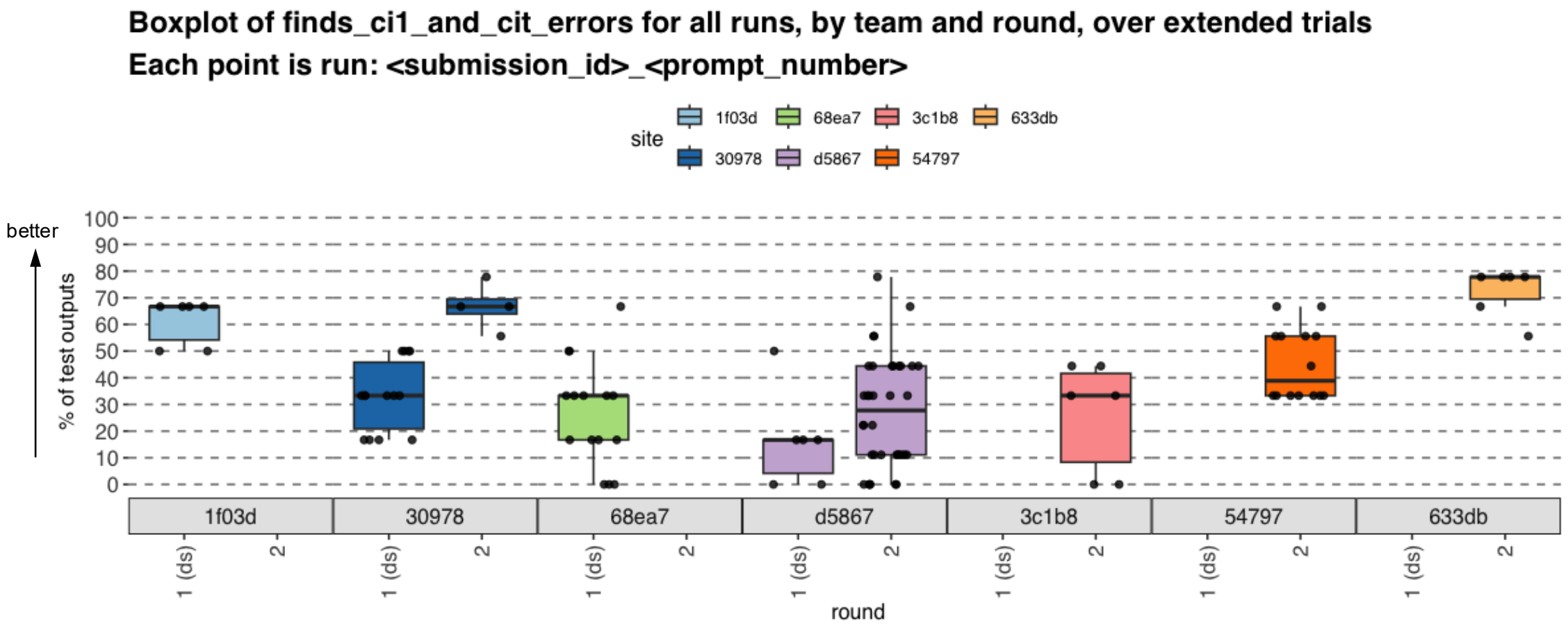
Comparing all teams over Round-1 (ds) and Round-2: finds_ci1_and_cit_errors over simple trials



Comparing all teams over Round-1 (ds), and Round-2: correct_tests over extended trials



Comparing all teams over Round-1 (ds) and Round-2: finds_ci1_and_cit_errors over extended trials



Summary

- Test correctness (metric correct_tests) is a valuable metric, as some test outputs are not correct even on simple problems
- Precise specifications are crucial for properly evaluating Super Intelligence systems

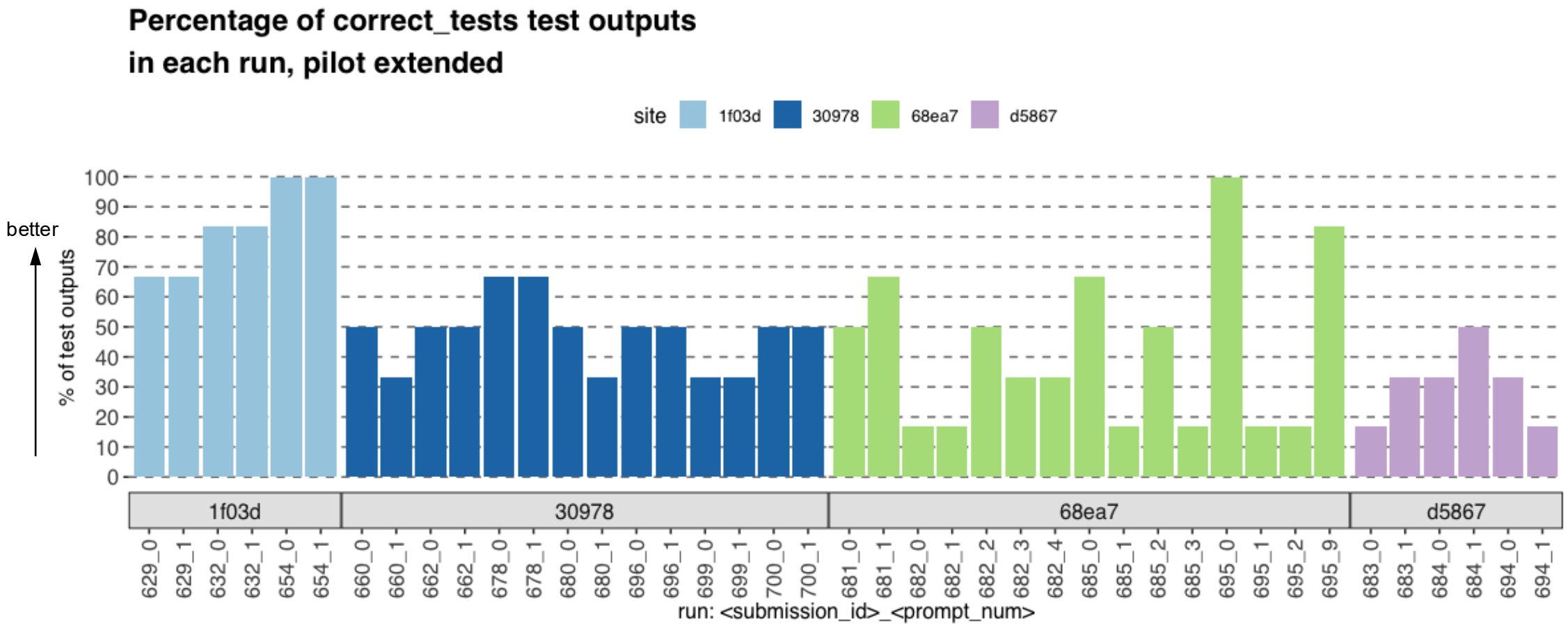
Questions?

Contact genai-poc@nist.gov

Appendix: Additional Round-1 (ds) Result Plots

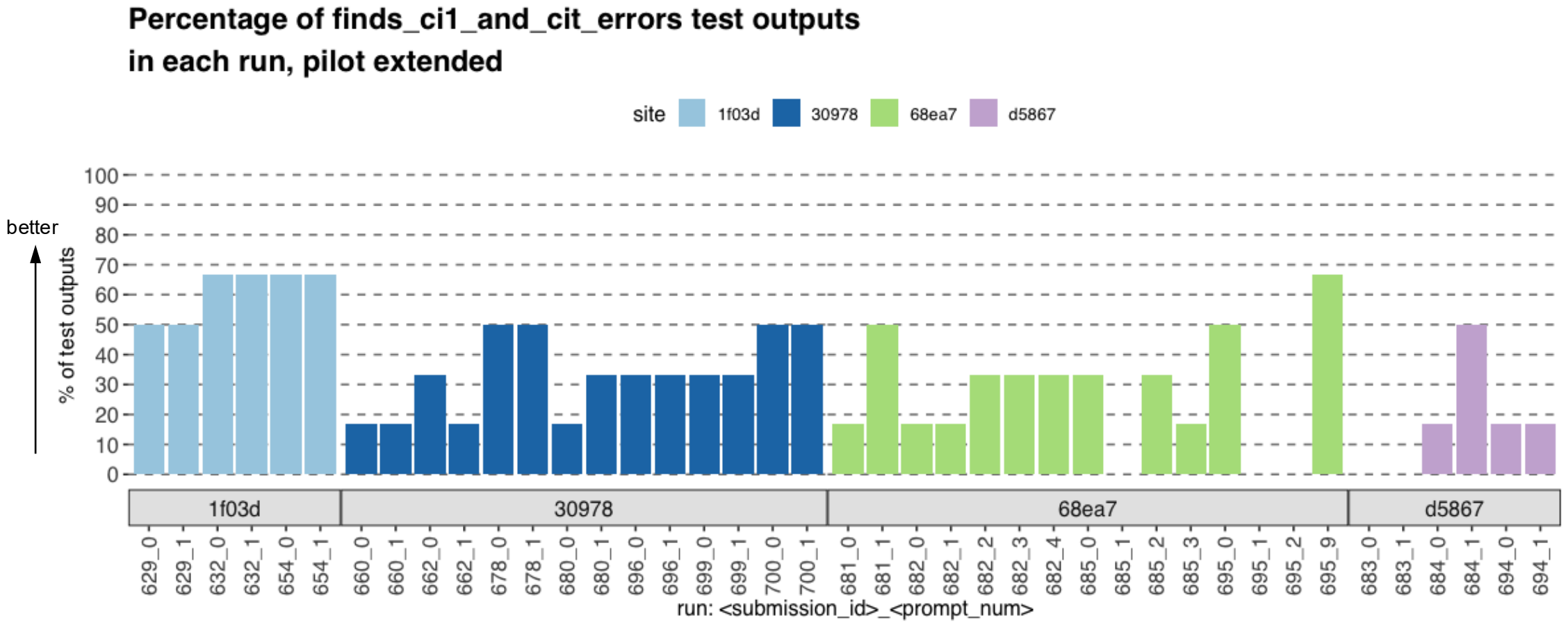
NIST INFORMATION TECHNOLOGY LABORATORY

Round-1 (ds): `correct_tests` over extended trials



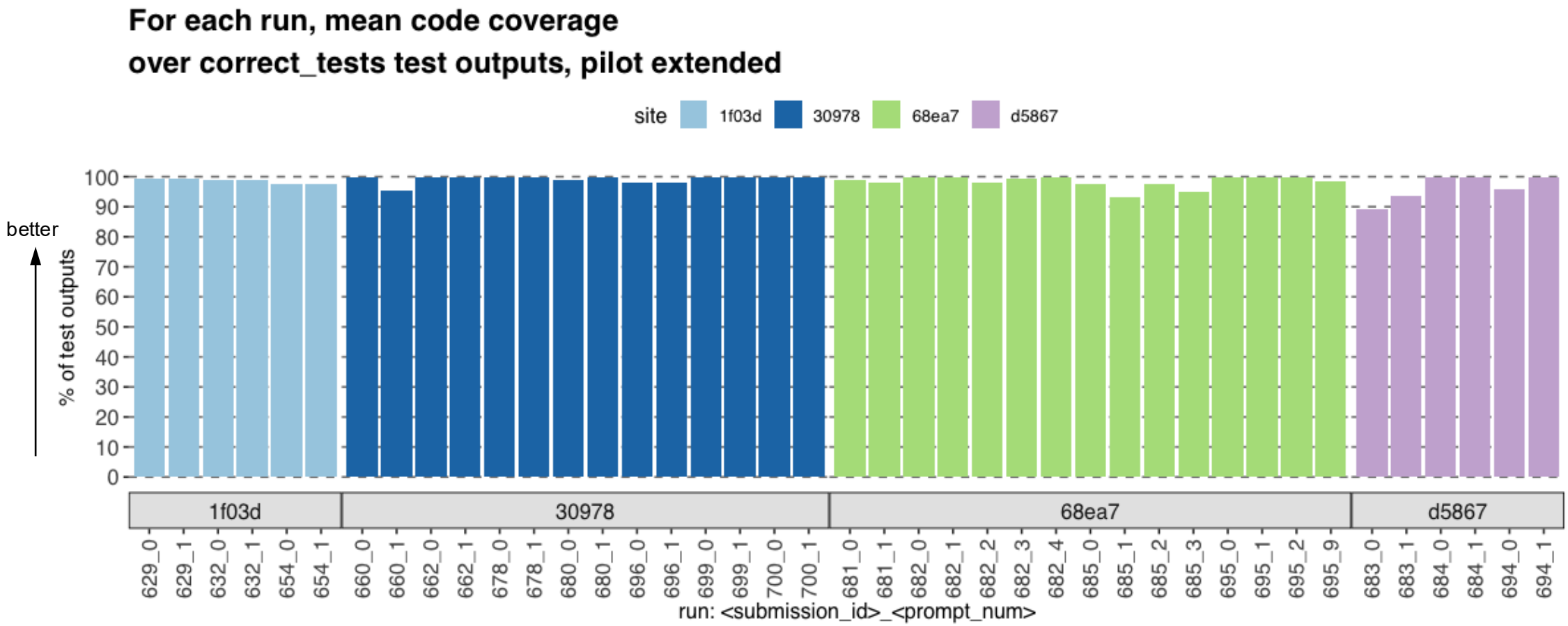
`correct_tests` (%): **% of test outputs that** find no errors on the correct program

Round-1 (ds): `finds_ci1_and_cit_errors` over extended trials



`finds_ci1_and_cit_errors` (%): **% of test outputs** that are `correct_tests` and find an error in both incorrect programs

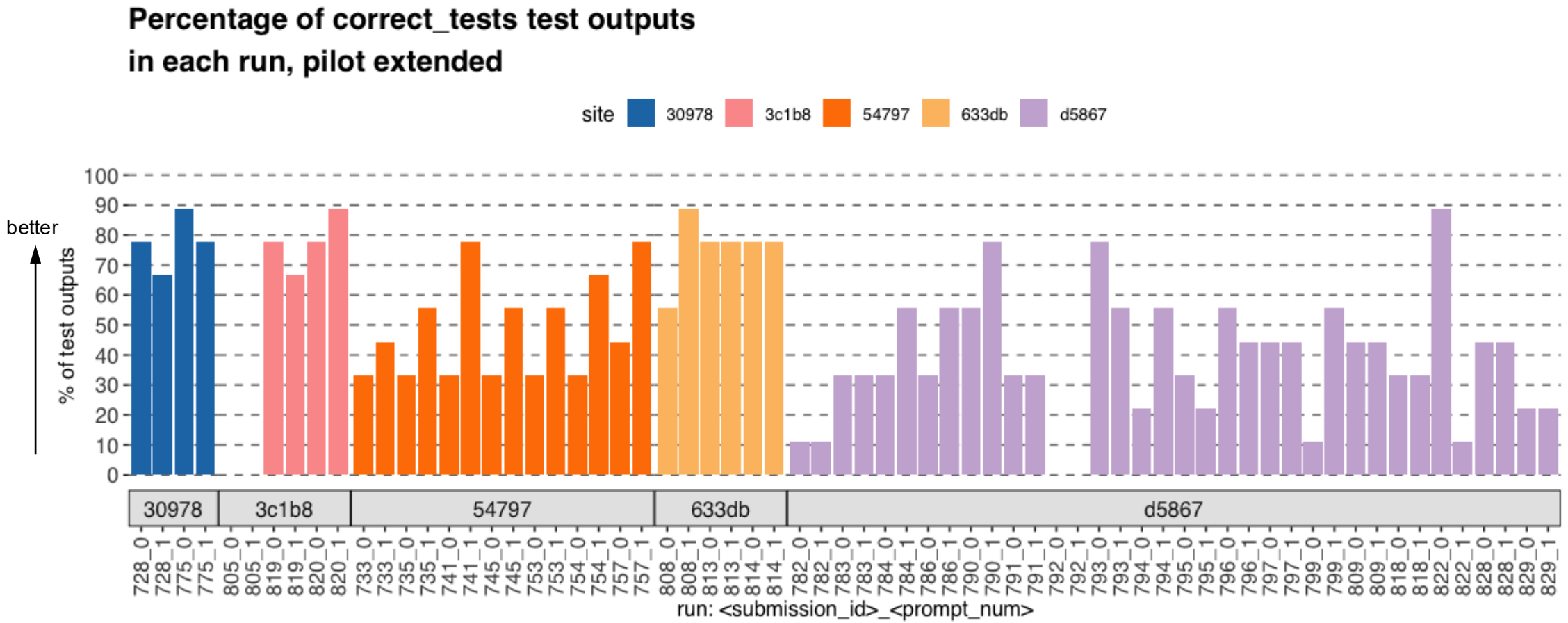
Round-1 (ds): cov over extended trials



cov (%): **mean % of coverage** over all test outputs

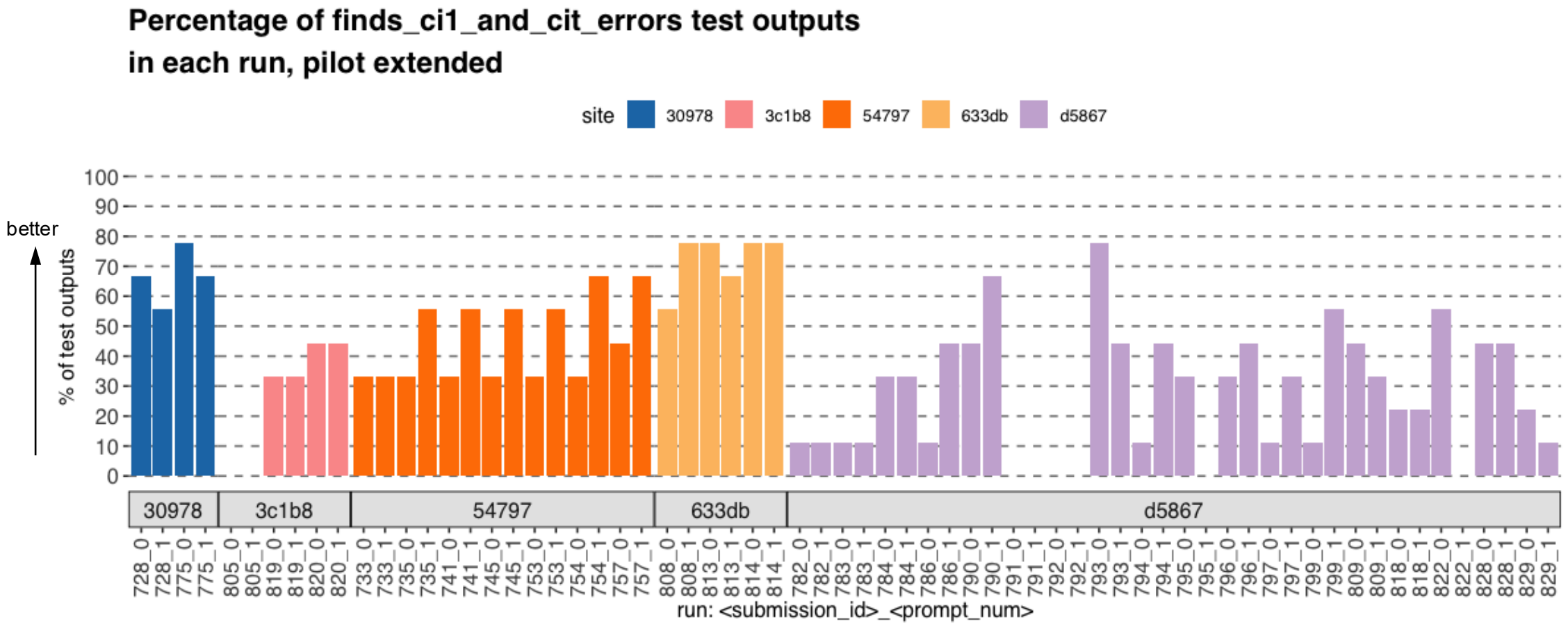
Appendix: Additional Round-2 Result Plots

Round-2: `correct_tests` over extended trials



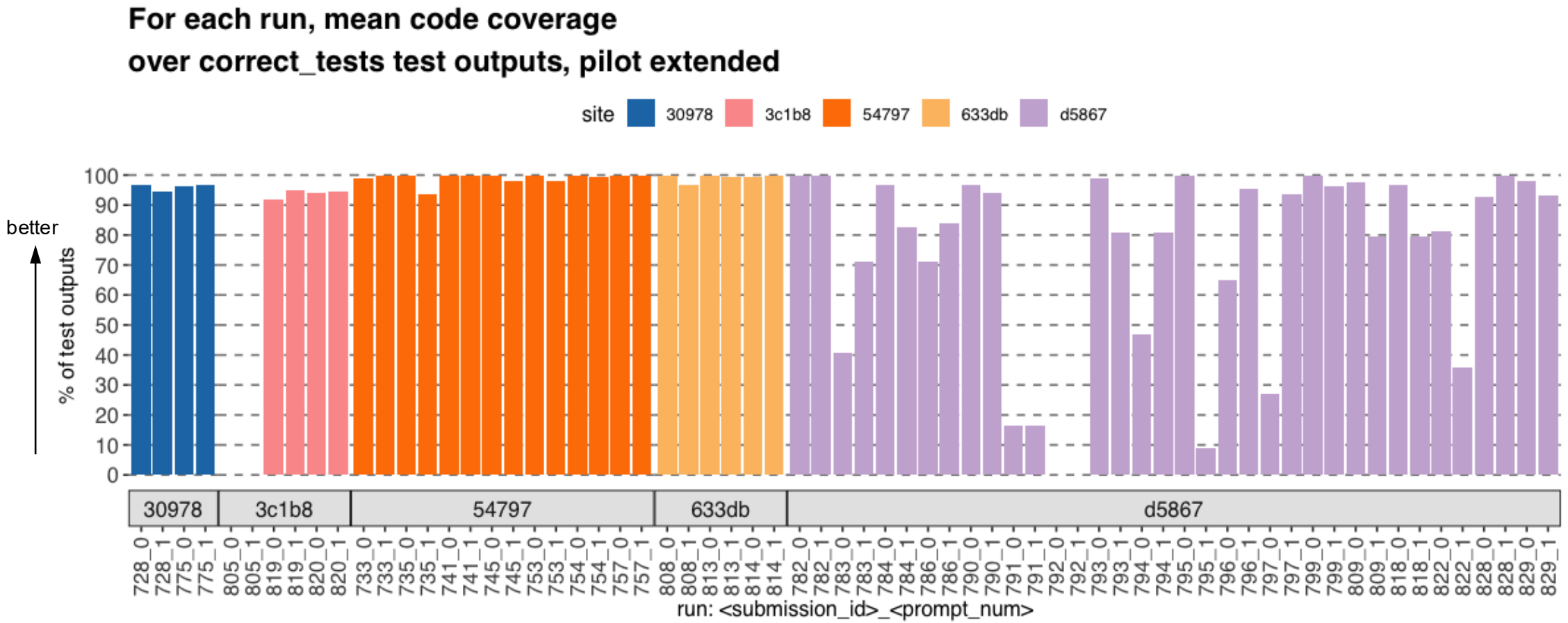
`correct_tests` (%): **% of test outputs that** find no errors on the correct program

Round-2 `finds_ci1_and_cit_errors` over extended trials



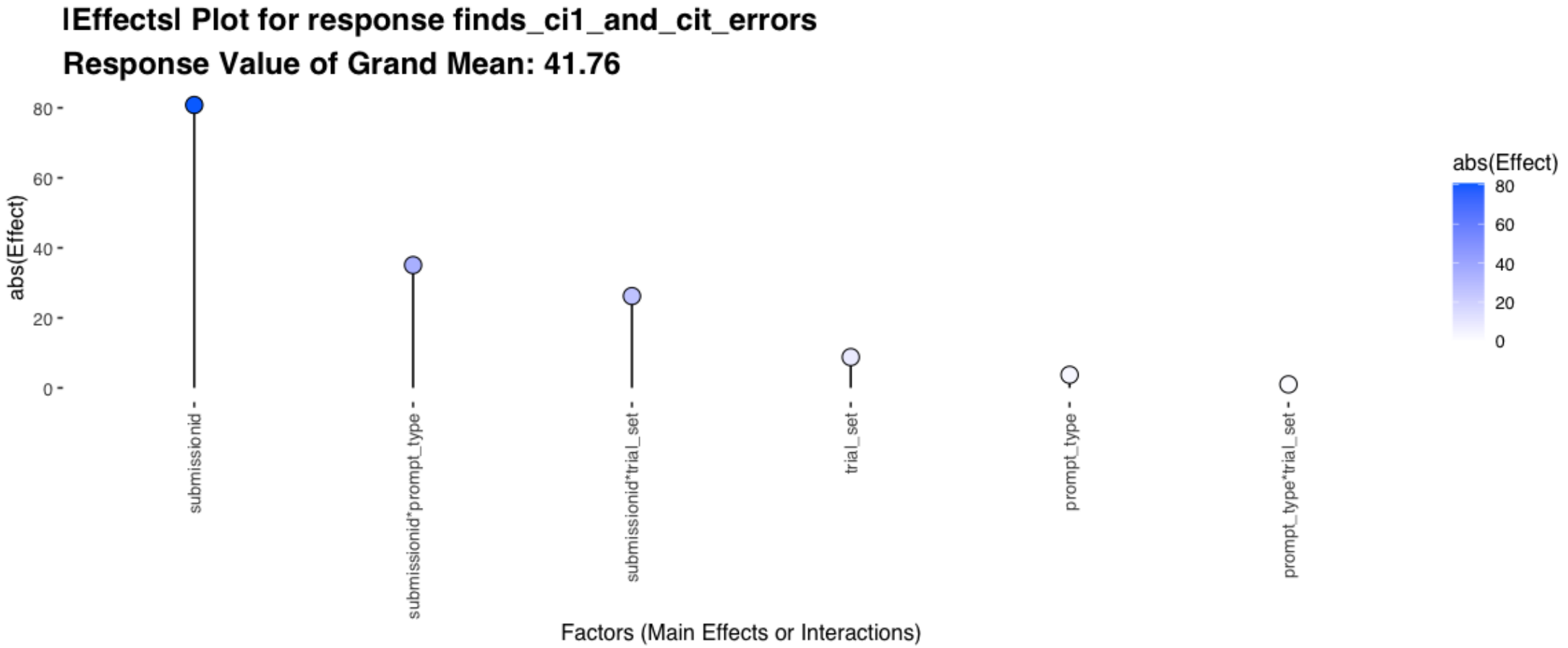
`finds_ci1_and_cit_errors` (%): **% of test outputs** that are `correct_tests` and find an error in both incorrect programs

Round-2 cov result: the median code coverage over extended trials

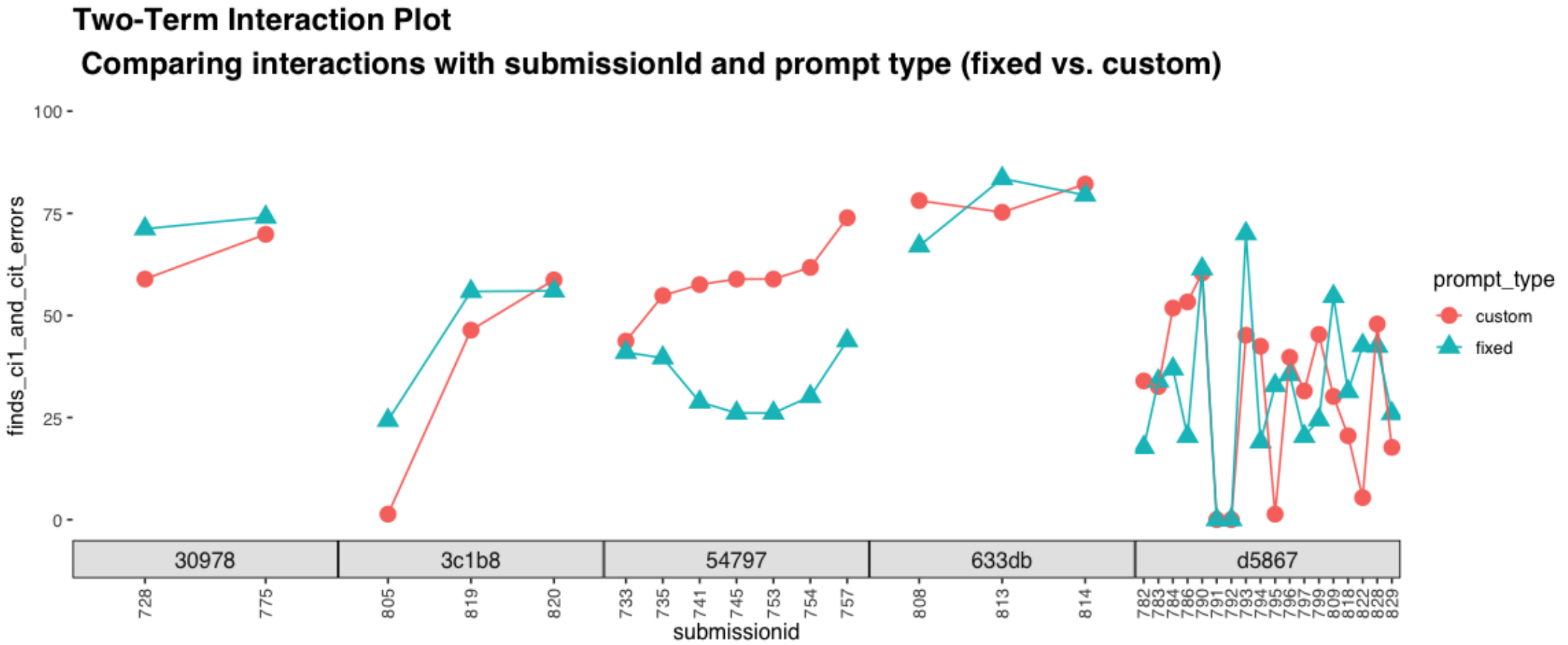


cov (%): **mean % of coverage** over all test outputs

Variability in submission scores exceeds influence of simple vs. extended (trial_set) or fixed vs. custom (prompt_type)



Variability in submission scores exceeds influence of simple vs. extended (trial_set) or fixed vs. custom (prompt_type)



Variability in submission scores exceeds influence of simple vs. extended (trial_set) or fixed vs. custom (prompt_type)

