



# Machine Learning in Production

# Testing in Production

# Logistics

- Project M1 dues TODAY
- Schedule debriefing meetings with your TA mentor, before next Tuesday
- Midterm exam next week (10/9), regular time and place for lectures, see #announcements for details
  - Questions based on shared scenario, apply concepts
  - All lectures and readings in scope, focus on concepts with opportunity to practice (e.g., recitations, homeworks, in-class exercises)
  - Closed book, but 6 sheets of notes (sorry, no ChatGPT)
  - Lab for next week (10/11) is cancelled



# Back to QA...

## Fundamentals of Engineering AI-Enabled Systems

**Holistic system view:** AI and non-AI components, pipelines, stakeholders, environment interactions, feedback loops

### Requirements:

- System and model goals
- User requirements
- Environment assumptions
- Quality beyond accuracy
- Measurement
- Risk analysis
- Planning for mistakes

### Architecture + design:

- Modeling tradeoffs
- Deployment architecture
- Data science pipelines
- Telemetry, monitoring
- Anticipating evolution
- Big data processing
- Human-AI design

### Quality assurance:

- Model testing
- Data quality
- QA automation
- Testing in production
- Infrastructure quality
- Debugging

### Operations:

- Continuous deployment
- Contin. experimentation
- Configuration mgmt.
- Monitoring
- Versioning
- Big data
- DevOps, MLOps

**Teams and process:** Data science vs software eng. workflows, interdisciplinary teams, collaboration points, technical debt

## Responsible AI Engineering

Provenance,  
versioning,  
reproducibility

Safety

Security and  
privacy

Fairness

Interpretability  
and explainability

Transparency  
and trust

Ethics, governance, regulation, compliance, organizational culture

# Learning Goals

- Design telemetry for evaluation in practice
- Understand the rationale for beta tests and chaos experiments
- Plan and execute experiments (chaos, A/B, shadow releases, ...) in production
- Conduct and evaluate multiple concurrent A/B tests in a system
- Perform canary releases
- Examine experimental results with statistical rigor
- Support data scientists with monitoring platforms providing insights from production data



# Readings

## Required Reading:

- Hulten, Geoff. "[Building Intelligent Systems: A Guide to Machine Learning Engineering.](#)" Apress, 2018, Chapters 14 and 15 (Intelligence Management and Intelligent Telemetry).

## Suggested Readings:

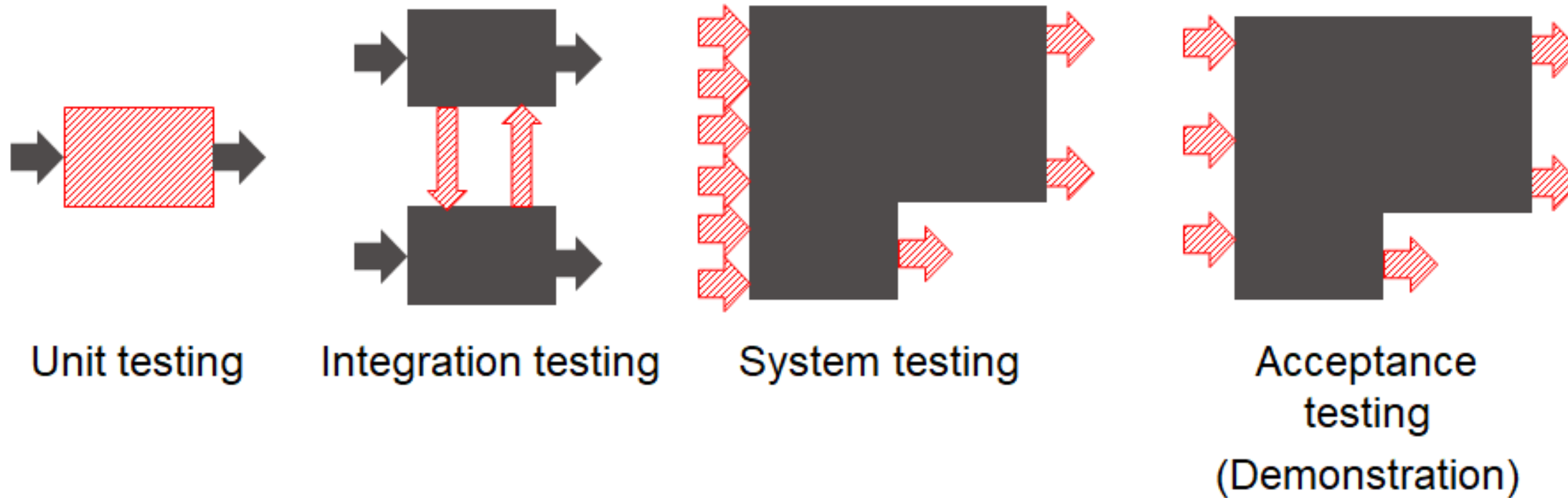
- Alec Warner and Štěpán Davidovič. "[Canary Releases.](#)" in [The Site Reliability Workbook](#), O'Reilly 2018
- Kohavi, Ron, Diane Tang, and Ya Xu. "[Trustworthy Online Controlled Experiments: A Practical Guide to A/B Testing.](#)" Cambridge University Press, 2020.

# From Unit Tests to Testing in Production

*(in traditional software systems)*

# Unit Test, Integration Tests, System Tests

These are all testing *before* deployment --- Manual or automated





Speaker notes

Testing before release. Manual or automated.



# Beta Testing: Move towards testing-during-deployment

Early release to select users, asking them to send feedback or report issues.

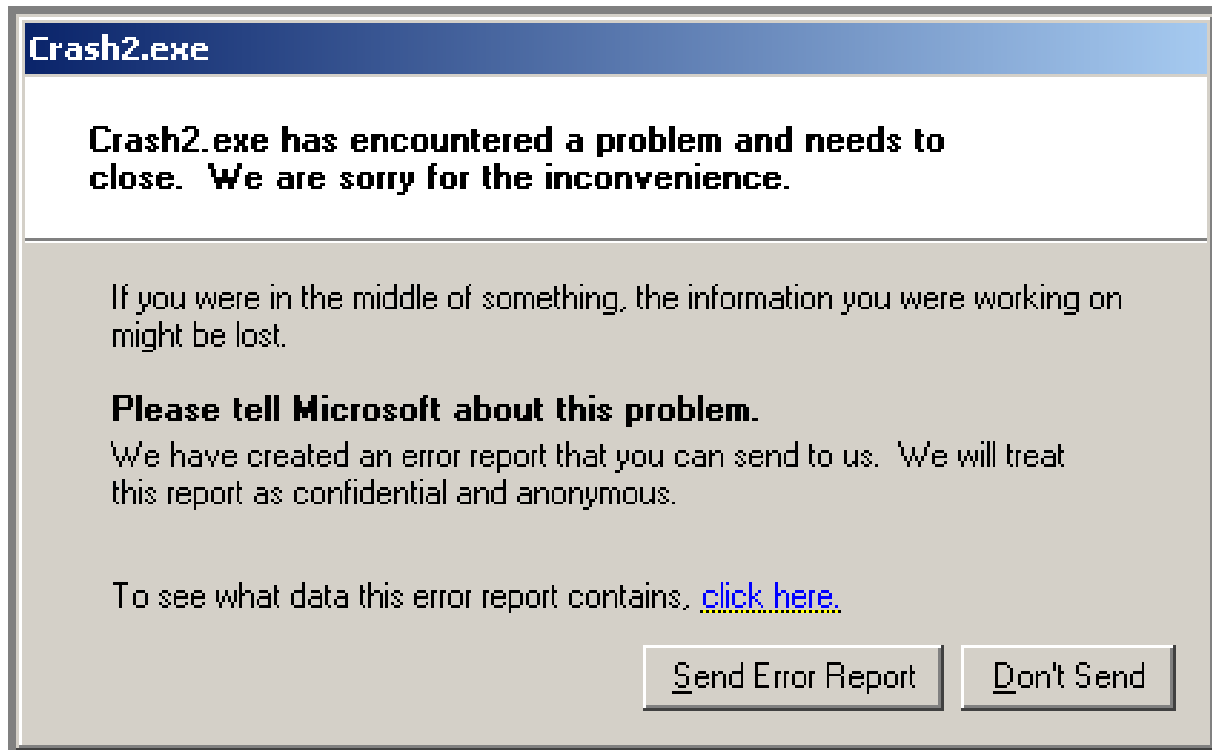
## Speaker notes

Early release to select users, asking them to send feedback or report issues. No telemetry in early days.



# Crash Telemetry: "Bugs" in Production

- **Telemetry:** The automated collection, transmission, and analysis of data about the performance, behavior, and usage of an application or system in real time.
- With internet availability, send crash reports home to identify problems "in production".



# Model Assessment in Production

Ultimate held-out evaluation data: Unseen real user data

# Recap: Limitations of Offline Model Evaluation

Training and test data drawn from the same population

- **i.i.d.: independent and identically distributed**
- leakage and overfitting problems quite common

Is the population representative of production data?

- If not or only partially or not anymore: Does the model generalize beyond training data?

# Model Assessment in Production

Live observation in the running system

Need **telemetry** to evaluate quality --- automated collection and monitoring of data on model performance, data quality, and user behavior to ensure continuous model accuracy, detect drift, and maintain system health in real-time.



# Discuss how to collect data

How do you gather data without being intrusive (i.e., labeling outcomes), without harming user experience?

- Was the house price predicted correctly?
- Was the ranking of search results good?
- Did the profanity filter remove the right blog comments?
- Was there cancer in the image?
- Was a Spotify playlist good?
- Was the weather prediction good?
- Was the translation correct?
- Did the self-driving car break at the right moment? Did it detect the pedestrians?

More:

- SmartHome: Does it automatically turn of the lights/lock the doors/close the window at the right time?
- Profanity filter: Does it block the right blog comments?
- News website: Does it pick the headline alternative that attracts a user's attention most?
- Autonomous vehicles: Does it detect pedestrians in the street?

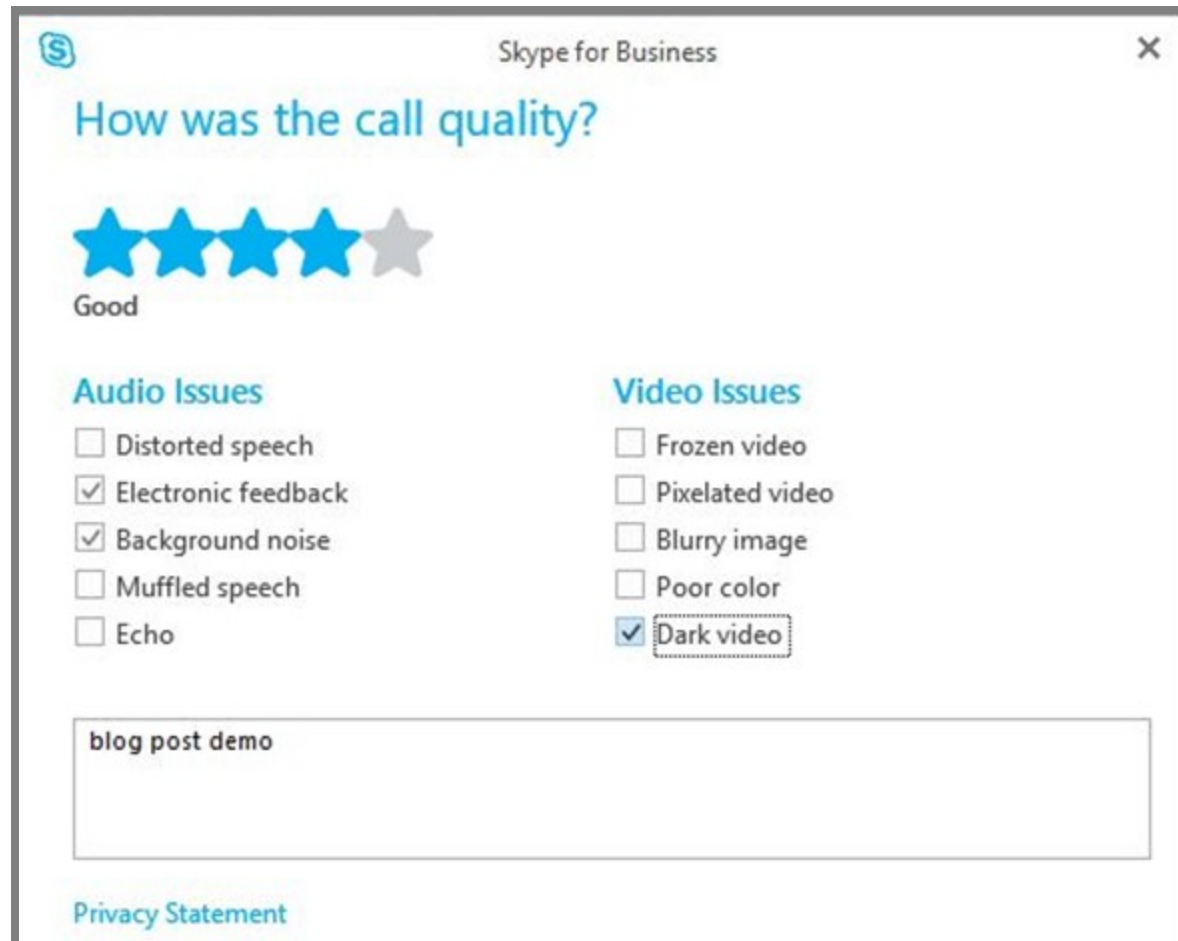
# Telemetry strategy: Manually Label Production Samples

Similar to labeling learning and testing data, have human annotators



# Telemetry strategy: Explicit Report

Expect only sparse feedback and expect negative feedback over-proportionally



Skype for Business

How was the call quality?

★★★★☆  
Good

**Audio Issues**

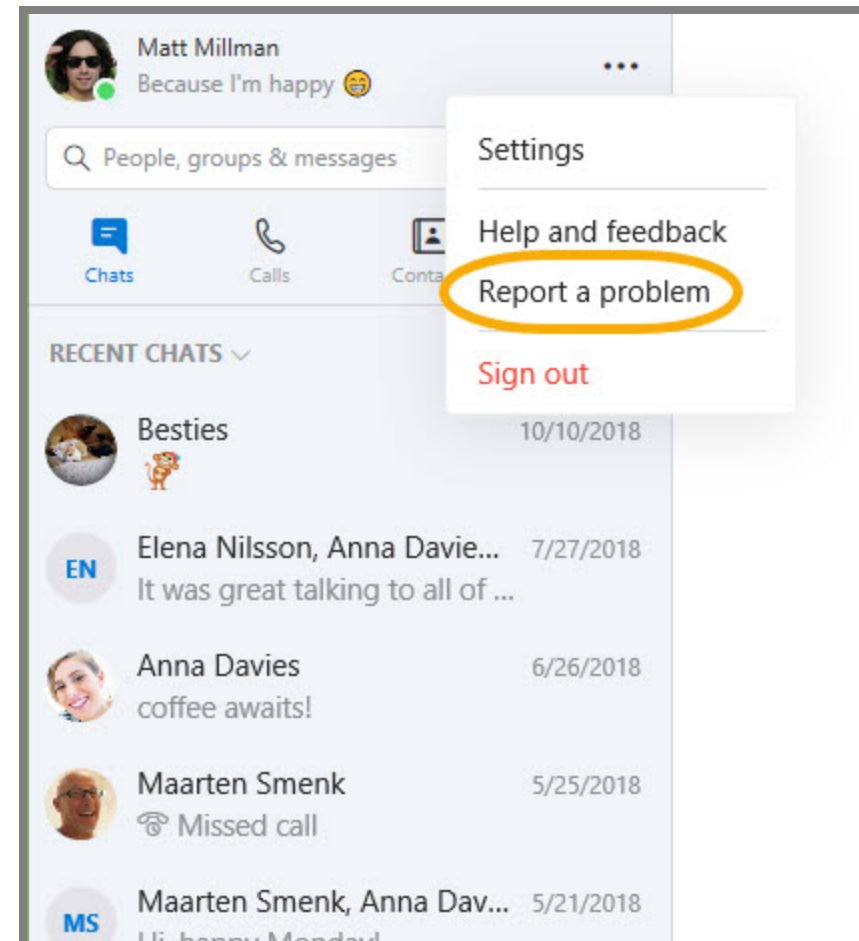
- Distorted speech
- Electronic feedback
- Background noise
- Muffled speech
- Echo

**Video Issues**

- Frozen video
- Pixelated video
- Blurry image
- Poor color
- Dark video

blog post demo

[Privacy Statement](#)



Matt Millman  
Because I'm happy 😊

People, groups & messages

Chats Calls Contacts

RECENT CHATS

- Besties 10/10/2018
- Elena Nilsson, Anna Davie... 7/27/2018  
It was great talking to all of ...
- Anna Davies 6/26/2018  
coffee awaits!
- Maarten Smenk 5/25/2018  
☎ Missed call
- Maarten Smenk, Anna Dav... 5/21/2018  
Hi, happy Monday!

Settings  
Help and feedback  
**Report a problem**  
Sign out

## Speaker notes

Expect only sparse feedback and expect negative feedback over-proportionally



# Telemetry strategy: Implicit Feedback

Collect user editing ("human-AI collaboration"); Can have indicators for low confidence predictions

## Speaker notes

Clever UI design allows users to edit transcripts. UI already highlights low-confidence words, can





# Telemetry strategy: Wait-and-See (Retrospective)

Can just wait 7 days to see actual outcome for all predictions



# Summary: Telemetry Strategies

- Manual/crowd-source labeling on production samples
- Explicit report: Ask users or allow users to complain
- Observe user reaction (implicit feedback)
- Wait and see

Ordered by -- How much efforts needed

# Measuring Model Quality with Telemetry

Three steps, again on measurements: (1) Metric, (2) data collection (telemetry), (3) operationalization

- Telemetry can provide insights for correctness
  - sometimes very accurate labels for real unseen data
  - sometimes only mistakes
  - sometimes delayed
  - often just samples
  - often just weak proxies for correctness
- Often sufficient to *approximate* precision/recall or other model-quality measures
- Mismatch to (static) evaluation set may indicate stale or unrepresentative data
- Trend analysis can provide insights even for inaccurate proxy measures

# Breakout: Design Telemetry in Production

Discuss how to collect telemetry (manual/crowd-source labeling, wait and see, ask users, observe user reaction), the metric to monitor, and how to operationalize.

Scenarios:

- Front-left: Amazon: Shopping app detects the shoe brand from photos
- Front-right: Google: Tagging uploaded photos with friends' names
- Back-left: Spotify: Recommended personalized playlists
- Back-right: Wordpress: Profanity filter to moderate blog posts

As a group post to #lecture and tag team members:

- *Quality metric:*
- *Data to collect:*
- *Operationalization:*

Speaker notes

about 30 minutes to here



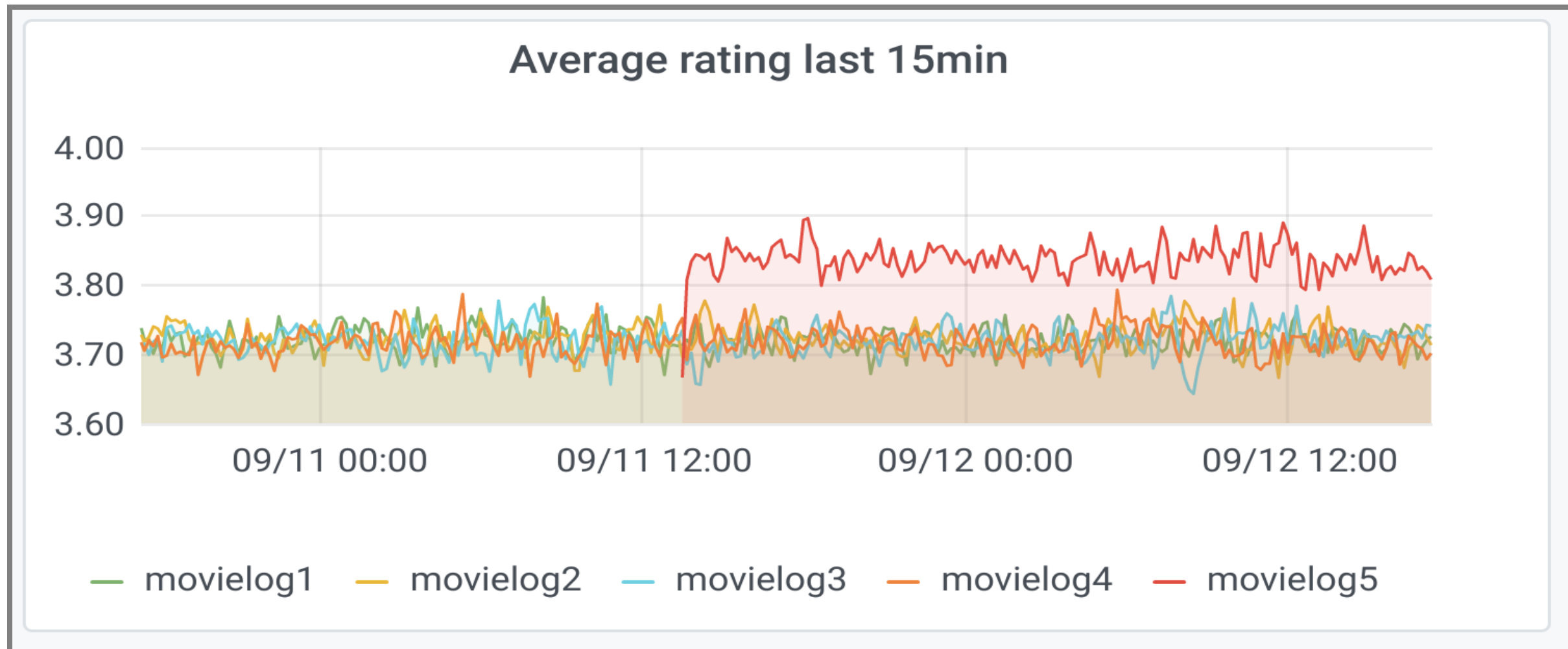
# Monitoring Model Quality in Production

- Monitor model quality together with other quality attributes (e.g., uptime, response time, load)
- Set up automatic alerts...
  - when model quality drops
  - Watch for jumps after releases (roll back after negative jump)
  - Watch for slow degradation (Stale models, data drift, feedback loops, adversaries)
- Debug common or important problems
  - Monitor characteristics of requests
  - Mistakes uniform across populations?
  - Challenging problems -> refine training, add regression tests

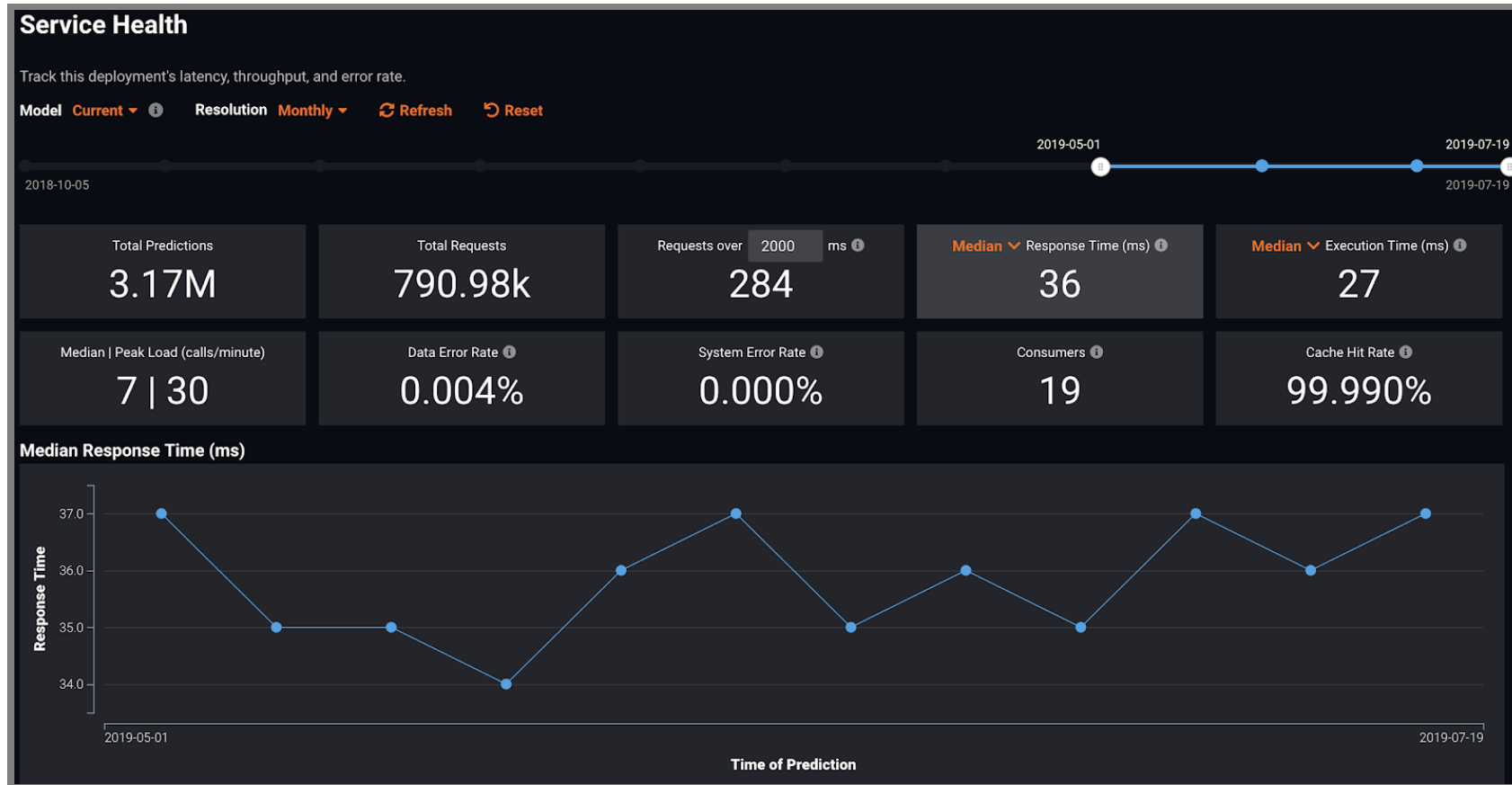


# Monitoring services: Prometheus and Grafana

# Grafana screenshot from Movie Recommendation Service



# Many commercial solutions



e.g. <https://www.datarobot.com/platform/mlops/>

≡ Many pointers: Ori Cohen "[Monitor! Stop Being A Blind Data-Scientist.](#)" Blog 2019

# Many different monitoring possibilities, e.g. Detecting Drift

Deploy monitoring at...

Training data

- Schema & distribution of incoming data
- Distribution of labels

Requests & predictions

- Schema & distribution of requests
- Distribution of predictions
- Quality of predictions

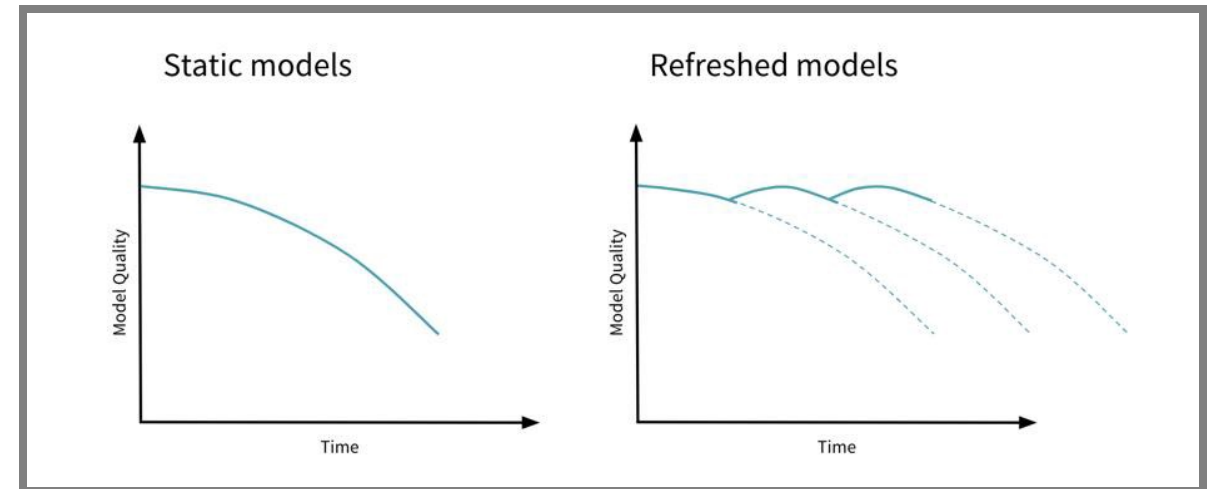
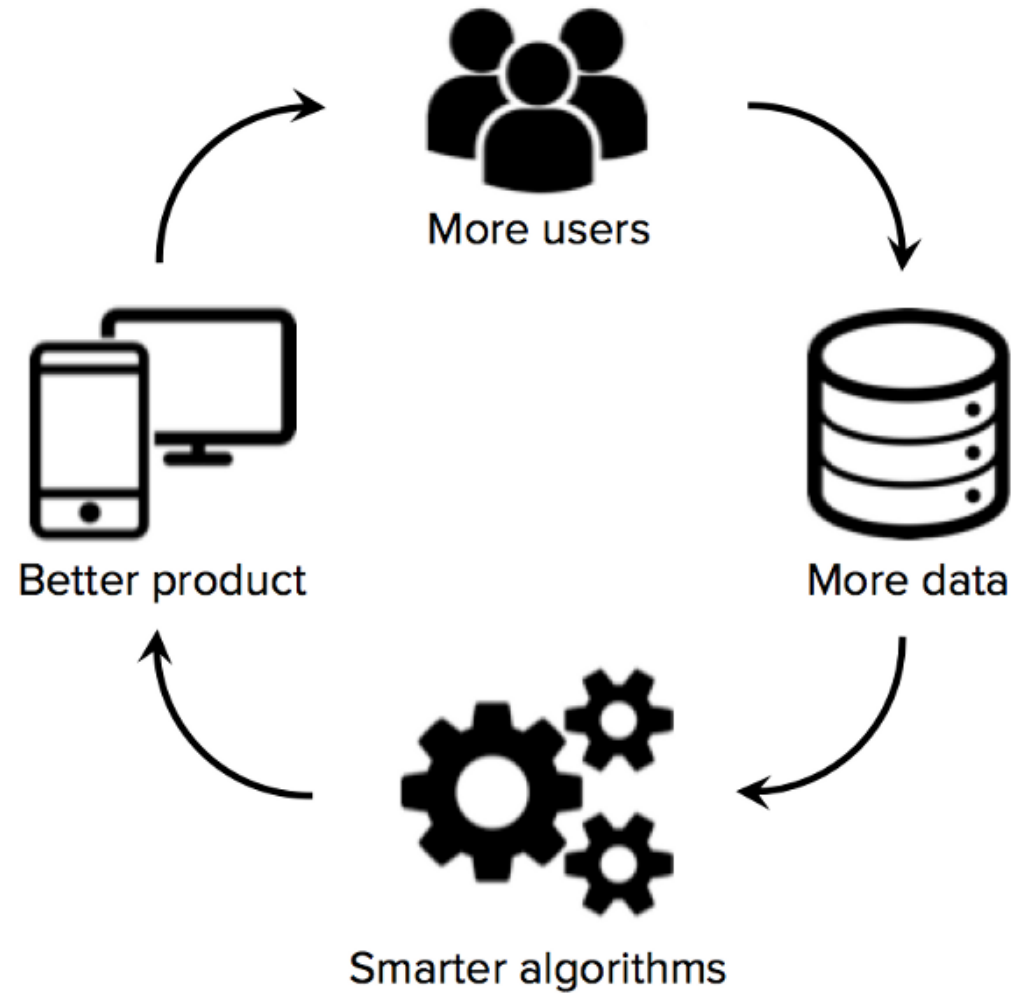


Image source: Joel Thomas and Clemens Mewald. [Productionizing Machine Learning: From Deployment to Drift Detection](#). Databricks Blog, 2019

# Monitoring without Ground Truth: Invariants/Assertions to Assure with Telemetry

- Consistency between multiple sources
  - e.g., multiple models agree, multiple sensors agree
  - e.g., text and image agree
- Physical domain knowledge
  - e.g., cars in video shall not flicker,
  - e.g., earthquakes should appear in sensors grouped by geography
- Domain knowledge about unlikely events
  - e.g., unlikely to have 3 cars in same location
- Stability
  - e.g., object detection should not change with video noise
- Input conforms to schema (e.g. boolean features)
- And all invariants from model quality lecture, including capabilities

# Telemetry for Training: The ML Flywheel



# Engineering Challenges for Telemetry

- Data volume and operating cost
  - e.g., record "all AR live translations"?
  - reduce data through sampling
  - reduce data through summarization (e.g., extracted features rather than raw data; extraction client vs server side)
- Isolating feedback for specific ML component + version
- Biased sampling / Rare events
- Privacy (related: Offline deployments)





# Breakout: Engineering Challenges in Telemetry

Discuss: Cost, privacy, rare events, bias

Scenarios:

- Front-left: Amazon: Shopping app feature that detects the shoe brand from photos
- Front-right: Google: Tagging uploaded photos with friends' names
- Back-left: Spotify: Recommended personalized playlists
- Back-right: Wordpress: Profanity filter to moderate blog posts

(can update slack, but not needed)



# Revisiting Model Quality vs System Goals

# Model Quality vs System Goals

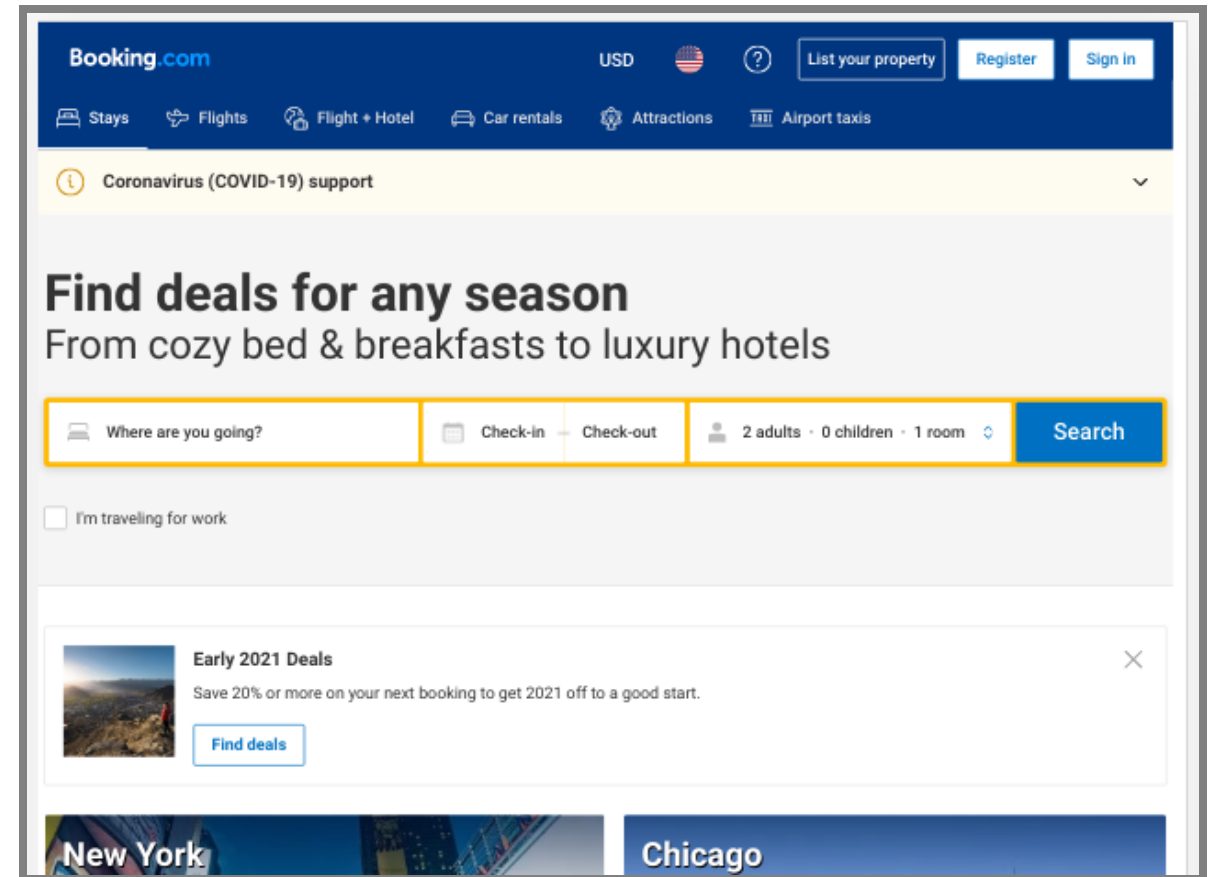
Telemetry can approximate model accuracy

Telemetry can directly measure system qualities, leading indicators, user outcomes

- define measures for "key performance indicators"
- clicks, buys, signups, engagement time, ratings
- operationalize with telemetry

# Model Quality vs System Quality: Booking.com

What's the system goal?



## Speaker notes

The conversion rate is a key performance metric used to measure the effectiveness of marketing efforts, advertising campaigns, or sales processes. It represents the percentage of users or visitors who complete a desired action (a "conversion") out of the total number of users who were exposed to an offer, website, or advertisement.

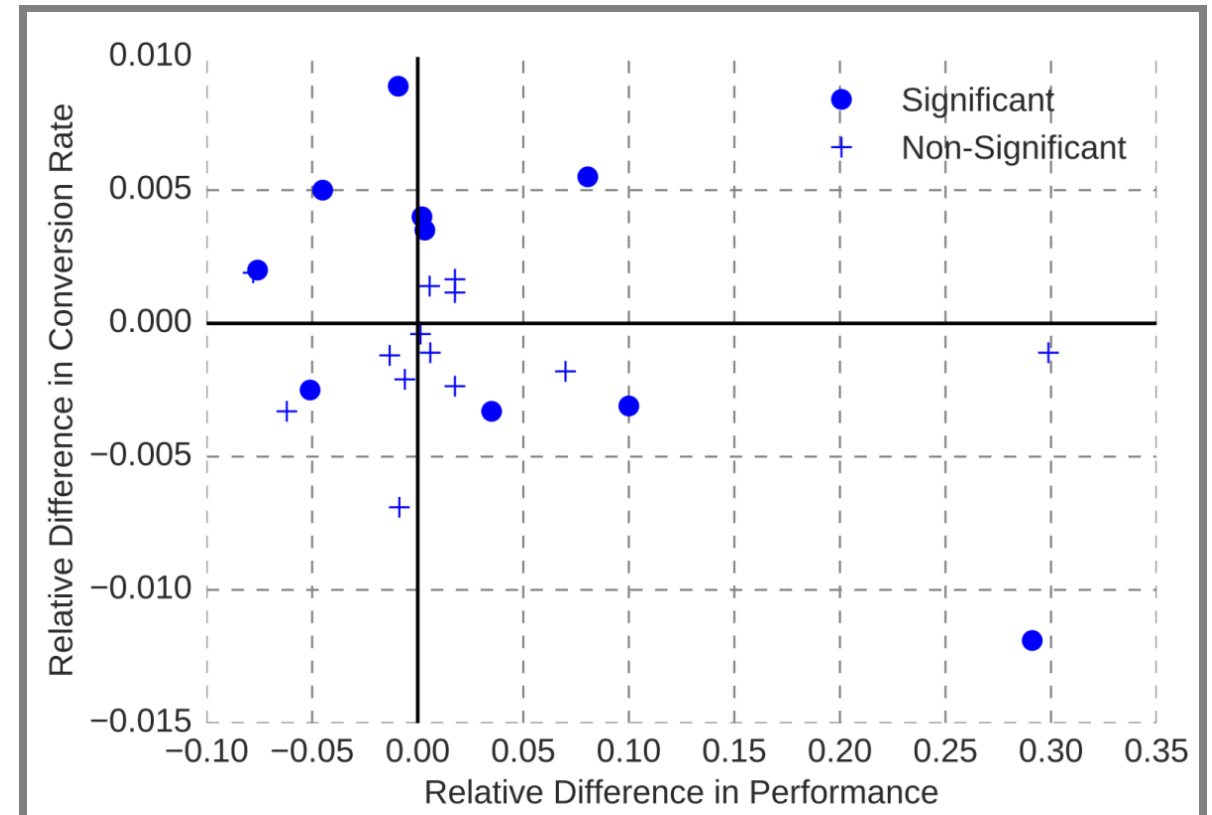
Bernardi, Lucas, et al. "150 successful machine learning models: 6 lessons learned at Booking.com." In Proc. Int'l Conf. Knowledge Discovery & Data Mining, 2019.



# Possible causes of model vs system conflict?

Observed no correlation between offline performance gain vs. business value gain, across many different models with different goals.

Possible reasons:



hypothesized

- model value saturated, little more value to be expected
- segment saturation: only very few users benefit from further improvement
- overoptimization on proxy metrics not real target metrics
- uncanny valley effect from "creepy AIs"



# Design Telemetry in Production

Discuss: What key performance indicator of the *system* to collect?

Scenarios:

- Front-left: Amazon: Shopping app feature that detects the shoe brand from photos
- Front-right: Google: Tagging uploaded photos with friends' names
- Back-left: Spotify: Recommended personalized playlists
- Back-right: Wordpress: Profanity filter to moderate blog posts



# Experimenting in Production

- A/B experiments
- Shadow releases / traffic teeing
- Blue/green deployment
- Canary releases
- Chaos experiments



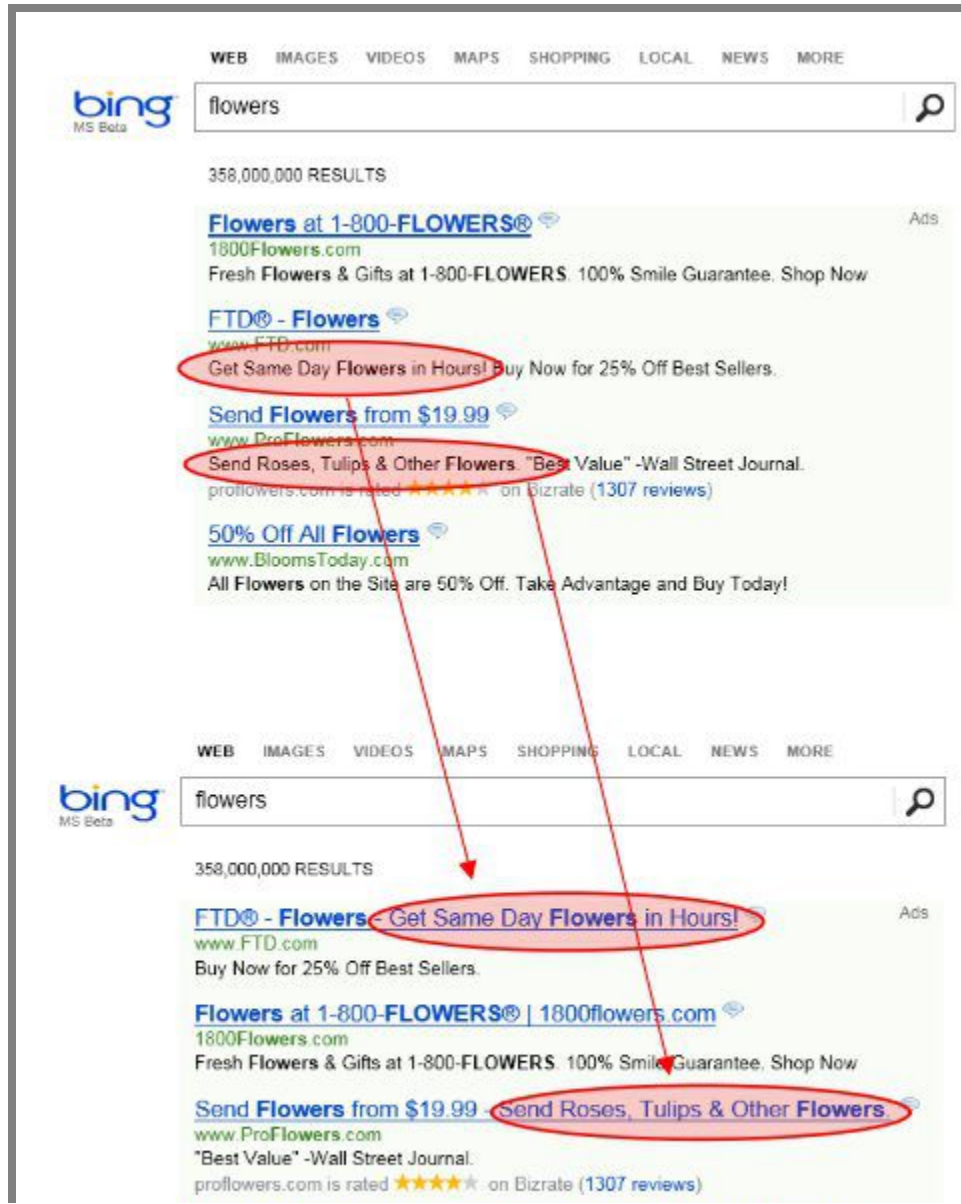
# A/B Experiments: What if...?

- ... we have plenty of subjects for experiments
- ... we could randomly assign to treatment/ control group without them knowing
- ... we could analyze small individual changes and keep everything else constant

Usage observable online, telemetry allows testing in production. Picture source: <https://www.designforfounders.com/ab-testing-examples/>



# Bing Experiment



- Experiment: Ad Display at Bing
- Suggestion deprioritized and left in the backlog for over 6 months
- Implemented & A/B tested in production
- Within 2h *revenue-too-high* alarm triggered.
  - Typically suggest serious bug (e.g., double billing)
  - But valid here: Revenue increase by 12% - \$100M annually in US
  - Did not hurt user-experience metrics
- Simple adjustment became one of the most profitable ideas in Bing's history!

From: Kohavi, Ron, Diane Tang, and Ya Xu.  
"Trustworthy Online Controlled Experiments: A Practical Guide to A/B Testing." 2020.

# A/B Experiment for ML Components?

- New product recommendation algorithm for web store?
- New language model in audio transcription service?
- New (offline) model to detect falls on smart watch?



# Implementing A/B Testing

Implement alternative versions of the system

- using feature flags (decisions in implementation)
- separate deployments (decision in router/load balancer)

Map users to treatment group

- Randomly from distribution
- Static user - group mapping
- Online service (e.g., [launchdarkly](#), [split](#))

Monitor outcomes *per group*

- Telemetry, sales, time on site, server load, crash rate

Speaker notes

divide them into groups





# Feature Flags (Boolean flags)

```
if (features.enabled(userId, "one_click_checkout")) {  
    // new one click checkout function  
} else {  
    // old checkout functionality  
}
```

- Good practices: tracked explicitly, documented, keep them localized and independent
- External mapping of flags to customers, who should see what configuration
  - e.g., 1% of users sees one\_click\_checkout, but always the same users; or 50% of beta-users and 90% of developers and 0.1% of all users

```
def isEnabled(user): Boolean = (hash(user.id) % 100) < 10
```

## Speaker notes

mapping somewhere One way of doing this randomly is hashing user ID random but stable same users always in the same group some offset to get a new sample for telemetry you need to know what group a user was in. Once you have a mapping from flags here then it's easier you can also use a load balancer to manage this you need an if statement somewhere chrome, facebook, if statements are in the backend/code



# Feature Flags as a Service

The screenshot displays a configuration interface for a Feature Flags as a Service. It is organized into several sections:

- Treatments:** A table with columns for Treatment, Default, and Description. It lists two treatments: 'on' (green square, radio button) and 'off' (red square, radio button).
- Whitelist:** A section indicating 0 user(s) or segments are targeted, with an 'Add whitelist' button.
- Traffic Allocation:** A slider set to 100% total User in Split.
- Targeting Rules:** Two rules are defined:
  - if:** user is in segment qa. Then serve on.
  - else if:** user is in segment beta\_testers. Then serve percentage split (50% on, 50% off).
- Default Rule:** Serve treatment of 'off'.



## Speaker notes

There are companies that do this as a service expose your database. "boolean option as a service" most groups in the past have done it themselves

launch darkley, pslit IO, open source libraries gets complicated when you want to do multiple experiments, factorial design gets complicated the bing team wrote a whole book



# Confidence in A/B Experiments (statistical tests)

## Group A

*classic personalized content recommendation model*

2158 Users

average 3:13 min time on site

## Group B

*updated personalized content recommendation model*

10 Users

average 3:24 min time on site

What's the problem of comparing the average?

## Speaker notes

What's the problem here? t test assumes a normal distribution

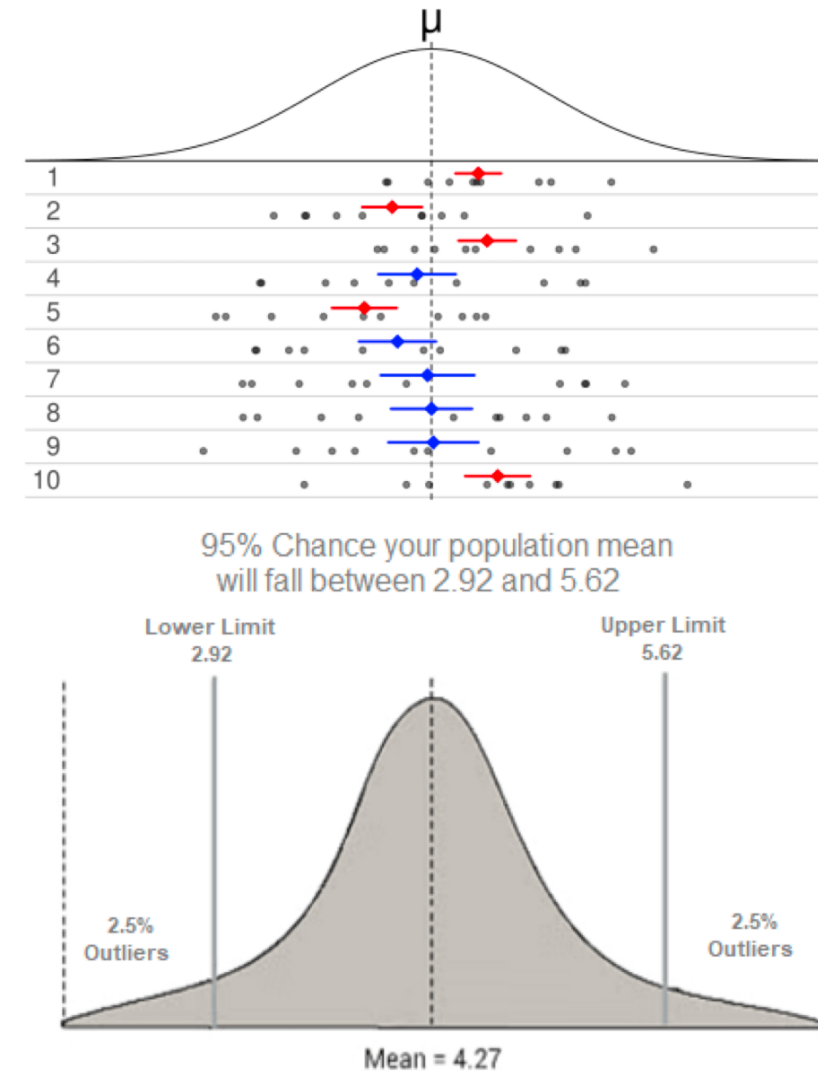
statistical tests to see if it's random event

who here knows about the t test? It's one of the standard tests for this kind of thing.



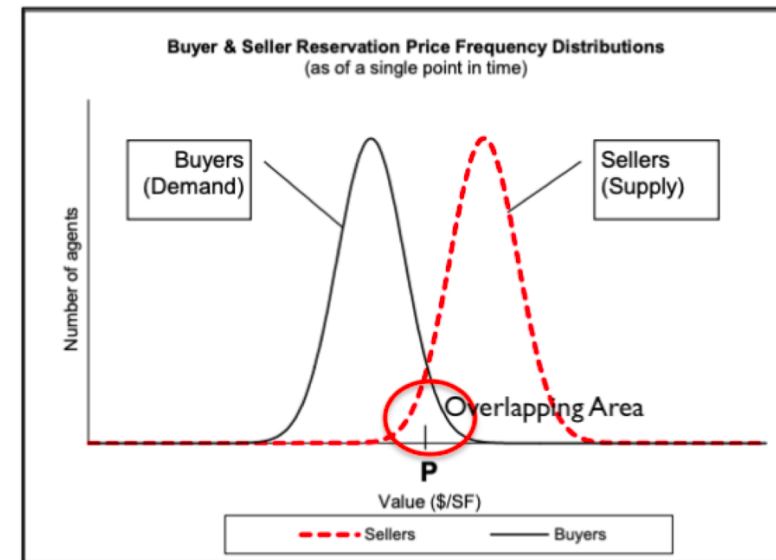
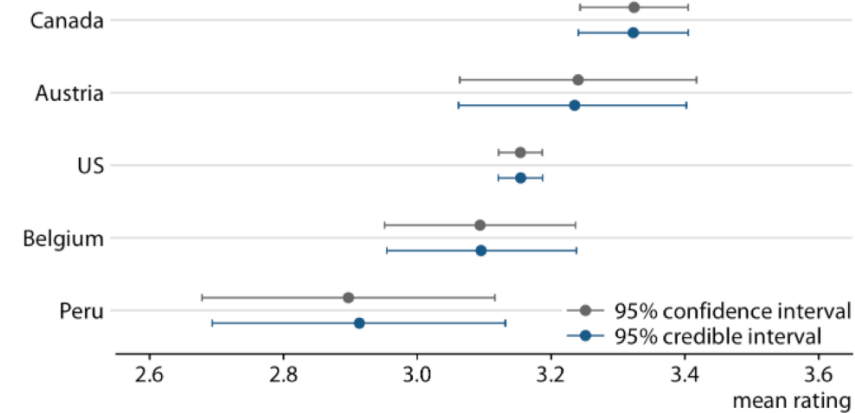
# Analyzing Results: Stats 101

- All the data have some **distribution**.
- **Normal distribution** is very common (e.g. students tend to get average score, not full score).
- When we draw samples from the same distribution, we can get different samples **just by chance**.
- Statistics tells us whether “interesting” patterns we observe **actually exist** in the samples, or whether it’s just sampling noise — use some numbers to express uncertainty.
- **Confidence interval**: if we can compute the average of some samples (average on-site time of 10 users), what’s the average of the population (whole user group)?



# Analyzing Results: Stats 101

- **Significance testing** also helps with comparison. When most of the confidence intervals overlap, we'd know there's no actual differences between groups.
- This is quantified by **p-value**: what is the probability a difference between two samples is by chance (lower = more likelihood of a significant difference).





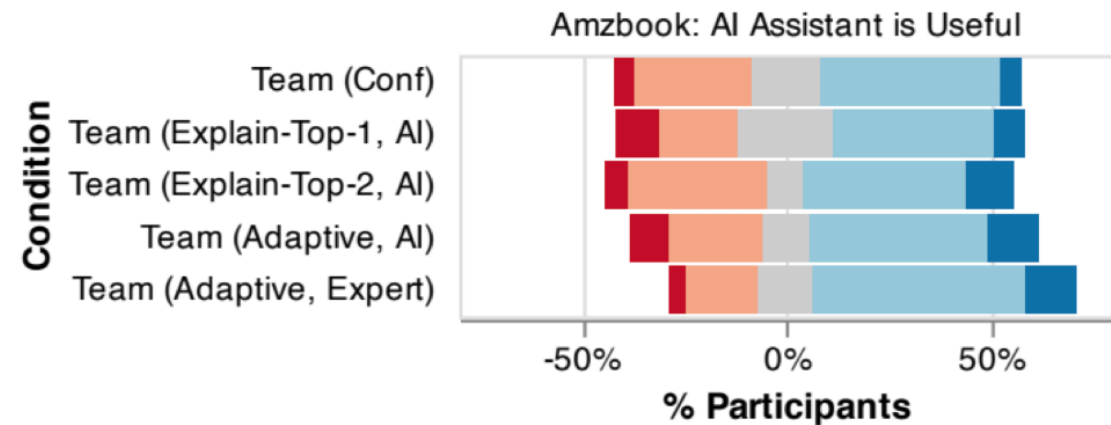
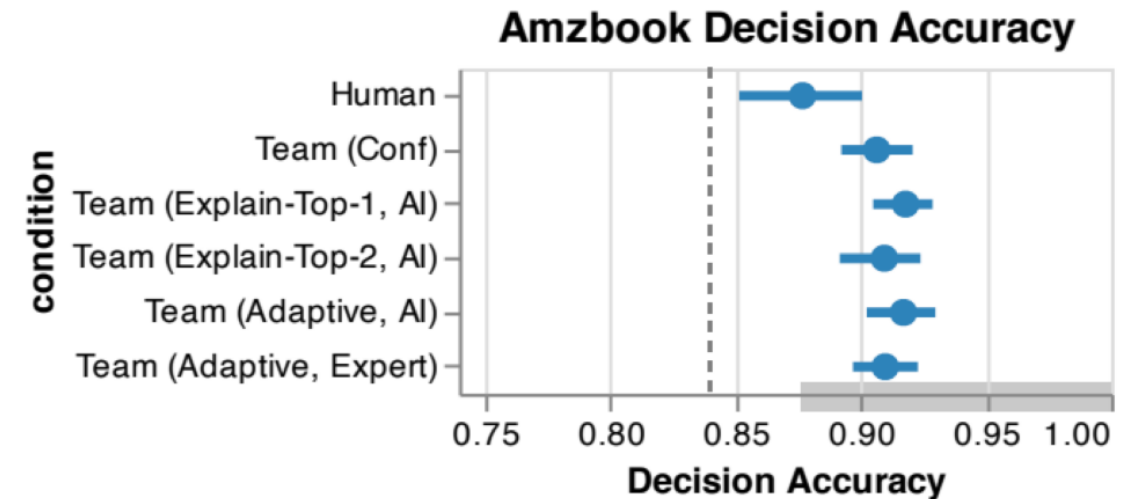
# Stats 101: How to compute p-value?

**Parametric tests:** Assume comparing normally distributed groups that have the same variances.

- Tests: t-test, ANOVA, & linear regression.
- For: model accuracy, human click streams
- More sensitive and powerful when the requirement is met.

**Non-parametric tests:** Does not assume normal distribution.

- For: categorical data like Likert Scale rating
- Tests: Wilcoxon signed-rank test compare users' nominal Likert Scale ratings



# t-test

We will ask for statistical test in M3 -- Many softwares implement it!

```
> t.test(x, y, conf.level=0.9)
```

```
Welch Two Sample t-test
```

```
t = 1.9988, df = 95.801, p-value = 0.04846
```

```
alternative hypothesis: true difference in means is  
not equal to 0
```

```
90 percent confidence interval:
```

```
0.3464147 3.7520619
```

```
sample estimates:
```

```
mean of x mean of y
```

# Decision tree of tests

Many many other factors, e.g., dependent vs. independent measures

# How many samples needed?

**Too few?**

Noise and random results!

**Too many?**

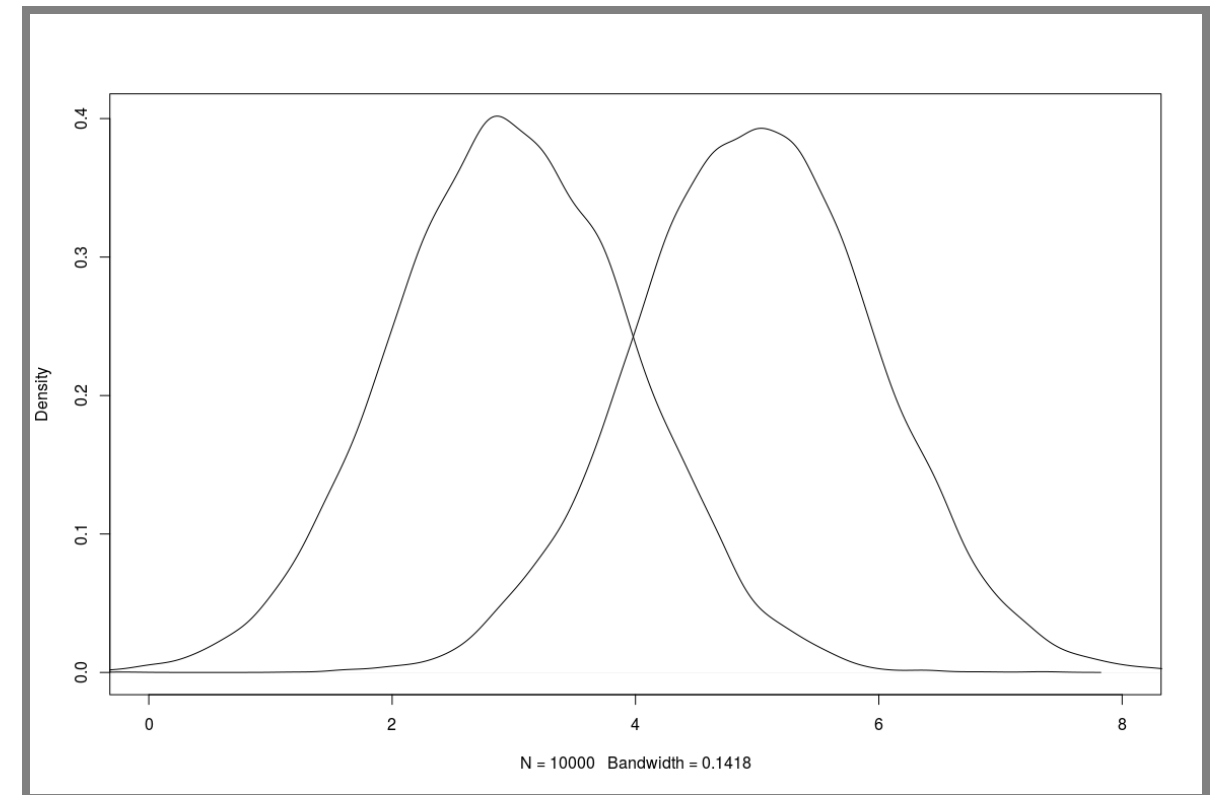
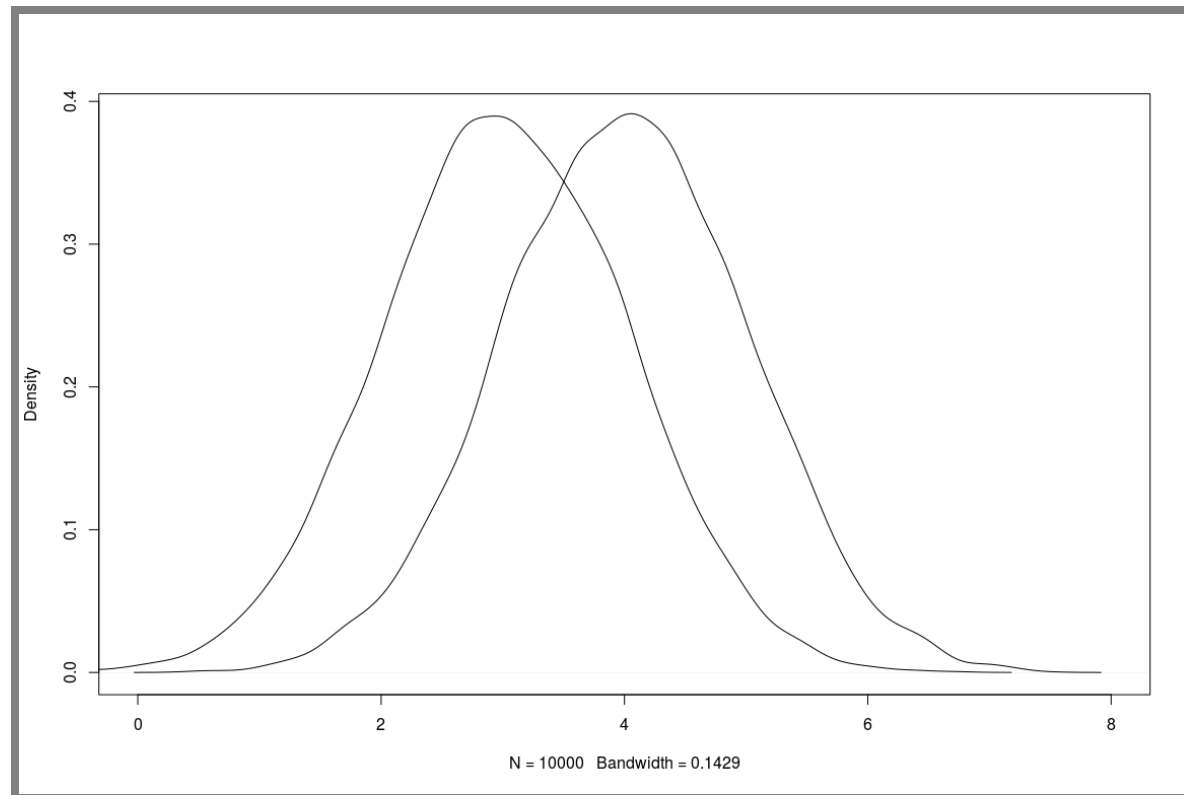
Risk of spreading bad designs!

50/50? New model vs. old model about 1:10 to here inherently risky, so you don't want too many in experiment

# Some factors that affect your sample size

Different effect size, same deviations

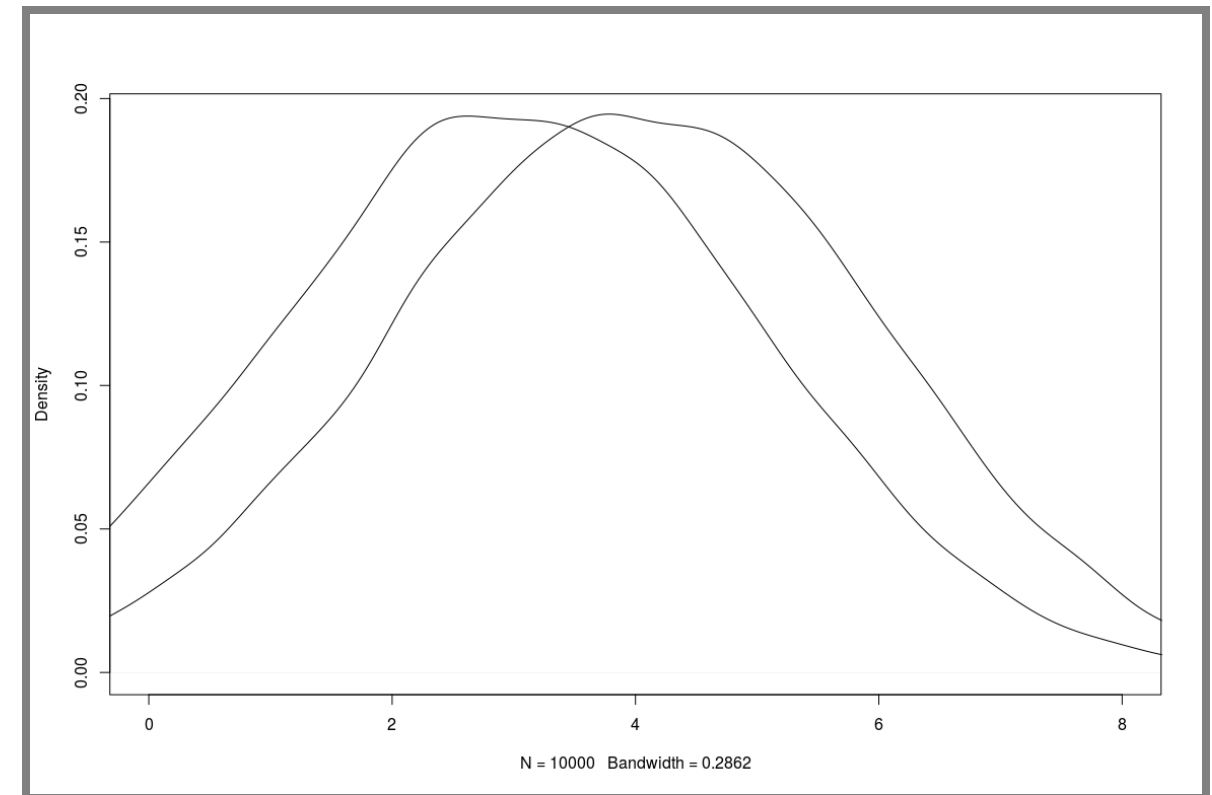
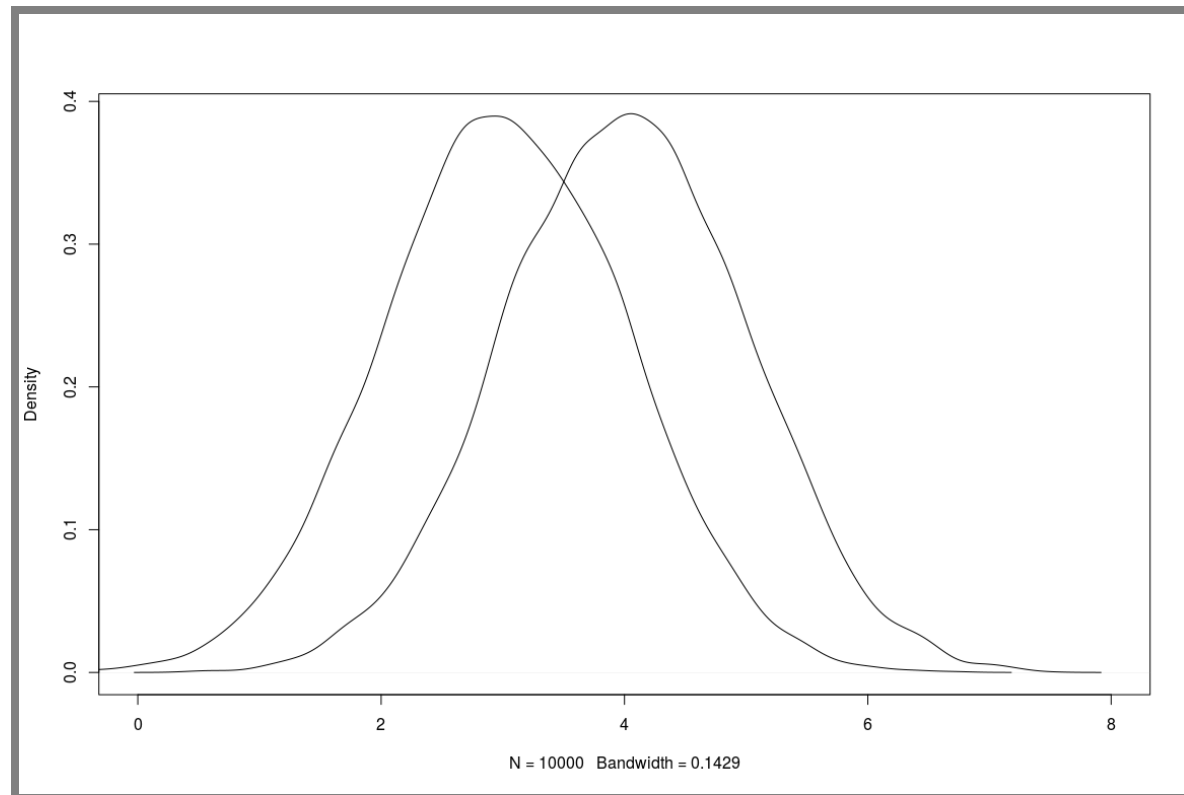
- Effect size: the magnitude of the difference between groups



# Some factors that affect your sample size

Same effect size, different deviations

- Less noise --> Easier to recognize



# Concurrent A/B testing

Multiple experiments at the same time

- Independent experiments on different populations -- interactions not explored
- Multi-factorial designs, well understood but typically too complex, e.g., not all combinations valid or interesting
- Grouping in sets of experiments (layers)

Further readings:



# Other Experiments in Production

Chaos experiments

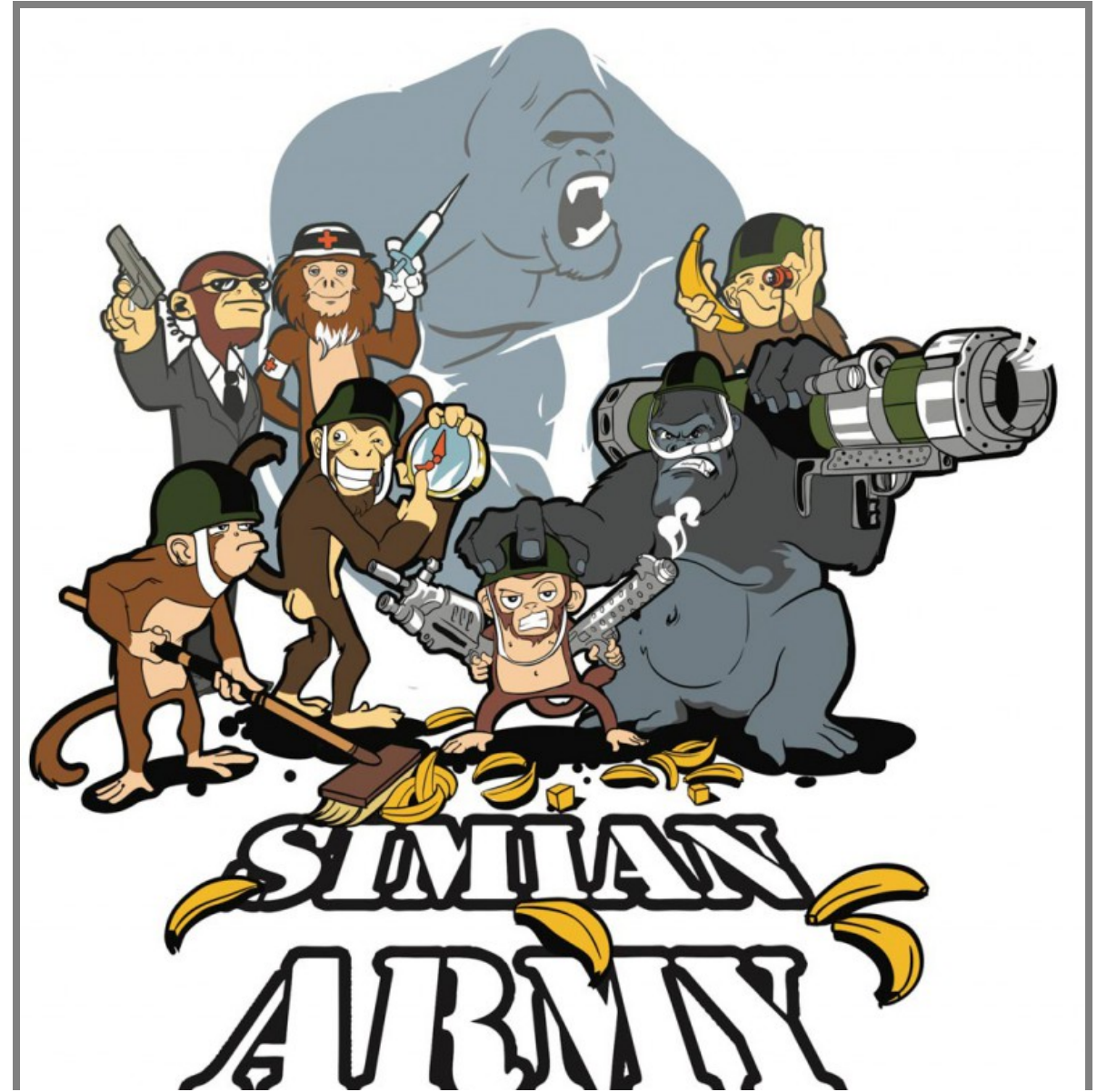
Shadow releases / traffic teeing

Canary releases

# Chaos Experiments

Deliberate introduction of faults in production to test **robustness**.

e.g. Netflix's Chaos Monkey -- randomly shuts down virtual machine instances, test automatic traffic re-route to healthy instances.



Deliberate introduction of faults in production to test robustness.



# Chaos Experiments for ML Components?



Speaker notes

Artificially reduce model quality, add delays, insert bias, etc to test monitoring and alerting infrastructure



# Shadow releases / traffic teeing

Run both models in parallel

Use predictions of old model in production

Compare differences between model predictions

If possible, compare against ground truth labels/telemetry

**Examples?**

# Canary Releases

Release new version to small percentage of population (like A/B testing)

Automatically roll back if quality measures degrade

Automatically and incrementally increase deployment to 100% otherwise



# Advice for Experimenting in Production

Minimize *blast radius* (canary, A/B, chaos expr)

Automate experiments and deployments

Allow for quick rollback of poor models (continuous delivery, containers, loadbalancers, versioning)

Make decisions with confidence, compare distributions

Monitor, monitor, monitor



# Summary

Production data is ultimate unseen validation data

Both for model quality and system quality

Telemetry is key and challenging (design problem and opportunity)

Monitoring and dashboards

Many forms of experimentation and release (A/B testing, shadow releases, canary releases, chaos experiments, ...) to minimize "blast radius"; gain confidence in results with statistical tests

# Further Readings

- On canary releases: Alec Warner and Štěpán Davidovič. “[Canary Releases.](#)” in [The Site Reliability Workbook](#), O’Reilly 2018
- Everything on A/B testing: Kohavi, Ron. [Trustworthy Online Controlled Experiments: A Practical Guide to A/B Testing](#). Cambridge University Press, 2020.
- A/B testing critiques: Josh Constine. [The Morality Of A/B Testing](#). Blog 2014; the [Center of Humane Technology](#); and the Netflix documentary [The Social Dilemma](#)
- Ori Cohen “[Monitor! Stop Being A Blind Data-Scientist.](#)” Blog 2019
- Jens Meinicke, Chu-Pan Wong, Bogdan Vasilescu, and Christian Kästner. [Exploring Differences and Commonalities between Feature Flags and Configuration Options](#). In Proceedings of the Proc. International Conference on Software Engineering ICSE-SEIP, pages 233–242, May 2020.

