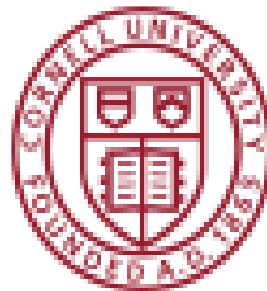


# Automated Assertion Generation and Regression Testing for Machine Learning Notebooks

**Elaine Yao**<sup>\*</sup>, Vedant Nimje<sup>†</sup>, Varun Viswanath<sup>‡</sup>, Saikat Dutta<sup>\*</sup>

<sup>\*</sup> Cornell University (USA), <sup>†</sup> VJTI (India), <sup>‡</sup> DJSCE (India)

**ASE 2026**



# Computational notebooks

## Step 1: Load and clean the dataset

Each row of `train.csv` is one scanned digit: `label` holds the digit a human transcribed, and the 784 `pixel*` columns hold the 28x28 grayscale image, with values in `[0, 255]`.

Text

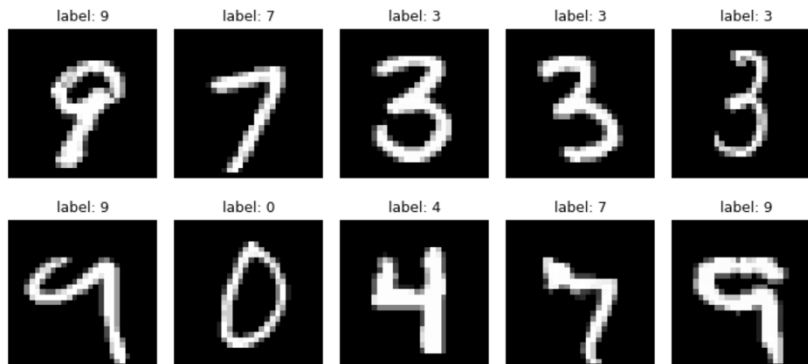
```
[6]: df = pd.read_csv("../input/train.csv")
```

```
[7]: df = df.drop(columns=["id", "source", "extra_note", "flag"])
```

...

```
[10]: plot_digit_samples(df)  
plt.show()
```

Code



Cell  
outputs

# Notebooks allow out-of-order execution

```
[1]: df = df.drop(columns=["id", "source", "extra_note", "flag"])
```

```
[2]: cutoff = 0.1 * df.max(numeric_only=True).max()  
df = remove_pixels_below_cutoff(df, cutoff)
```

```
[3]: df.mean().mean()
```

33.38

```
[2]: df = df.drop(columns=["id", "source", "extra_note", "flag"])
```

```
[1]: cutoff = 0.1 * df.max(numeric_only=True).max()  
df = remove_pixels_below_cutoff(df, cutoff)
```

```
[3]: df.mean().mean()
```

25.31

# Effect of out-of-order execution: Stale states

Old

```
[ ]: import pandas as pd  
    df = pd.read_csv('train_sample.csv')
```

```
[ ]: df.shape
```

# Effect of out-of-order execution: Stale states



```
[1]: import pandas as pd  
df = pd.read_csv('train_sample.csv')
```

```
[2]: df.shape
```

(1000, 785)

# Effect of out-of-order execution: Stale states

Old

```
[1]: import pandas as pd  
df = pd.read_csv('train_sample.csv')
```

```
[2]: df.shape
```

(1000, 785)

New

```
[1]: import pandas as pd  
df = pd.read_csv('train.csv')
```

```
[2]: df.shape
```

(1000, 785)



Forgot to rerun the cell

# Effect of out-of-order execution: Stale states

Old

```
[1]: import pandas as pd  
df = pd.read_csv('train_sample.csv')
```

```
[2]: df.shape
```

(1000, 785)



New

```
[3]: import pandas as pd  
df = pd.read_csv('train.csv')
```

```
[4]: df.shape
```

(~~60000~~, 785)



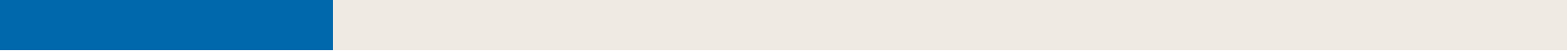
Silent regressions!

# Notebooks get messy easily

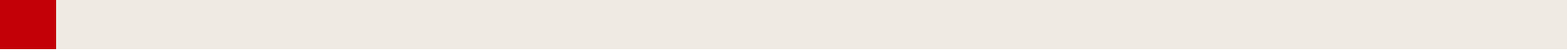
**1.4M** notebooks analyzed<sup>[1]</sup>



**24%** ran end-to-end without any execution errors<sup>[1]</sup>



**4%** reproduced the original results<sup>[1]</sup>



Hard to **reuse** and  
**reproduce** results with

[1] Pimentel, João Felipe, et al. "A large-scale study about quality and reproducibility of jupyter notebooks." *2019 IEEE/ACM 16th international conference on mining software repositories (MSR)*. IEEE, 2019.

# Current (Ad-hoc) testing practices

## Print statements

```
[12]: # Sanity checks before training
print("X_train shape:", X_train.shape)
print("X_test shape:", X_test.shape)
print("pixel range:", X_train.min(), "-", X_train.max())
print("label range:", y_train.min(), "-", y_train.max())
print("any NaN in X_train:", np.isnan(X_train).any())
print("train label counts:")
print(pd.Series(y_train).value_counts().sort_index())
```

```
X_train shape: (1500, 28, 28, 1)
X_test shape: (500, 28, 28, 1)
pixel range: 0.0 - 0.00392156862745098
label range: 0 - 9
any NaN in X_train: False
train label counts:
0    162
1    150
2    145
3    165
4    157
5    126
6    160
7    166
8    131
9    138
Name: count, dtype: int64
```

Cluttered.  
Needs manual inspection

## Python `assert`

```
assert X_train.shape == (60000, 785)
assert X_test.shape == (10000, 785)
assert X_train.min() >= 0.0 \
       and X_train.max() <= 1.0
assert y_train.min() >= 0.0 \
       and y_train.max() <= 1.0
assert not np.isnan(X_train).any()
```

✗ integrated with pytest.  
Disrupt the workflow.  
General assertions

How can we test notebooks more systematically?

# Our work: NBTest

## ① Regression testing framework

**NBTest-lib** Python library

4 generic  
assertion APIs

7 data-related  
assertion APIs

11  
in total

# How NBTest-lib assertions catch regressions

```
[2] : df = df.drop(columns=["id", "source", "extra_note", "flag"])
```

```
[1] : cutoff = 0.1 * df.max(numeric_only=True).max()  
df = remove_pixels_below_cutoff(df, cutoff)
```

```
[3] : nbtest.assert_df_mean(df, 33.38, test_id='1')
```

---

AssertionError

Traceback (most recent call last)

Cell In[14], line 1

```
----> 1 nbtest.assert_df_mean(df, 33.38, test_id='1')
```

# Many assertions can clutter the notebook

```
df = pd.read_csv("../input/train.csv")
```

```
df = df.drop(columns=["id", "source", "extra_note", "flag"])
```

```
cutoff = 0.1 * df.max(numeric_only=True).max()  
df = remove_pixels_below_cutoff(df, cutoff)
```

```
nbtest.assert_df_mean(df, 33.38, test_id='1')  
nbtest.assert_equal(df.shape, (2000, 785), test_id='2')  
nbtest.assert_equal(df.columns[0], 'label', test_id='3')  
nbtest.assert_column_names(df, ['label', 'pixel0', 'pixel783'], test_id='4')  
nbtest.assert_column_types(df, {'label': 'int64',  
                                'pixel0': 'int64', 'pixel783': 'int64'},  
                             test_id='5')  
nbtest.assert_nanmean(df, 33.38266687898089, test_id='6')  
nbtest.assert_nanvar(df, 6227.552031664833, test_id='7')  
nbtest.assert_equal(df.max(numeric_only=True).max(), 255, test_id='8')  
nbtest.assert_equal((df == 0).all().sum(), 170, test_id='9')  
nbtest.assert_allclose(df['pixel350'].mean(), 84.039, test_id='10')
```

# Our work: NBTest

## ① Regression testing framework

**NBTest-lib** Python library

4 generic  
assertion APIs

7 data-related  
assertion APIs

11  
in total

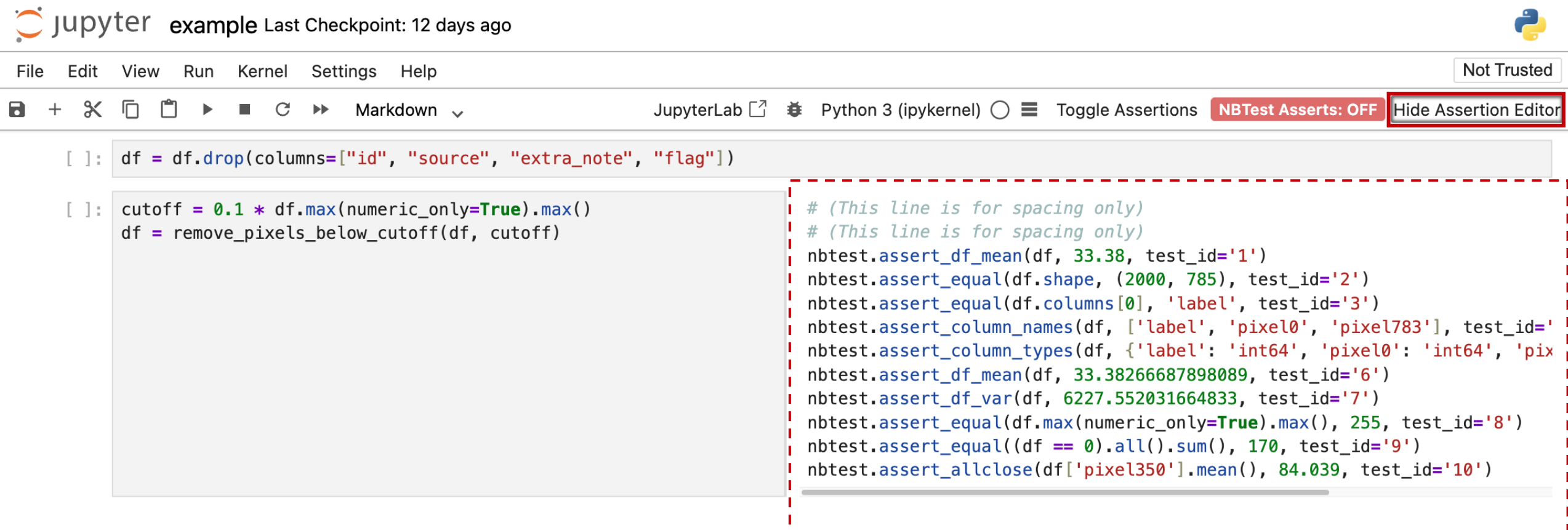
**NBTest-lab-extension** JupyterLab plugin

| Assertion visibility

| Enable / disable  
assertion execution

# NBTest-lab-extension

## Assertion panel shown



The screenshot shows a JupyterLab interface. At the top, the Jupyter logo and 'example' are on the left, and 'Last Checkpoint: 12 days ago' is on the right. Below this is a menu bar with 'File', 'Edit', 'View', 'Run', 'Kernel', 'Settings', and 'Help'. To the right of the menu bar is a 'Not Trusted' badge. Below the menu bar is a toolbar with icons for saving, adding, deleting, copying, pasting, running, and a 'Markdown' dropdown. To the right of the toolbar is 'JupyterLab' with an external link icon, a Python 3 (ipykernel) badge, a 'Toggle Assertions' button, and a red 'NBTest Asserts: OFF' button. To the right of the red button is a 'Hide Assertion Editor' button, which is highlighted with a red dashed border. Below the toolbar is a code cell with two lines of Python code:

```
[ ]: df = df.drop(columns=["id", "source", "extra_note", "flag"])

[ ]: cutoff = 0.1 * df.max(numeric_only=True).max()
df = remove_pixels_below_cutoff(df, cutoff)
```

To the right of the code cell is an assertion panel, also highlighted with a red dashed border. It contains the following nbtest assertions:

```
# (This line is for spacing only)
# (This line is for spacing only)
nbtest.assert_df_mean(df, 33.38, test_id='1')
nbtest.assert_equal(df.shape, (2000, 785), test_id='2')
nbtest.assert_equal(df.columns[0], 'label', test_id='3')
nbtest.assert_column_names(df, ['label', 'pixel0', 'pixel783'], test_id='4')
nbtest.assert_column_types(df, {'label': 'int64', 'pixel0': 'int64', 'pixel783': 'int64'}, test_id='5')
nbtest.assert_df_mean(df, 33.38266687898089, test_id='6')
nbtest.assert_df_var(df, 6227.552031664833, test_id='7')
nbtest.assert_equal(df.max(numeric_only=True).max(), 255, test_id='8')
nbtest.assert_equal((df == 0).all().sum(), 170, test_id='9')
nbtest.assert_allclose(df['pixel350'].mean(), 84.039, test_id='10')
```

Cell-scoped assertions

# NBTest-lab-extension

## Assertion panel hidden

 jupyter example Last Checkpoint: 12 days ago 

File Edit View Run Kernel Settings Help Not Trusted

         Markdown ▾

JupyterLab   Python 3 (ipykernel)   Toggle Assertions **NBTest Asserts: OFF** Show Assertion Editor

```
[ ]: df = df.drop(columns=["id", "source", "extra_note", "flag"])

[ ]: cutoff = 0.1 * df.max(numeric_only=True).max()
df = remove_pixels_below_cutoff(df, cutoff)
```

# NBTest-lab-extension

## Enable/Disable assertion execution during development

 jupyter example Last Checkpoint: 12 days ago 

File Edit View Run Kernel Settings Help Trusted

 +         Markdown  JupyterLab  Python 3 (ipykernel)   Toggle Assertions **NBTest Asserts: ON** Hide Assertion Editor

```
[9]: df = df.drop(columns=["id", "source", "extra_note", "flag"])

[8]: cutoff = 0.1 * df.max(numeric_only=True).max()
df = remove_pixels_below_cutoff(df, cutoff)

# (This line is for spacing only)
# (This line is for spacing only)
nbtest.assert_df_mean(df, 33.38, test_id='1')
nbtest.assert_equal(df.shape, (2000, 785), test_id='2')
nbtest.assert_equal(df.columns[0], 'label', test_id='3')
nbtest.assert_column_names(df, ['label', 'pixel0', 'pixel783'], test_id='4')
nbtest.assert_column_types(df, {'label': 'int64', 'pixel0': 'int64', 'pixel783': 'int64'}, test_id='5')
nbtest.assert_df_mean(df, 33.38266687898089, test_id='6')
nbtest.assert_df_var(df, 6227.552031664833, test_id='7')
nbtest.assert_equal(df.max(numeric_only=True).max(), 255, test_id='8')
nbtest.assert_equal((df == 0).all().sum(), 170, test_id='9')
nbtest.assert_allclose(df['pixel350'].mean(), 84.039, test_id='10')
```

-----  
AssertionError Traceback (most recent call last)  
Cell In[8], line 3  
1 cutoff = 0.1 \* df.max(numeric\_only=True).max()  
2 df = remove\_pixels\_below\_cutoff(df, cutoff)  
----> 3 nbtest.assert\_df\_mean(df, 33.38, test\_id='1')

# NBTest-lab-extension

## Enable/Disable assertion execution during development

jupyter example Last Checkpoint: 12 days ago



File Edit View Run Kernel Settings Help

Trusted

Code

JupyterLab

Python 3 (ipykernel)

Toggle Assertions

NBTest Asserts: OFF

Hide Assertion Editor

```
[8]: df = df.drop(columns=["id", "source", "extra_note", "flag"])
```

```
[7]: cutoff = 0.1 * df.max(numeric_only=True).max()
df = remove_pixels_below_cutoff(df, cutoff)
```

```
# (This line is for spacing only)
# (This line is for spacing only)
nbtest.assert_df_mean(df, 33.38, test_id='1')
nbtest.assert_equal(df.shape, (2000, 785), test_id='2')
nbtest.assert_equal(df.columns[0], 'label', test_id='3')
nbtest.assert_column_names(df, ['label', 'pixel0', 'pixel783'], test_id='4')
nbtest.assert_column_types(df, {'label': 'int64', 'pixel0': 'int64', 'pi
nbtest.assert_df_mean(df, 33.38266687898089, test_id='6')
nbtest.assert_df_var(df, 6227.552031664833, test_id='7')
nbtest.assert_equal(df.max(numeric_only=True).max(), 255, test_id='8')
nbtest.assert_equal((df == 0).all().sum(), 170, test_id='9')
nbtest.assert_allclose(df['pixel350'].mean(), 84.039, test_id='10')
```

# Our work: NBTest

## ① Regression testing framework

**NBTest-lib** Python library

4 generic  
assertion APIs

7 data-related  
assertion APIs

11  
in total

**NBTest-lab-extension** JupyterLab plugin

| Assertion visibility

| Enable / disable  
assertion execution

## ② Automated assertion generation

**NBTestGen**

notebook



notebook with assertions

# How NBTest components work together

## 1 Generate assertions with

**NBTest-lib**

Manual

or

**NBTestGen**

Automated

Notebook with assertions

...

```
nbtest.assert_column_types(...)
nbtest.assert_shape(...)
```

...

## 2 Put them to work

**NBTest-lab-extension**

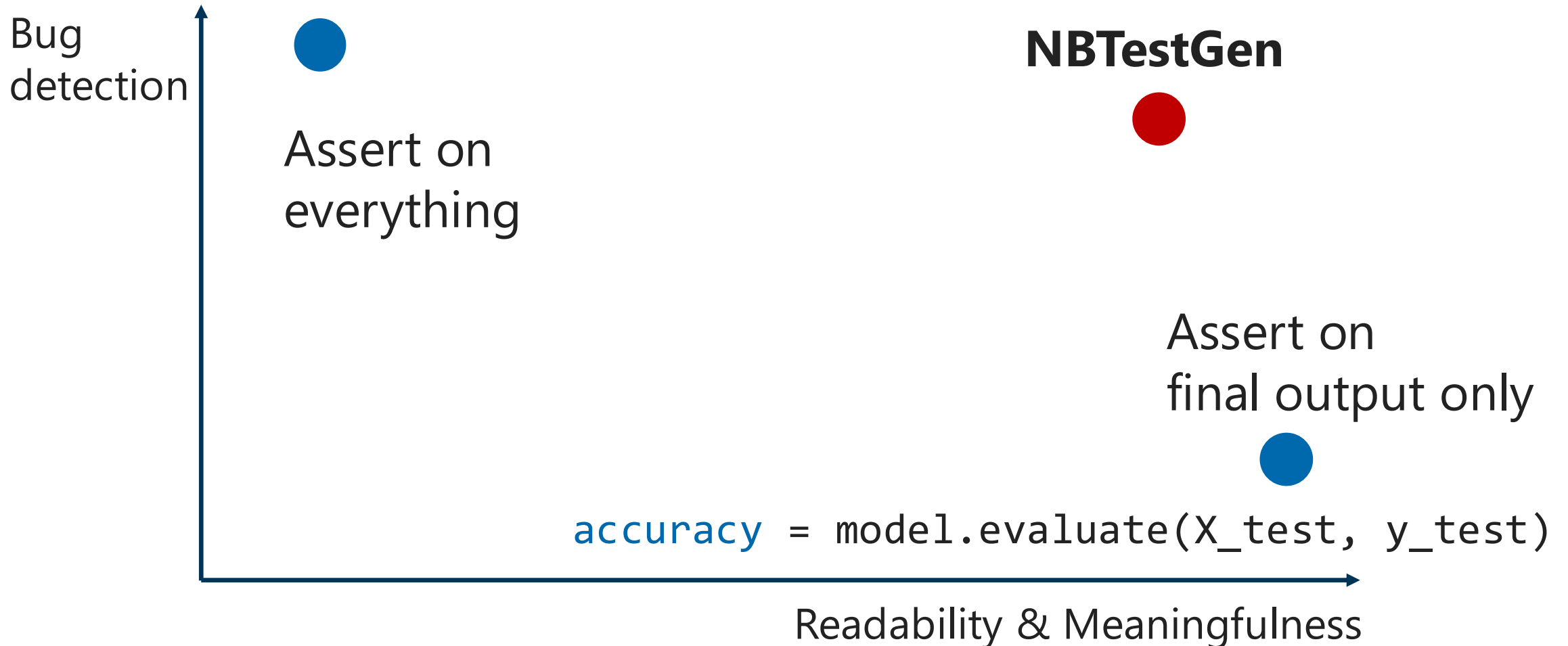
While developing

**pytest** in CI

```
% pytest -v --nbtest ./digits.ipynb
=== test session starts ===
./digits.ipynb ::0 PASSED [ 12%]
./digits.ipynb ::1 PASSED [ 25%]
./digits.ipynb ::2 PASSED [ 37%]
...
./digits.ipynb ::6 PASSED [ 87%]
./digits.ipynb ::7 FAILED [100%]
```

# NBTestGen – Challenge 1

Which variables and computations to assert?



# NBTestGen – Challenge 1

Which variables and computations to assert?

## Assertions written by developers<sup>[1]</sup>

```
assert y_valid.shape == (1132,)
assert np.isclose(stdev_norm, 1.0, atol=1e-16)
assert np.all(np.unique(X_train['smoke'].values) ==
               np.array([0, 1]))
```

```
assert isinstance(model, RandomForestClassifier)
assert self.encoder[0].weight.size() ==
       self.vgg16[2].weight.size()
```

```
assert np.isclose(accuracy, 0.967)
```

## Integrity checks

Dataset

Model Architecture

Model Performance

# Generate Dataset Integrity assertions

```
(train_df, test_df) = train_test_split(df)
```

```
...
```

```
Generated by NBTestGen
```

```
...
```

```
nbtest.assert_column_types(train_df, ['int64', ...])
```

```
nbtest.assert_shape(train_df, (30240, 785))
```

```
nbtest.assert_nanmean(train_df, 545.83)
```

① Tracked APIs called

`train_test_split`

② Targeted variables

`train_df` `test_df`

③ Dataset integrity assertions

# Generate Model Architecture integrity assertions

```
model = Sequential([...,  
                    MaxPooling2D(...), ...])  
...
```

Generated by NBTestGen

```
...  
nbtest.assert_equal(  
    json.loads(model.to_json(),  
    {"class_name": "Sequential",  
     "config": {..., "layers": [...,  
    {"class_name": "Maxpooling2D"}...]}...}))
```

- 1 Tracked APIs called  
`tensorflow.keras.Sequential`
- 2 Targeted variables  
`model`
- 3 Model architecture integrity assertions

# Generate Model Performance Integrity assertions

```
(, accuracy) = model.evaluate(X_test, y_test)
```

```
'''
```

```
Generated by NBTestGen
```

```
'''
```

```
nbtest.assert_allclose(accuracy, 0.964, atol=0.014)
```

- 1 Tracked APIs called

evaluate

- 2 Targeted variables

accuracy

- 3 Model performance integrity assertions

# NBTestGen – Challenge 2

## Nondeterminism

### 1 Is this variable non-deterministic?

(Num. of runs)  $n = 30$

Same value in every run **Deterministic**

(0.842, 0.842, 0.842, ...)

```
assert_equal(acc, 0.842)
```

Value differs **Non-deterministic**

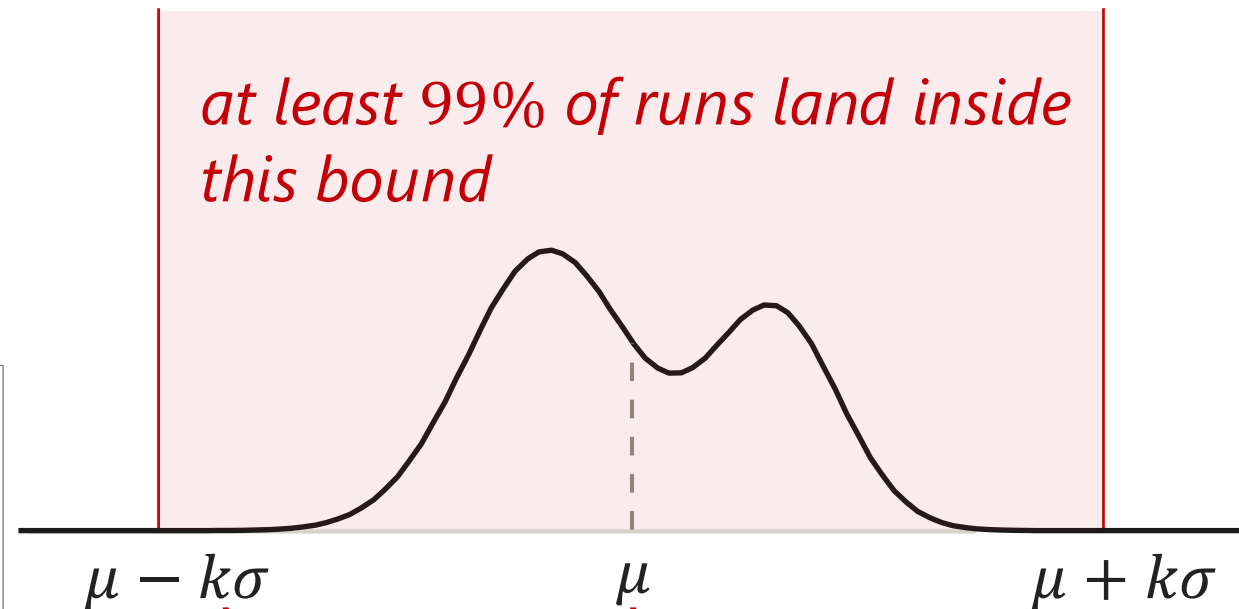
(0.914, 0.908, 0.921, ...)

```
assert_allclose(acc,  $\mu$ ,  $k\sigma$ )
```

### 2 Chebyshev's Inequality

(Confidence level)  $c = 0.99$

*at least 99% of runs land inside this bound*



Any distribution

# Evaluation

**RQ1&2:** How does NBTestGen's passing rate and mutation score change with different confidence thresholds?

**RQ3:** How effective are NBTestGen's assertions in detecting real-world regressions?

**RQ4:** How does NBTestGen compare with nbval?

NBTestGen generates 50.3% more assertions than nbval, with a 13.9% higher passing rate.

# Evaluation setup

## DATASET

**585** Kaggle  
notebooks

## CONFIGURATIONS

|                  | $n$ (Num. of runs) | $c$ (Confidence level) |
|------------------|--------------------|------------------------|
| <b>RQ1</b>       | 30                 | 0.5, 0.7, 0.99, 0.999  |
| <b>RQ2 – RQ4</b> | 30                 | 0.99                   |

## MUTATION TESTING

### Data-based mutations

4

Add Outliers<sup>[1]</sup>

Repeat Data<sup>[2]</sup>

Add Nulls<sup>[1]</sup>

Modify labels<sup>[2]</sup>

### Code-based mutations

4

Remove Zero Grad<sup>[3]</sup>

Modify Hyper-parameters<sup>[1]</sup>

Remove Hyper-parameters<sup>[1]</sup>

Remove Layers<sup>[2]</sup>

[1] Zimmermann et al., *Common Errors in Machine Learning Projects: A Second Look* (Koli Calling, 2023)

[2] Ma et al., *DeepMutation: Mutation Testing of Deep Learning Systems* (ISSRE, 2018)

[3] Liblit et al., *Shifting Left for Early Detection of Machine-Learning Bugs* (International Symposium on Formal Methods, 2023)

# RQ1&2: Number and Passing rate of NBTestGen's assertions

21,542 assertions generated. Avg. 36.82 per notebook.

535 / 585 notebooks contain at least one assertion.

21,526 assertions with 100% passing rate

16 assertions with (50 – 100)% passing rate

10

6

Overly strict bounds

Highly irregular  
distributions

# Confidence Level vs Passing Rate & Mutation Scores

| Confidence level | Passing rate | Mutation score | FPR | FNR |
|------------------|--------------|----------------|-----|-----|
| 0.5              |              |                |     |     |
| 0.7              |              |                |     |     |
| 0.99             |              |                |     |     |
| 0.999            |              |                |     |     |

False Positive Ratios (**FPR**) - % failed assertions not caused by mutations

False Negative Ratios (**FNR**) - % passed assertions that fail to catch mutations

# Confidence Level vs Passing Rate & Mutation Scores

| Confidence level | Passing rate  | Mutation score  | FPR  | FNR  |
|------------------|--------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------|
| <b>0.5</b>       | 96.93%                                                                                           | 0.82                                                                                               | 0.1342                                                                                  | 0.18                                                                                    |
| <b>0.7</b>       | 98.58%                                                                                           | 0.78                                                                                               | 0.0712                                                                                  | 0.22                                                                                    |
| <b>0.99</b>      | 100%                                                                                             | 0.72                                                                                               | 0.0052                                                                                  | 0.28                                                                                    |
| <b>0.999</b>     | 100%                                                                                             | 0.71                                                                                               | 0.0042                                                                                  | 0.29                                                                                    |

 higher the better     lower the better

← tighter bound, lower passing rate, higher mutation score

False Positive Ratios (**FPR**) - % failed assertions not caused by mutations

False Negative Ratios (**FNR**) - % passed assertions that fail to catch mutations

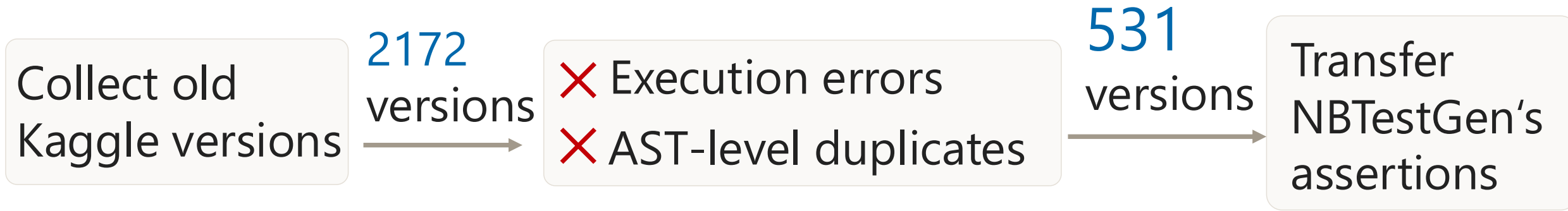
# Confidence Level vs Passing Rate & Mutation Scores

| Default | Confidence level | Passing rate  | Mutation score  | FPR  | FNR  |
|---------|------------------|--------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------|
|         | <b>0.5</b>       | 96.93%                                                                                           | 0.82                                                                                               | 0.1342                                                                                  | 0.18                                                                                    |
|         | <b>0.7</b>       | 98.58%                                                                                           | 0.78                                                                                               | 0.0712                                                                                  | 0.22                                                                                    |
|         | <b>0.99</b>      | 100%                                                                                             | 0.72                                                                                               | 0.0052                                                                                  | 0.28                                                                                    |
|         | <b>0.999</b>     | 100%                                                                                             | 0.71                                                                                               | 0.0042                                                                                  | 0.29                                                                                    |

← tighter bound, lower passing rate, higher mutation score

# RQ3: Detecting real-world regressions with NBTestGen's assertions

## SETUP



## RESULTS

370/531 (69.68%) versions killed.

➡ Effective in detecting silent regressions in notebooks.

# User Study

17 users found NBTest **intuitive** (Rating 4.3/5) and **useful** (Rating 4.24/5) in checking ML properties.

“ I often ran into inexplicable errors or couldn't tell which cell execution had overwritten a value. NBTest makes it easy to reliably validate the correctness of my data.

“ extend to a vscode plugin? (I would love to use it!)

“ broaden with more assertion types

# Real-world Adoption



SHAP

☆ Star 25.7k

**Integrated** NBTest into its CI pipeline



**fg-data-profiling**

☆ Star 13.7k

“ ...great to see how it performs in practice with our notebooks, ..., tag it as part of roadmap so the community is aware that this is something we're keen to integrate”



**bionemo-recipes**

☆ Star 850

“ ... happy to keep up with the latest notebook test infra ... how this work with one of our notebooks? ...”

# Summary

- **NBTest-lib:** First regression testing framework for ML notebooks
- **NBTestGen:** First automated assertion generation for ML notebooks
- NBTest assertions are effective in catching regressions
- NBTest has been adopted in real world



[elainey@cs.cornell.edu](mailto:elainey@cs.cornell.edu)



<https://github.com/seal-research/NBTest>