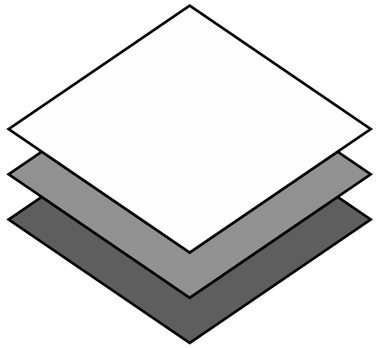
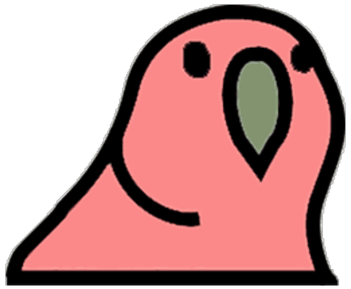


Beyond the Black Box: Testing and Explaining ML Systems

Test score < target performance (in school)



Attention Is All You Need

Ashish Vaswani* Google Brain avaswani@google.com
Noam Shazeer* Google Brain noam@google.com
Niki Parmar* Google Research nikip@google.com
Jakob Uszkoreit* Google Research usz@google.com
Llion Jones* Google Research llion@google.com
Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu
Łukasz Kaiser* Google Brain lukaszkaiser@google.com
Illia Polosukhin* ‡ illia.polosukhin@gmail.com

Abstract

The dominant sequence transduction models are based on complex recurrent or convolutional neural networks that include an encoder and a decoder. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely. Experiments on two machine translation tasks show these models to be superior in quality while being more parallelizable and requiring significantly less time to train. Our model achieves 28.4 BLEU on the WMT 2014 English-to-German translation task, improving over the existing best results, including ensembles, by over 2 BLEU. On the WMT 2014 English-to-French translation task, our model establishes a new single-model state-of-the-art BLEU score of 41.8 after training for 3.5 days on eight GPUs, a small fraction of the training costs of the best models from the literature. We show that the Transformer generalizes well to other tasks by applying it successfully to English constituency parsing both with large and limited training data.

1 Introduction

Recurrent neural networks, long short-term memory [13] and gated recurrent [7] neural networks in particular, have been firmly established as state of the art approaches in sequence modeling and

*Equal contribution. Listing order is random. Jakob proposed replacing RNNs with self-attention and started the effort to evaluate this idea. Ashish, with Illia, designed and implemented the first Transformer models and has been crucially involved in every aspect of this work. Noam proposed scaled dot-product attention, multi-head attention and the parameter-free position representation and became the other person involved in nearly every detail. Niki designed, implemented, tuned and evaluated countless model variants in our original codebase and tensor2tensor. Llion also experimented with novel model variants, was responsible for our initial codebase, and efficient inference and visualizations. Lukasz and Aidan spent countless long days designing various parts of and implementing tensor2tensor, replacing our earlier codebase, greatly improving results and massively accelerating our research.

†Work performed while at Google Brain.

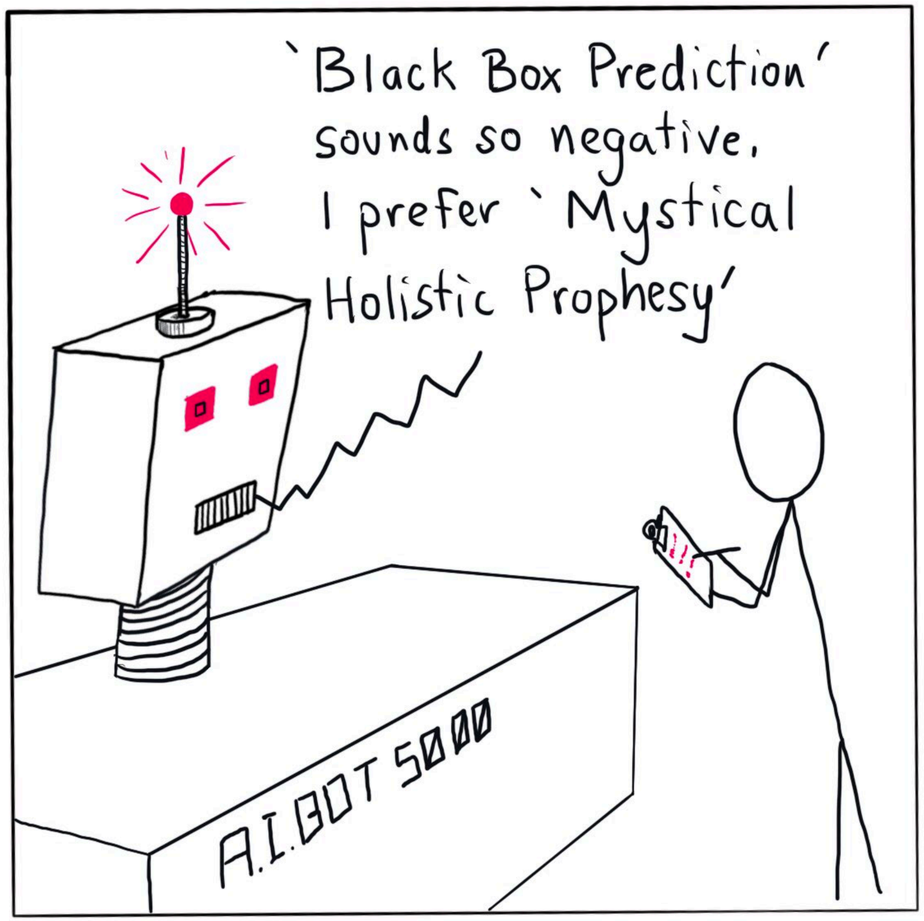
‡Work performed while at Google Research.

31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA.

Test score < target performance (real world)

How do we know we can trust it?

What's wrong with the black box?



What's wrong with the black box?

What does good test set performance tell us?

*If test data and production data come from the **same distribution**, then **in expectation** the performance **of your model** on **your evaluation metrics** will be the same*

- Real-world distribution doesn't always match the offline distribution
  - Drift, shift, and malicious users
- Expected performance doesn't tell the whole story
  - Slices and edge cases
- Your model ≠ your machine learning system
- You're probably not optimizing the exact metrics you care about

How bad is it, really?

"I think the single biggest thing holding back the autonomous vehicle industry today is that, even if we had a car that worked, no one would know, because no one is confident that they know how to properly evaluate it"

Former ML engineer at autonomous vehicle company

Goal of this lecture

- Introduce concepts and methods to help you, your team, and your users:
  - Understand at a deeper level how well your model is performing
  - Become more confident in your model's ability to perform well in production
  - Understand the model's **performance envelope**, i.e., where you should expect it to perform well and where not

Outline

- Software testing
- Testing machine learning systems
- Explainable and interpretable AI
- What goes wrong in the real world?

Questions?

Outline

- **Software testing**
- Testing machine learning systems
- Explainable and interpretable AI

Software testing

- Types of tests
- Best practices
- Testing in production
- CI/CD

Software testing

- **Types of tests**
- Best practices
- Testing in production
- CI/CD

Basic types of software tests

- **Unit tests**: test the functionality of a single piece of the code (e.g., a single function or class)
- **Integration tests**: test how two or more units perform when used together (e.g., test if a model works well with a preprocessing function)
- **End-to-end tests**: test that the entire system performs when run together, e.g., on real input from a real user

Other types of software tests

- Mutation tests
- Contract tests
- Functional tests
- Fuzz tests
- Load tests
- Smoke tests
- Coverage tests
- Acceptance tests
- Property-base tests
- Usability tests
- Benchmarking
- Stress tests
- Shadow tests
- Etc

Software testing

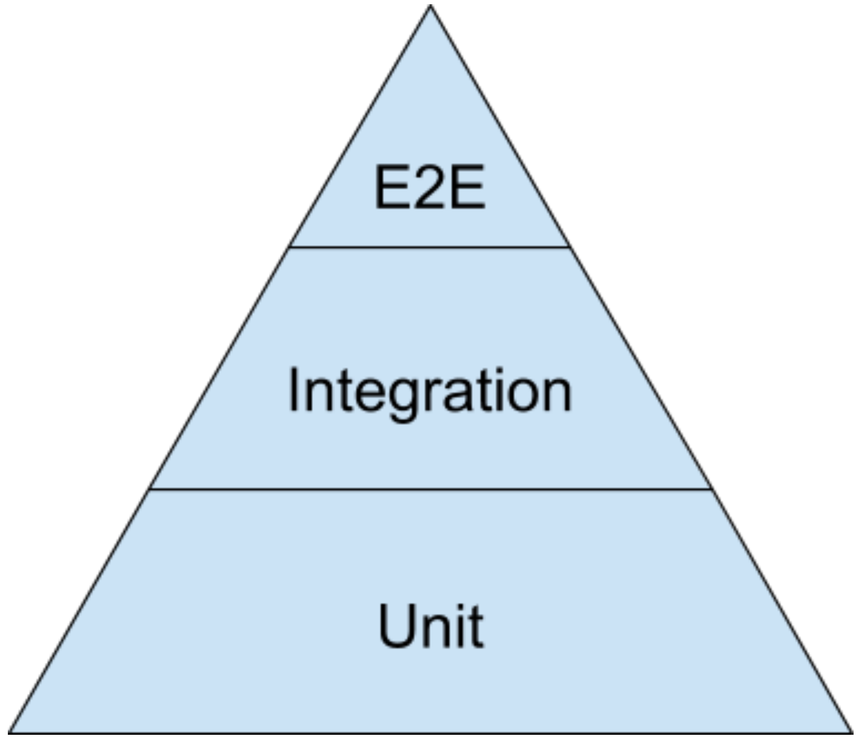
- Types of tests
- **Best practices**
- Testing in production
- CI/CD

Testing best practices

- Automate your tests
- Make sure your tests are reliable, run fast, and go through the same code review process as the rest of your code
- Enforce that tests must pass before merging into the main branch
- When you find new production bugs, convert them to tests
- Follow the test pyramid

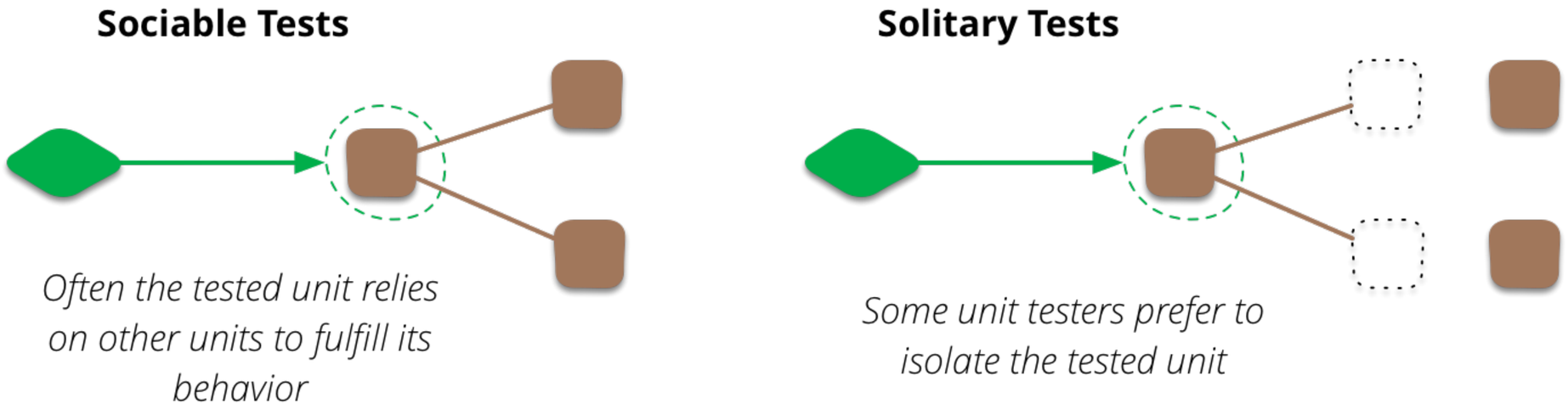
Best practice: the testing pyramid

- Compared to E2E tests, unit tests are:
  - Faster
  - More reliable
  - Better at isolating failures
- Rough split: 70/20/10



https://testing.googleblog.com/2015/04/just-say-no-to-more-end-to-end-tests.html

Controversial best practice: solitary tests



- Solitary testing (or mocking) doesn't rely on real data from other units
- Sociable testing makes the implicit assumption that other modules are working

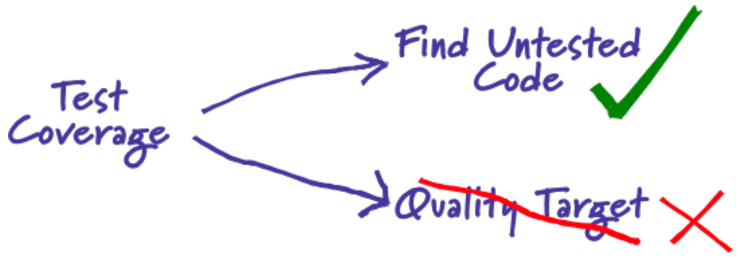
https://martinfowler.com/bliki/UnitTest.html

Controversial best practice: test coverage

Coverage report: 39.90%

Module ↑	statements	missing	excluded	branches	partial	coverage
cogapp/__init__.py	2	0	0	0	0	100.00%
cogapp/__main__.py	3	3	0	0	0	0.00%
cogapp/backward.py	22	6	0	4	2	69.23%
cogapp/cogapp.py	485	215	1	200	28	49.34%
cogapp/makefiles.py	27	20	0	14	0	17.07%
cogapp/test_cogapp.py	790	549	6	20	0	30.00%
cogapp/test_makefiles.py	71	53	0	6	0	23.38%
cogapp/test_whiteutils.py	69	50	0	0	0	27.54%
cogapp/whiteutils.py	45	5	0	34	4	88.61%
Total	**1514**	**901**	**7**	**278**	**34**	**39.90%**

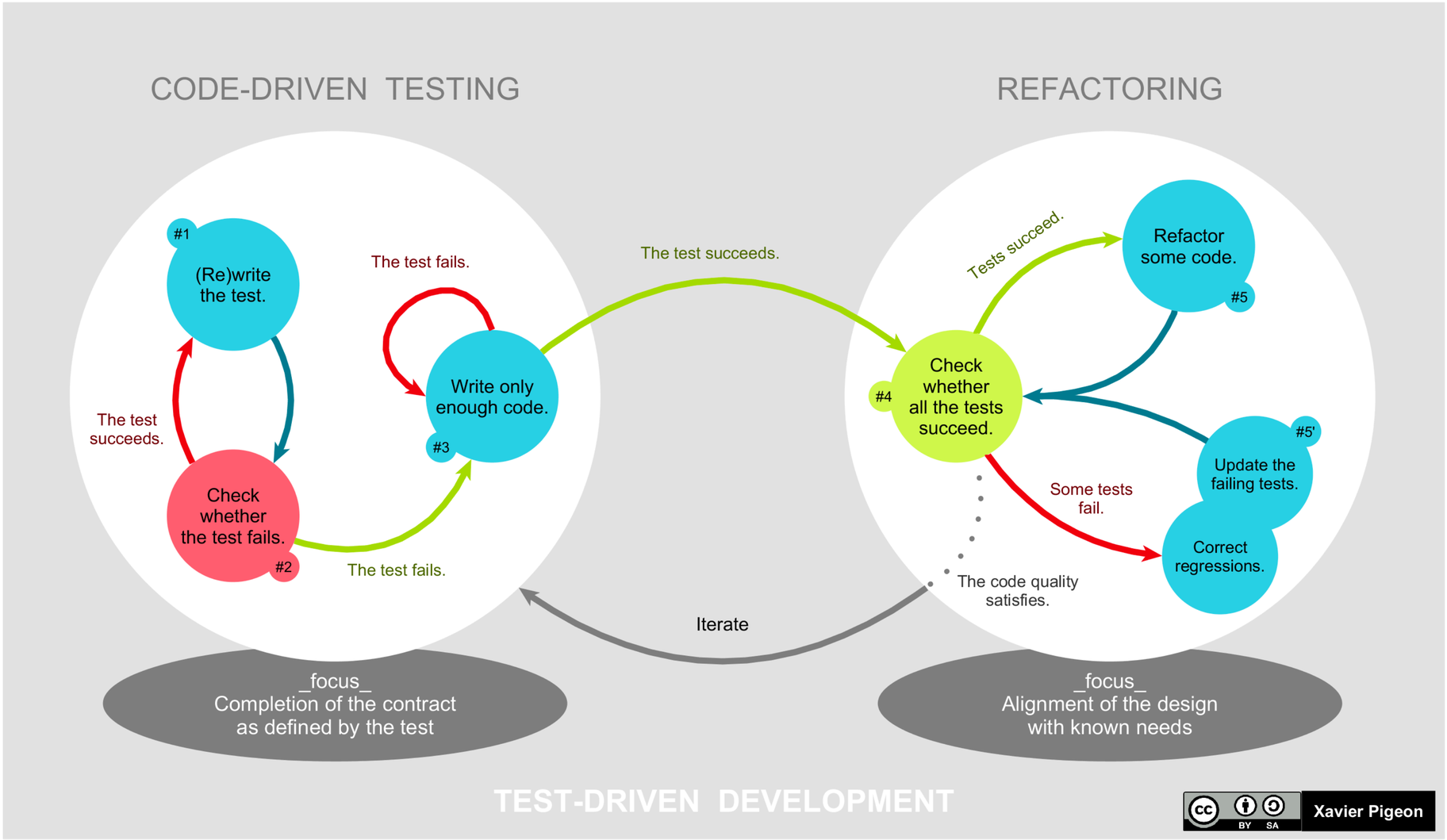
coverage.py v5.4, created at 2021-01-24 17:48 -0500



- Percentage of lines of code called in at least one test
- Single metric for quality of testing
- But it doesn't measure the right things — in particular, test quality

https://martinfowler.com/bliki/TestCoverage.html

Controversial best practice: test-driven development



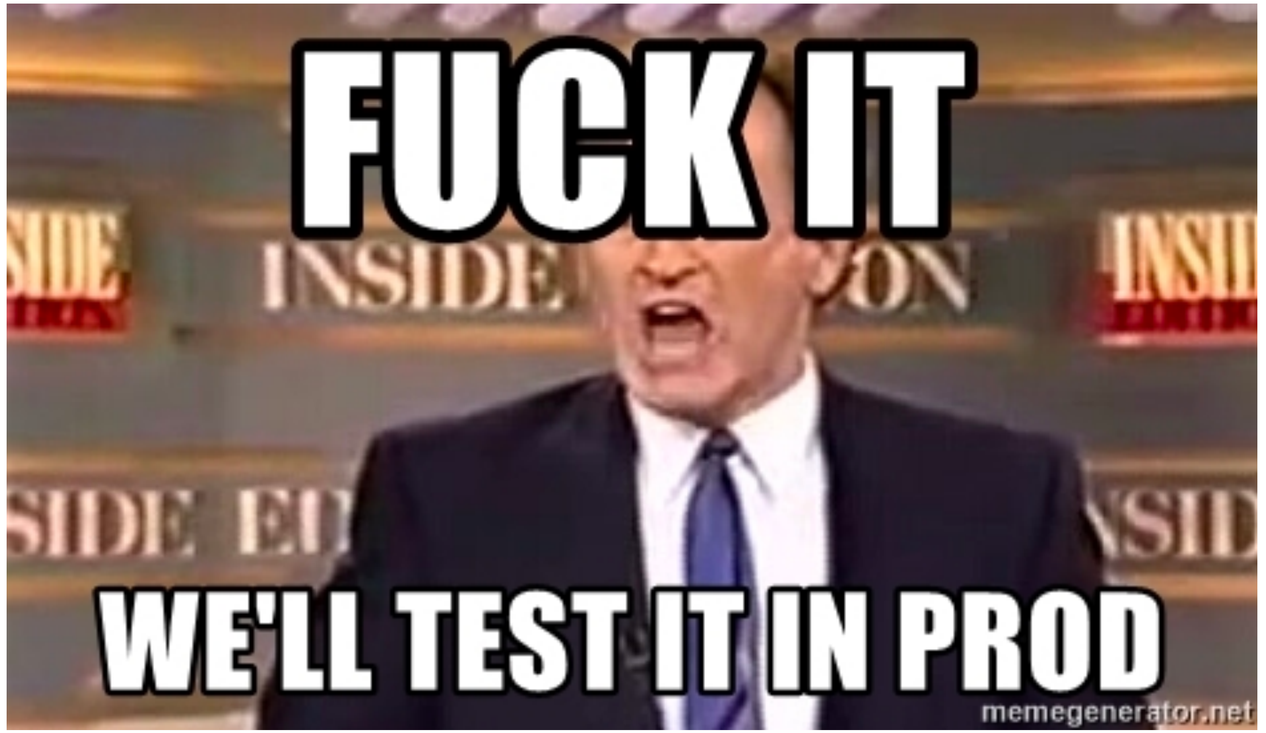
https://martinfowler.com/bliki/TestCoverage.html

Questions?

Software testing

- Types of tests
- Best practices
- **Testing in production**
- CI/CD

Test in prod, really?



Test in prod, really?

- Traditional view:
  - The goal of testing is to prevent shipping bugs to production
  - Therefore, by definition you must do your testing offline, before your system goes into production
- However:
  - “Informal surveys reveal that the percentage of bugs found by automated tests are surprisingly low. Projects with significant, well-designed automation efforts report that regression tests find about 15 percent of the total bugs reported (Marick, online).” — Lessons Learned in Software Testing
  - Modern distributed systems just make it worse

The case for testing in production

Bugs are inevitable, might as well set up the system so that users can help you find them

The case for testing in production

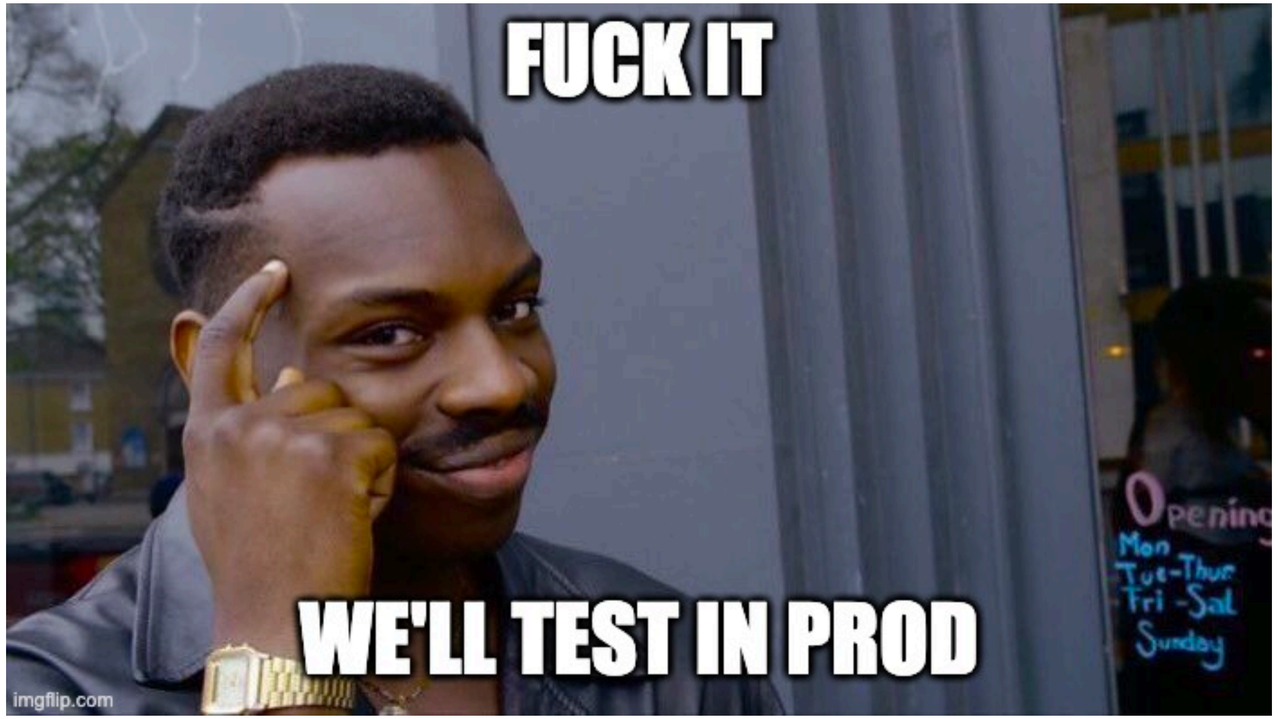
With sufficiently advanced monitoring & enough scale, it's a realistic strategy to write code, push it to prod, & watch the error rates. If something in another part of the app breaks, it'll be apparent very quickly in your error rates. You can either fix or roll back. You're basically letting your monitoring system play the role that a regression suite & continuous integration play on other teams.

https://twitter.com/sarahmei/status/868928631157870592?s=20

How to test in prod

- Canary deployments
- A/B Testing
- Real user monitoring
- Exploratory testing

Test in prod, really.



Questions?

Software testing

- Types of tests
- Best practices
- Testing in production
- **CI/CD**

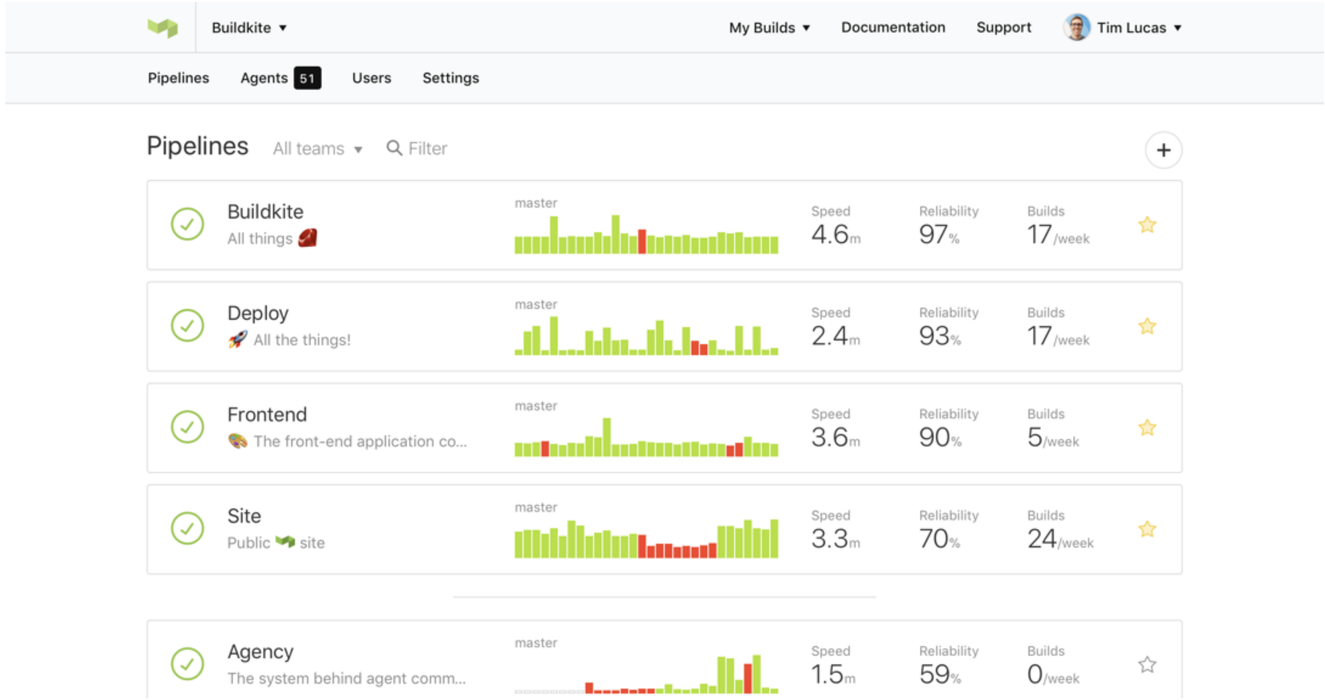
CircleCI / Travis / Github Actions / etc

- SaaS for continuous integration
- Integrate with your repository: every push kicks off a cloud job
- Job can be defined as commands in a Docker container, and can store results for later review
- No GPUs
- CircleCI / Github Actions have free plans
- Github actions is super easy to integrate — would start here

Jenkins / Buildkite

- Nice option for running CI on your own hardware, in the cloud, or mixed
- Good option for a scheduled training test (can use your GPUs)
- Very flexible

```
$ buildkite-agent start
Starting buildkite-agent with PID: 11301
Registering agent with Buildkite...
Successfully registered agent "docker-builder-1
Waiting for work...
Assigned job 69f662c7-b990-489e-8e77-8e02550
```



Questions?

Outline

- Software testing
- **Testing machine learning systems**
- Explainable and interpretable AI

ML systems are software, so test them like software?

Software is…

- Code
- Written by humans to solve a problem
- Prone to “loud” failures
- Relatively static

… But ML systems are

- Code + data
- Compiled by optimizers to satisfy a proxy metric
- Prone to silent failures
- Constantly changing

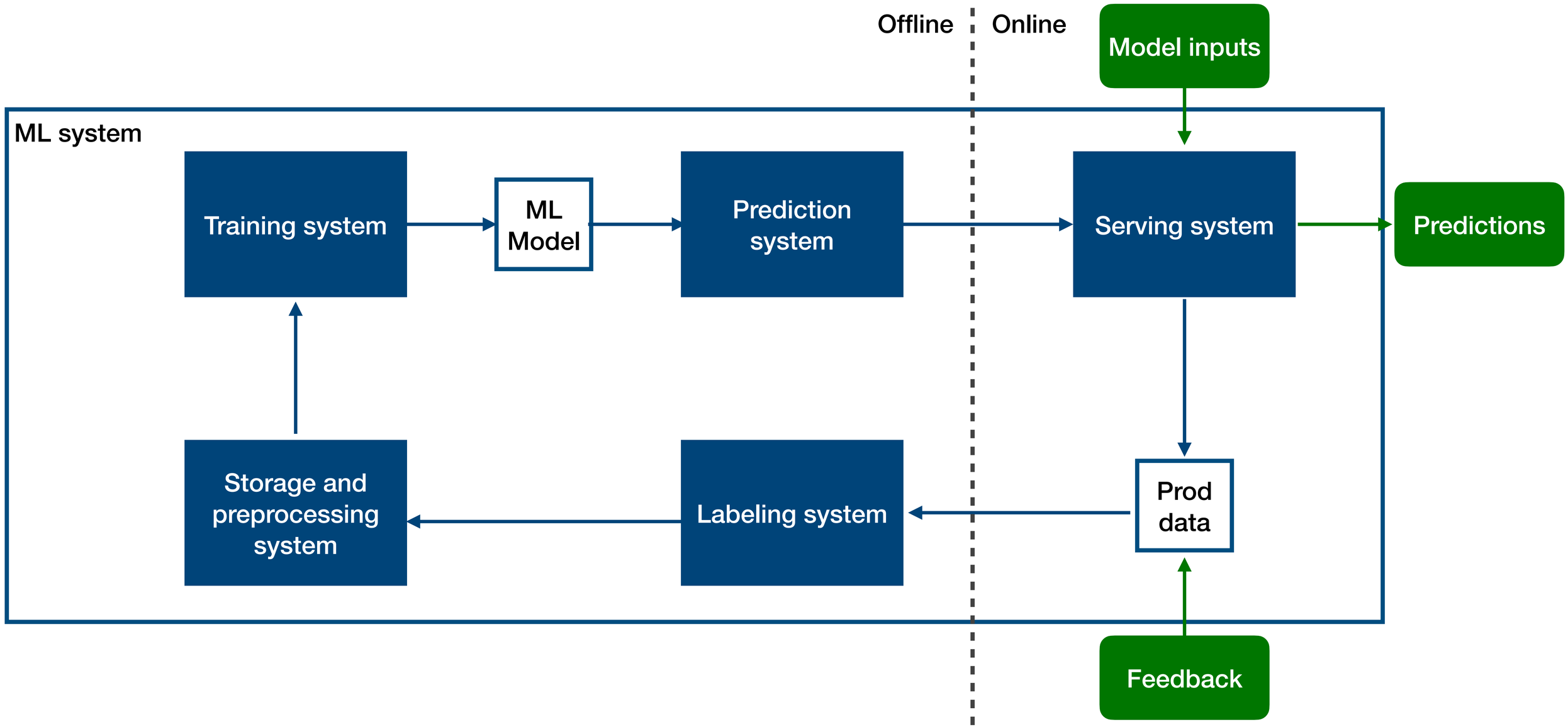
Common mistakes in testing ML systems

- Only testing the model, not the entire ML system
- Not testing the data
- Not building a granular enough understanding of the performance of the model before deploying
- Not measuring the relationship between model performance metrics and business metrics
- Relying too much on automated testing
- Thinking offline testing is enough — not monitoring or testing in prod

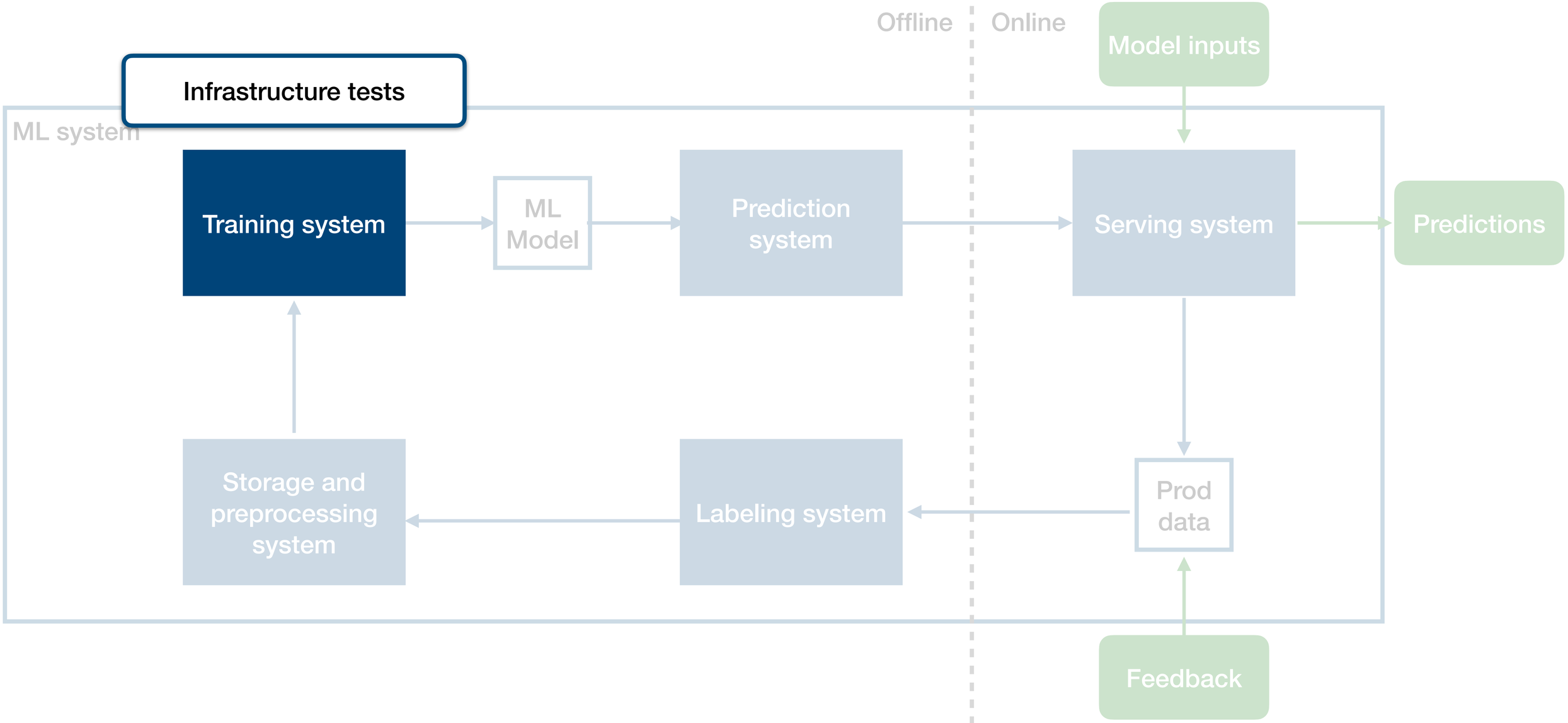
Common mistakes in testing ML systems

- **Only testing the model, not the entire ML system**
- Not testing the data
- Not building a granular enough understanding of the performance of the model before deploying
- Not measuring the relationship between model performance metrics and business metrics
- Relying too much on automated testing
- Thinking offline testing is enough — not monitoring or testing in prod

ML Models vs ML Systems



ML Models vs ML Systems

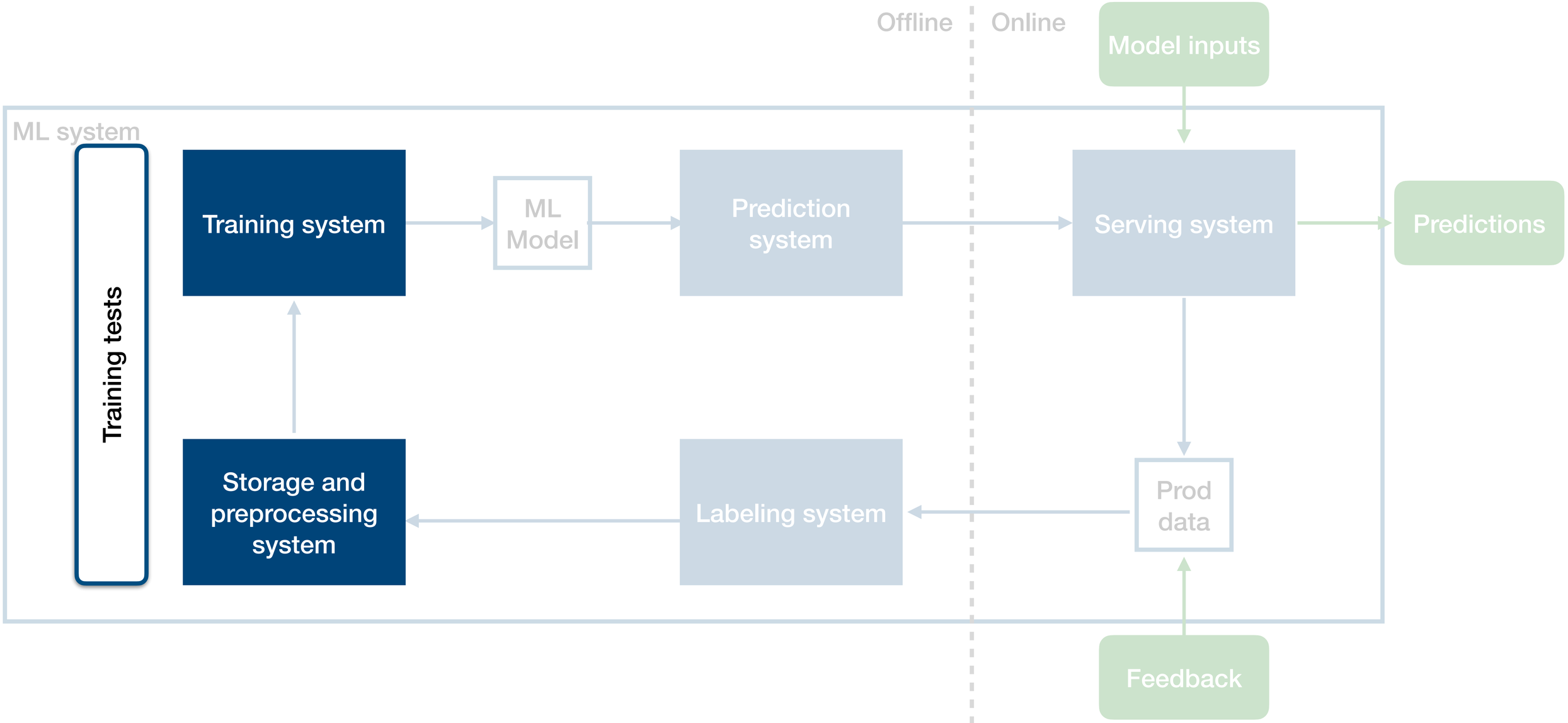


Infrastructure tests — unit tests for training code

- **Goal:** avoid bugs in your training pipelines
- **How?**
  - Unit test your training code like you would any other code
  - Add **single batch** or **single epoch** tests that check performance after an abbreviated training run on a tiny dataset
  - Run frequently during development

Questions?

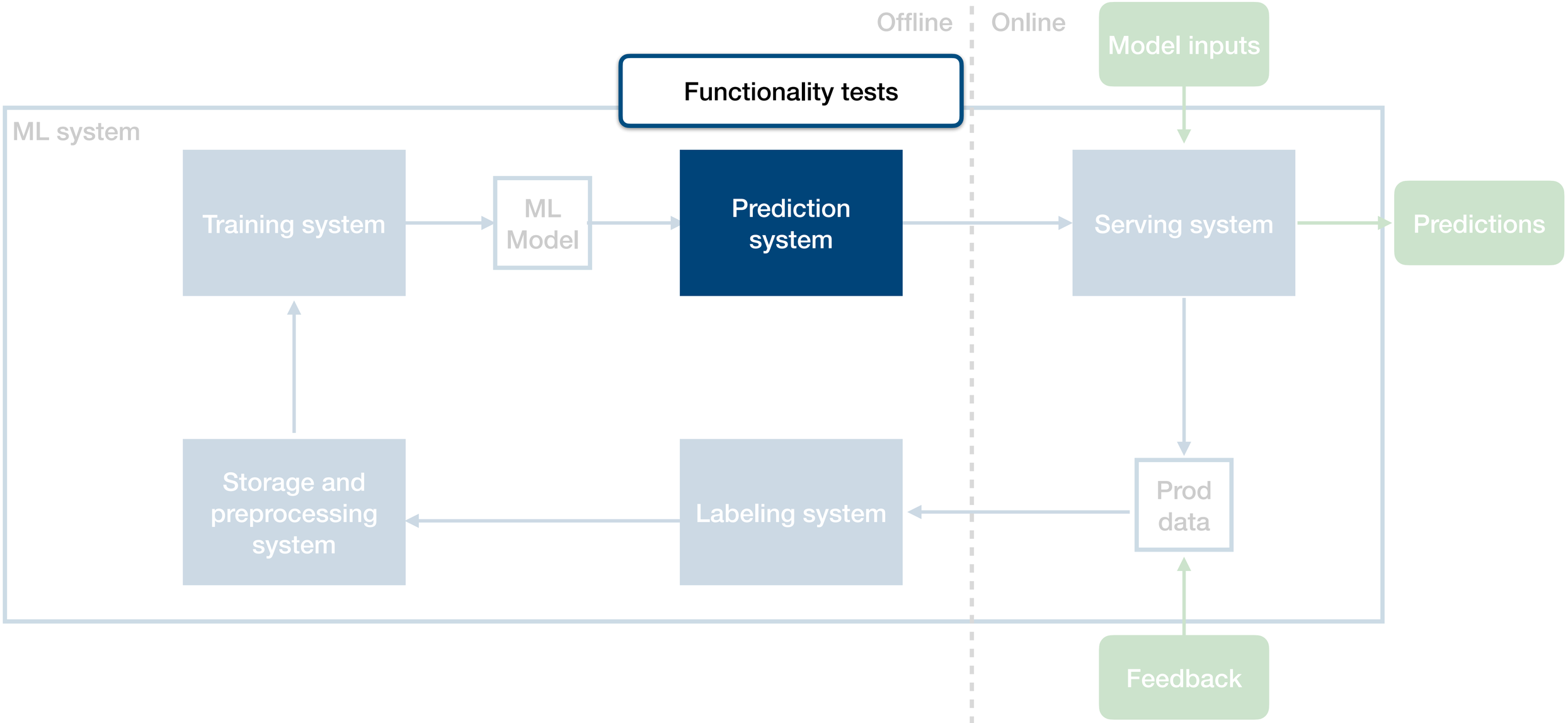
ML Models vs ML Systems



Training tests

- **Goal:** make sure training is reproducible
- **How?**
  - Pull a fixed dataset and run a full or abbreviated training run
  - Check to make sure model performance remains consistent
  - Consider pulling a sliding window of data
  - Run periodically (nightly for frequently changing codebases)

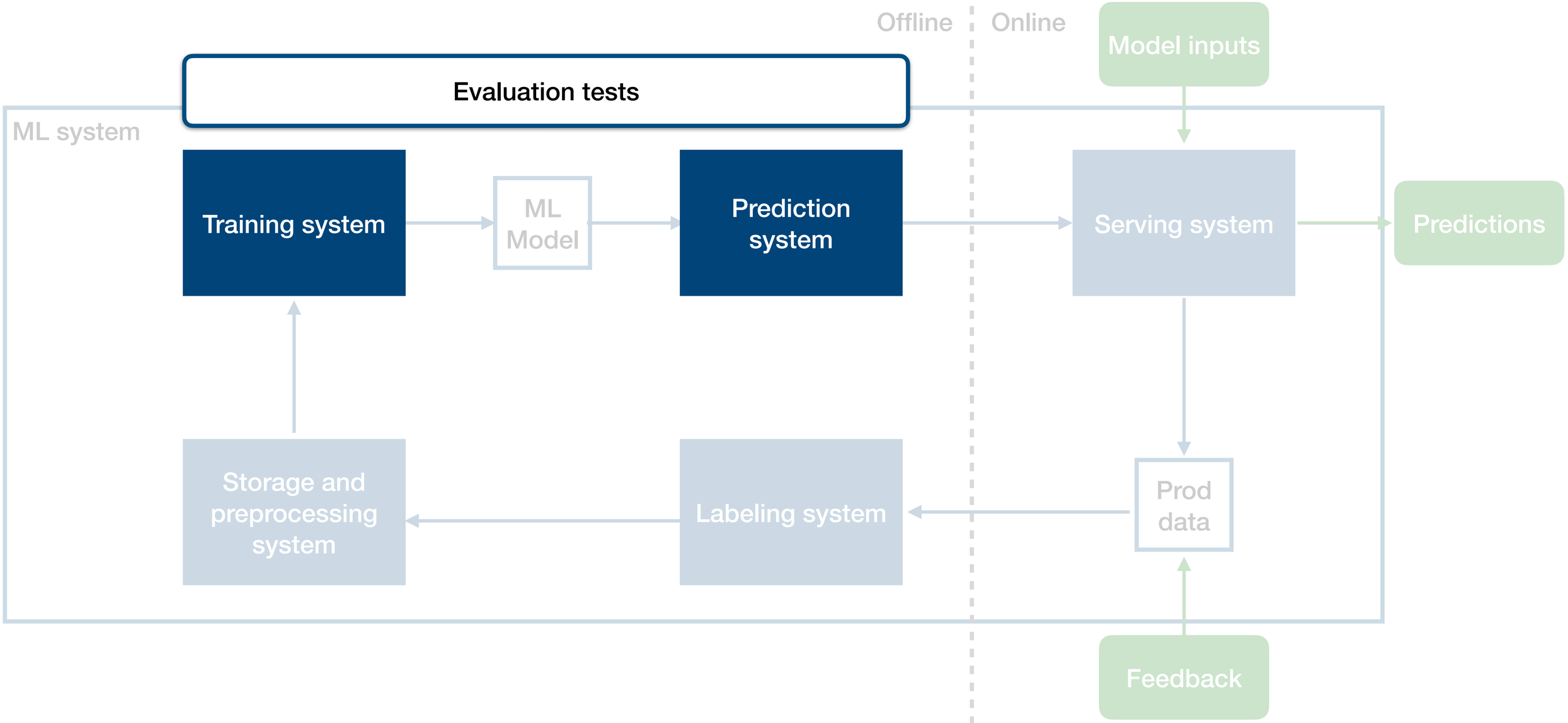
ML Models vs ML Systems



Functionality tests: unit tests for prediction code

- **Goal:** avoid regressions in the code that makes up your prediction infra
- **How?**
  - Unit test your prediction code like you would any other code
  - Load a pretrained model and test prediction on a few key examples
  - Run frequently during development

ML Models vs ML Systems



Evaluation tests

- **Goal:** make sure a new model is ready to go into production
- **How?**
  - Evaluate your model on all of the **metrics, datasets, and slices** that you care about
  - Compare the new model to the old one and your baselines
  - Understand the **performance envelope** of the new model
  - Run every time a new candidate model is created and considered for production

Eval tests: more than just your val score

- **Look at all of the metrics you care about**
  - Model metrics (precision, recall, accuracy, L2, etc)
  - Behavioral tests
  - Robustness tests
  - Privacy and fairness tests
  - Simulation tests

Behavioral testing metrics

- **Goal:** make sure the model has the invariances we expect
- Types:
  - Invariance tests: assert that change in inputs shouldn't affect outputs
  - Directional tests: assert that change in inputs should affect outputs
  - Min functionality tests: certain inputs and outputs should always produce a given result
- Mostly used in NLP

Capability	Min Func Test	INVariance	DIRectional
Vocabulary	Fail. rate=15.0%	16.2%	C 34.6%
NER	0.0%	B 20.8%	N/A
Negation	A 76.4%	N/A	N/A
...			

Test case	Expected	Predicted	Pass?
A Testing **Negation** with ***MFT***	Labels: negative, positive, neutral		
`Template: I {NEGATION} {POS_VERB} the {THING}.`			
I can't say I recommend the food.	neg	pos	x
I didn't love the flight.	neg	neutral	x
...			
		Failure rate = 76.4%	
B Testing **NER** with ***INV***	Same pred. (`inv`) after removals / additions		
@AmericanAir thank you we got on a different flight to [Chicago → Dallas].	`inv`	pos / neutral	x
@VirginAmerica I can't lose my luggage, moving to [Brazil → Turkey] soon, ugh.	`inv`	neutral / neg	x
...			
		Failure rate = 20.8%	
C Testing **Vocabulary** with ***DIR***	Sentiment monotonic decreasing (↓)		
@AmericanAir service wasn't great. You are lame.	↓	neg / neutral	x
@JetBlue why won't YOU help them?! Ugh. I dread you.	↓	neg / neutral	x
...			
		Failure rate = 34.6%	

Beyond Accuracy: Behavioral Testing of NLP models with CheckList

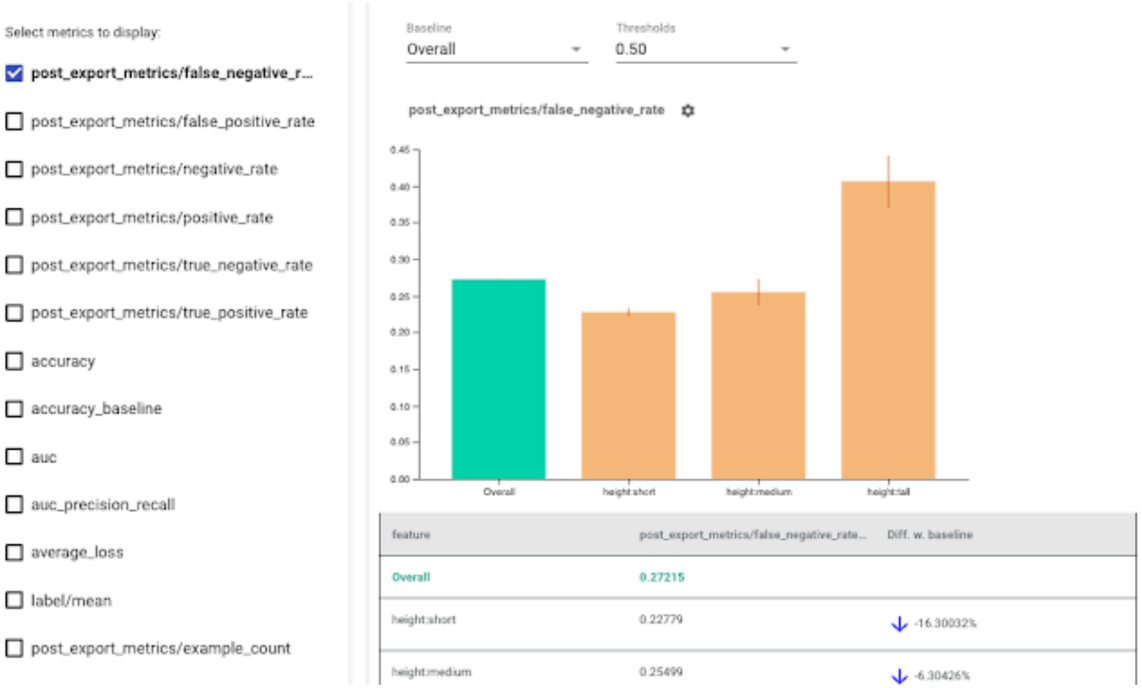
https://arxiv.org/abs/2005.04118

Robustness metrics

- Goal: understand the **performance envelope** of the model, i.e., where you should expect the model to fail, e.g.,
  - Feature importance
  - Sensitivity to staleness
  - Sensitivity to drift (how to measure this?)
  - Correlation between model performance and business metrics (on an old version of the model)
- Very underrated form of testing

Privacy and fairness metrics

https://ai.googleblog.com/2019/12/fairness-indicators-scalable.html



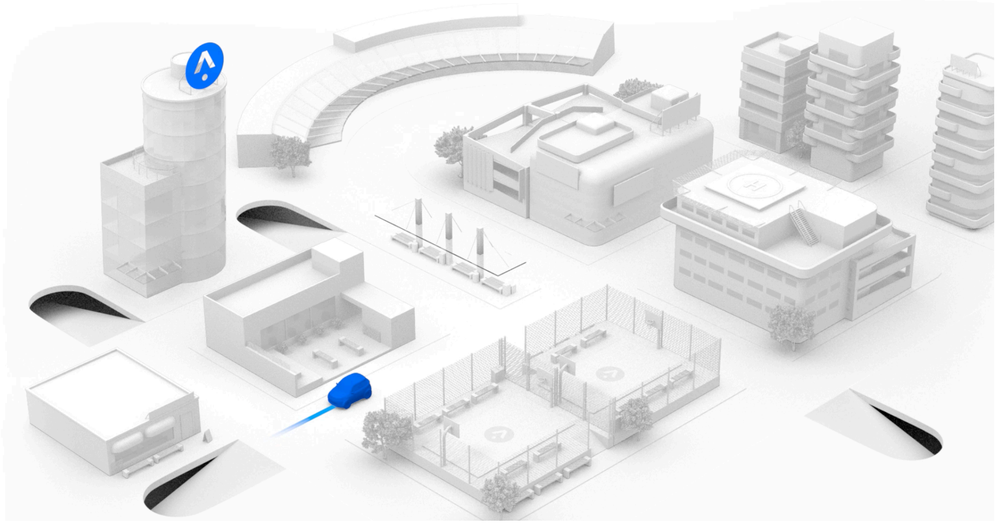
	Definition	Paper	Citation #	Result
3.1.1	Group fairness or statistical parity	[12]	208	×
3.1.2	Conditional statistical parity	[11]	29	✓
3.2.1	Predictive parity	[10]	57	✓
3.2.2	False positive error rate balance	[10]	57	×
3.2.3	False negative error rate balance	[10]	57	✓
3.2.4	Equalised odds	[14]	106	×
3.2.5	Conditional use accuracy equality	[8]	18	×
3.2.6	Overall accuracy equality	[8]	18	✓
3.2.7	Treatment equality	[8]	18	×
3.3.1	Test-fairness or calibration	[10]	57	✓
3.3.2	Well calibration	[16]	81	✓
3.3.3	Balance for positive class	[16]	81	✓
3.3.4	Balance for negative class	[16]	81	×
4.1	Causal discrimination	[13]	1	×
4.2	Fairness through unawareness	[17]	14	✓
4.3	Fairness through awareness	[12]	208	×
5.1	Counterfactual fairness	[17]	14	–
5.2	No unresolved discrimination	[15]	14	–
5.3	No proxy discrimination	[15]	14	–
5.4	Fair inference	[19]	6	–

Table 1: Considered Definitions of Fairness

Fairness Definitions Explained (https://fairware.cs.umass.edu/papers/Verma.pdf)

Simulation tests

- **Goal:** understand how the performance of the model could affect the rest of the system
- Useful when your model affects the world
- Main use is AVs and robotics
- This is hard: need a model of the world, dataset of scenarios, etc

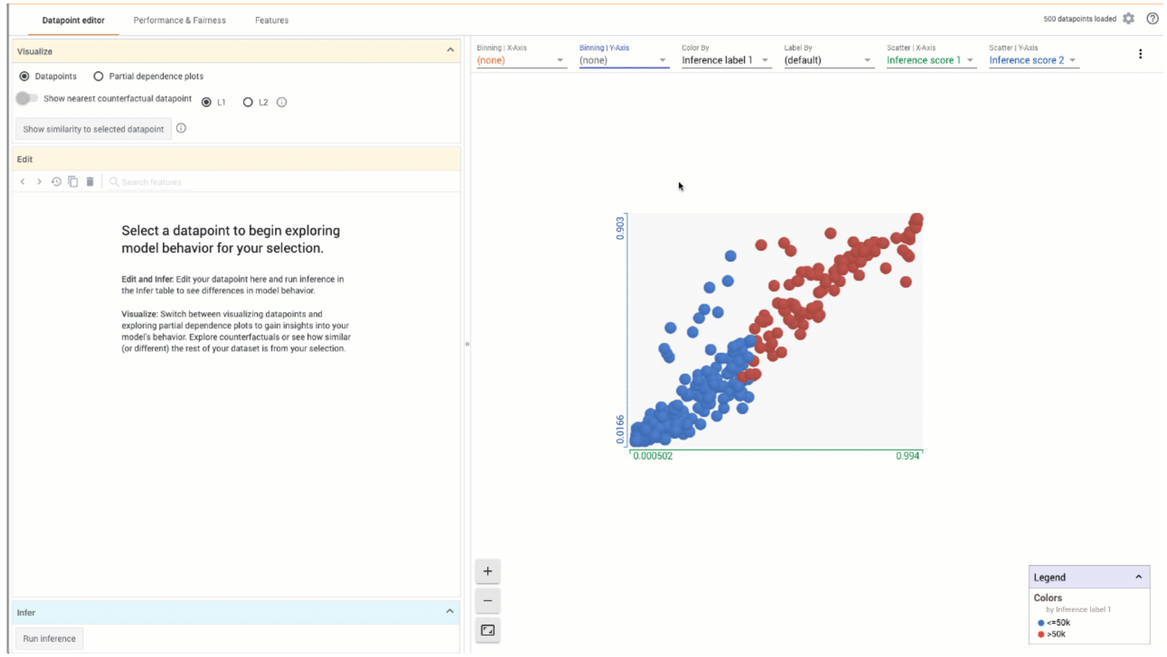


Eval tests: more than just your val score

- **Evaluate metrics on multiple slices of data**
  - Slices = mappings of your data to categories
    - E.g., value of the “country” feature
    - E.g., “age” feature within intervals {0 - 18, 18 - 30, 30-45, 45-65, 65+}
    - E.g., arbitrary other machine learning model like a noisy cat classifier

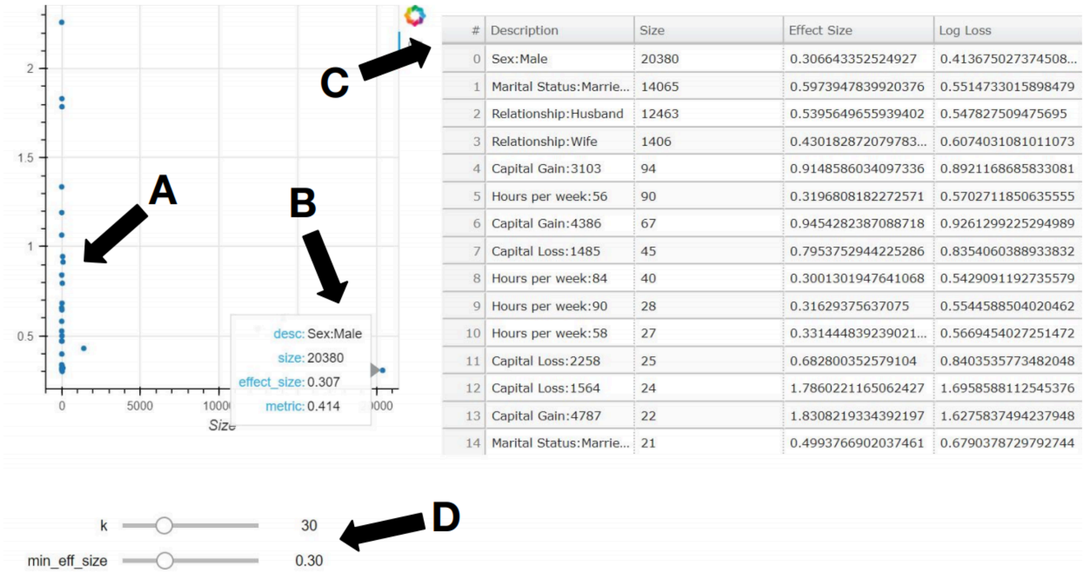
Surfacing slices

What-if tool: visual exploration tool for model performance slicing



The What-If Tool: Interactive Probing of Machine Learning Models

SliceFinder: clever way of computing metrics across all slices & surfacing the worst performers



#	Description	Size	Effect Size	Log Loss
0	Sex:Male	20380	0.306643352524927	0.413675027374508...
1	Marital Status:Marrie...	14065	0.5973947839920376	0.5514733015898479
2	Relationship:Husband	12463	0.5395649655939402	0.547827509475695
3	Relationship:Wife	1406	0.430182872079783...	0.6074031081011073
4	Capital Gain:3103	94	0.9148586034097336	0.8921168685833081
5	Hours per week:56	90	0.3196808182272571	0.5702711850635555
6	Capital Gain:4386	67	0.9454282387088718	0.9261299225294989
7	Capital Loss:1485	45	0.7953752944225286	0.8354060388933832
8	Hours per week:84	40	0.3001301947641068	0.5429091192735579
9	Hours per week:90	28	0.31629375637075	0.5544588504020462
10	Hours per week:58	27	0.331444839239021...	0.5669454027251472
11	Capital Loss:2258	25	0.682800352579104	0.8403535773482048
12	Capital Loss:1564	24	1.7860221165062427	1.6958588112545376
13	Capital Gain:4787	22	1.8308219334392197	1.6275837494237948
14	Marital Status:Marrie...	21	0.4993766902037461	0.6790378729792744

Fig. 3: The Slice Finder GUI helps users quickly browse through problematic slices by effect size and slice size on a scatter plot (A) and see a slice summary by hovering over any point (B); the user can sort slices by any metric and select on the scatter plot or table view. The selections are highlighted on the linked views (C). The user can also explore the top-k large problematic slices by varying the effect size threshold using the slider (*min_eff_size*) on the bottom left corner (D).

Automated Data Slicing for Model Validation: A Big data - AI Integration Approach

Eval tests: more than just your val score

- **Maintain evaluation datasets for all of the distinct data distributions you need to measure**
  - Your main validation set should mirror your test distribution
  - When to add new datasets?
    - When you collect datasets to specify specific edge cases (e.g., “left hand turn” dataset)
    - When you run your production model on multiple modalities (e.g., “San Francisco” and “Miami” datasets, English and French datasets)
    - When you augment your training set with data not found in production (e.g., synthetic data)

Evaluation reports

Main eval set

Slice	Accuracy	Precision	Recall
Aggregate	90%	91%	89%
Age <18	87%	89%	90%
Age 18-45	85%	87%	79%
Age 45+	92%	95%	90%

NYC users

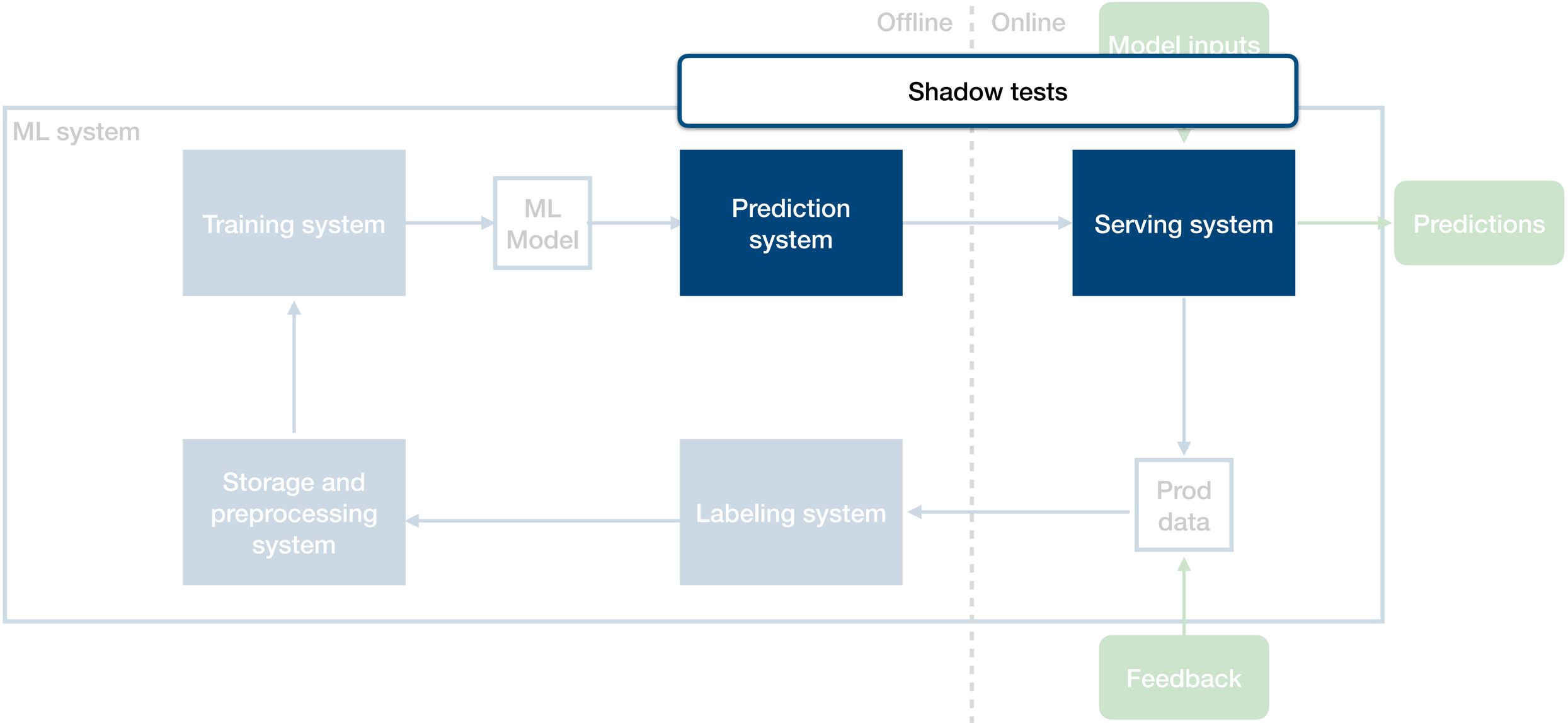
Slice	Accuracy	Precision	Recall
Aggregate	94%	91%	90%
New accounts	87%	89%	90%
Frequent users	85%	87%	79%
Churned users	50%	36%	70%

Deciding whether the evaluations pass

- Compare new model to **previous model** and **and a fixed older model**
  - Set threshold on the diffs between new and old for most metrics
  - Set thresholds on the diffs between slices
  - Set thresholds against the fixed older model to prevent slower performance “leaks”

Questions?

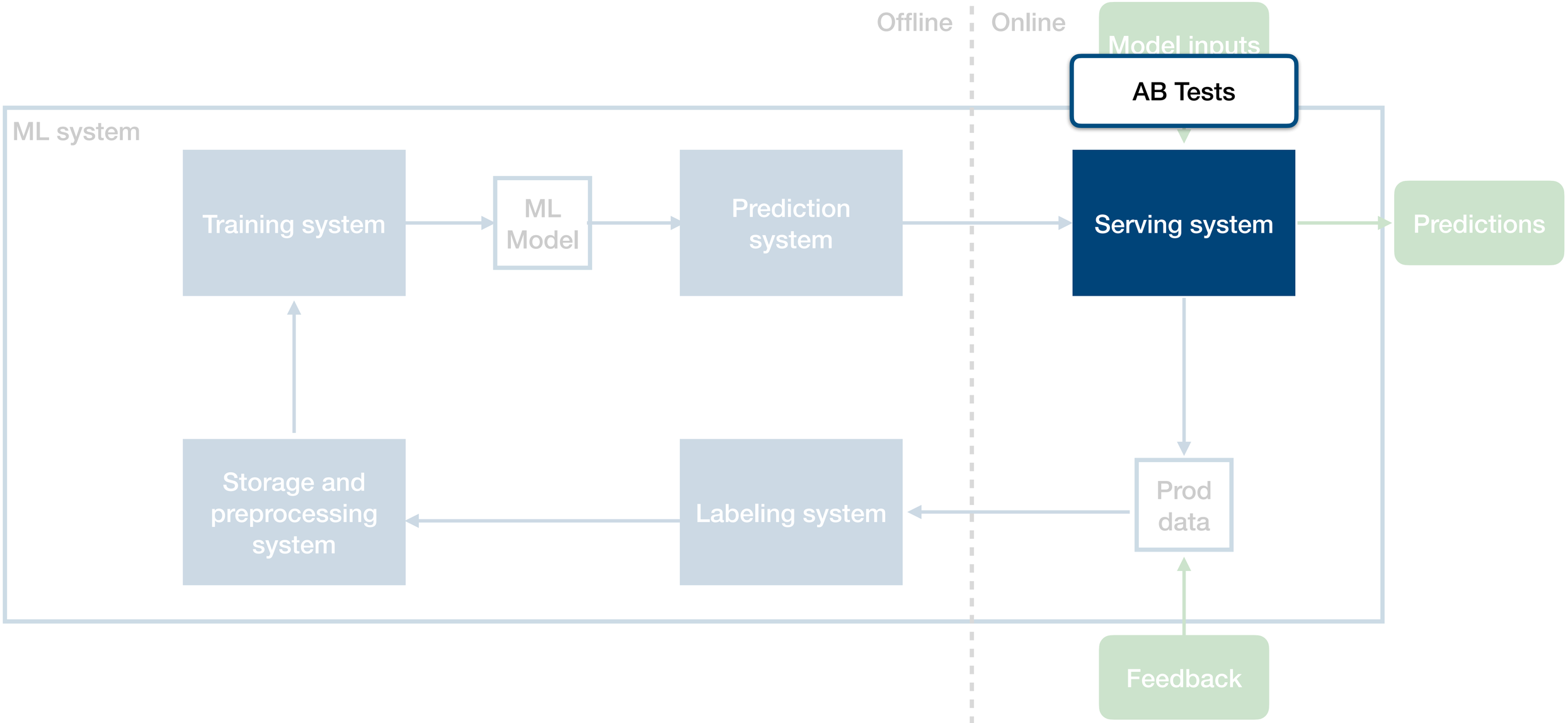
ML Models vs ML Systems



Shadow tests: catch production bugs before they hit users

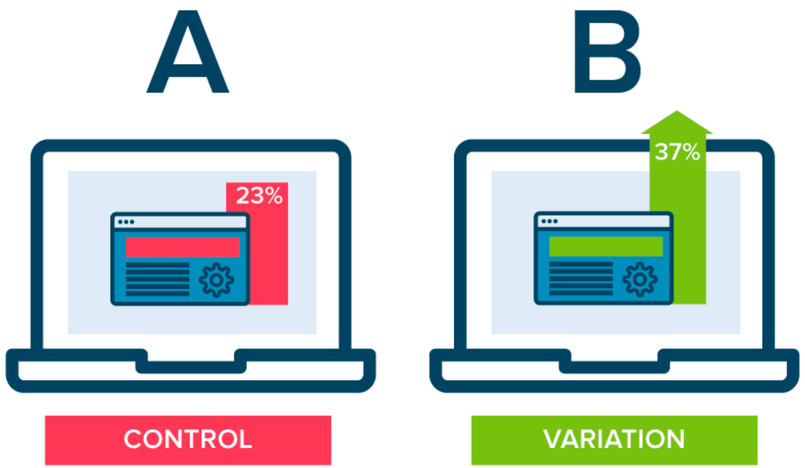
- **Goals:**
  - Detect bugs in the production deployment
  - Detect inconsistencies between the offline and online model
  - Detect issues that appear on production data
- **How?**
  - Run new model in production system, but don’t return predictions to users
  - Save the data and run the offline model on it
  - Inspect the prediction distributions for old vs new and offline vs online for inconsistencies

ML Models vs ML Systems



AB Testing: test users' reaction to the new model

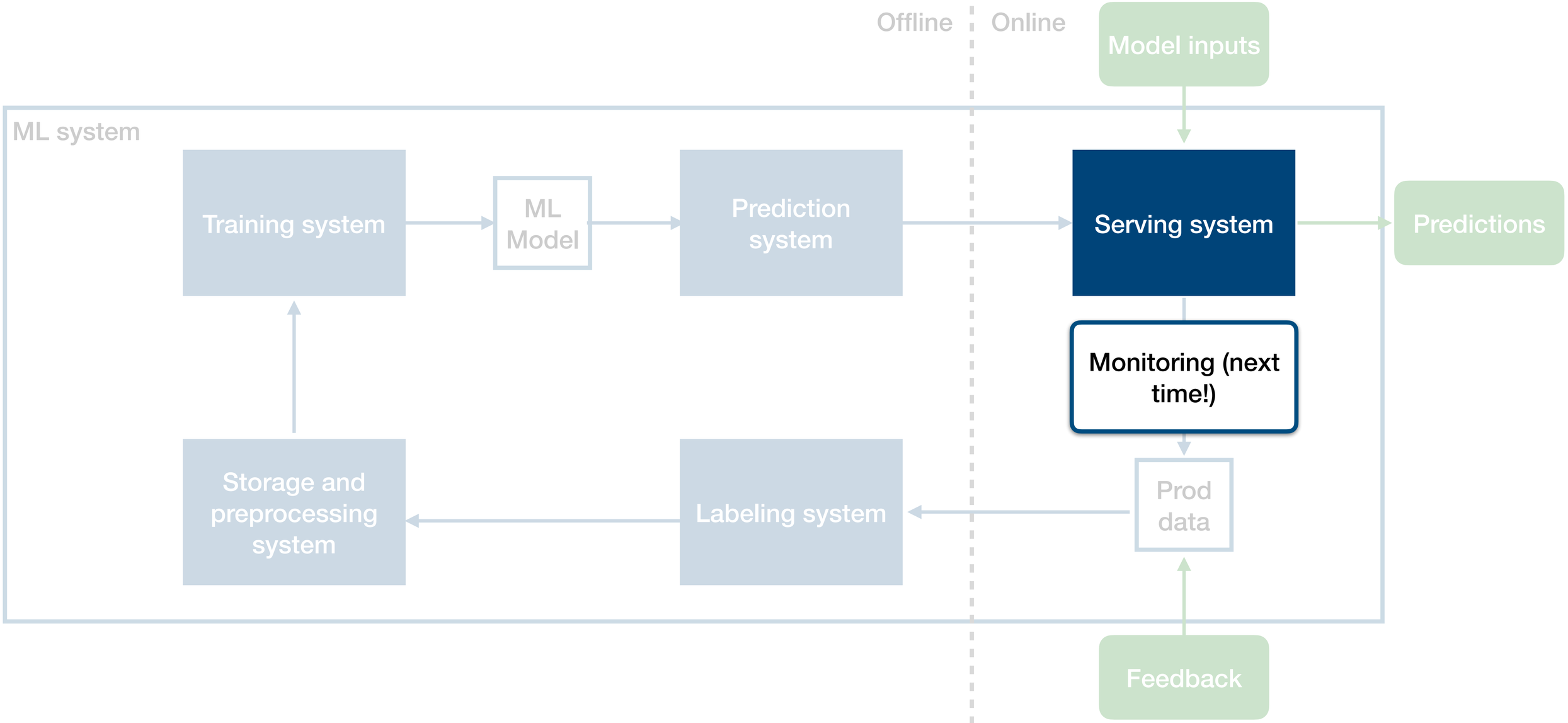
- **Goals:**
  - Understand how your new model affects user and business metrics
- **How?**
  - Start by "canarying" model on a tiny fraction of data
  - Consider using a more statistically principled split
  - Compare metrics on the two cohorts



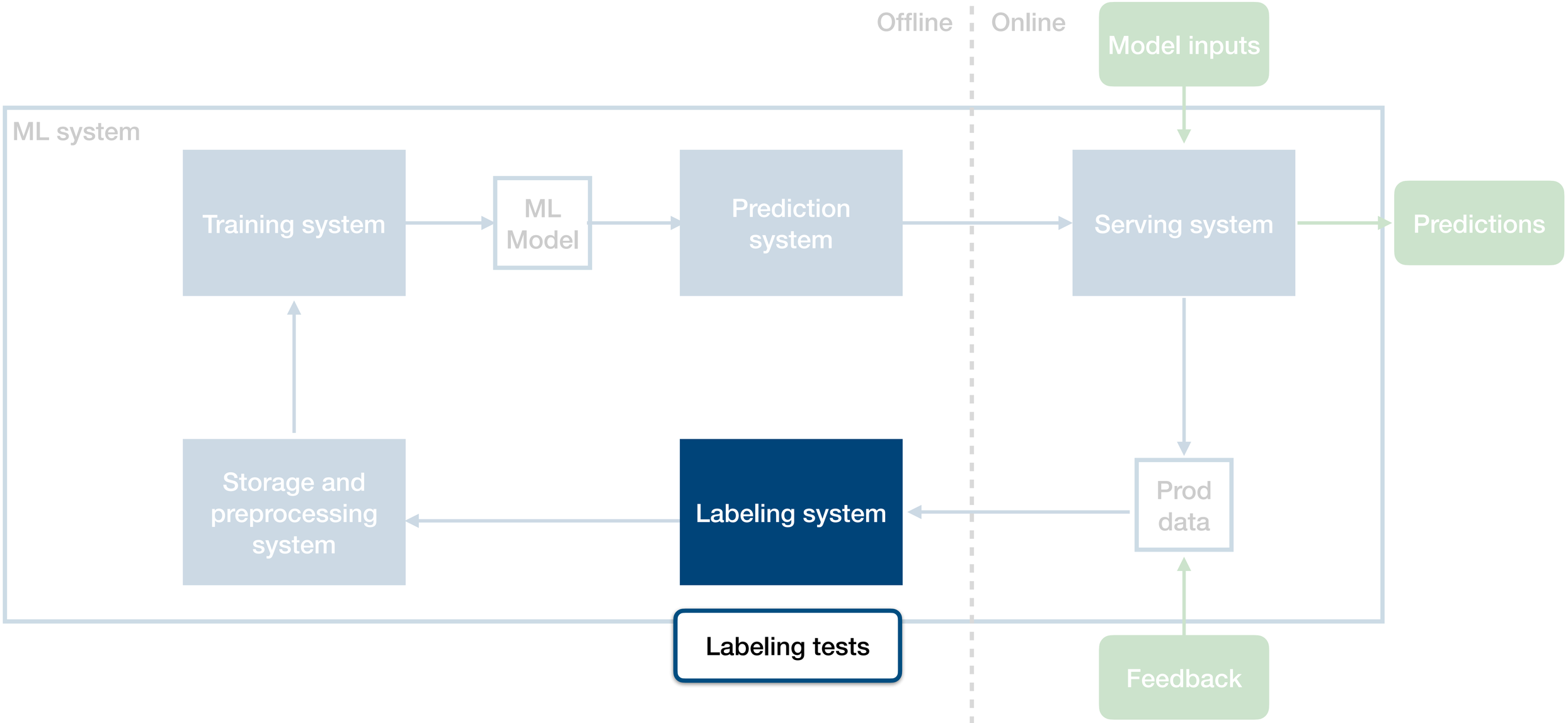
https://levelup.gitconnected.com/the-engineering-problem-of-a-b-testing-ac1adfd492a8

Questions?

ML Models vs ML Systems



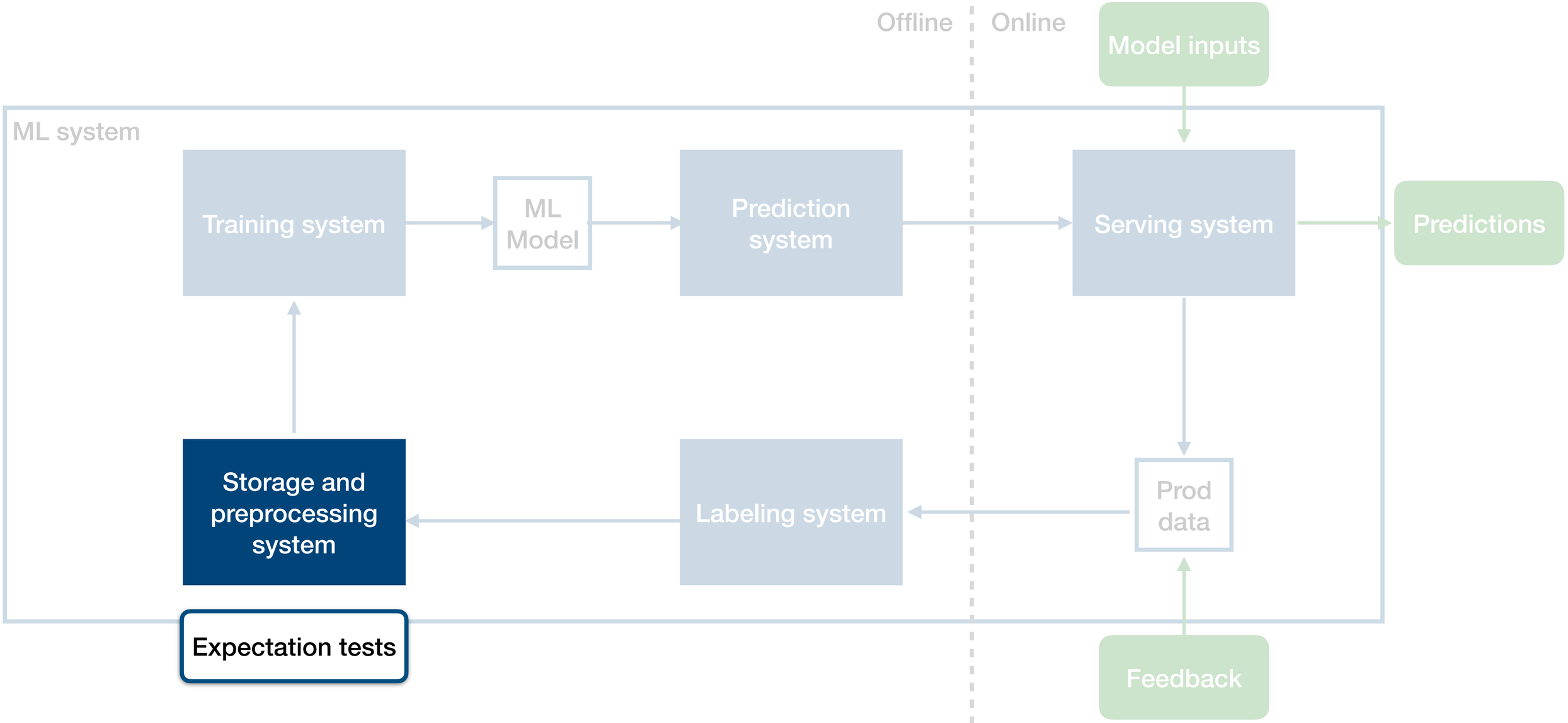
ML Models vs ML Systems



Labeling tests: get more reliable labels

- **Goals:**
  - Catch poor quality labels before they corrupt your model
- **How?**
  - Train and certify the labelers
  - Aggregate labels of multiple labelers
  - Assign labelers a “trust score” based on how often they are wrong
  - Manually spot check the labels from your labeling service
  - Run a previous model on new labels and inspect the ones with biggest disagreement

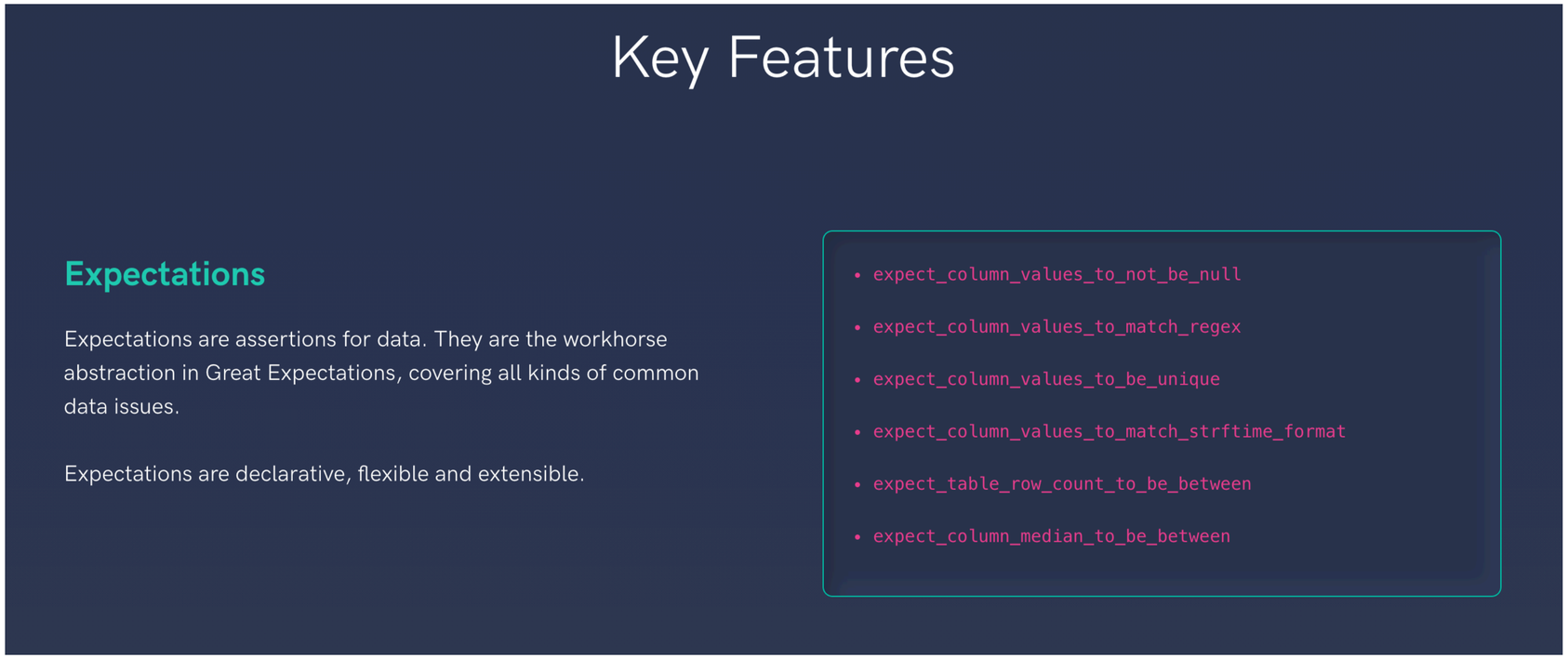
ML Models vs ML Systems



Expectation tests: unit tests for data

- **Goals:**
  - Catch data quality issues before they make your way into your pipeline
- **How?**
  - Define rules about properties of each of your data tables at each stage in your data cleaning and preprocessing pipeline
  - Run them when you run batch data pipeline jobs

What kinds of expectations?



https://greatexpectations.io/

How to set expectations?

- Manually
- Profile a dataset you expect to be high quality and use those thresholds
- Both

Challenges operationalizing ML tests

- **Organizational:** data science teams don't always have the same norms around testing & code review
- **Infrastructure:** cloud CI/CD platforms don't have great support for GPUs or data integrations
- **Tooling:** no standard toolset for comparing model performance, finding slices, etc
- **Decision making:** Hard to determine when test performance is "good enough"

How to test your ML system the right way

- Test each part of the ML system, not just the model
- Test code, data, and model performance, not just code
- Testing model performance is an art, not a science
- The goal of testing model performance is to **build a granular understanding** of how well your model performs, and where you don't expect it to perform well
- Build up to this gradually! Start here:
  - Infrastructure tests
  - Evaluation tests
  - Expectation tests

Questions?

Outline

- Software testing
- Testing machine learning systems
- **Explainable and interpretable AI**

Some definitions

- **Domain predictability:** the degree to which it is possible to detect data outside the model's domain of competence
- **Interpretability:** the degree to which a human can consistently predict the model's result [1]
- **Explainability:** the degree to which a human can understand the cause of a decision [2]

[1] Kim, Been, Rajiv Khanna, and Oluwasanmi O. Koyejo. "Examples are not enough, learn to criticize! Criticism for interpretability." Advances in Neural Information Processing Systems (2016).

[2] Miller, Tim. "Explanation in artificial intelligence: Insights from the social sciences." arXiv Preprint arXiv:1706.07269. (2017).

Making models interpretable / explainable

- Use an interpretable family of models
- Distill the complex model to an interpretable one
- Understand the contribution of features to the prediction (feature importance, SHAP)
- Understand the contribution of training data points to the prediction

Making models interpretable / explainable

- **Use an interpretable family of models**
- Distill the complex model to an interpretable one
- Understand the contribution of features to the prediction (feature importance, SHAP)
- Understand the contribution of training data points to the prediction

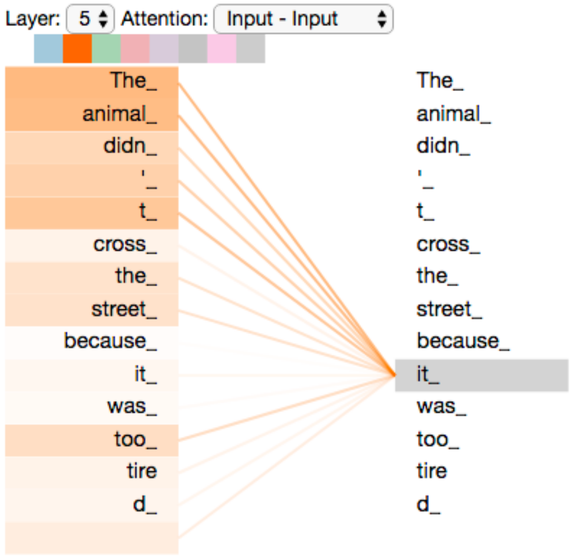
Interpretable families of models

- Linear regression
- Logistic regression
- Generalized linear models
- Decision trees

Interpretable **and** explainable (to the right user)

Not very powerful

What about attention?



A woman is throwing a frisbee in a park.

A dog is standing on a hardwood floor.

A stop sign is on a road with a mountain in the background.

A little girl sitting on a bed with a teddy bear.

A group of people sitting on a boat in the water.

A giraffe standing in a forest with trees in the background.

Interpretable

Not explainable

Attention maps are not reliable explanations

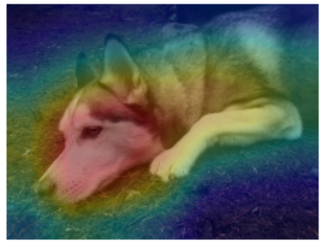
	Test Image	Evidence for Animal Being a Siberian Husky	Evidence for Animal Being a Transverse Flute
Explanations Using Attention Maps			

Figure 2: Saliency does not explain anything except where the network is looking. We have no idea why this image is labeled as either a dog or a musical instrument when considering only saliency. The explanations look essentially the same for both classes. Figure credit: Chaofan Chen and [28].

Questions?

Making models interpretable / explainable

- Use an interpretable family of models
- **Distill the complex model to an interpretable one**
- Understand the contribution of features to the prediction (feature importance, SHAP)
- Understand the contribution of training data points to the prediction

Surrogate models

- Train an interpretable **surrogate** model on your data and your original model's predictions
- Use the surrogate's interpretation as a proxy for understanding the underlying model

Easy to use, general

If your surrogate is good, why not use it? And how do we know it decides the same way

Local surrogate models (LIME)

- Pick a data point to explain the prediction for
- Perturb the data point and query the model for the prediction
- Train a surrogate on the perturbed dataset

Widely used in practice.
Works for all data types

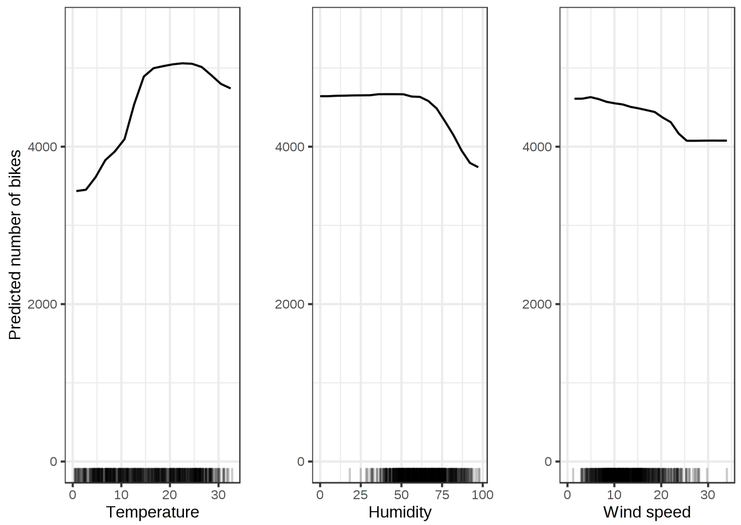
Hard to define the perturbations.
Explanations are unstable and can be manipulated

Making models interpretable / explainable

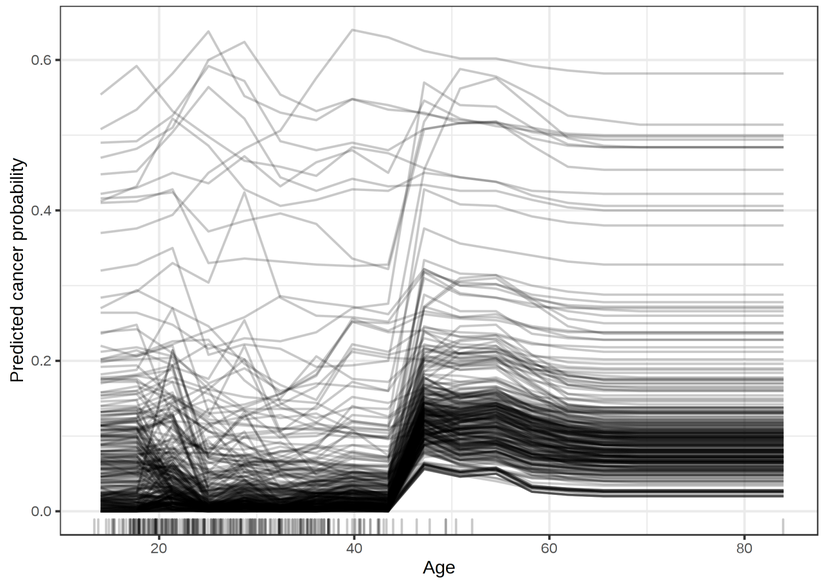
- Use an interpretable family of models
- Distill the complex model to an interpretable one
- **Understand the contribution of features to the prediction**
- Understand the contribution of training data points to the prediction

Data viz for feature importance

Partial dependence plot



Individual conditional expectation



Permutation feature importance

- Pick a feature
- Randomize its order in the dataset and see how that affects performance

Easy to use. Widely used in practice

Doesn’t work for high-dim data. Doesn’t capture feature interdependence

SHAP (SHapley Additive exPlanations)

- Intuition:
  - How much does the pretense of this feature affect the value of the classifier, independent of what other features are present?

Works for a variety of data. Mathematically principled

Tricky to implement. Still doesn't provide explanations

Gradient-based saliency maps

- Pick an input (e.g., image)
- Perform a forward pass
- Compute the gradient with respect to the pixels
- Visualize the gradients

Easy to use. Many variants that produce more interpretable results.

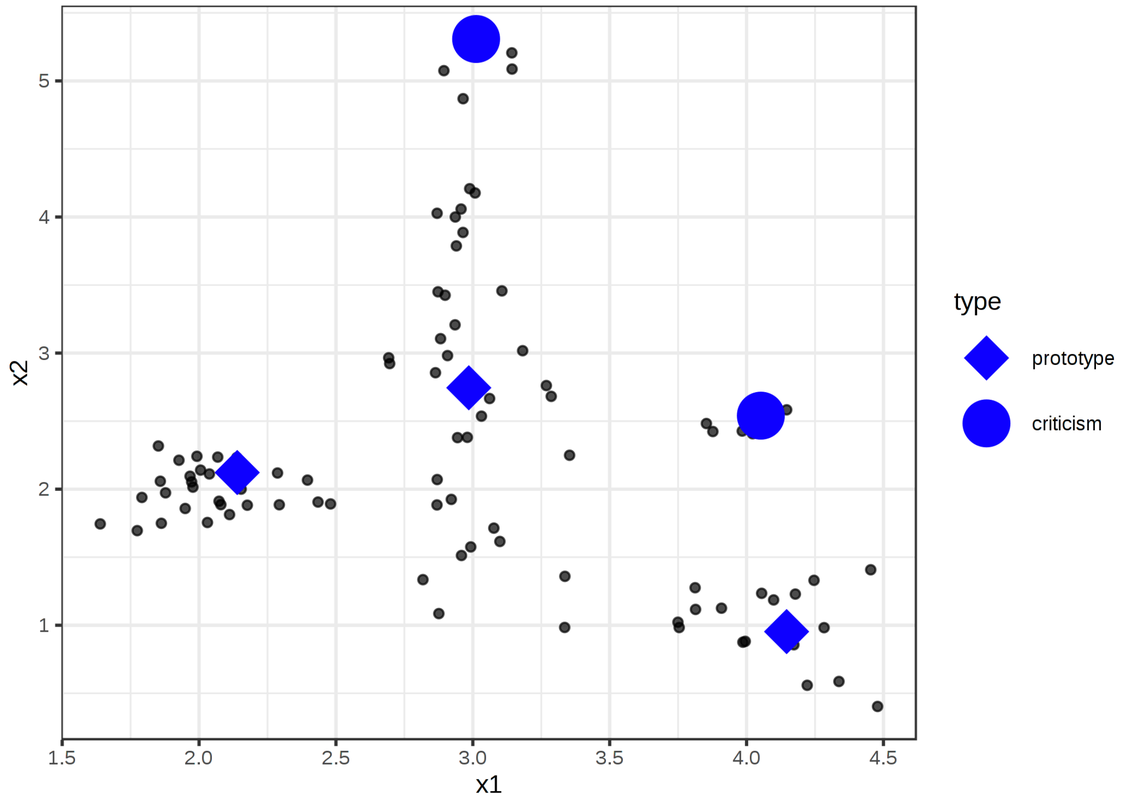
Difficult to know whether the explanation is correct. Fragile. Unreliable.

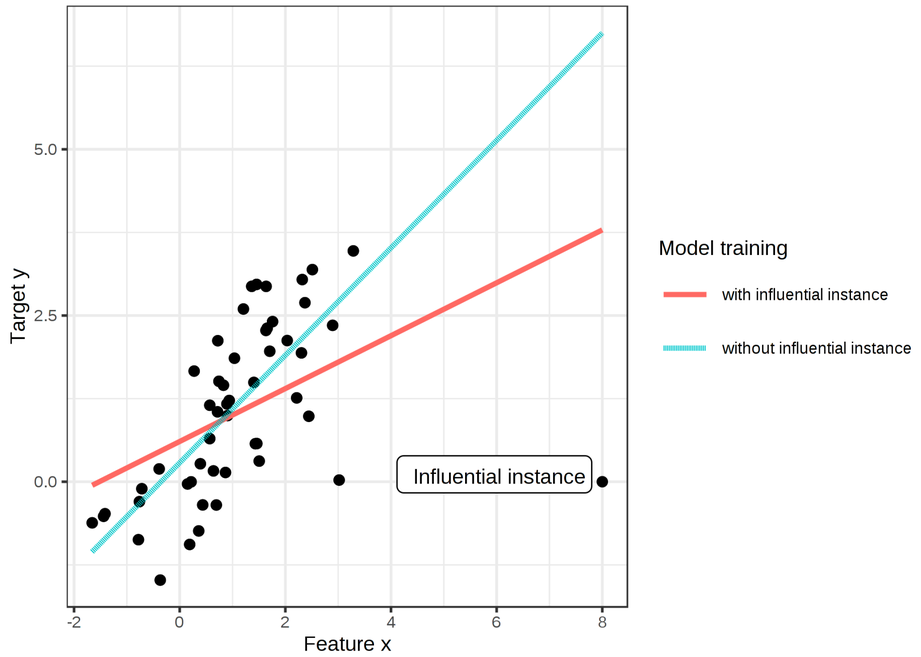
Questions?

Making models interpretable / explainable

- Use an interpretable family of models
- Distill the complex model to an interpretable one
- Understand the contribution of features to the prediction
- **Understand the contribution of training data points to the prediction**

Prototypes and criticisms





Do you need “explainability”?

- Regulators say my model needs to be explainable
  - What do they mean?
  - But yeah you should probably do explainability
- Users want my model to be explainable
  - Can a better product experience alleviate the need for explainability?
  - How often will they be interacting with the model? If frequently, maybe interpretability is enough?
- I want to deploy my models to production more confidently
  - You really want domain predictability
  - However, interpretable visualizations could be a helpful tool for debugging

Can you explain deep learning models?

- Not really
  - Known explanation methods aren’t faithful to original model performance
  - Known explanation methods are unreliable
  - Known explanations don’t fully explain models’ decisions

Stop Explaining Black Box Machine Learning Models for High Stakes Decisions and Use Interpretable Models Instead (https://arxiv.org/pdf/1811.10154.pdf)

Conclusion

- If you need to explain your model's predictions, use an interpretable model family
- Explanations for deep learning models can produce interesting results but are unreliable
- Interpretability methods like SHAP, LIME, etc can help users reach the interpretability threshold faster
- Interpretable visualizations can be helpful for debugging

Resources

- ML Test Score Paper (https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/aad9f93b86b7addfea4c419b9100c6cdd26cacea.pdf)
- Behavioral testing paper (https://arxiv.org/abs/2005.04118)
- Jeremy Jordan's effective testing (https://www.jeremyjordan.me/testing-ml/)
- Robustness Gym (https://arxiv.org/abs/2101.04840)
- Made with ML's guide to testing (https://madewithml.com/courses/applied-ml/testing/)
- Eugene Yan's practical guide to maintaining machine learning (https://eugeneyan.com/writing/practical-guide-to-maintaining-machine-learning/)
- Chip Huyen's CS329 lecture on evaluating models (https://stanford-cs329s.github.io/syllabus.html)
- Interpretible ML Book (https://christophm.github.io/interpretable-ml-book/)

Thank you!