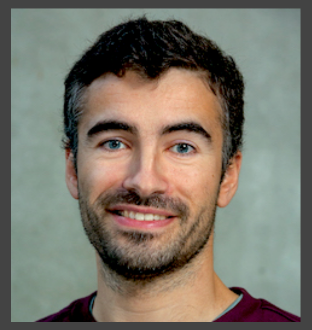
ML Testing

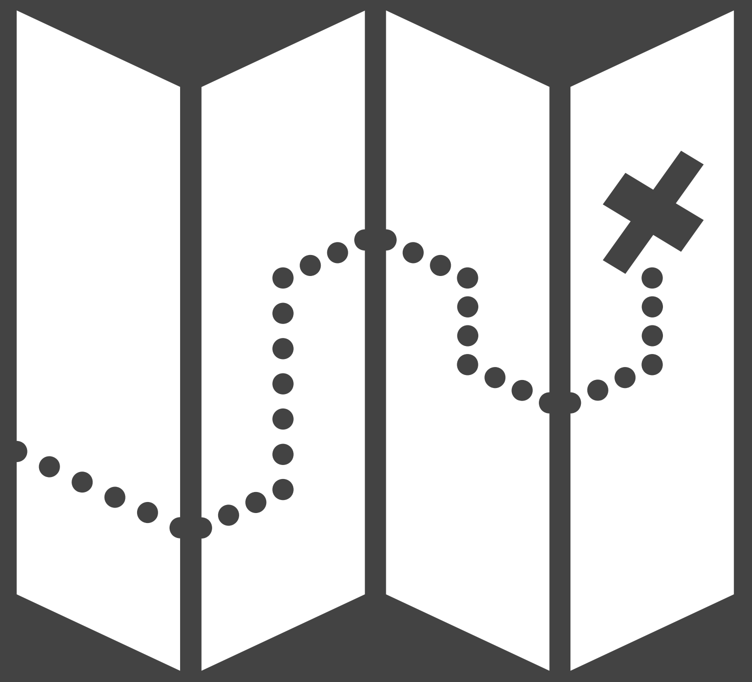
Release Engineering for Machine Learning Applications (REMLA, CS4295)



Luís Cruz
L.Cruz@tudelft.nl

Sebastian Proksch
S.Proksch@tudelft.nl

Outline



- ML Testing Landscape
- What to test?
- How to test?
- Mutamorphic Testing
- Tools and Resources

ML Testing Landscape

Machine Learning Testing: Survey, Landscapes and Horizons

Jie M. Zhang, Mark Harman, Lei Ma, Yang Liu

Abstract—This paper provides a comprehensive survey of Machine Learning Testing (ML testing) research. It covers 138 papers on testing properties (e.g., correctness, robustness, and fairness), testing components (e.g., the data, learning program, and framework), testing workflow (e.g., test generation and test evaluation), and application scenarios (e.g., autonomous driving, machine translation). The paper also analyses trends concerning datasets, research trends, and research focus, concluding with research challenges and promising research directions in ML testing.

Index Terms—machine learning, software testing, deep neural network,

https://arxiv.org/abs/1906.10742

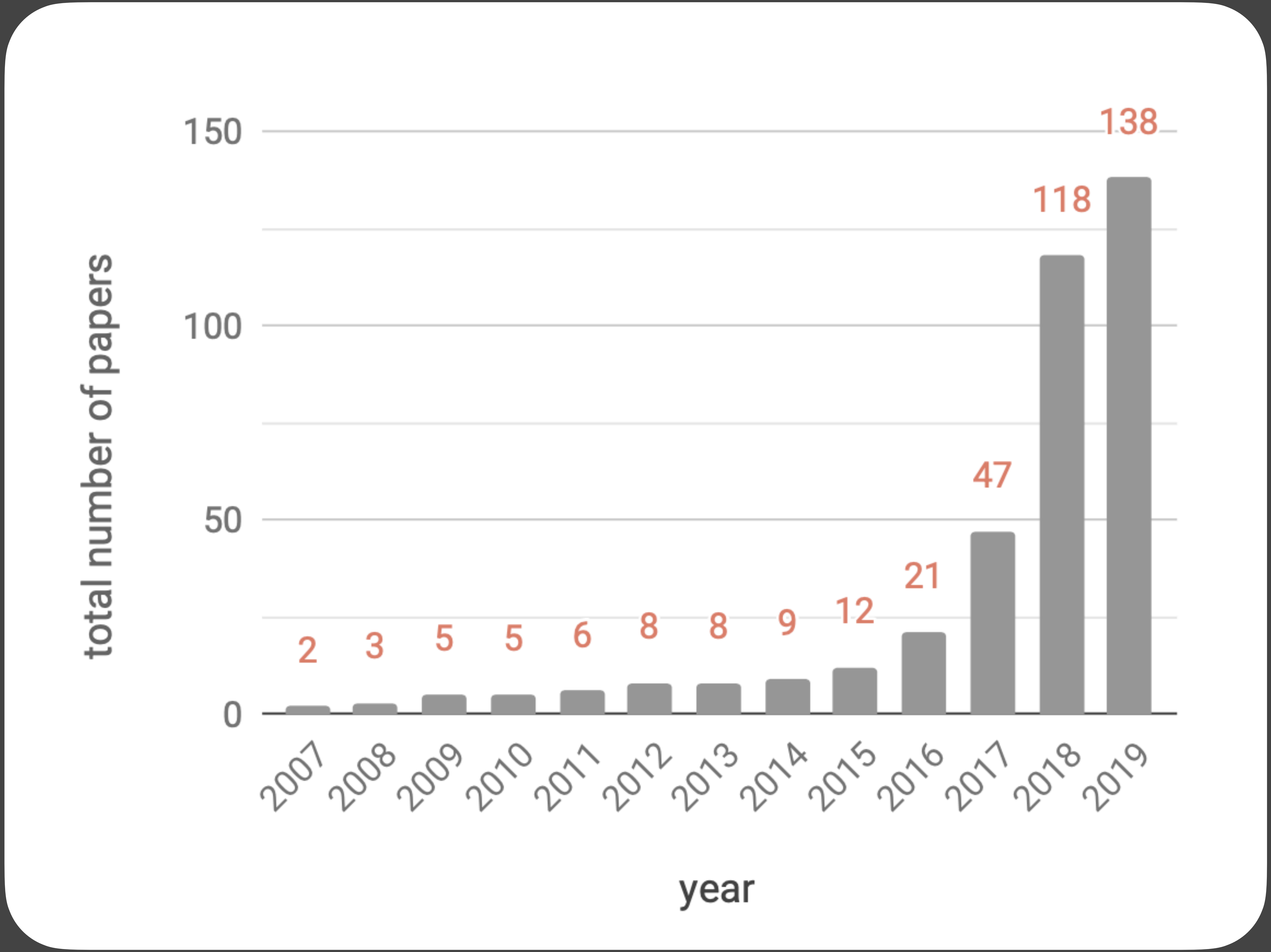
ML Testing
Zhang et al. (2019)

- Definition 1 – **ML Bug:** any imperfection in a machine learning item that causes a discordance between the existing and the required conditions.

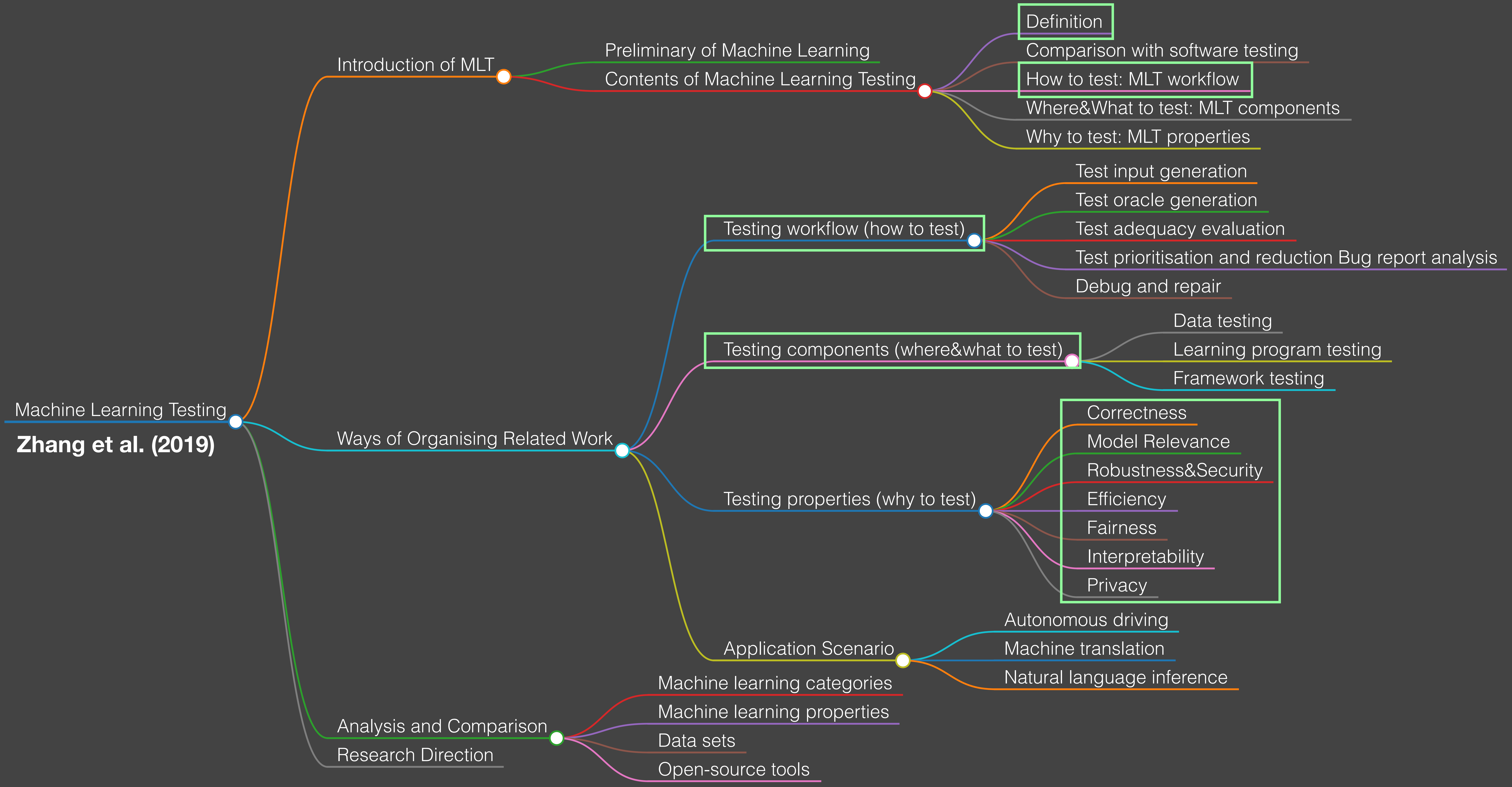
- Definition 2 – **ML Testing:** any activities designed to reveal machine learning bugs.

ML Testing Publications during 2007–2019

(⚠️ Cumulative plot)



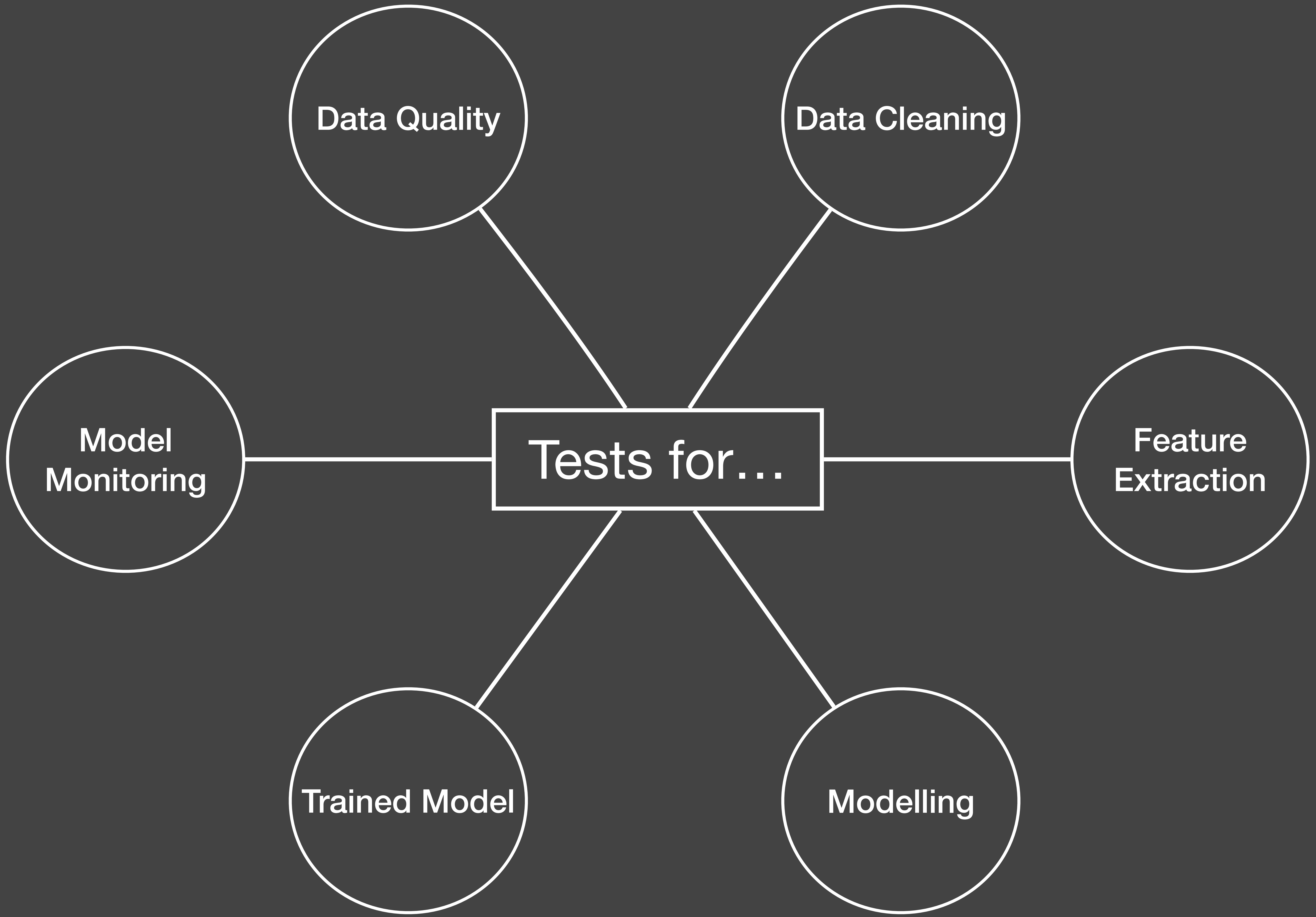
Zhang et al. (2019)



Machine Learning Testing
Zhang et al. (2019)
Introduction of MLT
Preliminary of Machine Learning
Contents of Machine Learning Testing
Definition
Comparison with software testing
How to test: MLT workflow
Where&What to test: MLT components
Why to test: MLT properties
Ways of Organising Related Work
Testing workflow (how to test)
Test input generation
Test oracle generation
Test adequacy evaluation
Test prioritisation and reduction Bug report analysis
Debug and repair
Testing components (where&what to test)
Data testing
Learning program testing
Framework testing
Testing properties (why to test)
Correctness
Model Relevance
Robustness&Security
Efficiency
Fairness
Interpretability
Privacy
Application Scenario
Autonomous driving
Machine translation
Natural language inference
Analysis and Comparison
Machine learning categories
Machine learning properties
Data sets
Open-source tools
Research Direction

Automated ML testing

- Ensure that all the executions are exactly the same, in the same environment (no more "it works on my machine, I swear!").
- Run tests faster: let's say that you have one thousand models in your pipeline. Running one by one for sure is not the best way to spend your time.
- Easier debugging: detect model's malfunctioning earlier, avoiding deploying it into production.


Data Quality
Data Cleaning
Model Monitoring
Tests for…
Feature Extraction
Trained Model
Modelling

What should we test?

ML Test Score

What's your ML Test Score? A rubric for ML production systems

Eric Breck, Shanqing Cai, Eric Nielsen, Michael Salib, D. Sculley
Google, Inc.
{ebreck, cais, nielsene, msalib, dsculley}@google.com

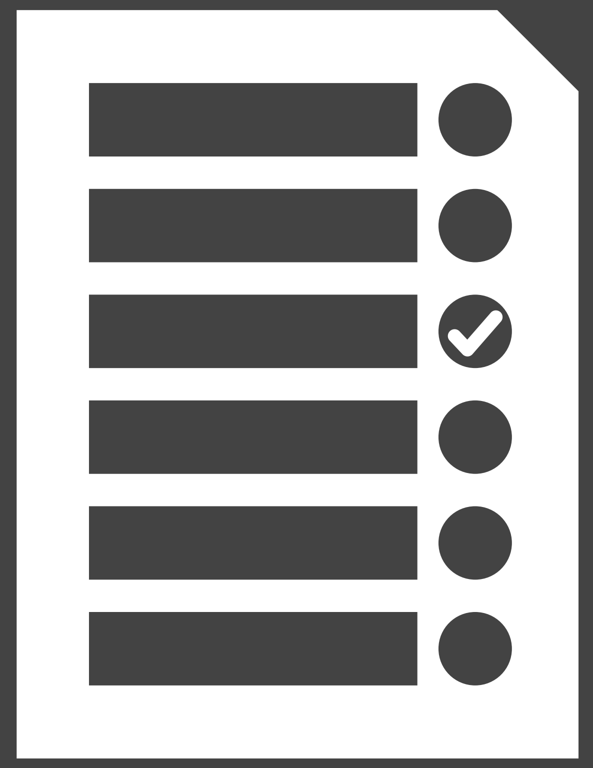
Abstract

Using machine learning in real-world production systems is complicated by a host of issues not found in small toy examples or even large offline research experiments. Testing and monitoring are key considerations for assessing the production-readiness of an ML system. But how much testing and monitoring is enough? We present an ML Test Score rubric based on a set of actionable tests to help quantify these issues.

https://research.google/pubs/pub45742/

ML Test Score

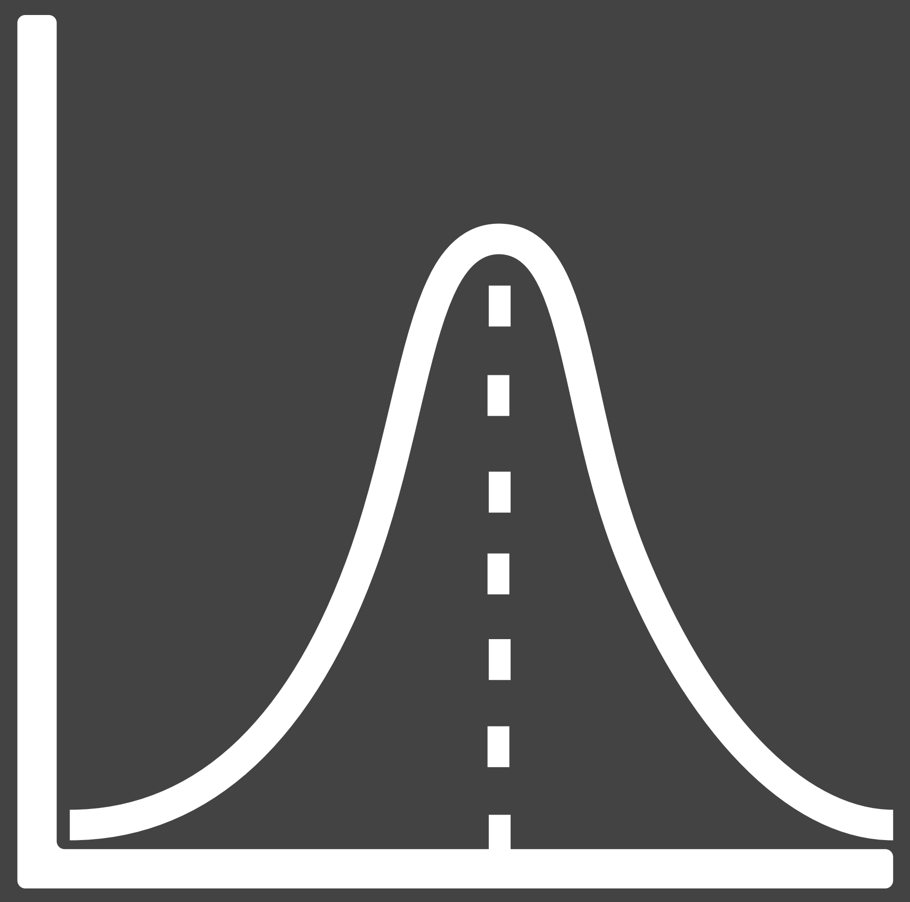
- 4 main angles:
  - Tests for features and data.
  - Tests for model development.
  - Tests for ML infrastructure.
  - Monitoring tests for ML.



⚠️ Disclaimer

Some tests are not covered but are not less important.
Check the paper for the full list.

Test that the distributions of each feature match your expectations.



Tests for Features and Data

Test the relationship between each feature and the target, and the pairwise correlations between individual signals.

 # Tests for Features and Data

Test the cost of each feature.

- Latency
- Memory usage
- More upstream data dependencies
- Additional instability

Test that a model does not contain any features that have been manually determined as unsuitable for use.

Tests for Features and Data

Test that your system maintains privacy controls across its entire data pipeline.

(not only in raw data but also in intermediate stages)

Tests for Features and Data

Test all code that creates input features, both in training and serving

E.g., methods used to clean date formats; methods use to remove stop words.

Tests for Features and Data

Test that every model specification undergoes a code review and is checked in to a repository

mllint might be useful here

 # Tests for Model Development

Test the relationship between offline proxy metrics and the actual impact metrics

For example, how does a 1% improvement in accuracy metrics translate into effects on business metrics (e.g., user satisfaction)?

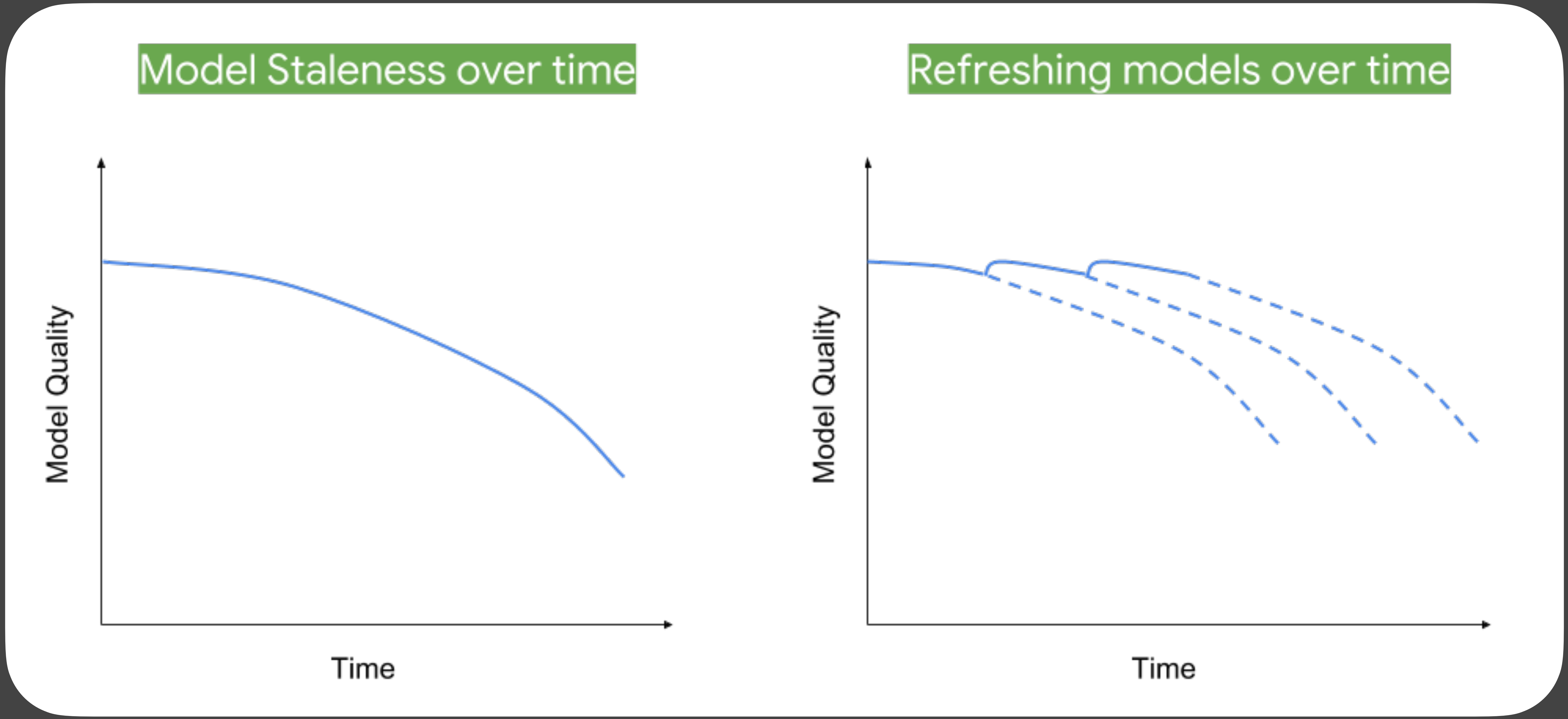
 # Tests for Model Development

Test the impact of each tunable hyperparameter.

What's the oracle?

 # Tests for Model Development

Test the effect of model staleness.

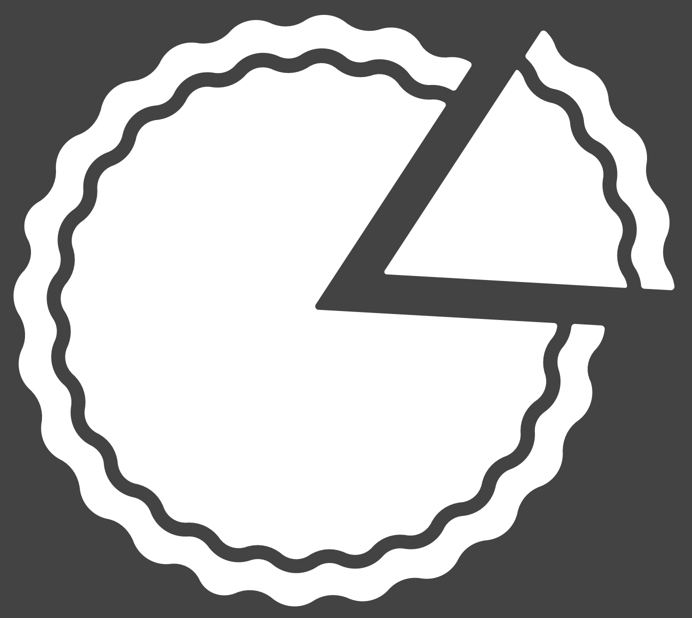


#figure credits: Clemens Mewald, 2018

Test against a simpler model as a baseline

Tests for Model Development

Test model quality on important data slices.



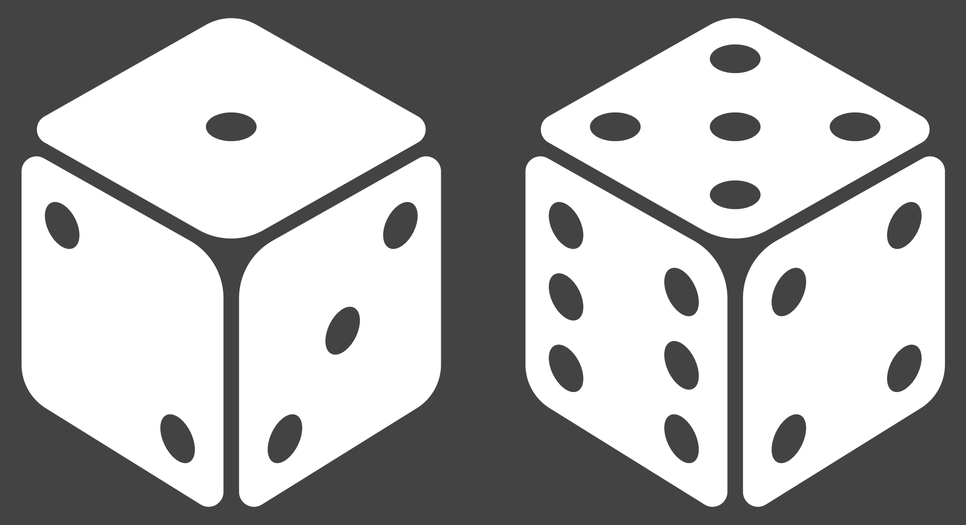
 # Tests for Model Development

Test the model for implicit bias.

Tests for Model Development

~~Test the reproducibility of training.~~

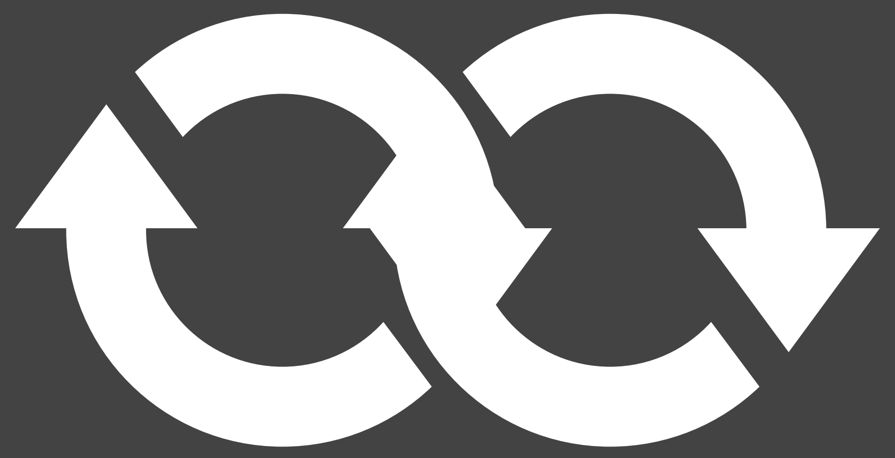
Test non-determinism robustness.



~~# Tests for ML Infrastructure~~

Tests for Model Development

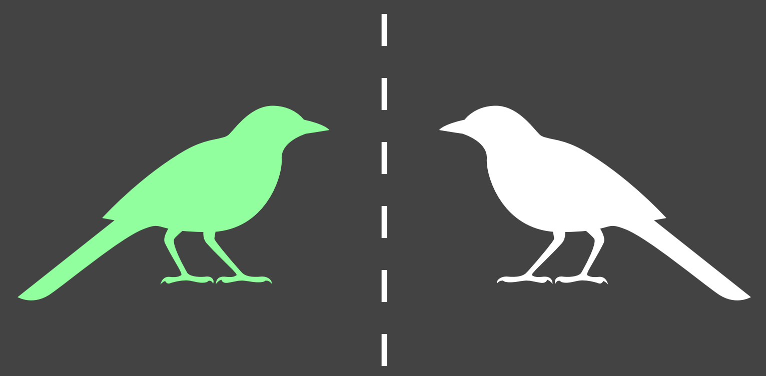
Integration test the full ML pipeline.



Tests for ML Infrastructure

Test models via a canary process before they enter production serving environments.

Example: AB testing



Tests for ML Infrastructure

Test how quickly and safely a model can be rolled back to a previous serving version.

Tests for ML Infrastructure

Test that data invariants hold in training and serving inputs.

E.g., **shape of distributions** of features should be the same in **training data** and **serving data**.

…

 # Monitoring Tests for ML

Test for model staleness.

Test for dramatic or slow-leak regressions in training speed, serving latency, throughput, or RAM usage.

Monitoring Tests for ML

Test for regressions in prediction quality on served data.

Monitoring Tests for ML

Final score

ML test score

- **1 point.** If you do the test manually.
- **2 points.** If you do the test automatically. ⭐
- Meaning of the score:
    - **0 points**: More of a prototype project than a productionized system.
    - **1-2 points**: Not totally untested, but it is worth considering the possibility of serious holes in reliability.
    - **3-4 points**: There’s basic productionization, but additional investment may be needed.
    - **5-6 points**: Reasonably tested, but it’s possible that more of those tests and procedures may be automated.
    - **7-10 points**: Strong levels of automated testing and monitoring, appropriate for critical systems.
    - **12+ points**: Exceptional levels of automated testing and monitoring.

But really, how should we test?

(a few basic examples)

❌ ✔ ❌ ❌ ✔ ✔

PyTest – basic example

Project structure

```
/my_project_folder
   ...
   /src
      train_model.py
   /tests
      test_trained_model.py
```

./tests/test_trained_model.py

```python
from sklearn.externals import joblib

def test_something_in_the_model():
    model = joblib.load('trained_dummy_model.sav')
    // ...
    assert ...
```

Run the tests

```
$ pytest
```

Duplicates

Unit test

./tests/test_data_cleaning.py

```
@pytest.fixture()
def df():
  df = pandas.read()
  yield df


def test_no_duplicates(df):
  assert len(df['id'].unique())==df.shape[0]
  assert df.groupby(['date','id']).size().max()==1
```

Preprocess methods

Unit test

./tests/test_data_cleaning.py

```
@pytest.fixture()
def df():
  df = pandas.read()
  yield df


def test_preprocess_missing_name(df):
  assert preprocess_missing_name("10019\n") is None

def test_preprocess_city(df):
  assert preprocess_city("amsterdam") == "Amsterdam"
  assert preprocess_city("AMS") == "Amsterdam"
  assert preprocess_city(" Amsterdam  ") == "Amsterdam"
```

Value ranges

Unit test

./tests/test_data_cleaning.py

```
@pytest.fixture()
def df():
  df = pandas.read()
  yield df

def test_value_ranges(df):
  assert all (df['percentage']<=1)
  assert df.groupby('name')['budget'].sum()<=1000
  assert all (df['height'] >= 0)
```

Test Non-determinism Robustness

Model Validation tests

- Performance stability when using different random seeds.
- If a model is performant, it should have little dependency on random variance.
- Make seed an attribute in the pipeline; test different seeds; assert for low variability.

```python
from sklearn.model_selection import train_test_split

SEED = 41

#...

(
    X_train,
    X_test,
    y_train,
    y_test
) = train_test_split(X, y, test_size=0.2, random_state=SEED)
```

Test Non-determinism Robustness

Unit test

./tests/test_data_cleaning.py

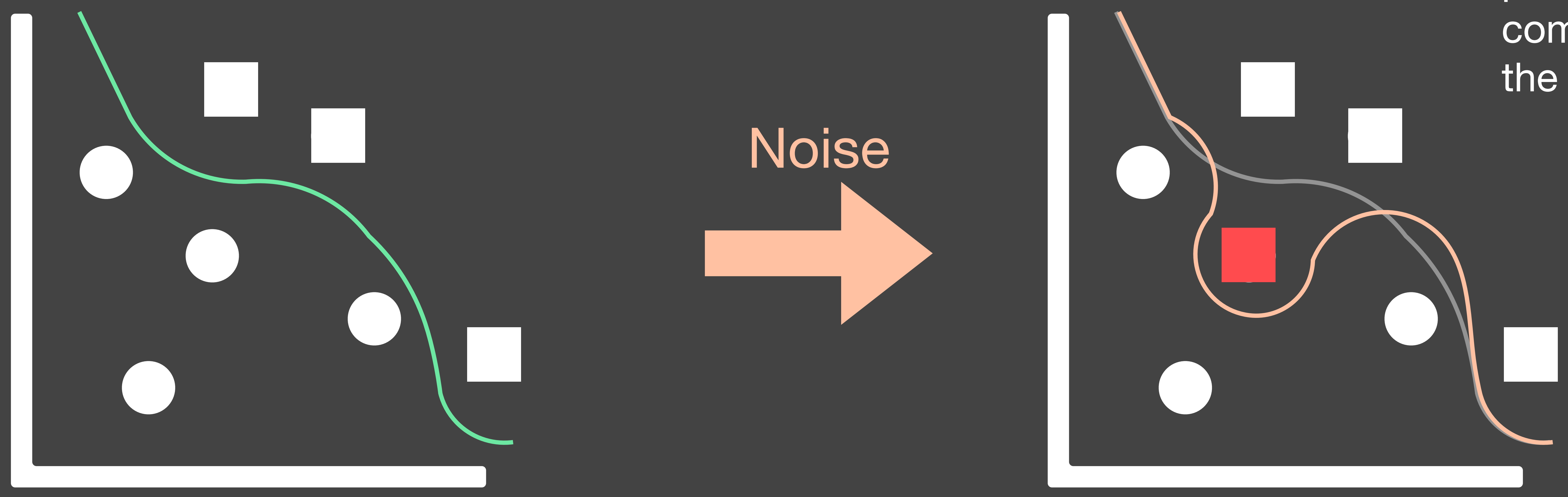
```
@pytest.fixture()
def trained_model():
  trained_model = joblib.load('trained_model.sav')
  yield trained_model

def test_nondeterminism_robustness(trained_model):
  original_score = evaluate_score(trained_model) # score between 0..100
  for seed in [1,2]:
    model_variant = train_model(random_state=seed)
    assert abs(original_score - score(model_variant)) <=0.03
```

Test Noise Robustness

- 1- What happens to the performance when we change a few training data points?
- 2- What happens to the performance when we add acceptable noise to test data points? (E.g., add typos)

A few noisy data points should not completely change the model



Test model quality on important data slices

Model validation tests

⚠️ **Warning!**

This test is highly dependent on the problem.

./tests/test_data_slice.py

```
@pytest.fixture()
def trained_model():
  trained_model = joblib.load('trained_model.sav')
  yield trained_model

@pytest.fixture()
def test_data():
  test_data = pandas.read_csv("test_data.csv")
  yield test_data

def test_data_slice(trained_model, test_data):
  original_score = evaluate_score(trained_model, test_data) # score between 0..1
  sliced_data = test_data[test_data['city'] == 'Delft']
  sliced_score = evaluate_score(trained_model, sliced_data)
  assert abs(original_score - sliced_data) <= 0.05
```

What else?

- A lot of work is yet to be done in this area:
  - There is not much documentation around this topic.
  - What to test? Practitioners are looking out for testing best practices.

?

Mutamorphic testing and repair

Automatic Testing and Improvement of Machine Translation

Zeyu Sun
Peking University
szy_@pku.edu.cn

Jie M. Zhang*
University College London
jie.zhang@ucl.ac.uk

Mark Harman
Facebook London
University College London
mark.harman@ucl.ac.uk

Mike Papadakis
University of Luxembourg
mike.papadakis@gmail.com

Lu Zhang
Peking University
zhanglucs@pku.edu.cn

ABSTRACT

This paper presents TransRepair, a fully automatic approach for testing and repairing the consistency of machine translation systems. TransRepair combines mutation with metamorphic testing to detect inconsistency bugs (without access to human oracles). It then adopts probability-reference or cross-reference to post-process the translations, in a grey-box or black-box manner, to repair the inconsistencies. Our evaluation on two state-of-the-art translators, Google Translate and Transformer, indicates that TransRepair has a high precision (99%) on generating input pairs with consistent trans-

Google Translate results for the language pair (English → Chinese)[1]. As can be seen from the figure, Google Translate translates 'good' into 'hen hao de' (which means 'very good') when the subject is 'men' or 'male students'. However, interestingly, but also sadly, it translates 'good' into 'hen duo' (which means 'a lot') when the subject is 'women' or 'female students'[2].

Such inconsistency may confuse users, and is also clearly *unfair* to female researchers in computer science; producing 'a lot' of research is clearly a more pejorative interpretation, when compared to producing 'very good' research. To avoid such unfair translations

https://arxiv.org/abs/1910.02688

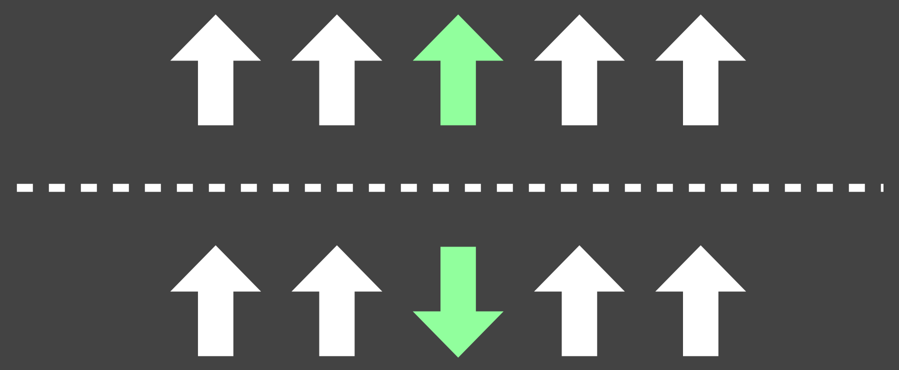
Mutamorphic testing

- Metamorphic testing + mutation
- Metamorphic testing: derive new test cases based on properties of the existing ones.
  E.g., commutative property:
  assertEqual(add(1, 2), 3) => assertEqual(add(2, 1), 3)
- Mutamorphic: the new test cases are not exactly based safe properties
- Black-/grey-box testing. No access to the model or its training codebase.
- Implemented for ML-based translators.
  "It is okay" and *"It is fine"* have a mutamorphic relationship (context-similar).

Mutamorphic Testing

1. Automatic Test Input Generation

Generate sentences by replacing 1 word with a context-similar word (Mutation).



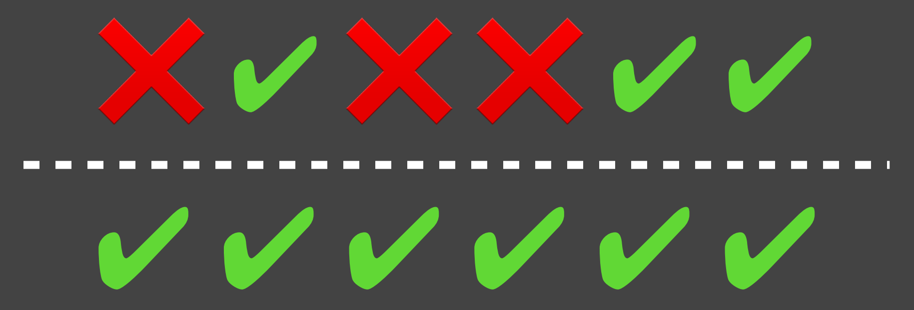
2. Automatic Test Oracle Generation

When the translation of the mutant and the original sentence are fairly different, we have a failing test.

3. Automatic Inconsistency Repair

Find a mutant sentence with a translation that we can use to replace the original translation.

(Only works for translations with similar structure and word types)

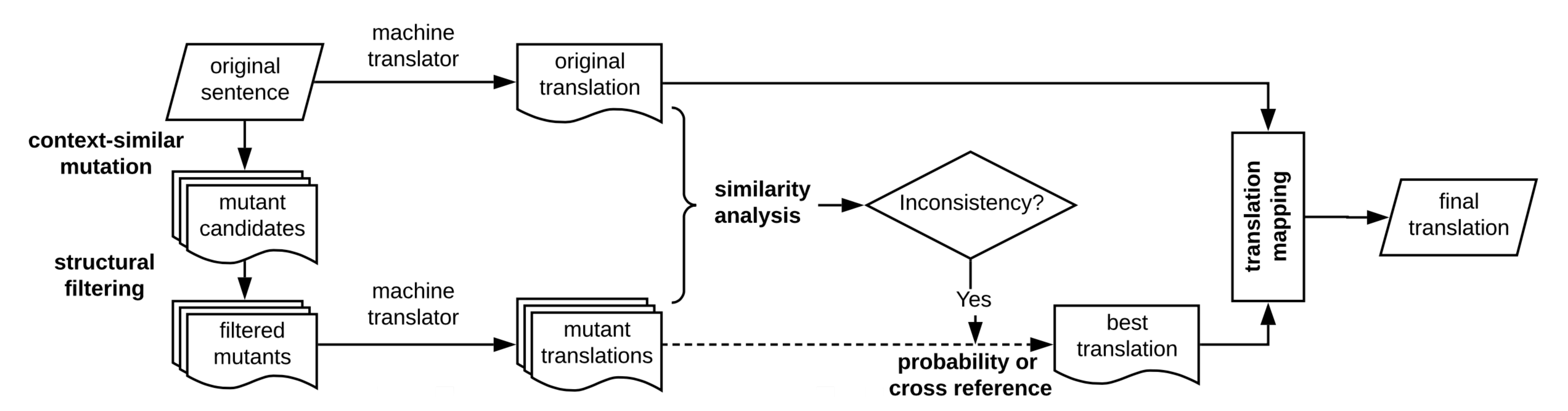


Mutamorphic Testing

1. Automatic Test Input Generation

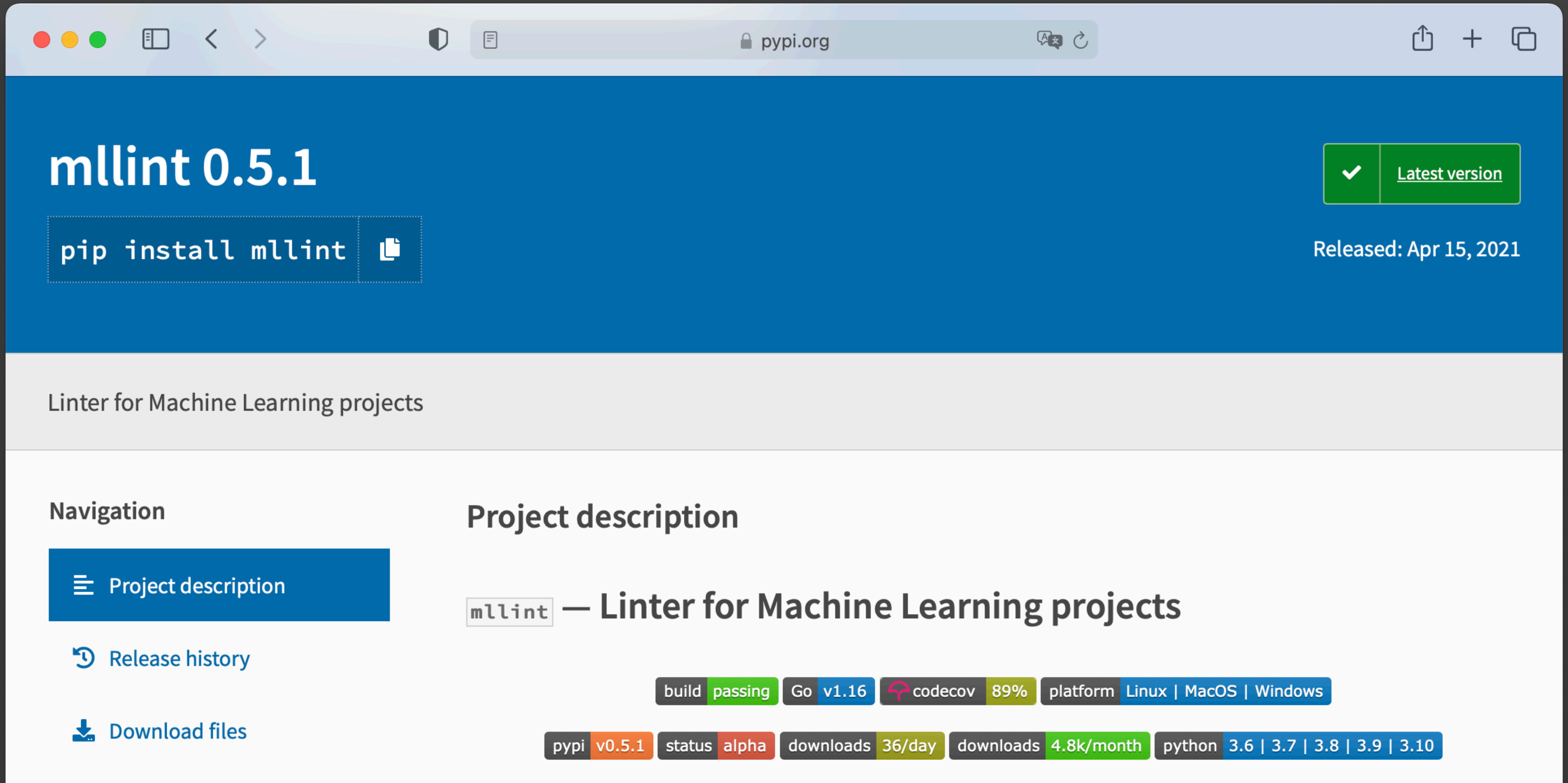
2. Automatic Test Oracle Generation

3. Automatic Inconsistency Repair



Useful tools

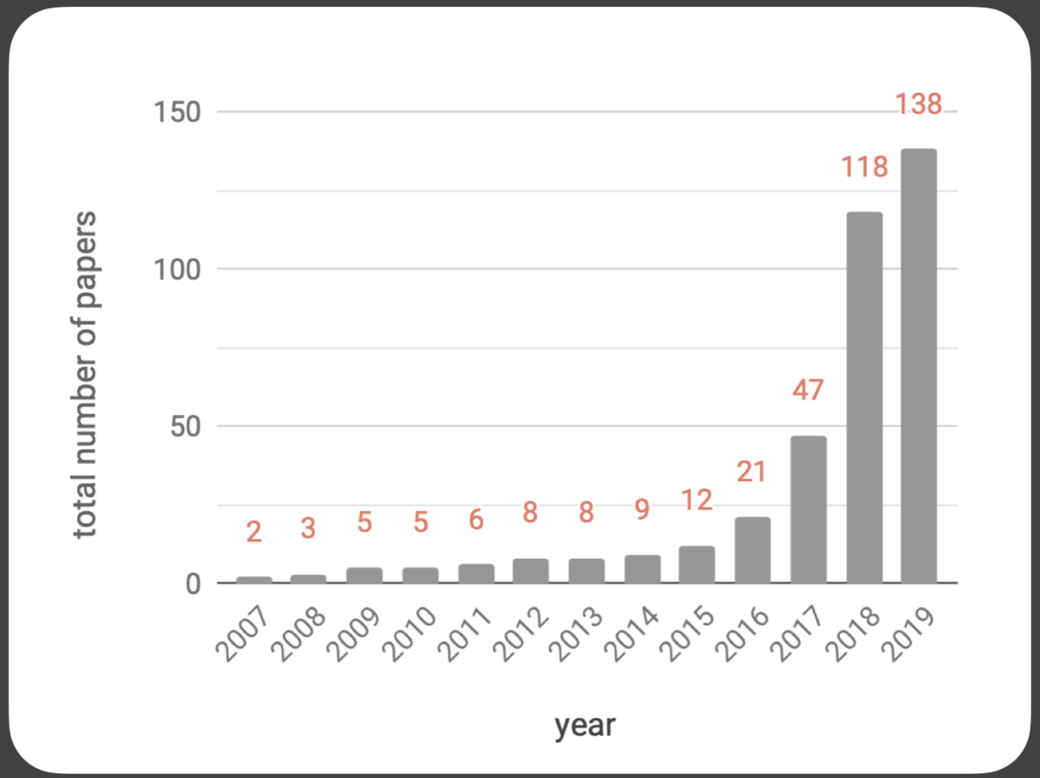
- Great Expectations. https://greatexpectations.io
- mllint. https://github.com/bvobart/mllint



Wrap-up

ML Testing Publications during 2007–2019

(⚠ Cumulative plot)



Zhang et al. (2019)

5

ML Test Score

- 4 main angles:
  - Tests for features and data.
  - Tests for model development.
  - Tests for ML infrastructure.
  - Monitoring tests for ML.

11

Test Non-determinism Robustness

Unit test

./tests/test_data_cleaning.py

```
@pytest.fixture()
def trained_model():
  trained_model = joblib.load('trained_model.sav')
  yield trained_model

def test_nondeterminism_robustness(trained_model):
  original_score = evaluate_score(trained_model) # score between 0..100
  for seed in [1,2]:
    model_variant = train_model(random_state=seed)
    assert abs(original_score - score(model_variant)) <=0.03
```

41

Mutamorphic Testing

1. Automatic Test Input Generation

Generate sentences by replacing 1 word with a context-similar word (Mutation).

2. Automatic Test Oracle Generation

When the translation of the mutant and the original sentence are fairly different, we have a failing test.

3. Automatic Inconsistency Repair

Find a mutant sentence with a translation that we can use to replace the original translation.

(Only works for translations with similar structure and word types)

47