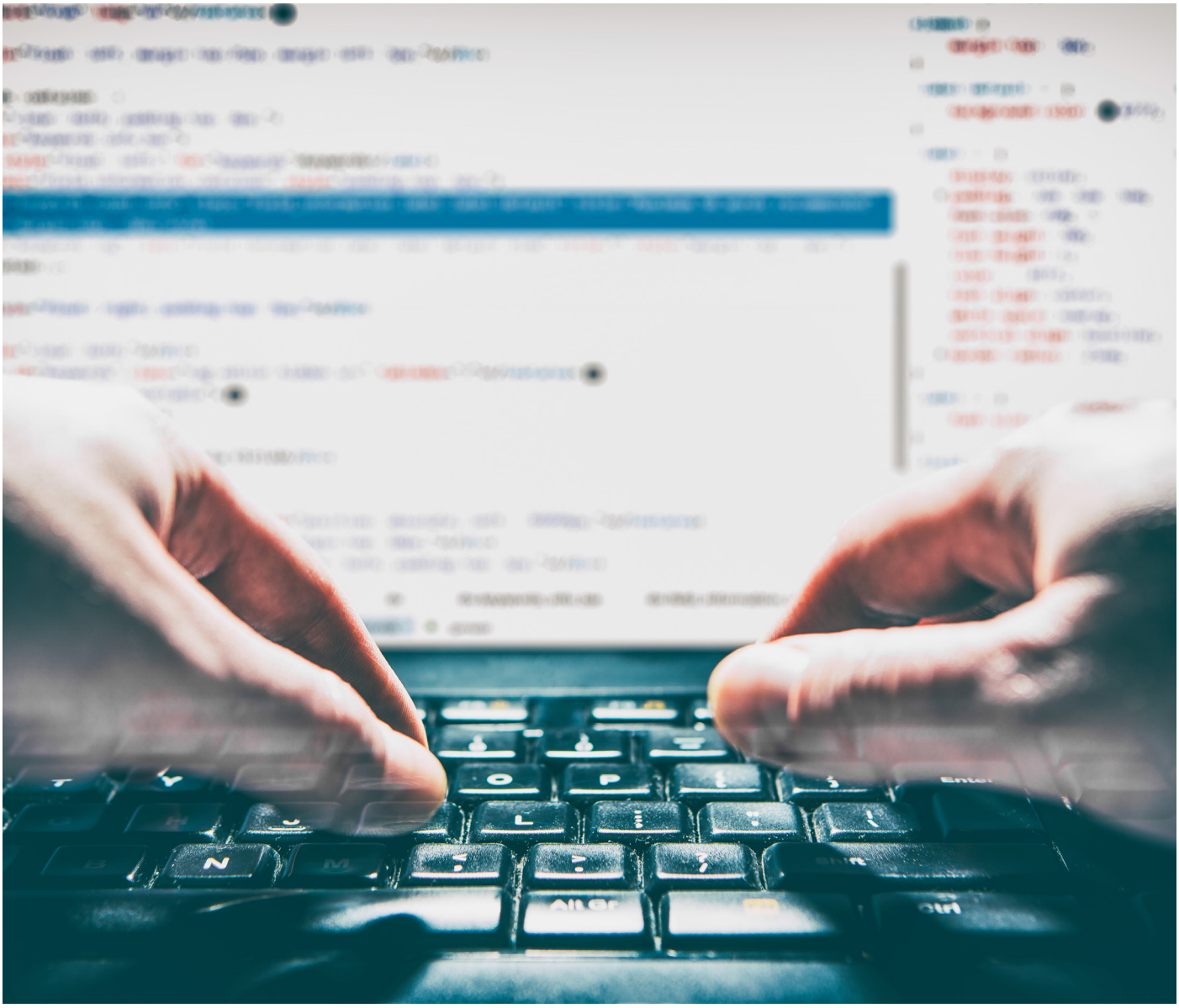
# THE MORNING PAPER
## QUARTERLY REVIEW

# EXPERIMENTATION, OPTIMISATION, AND LEARNING



Hard-won lessons from large-scale experiments at Amazon, Booking.com, LinkedIn Microsoft and Google.

InfoQ

# CONTENTS

*Issue #6, September 2017*

# WELCOME...

*Adrian Colyer*

I'm sure you've come across the phrase "cloud-native" to describe an approach to IT that assumes and embraces the characteristics of cloud platforms. While cloud-native has been stealing all the attention recently, there's another shift that's just as important, the move towards becoming a "data-native" organisation. For this edition of The Morning Paper Quarterly Review, I've chosen a set of papers that illustrates what the data-natives are up to: how they embed experimentation, optimisation, and learning into everything they do. We might have thought that continual delivery was the end game but for data natives, this is just the necessary prerequisite.

Cloud-natives move fast, data-natives move smart — they know where they're heading and they are able to continuously correct course. Getting to this level of understanding requires the definition of key metrics backed by instrumentation, collection and computation, and interpretation. In "A Dirty Dozen: Twelve Common Metric Interpretation Pitfalls in Online Controlled Experiments", Microsoft shares the four classes of metrics you need and how to safely interpret what they tell you. These are hard-won lessons from running thousands of experiments a year across Bing, MSN, Skype, Office, Xbox, and so on.

My second choice, "Seven Rules of Thumb for Web Site Experimenters", is in the same vein, and also draws on lessons from giants, such as Amazon, LinkedIn, and Booking.com as well as Microsoft. The thing that really caught my attention in this paper is just how much performance matters — so much so that we need to calibrate for any performance impact introduced as an artefact of an experiment (for example, a quick initial but slower implementation designed to test an idea). Another great insight from the paper is that "simple designs are best" — when we're trying to understand and interpret what's going on, keeping things simple helps us move faster.

If the first two paper choices inspire us to begin a data-native transformation, where do we start? In "The Evolution of Continuous Experimentation in Software Product Development", Microsoft shares a proven roadmap they've used with multiple internal teams to take them from data-illiterate to data-driven at scale. They break things down into crawl, walk, run, and fly stages across the evolution of the technology, organisation, and business. Accompanying this journey, we're going to need an experimentation platform that grows in sophistication across the stages. Where does our organisation fit in the "experimentation evolution model"?

Experimentation is just one path to optimisation. In "Google Vizier: A Service for Black-Box Optimization", a team from Google describes how they built the Vizier optimisation service that is used to optimise just about everything at Google: from online experiments to the hyper-parameters of machine-learning systems and even the recipe for the cookies served in the cafeteria! Vizier runs the entire hyper-parameter tuning workload across all of Alphabet.

My final choice for this edition is "TFX: A TensorFlow-Based Production-Scale Machine Learning Platform". All of the machine-learning models we've been optimising need to be trained, validated, and served somehow. For that, we're going to need a machine-learning platform. At Google, TFX reduced the time to production for machine-learning models from the order of months to weeks. This paper gives a great breakdown of the key functionality to look for in a machine-learning platform. Data-natives wouldn't be seen dead without one. ;)

Optimizely

# Implementing the Right Experimentation Solution

## How to Choose Whether to Build or Buy


- The key questions product teams must ask before choosing to build or buy
- The most common requirements that enterprise teams have for experimentation
- How to avoid key mistakes and plan for success when building an internal system

Download White Paper

# A DIRTY DOZEN:
## TWELVE COMMON METRIC INTERPRETATION PITFALLS IN ONLINE CONTROLLED EXPERIMENTS

**Pure gold! "A Dirty Dozen: Twelve Common Metric Interpretation Pitfalls in Online Controlled Experiments" by Dmitriev et al. (KDD 2017) has 12 wonderful lessons in how to avoid expensive mistakes in companies that are trying their best to be data-driven.**

*Dmitriev et al. (KDD 2017)*

I offer a huge "thank you" to the team from Microsoft for sharing their hard-won experiences with us:

*In our experience of running thousands of experiments with many teams across Microsoft, we observed again and again how incorrect interpretations of metric movements may lead to wrong conclusions about the experiment's outcome, which if deployed could hurt the business by millions of dollars.* ***(All quotes from Dmitriev et al. 2017.)***

(Warning: I definitely exceeded by usual length target with this post — there was just too much good stuff that I didn't want to leave out. I hope you find the extra time it takes to read through worth it!)

Before we dive into the details, there are a couple of interesting data points about experimentation at Microsoft I wanted to draw your attention to. Microsoft has an experimentation system used across Bing, MSN, Cortana, Skype, Office, Xbox, Edge, Visual Studio, and so on, that runs "thousands of experiments a year". This isn't just fiddling on the margins: "At Microsoft, it is not uncommon to see experiments that impact annual revenue by millions of dollars, sometimes tens of millions of dollars."

If we're going to talk about metric interpretation, first we'd better have some metrics. Section 4 of the paper has some excellent guidelines concerning the types of metrics we should be collecting. We're going to need a collection of metrics such as these in

***"In our experience of running thousands of experiments with many teams across Microsoft, we observed again and again how incorrect interpretations of metric movements may lead to wrong conclusions about the experiment's outcome, which if deployed could hurt the business by millions of dollars."***

order to diagnose and avoid the pitfalls.

## LET'S TALK ABOUT METRICS

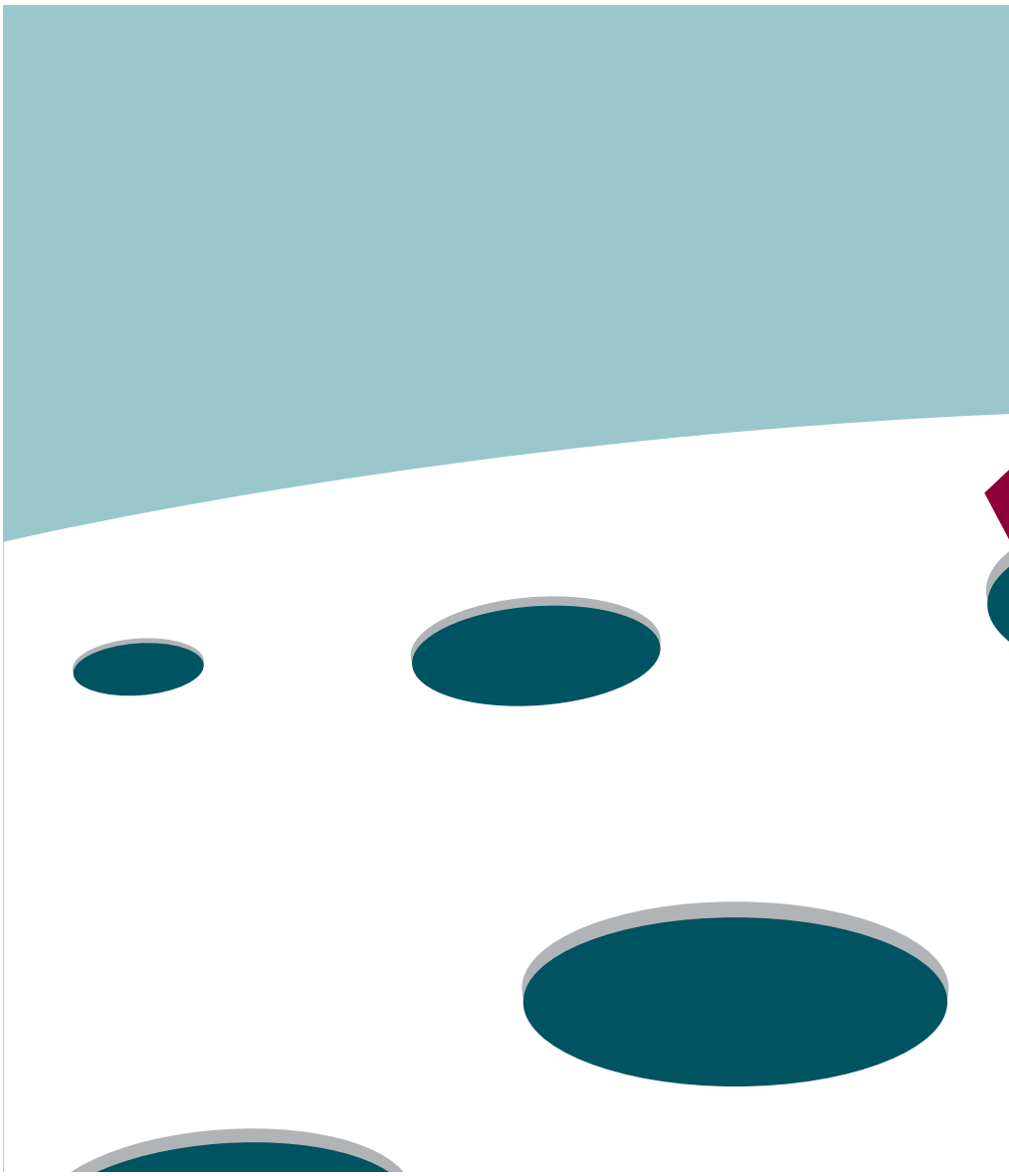
***Most teams at Microsoft compute hundreds of metrics to analyze the results of an experiment. While only a handful of these metrics are used to make a ship decision, we need the rest of the metrics to make sure we are making the correct decision.***

There are four primary classes of metrics that Microsoft collects:

**Data-quality metrics:** These help to determine whether an experiment was correctly configured and run, such that the results can be trusted. An important example is the ratio of the number of users in the treatment to the number of users in the control. You need this metric to avoid the first pitfall.

**Overall evaluation criteria (OEC) metrics:** These are the leading metrics that determine whether the experiment was successful. These metrics can be measured during the short duration of an experiment and are also "indicative of long-term business value and user satisfaction". In the ideal case, a product has just one OEC metric.

**Guardrail metrics:** "In addition to OEC metrics, we have found that there is a set of metrics which are not clearly indicative of success of the feature being tested, but which we do not want to significantly harm when making a ship decision." These are the guardrail metrics — on MSN and Bing, for example, page load time is a guardrail metric.

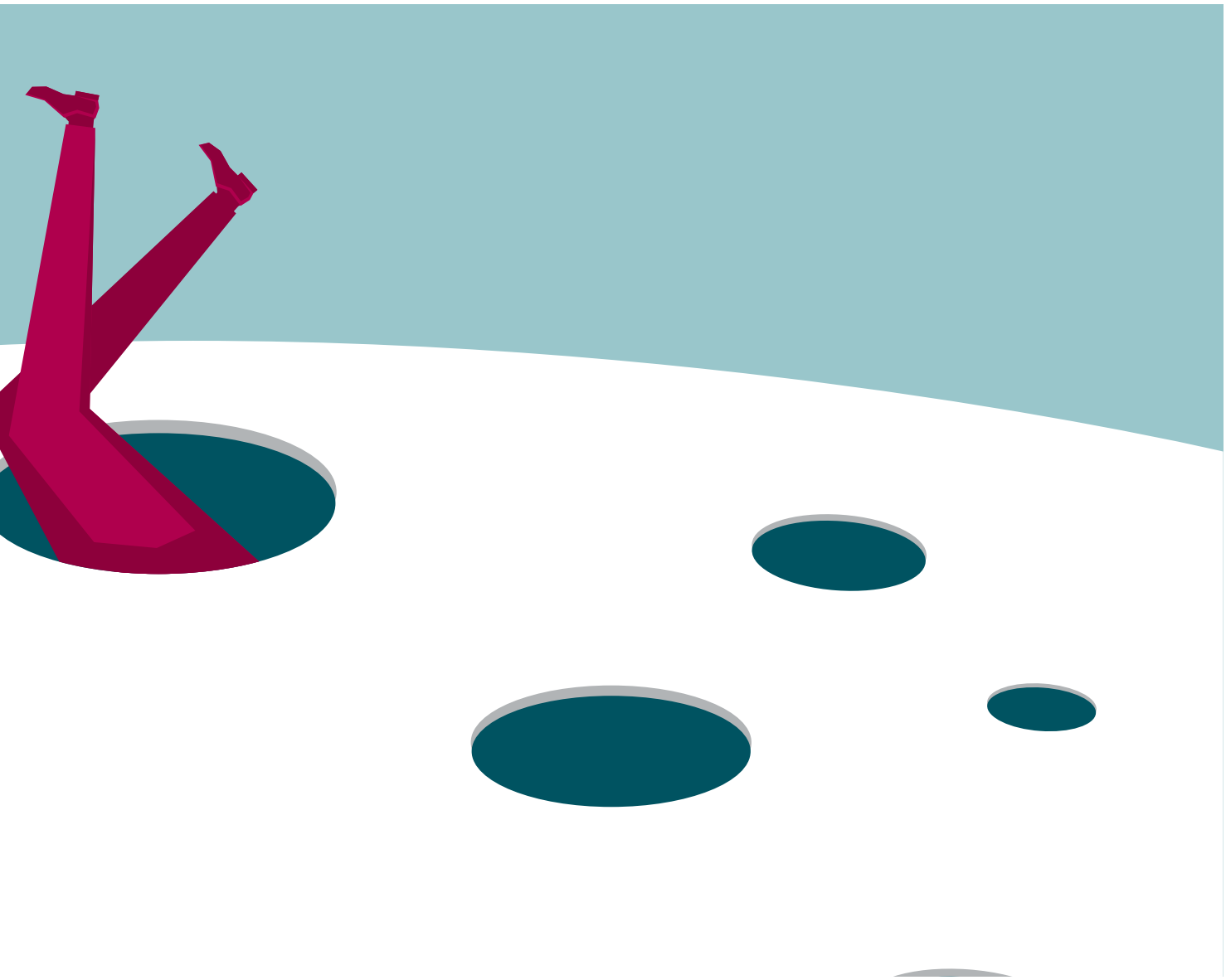


**Local feature and diagnostic metrics:** These measure the usage and functionality of individual features of a product. For example, click-through rates for individual elements on a webpage. These metrics often help to diagnose/explain where OEC movement (or lack of it) is coming from. They can also reveal situations where improvements in one area harm other areas.

## PITFALL #1: METRIC SAMPLE RATIO MISMATCH

Here's a puzzler for you:

***We ran an experiment on the MSN.com homepage where, when users clicked on a link, in treatment the destination page opened in a new browser tab while in control it opened in the same tab. The results showed 8.32% increase in Page Load Time (PLT) of the MSN.com homepage. How could this one-line JavaScript change cause such a large performance degradation?***



The PLT metric is simply the average load time for all pages in a variant (control or treatment), i.e., sum of all page load times for a variant, divided by number of page loads in the variant. (I know you know what average means, there's a reason I'm spelling it out!) This gives us a clue that the number of page loads matters. With results opening in the same tab, users often use the back button to go back to the results page — this results in more page loads (versus keeping the results tab open in the treatment), and those loads are faster due to the browser cache.

***Clearly, the set of page loads over which the metric was computed were different between the treatment and the control. In a situation like this the metric value cannot be trusted.***

This is a "sample ratio mismatch", and if it happens, all bets are off. Without being aware of this, we can make misinformed decisions.

***...It is critical that the experimentation system automatically detects sample ratio mismatches and warns the user about them.... A good start is to look separately at the numerator and denominator of the metric.***

## PITFALL #2: MISINTERPRETATION OF RATIO METRICS

On the MSN homepage, a module located close to the bottom of the main page was moved to a higher position on the page so that users needed to scroll less to reach it. The click-through rate (CTR) of the module decreased by 40%! That doesn't sound right!

Take a look at the metric:

$$Avg\ CTR/User = \frac{\sum_{users}\left(\frac{\#\ clicks\ on\ the\ module}{\#\ impressions\ of\ the\ module}\right)}{\sum_{users} 1}$$

If we think about that for a moment, we'll realise that many more users are seeing the module after it was moved higher. In fact, both the number of times the module was seen and the number of clicks on it increased. It's just that the former went up more than the latter. Overall page CTR stayed flat, but revenue actually increased because the promoted module was more monetize-able than those that were pushed down to make room for it. So a 40% decrease in CTR turned out to be a good thing!

***Ratio metrics can be misleading or potentially invalid if the denominator of the metric changes.***

The authors' preferred way to compute ratio metrics is as the average of ratios (as in the metric example in this section). This tends to give higher sensitivity and greater resilience to outliers.

***To detect denominator mismatch in ratio metrics, we recommend to always define count metrics for the numerator and the denominator, and provide those in the result alongside the ratio metric.***

## PITFALL #3: TELEMETRY-LOSS BIAS

Skype experimented with a protocol change for delivering push notifications. The result showed strong, statistically significant changes in some call-related metrics such as fraction of successfully connected attempted calls. But the notification mechanism has absolutely nothing to do with calls!

The push notifications were waking up the device, allowing it enough time to check whether or not it was on Wi-Fi, and, if so, to prepare a telemetry batch and send it over. Server-side metrics were unchanged (total capture), but more client-side metrics were being delivered.

***Since the new events that make it (or events that are lost) are typically a highly biased set, telemetry loss bias practically invalidates any metrics that are based on the affected events.... Therefore, for every client event used in metrics, a Data Quality metric measuring the rate of the event loss should be created.***

The simplest way is to compare to a corresponding server event if we have one. In the absence of that, we can assign sequence numbers to the client side and look for gaps in the sequence.

## PITFALL #4: ASSUMING UNDERPOWERED METRICS HAD NO CHANGE

In one MSN.com experiment, page views rose by 0.5% (that's a meaningful impact on the business), but the associated p-value was not statistically significant. Should we assume there was no true impact on page views? An investigation revealed that the experiment did not have enough "power" for the metric.

***Power is the probability of rejecting the null hypothesis given that the alternative hypothesis is true. Having a higher power implies that an experiment is more capable of detecting a statistically significant change when there is indeed a change.***

Before an experiment, we should conduct an a priori power analysis to estimate sufficient sample

sizes — at the very least for OEC and guardrail metrics. Often, the sample size will be the same for typical experiments on a given product — e.g., for Bing experiments in the US, the rule of thumb is at least 10% of users for one week.

***We recommend to have at least 80% power to detect small enough changes in success and guardrail metrics to properly evaluate the outcome of an experiment.***

## PITFALL #5: CLAIMING SUCCESS WITH A BORDERLINE P-VALUE

A Bing.com experiment gave a statistically significant positive increase in one of the OEC metrics, with a p-value of 0.029. Shall we celebrate and ship it?

***In Bing.com whenever key metrics move in a positive direction we always run a certification flight which tries to replicate the results of the experiment by performing an independent run of the same experiment.***

Even in A/A experiments, we can watch the behaviour of metrics flip between statistically significant and not. (Clearly they aren't, by construction!) Always rerun experiments that have produced borderline p-values; if the effects continue to fall on the borderline and the traffic can't be increased, we can use Fisher's method to obtain a more reliable conclusion.

## PITFALL #6: CONTINUOUS MONITORING AND EARLY STOPPING

You've probably heard about this one.... If our experiment is showing statistically significant results before the scheduled finishing time, can we stop it? Likewise, if an experiment isn't showing the effect we hoped for, can we keep running it for longer just in case? No and no.

***In our experience, stopping early or extending the experiment are both very common mistakes experiment owners make. These mistakes are subtle enough to even make it into recommended practices in some A/B testing books. The issue is exacerbated by the fact that in practice continuous experiment monitoring is essentially a requirement to be able to shut down the experiment quickly in case a strong degradation in user experience is observed.***

You can either (a) train experiment owners not to do this, (b) adjust p-values to account for extra checking, or (c) use a better test than p-values. For example, a Bayesian framework naturally allows for continuous monitoring and early stopping.

## PITFALL #7: ASSUMING THE METRIC MOVEMENT IS HOMOGENEOUS

Sometimes treatments have differing affects on different subsets of the treatment groups (e.g., users in different countries or the feature not working correctly in certain browsers). If you're not aware of such effects, you may end up shipping a feature that improves user experience overall but substantially hurts experience for a group of users.

***Due to a large number of metrics and segments that need to be examined in order to check whether there is a heterogenous treatment effect, it's pretty much impossible to reliably detect such effects via a manual analysis.***

The Microsoft system automatically analyses every experiment scorecard and warns if it finds heterogenous treatment effects.

## PITFALL #8: SEGMENT (MIS) INTERPRETATION

Segmenting users is pretty common, but we need to interpret metric movements on segments with extra care. Simpson's paradox shows how it is possible for a metric to go up in every individual segment but still go down overall, for example. The condition used for defining a segment must not be impacted by the treatment.

***If statistically significant difference is observed (in sample ratios), the results for that segment group, and often for all other groups in the segment, are invalid and should be ignored. A common way to run into Simpson's paradox is to keep recursively segmenting the users until a statistically significant difference is found.***

Recursive segmentation is also an instance of the multiple comparisons problem.

## PITFALL #9: IMPACT OF OUTLIERS

The number of images in the slideshow at the top of the MSN.com homepage was increased. This resulted in a significant decrease in engagement, the opposite of what was expected. It turned out that the experiment

increased engagement so much that real users were being labelled as bots!

***Outliers can skew metric values and increase the variance of the metric making it more difficult to obtain statistically significant results. Because of this, it is common to apply some kind of filtering logic to reduce the outlier impact. As the example above shows, however, one needs to ensure that all variants in the experiment are impacted by the outlier filtering logic in the same way.***

The authors recommend always having a metric that counts the number of values affected by outlier handling logic in the paper's "Data Quality Metrics" section.

## PITFALL #10: NOVELTY AND PRIMACY EFFECTS

We added a shiny thing and engagement went up. Yay!

***When we looked at the experiment results for each day segment, we found that the percent delta in clicks on top sites between treatment and control declined quickly, suggesting a novelty effect.***

What we really want, of course, is to understand the long-term impact on the business and users. Ignoring novelty effects can lead to an investment in features that aren't worth it in the long run. On the flip side of the coin, primacy effects are those where initial treatment effects appear small in the beginning but increase over time (e.g., as a machine-learning system better adapts). When we suspect a feature has primacy effects, we can do a "warm start" so that the marginal improvement in performance over time is smaller.

***We also recommend segmenting treatment effect by different days of the experiment, or different user visits, to see if the treatment effect changes over time.***

## PITFALL #11: INCOMPLETE FUNNEL METRICS

For funnel metrics, we need to measure all parts of the process. And with final success rates often as low as 1%, we need to take care to ensure that the metrics at every step are sufficiently powered to detect any significant changes.

***At every step of the funnel we need to ensure that the success rates are compared and not just the raw clicks or user counts. In addition, we should measure both conditional and unconditional success rate metrics. Conditional success rates are defined as the proportion of users that complete the given step among users that attempted to start the step. Unconditional success rates compute the success rate taking into consideration all users who started at the top of the funnel.***

## PITFALL #12: FAILING TO APPLY TWYMAN'S LAW

If we see an unexpected metric movement, positive or negative, it normally means there is an issue. For example, a too-good-to-be-true jump in number of clicks turned out to be because users were confused and clicking around trying to figure things out!

***Twyman's law says that any figure that looks interesting or different is usually wrong.***

Even surprising results that appear to be positive should be treated with scepticism.

***At Microsoft, we have configured automated alerts and auto-shutdown of the experiments if we detect unexpected large metrics movements in both the positive and negative directions.***

## FURTHER READING...

If you enjoy this paper, you might also like the following.

"Measuring Metrics", Dmitriev and Wu (CIKM 2016): More on defining and evaluating good metrics that capture long-term company goals.

"Data-Driven Metric Development for Online Controlled Experiments: Seven Lessons Learned", Deng and Shi (KDD 2016): How to create metrics you can trust.

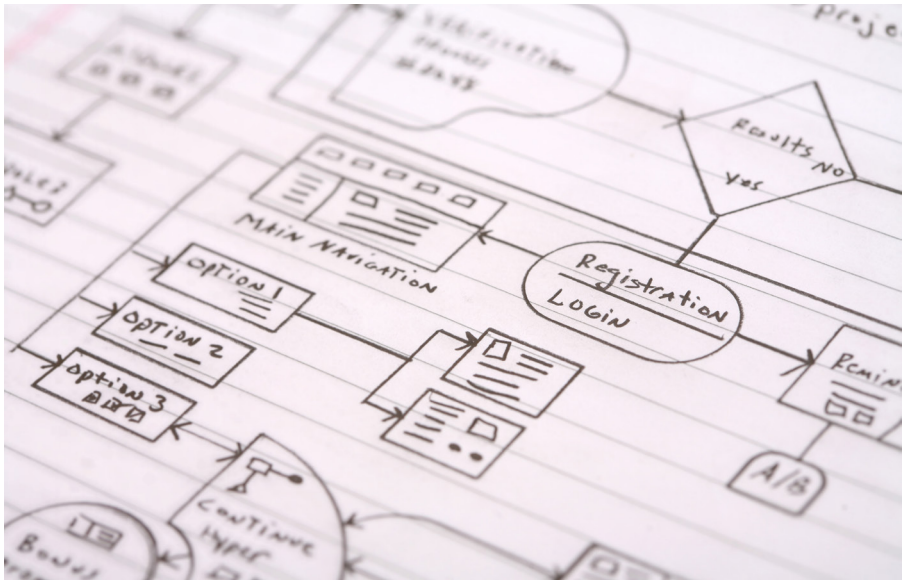
"Principles for the Design of Online A/B Metrics", Machmouchi and Buscher (SIGIR 2016): A short two-pager that introduces the idea of metric hierarchies.

"A Dirty Dozen: Twelve P-Value Misconceptions", Goodman (Seminars in Hematology 25:135-140, 2008): A catalogue of 12 p-value pitfalls, and the inspiration for this paper.

# SEVEN RULES OF THUMB FOR WEB SITE EXPERIMENTERS

**Following 12 metric-interpretation pitfalls, we're looking at "Seven Rules of Thumb for Web Site Experimenters" from Kohavi et al. (KDD 2014).**

*Kohavi et al. (KDD 2014)*




***"At Bing, two other small changes, which are confidential, took days to develop, and each increased ad revenues by about $100M annually."***

There's a little bit of duplication here, but the paper is packed with great real-world examples, and there is some very useful new material, especially around the impact of performance.

*Having been involved in running thousands of controlled experiments at Amazon, Booking.com, LinkedIn, and multiple Microsoft properties, we share seven rules of thumb for experimenters, which we have generalized from these experiments and their results.... To support these rule of thumb, we share multiple real examples, most being shared in a public paper for the first time.* ***(All quotes from Dmitriev et al. 2017.)***

For each example:

- the data sources are the actual web sites discussed;
- the user samples were all uniformly randomly selected from the triggered population;
- sample sizes are at least in the hundreds of thousands of users, with most experiments involving millions;
- all results were statistically significant with p-values below 0.05, and usually far below that; and

- the authors have personal experience with each of them.

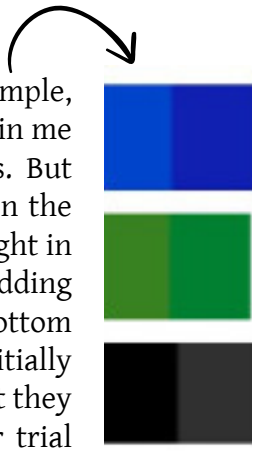
The seven rules of thumb are:

1. Small changes can have a big impact on key metrics.
2. Changes rarely have a big positive impact on key metrics.
3. Your mileage WILL vary.
4. Speed matters A LOT.
5. Reducing abandonment is hard; shifting clicks is easy.
6. Avoid complex designs: iterate.
7. Have enough users.

Rules four and six contain important messages with wide impact across the board: simplicity and performance are both really important!

## 1. SMALL CHANGES CAN HAVE A BIG IMPACT ON KEY METRICS

The message here is not to ignore what seem to be small changes, as sometimes they really can make a big impact. For example, opening search-result links in new tabs (once the team had got over the puzzle of the apparent page load-time increase) was a trivial coding change that ended up increasing clicks per user by 5%. That makes it one of the best features that MSN has ever implemented in terms of increasing user engagement.



Then there's the colour example, the like of which the cynic in me is all too ready to dismiss. But moving from the colours on the left to the colours on the right in the image below ended up adding over $10M to the annual bottom line. The results were initially treated with scepticism, but they held during a much larger trial with 32 million users.

***At Bing, two other small changes, which are confidential, took days to develop, and each increased ad revenues by about $100 million annually.***

So breakthroughs from small changes can happen, but they're rare. The risk of only focusing on small changes is incrementalism. Make room for big bets too!

## 2. CHANGES RARELY HAVE A BIG POSITIVE IMPACT ON KEY METRICS

Be suspicious when they do! (See Twyman's law.) If you see improvements in a segment metric, dilute the effect by the segment size. For example, a 10% improvement to a 1% segment has an overall impact of approximately 0.1%.

Most of our seemingly good results will be false positives. You've probably seen the common textbook example that works out the true chance after a positive test that you have some nasty medical condition. It's the same thing here, and we can use Bayes's rule in a similar way to

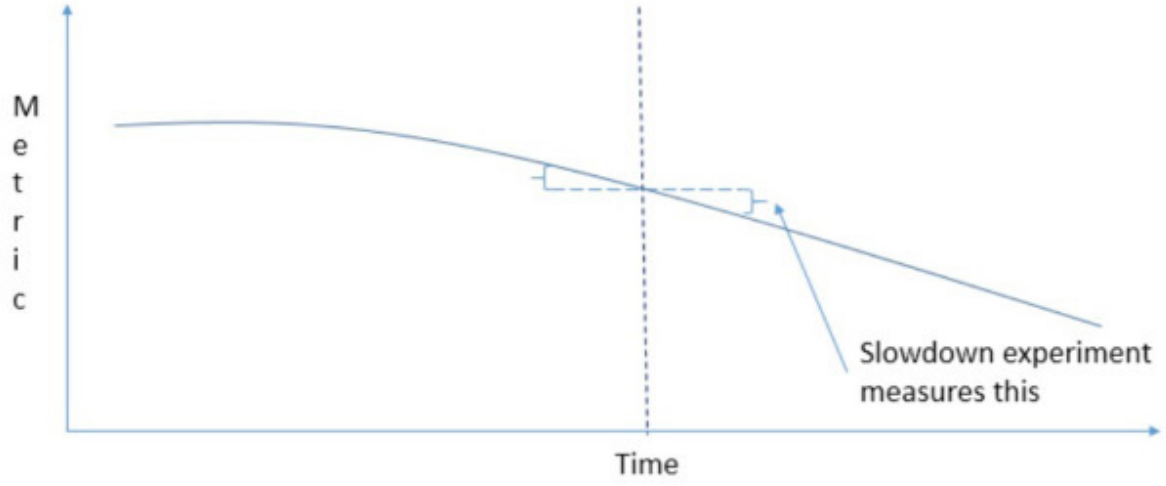



**Figure 3: Typical relationship between performace (time) and a metric of interest**

estimate our true chances of having struck gold.

***If the probability of a true positive effect is low, i.e., most ideas fail to move key metrics in a positive direction, then the probability of a true effect when the p-value is close to 0.05 is still low.***

The true rate of breakthrough improvements at Bing is estimated to be about one in 500, which makes the chance of an apparent breakthrough being the real deal about 3.1%!

One cunning way to move the odds in our favour is to observe changes deployed by statistically savvy competitors and try them on our own site:

***If our success rate on ideas at Bing is about 10-20%, in line with other search engines, the success rate of experiments from the set of features that the competition has tested and deployed to all users is higher.***

## 3. YOUR MILEAGE WILL VARY

In short, just because someone reported a great result elsewhere, it doesn't mean it's necessarily going to work in our own situation. Treat such case studies as idea generators but always test for yourself.

## 4. SPEED MATTERS A LOT

***How important is performance? Critical. At Amazon, 100-msec slowdown decreased sales by 1% as shared by Greg Linden. A talk by speakers from Bing and Google showed the significant impact of performance on key metrics.***

Performance is so important that when it improves, key metrics improve, and when it declines, key metrics decline. In this sense, it dominates many other factors.

***Web site developers that evaluate features using controlled experiments quickly realize web site performance, or speed, is critical. Even a slight delay to the page performance may impact key metrics in the treatment.***

Here's one way this can bite us. We want to test a new feature: for example, adding personalised recommendations to a page to try to increase engagement. Since we're just experimenting at the moment, the initial implementation might not be fully optimised. Let's say it adds 200 milliseconds to the page load time. Contrary to expectation, when we run the experiment, engagement goes down! Does that mean personalisation is a bad idea? Not necessarily. The load-time increase may

be having a bigger impact than the effect we're actually trying to measure. (We assume here that a faster implementation is indeed possible but takes more effort). So how can we untangle the impact of the performance change from experiment?

The authors have a cunning suggestion: run an experiment that injects a false delay in order to measure the performance impact's contribution!

***The best way to quantify the impact of performance is to isolate just that factor using a slowdown experiment, i.e., add a delay. Figure 3 shows a graph of depicting a common relationship between time (performance) and a metric of interest (e.g., click-through rate per page, success rate per session, or revenue per user). Typically, the faster the site, the better (higher in this example) the metric value.***

Using this information, we can now isolate the contribution from the performance change in each experiment that introduces a delay to see the true underlying impact.

Another observation here is that we clearly need to continuously monitor site performance to know when it changes and if treatments affect it.

Of course, sometimes site performance improves as well. Is it worth investing in such efforts?

***We can assess the impact to key metrics if the site were faster, helping us evaluate the ROI (return on investment) of such efforts. Using a linear approximation (1st-order Taylor expansion), we can assume the impact of the metric is similar in both directions (slowdown and speedup). As shown in Figure 3, we assume that the vertical delta on the right is similar to that on the left. By running slowdown experiments with different slowdown amounts, we have confirmed that a linear approximation is very reasonable for Bing.***

As an engineer at heart, I find it very satisfying that something like site performance matters so much. It's nice that, for example, changing CSS colours can help, but knowing that solid engineering likely makes a bigger and much more predictable impact feels good. In fact, if we look at the experiment success rates reported by Microsoft, then we have an overall success rate of just under one in three after accounting for false positives. If you want to move the needle on key metrics, by far the most certain thing you can invest in is improving your site performance. Nothing else seems to come close to the success from that. If you've run any of your own slowdown experiments or have data on the performance effect in your own organisations that you're able to share, I'd love to hear about it.

Disclaimer time: Accel is an investor in SKIPJAQ, and I'm a personal investor as well. SKIPJAQ improves your site performance by optimising the configuration of the stack (no code changes required). Run your own slowdown experiments and see what a performance improvement might mean in your own business. If it looks promising, why not give SKIPJAQ a try?

Here are some collected data points from the paper about the importance of performance:

- Slowdown experiments at Bing showed a 250-msec delay at the server reduces revenue at about 1.5% and click-through rate by 0.25%. "This is a massive impact," state the authors. A 500-msec delay would impact revenue by about 3%.
- An experiment at Google showed a delay of 100 to 400 msec caused searches per user to decline 0.2% to 0.6%.

***A slowdown experiment at Bing slowed 10% of users by 100 msec and another 10% by 250 msec for two weeks. The results of this controlled experiment showed that every 100 msec speedup improves revenue by 0.6%. The following phrasing resonated extremely well in our organization (based on translating the above to profit): an engineer that improves server performance by 10 msec (that's 1/30 of the speed that our eyes blink) more than pays for his fully loaded annual costs. Every millisecond counts.***

If we drill down further, we'll find that beyond baseline improvements, we can get bigger bang for our buck by improving performance of the most important

parts of the page. This is the whole "what is a good page load-time metric?" discussion. Bing's key time-related metric is time to success (TTS), which sidesteps some of these issues. Success is considered to be a clicked search-result link from which the user does not return within 30 seconds.

## 5. REDUCING ABANDONMENT IS HARD; SHIFTING CLICKS IS EASY

It's relatively easy to find improvements that make one part of your page/site better, but often these come at the expense of other parts of the page. Whether or not the results should then be considered positive needs to be evaluated in the global context.

> ***This rule of thumb is extremely important because we have seen many experiments (at Microsoft, Amazon, and reported by others) where a module or widget was added to the page with relatively good click-through rates. The claim is made that new module is clearly good for users because users are clicking. But if the module simply cannibalized other areas of the page,... it is only useful if those clicks are better, however "better" is defined for the site (e.g., they lead to higher success or purchases, etc.). Phrased differently: local improvements are easy; global improvements are much harder.***

## 6. AVOID COMPLEX DESIGNS: ITERATE

This one is simple, but it's an important point in the world of data-driven companies.

> *Good experimental design is vital to getting the best results from experiments.... Our experience is that* ***simple designs are best*** *in the online world and given the many pitfalls, they are easier to understand, run sanity checks, and thus more trustworthy. A complex design is usually not only unnecessary, but can hide bugs.* ***(Emphasis mine)***

Beyond simple site design, simple experiment design also wins. The authors give several examples of situations where too much was changed/tested at once, and when the results tanked, it was really hard to determine why.

> ***While the literature on multi-variable testing (MVT) is rich, and commercial products tout their MVT capabilities, we usually find it more beneficial to run simple uni-variable (e.g., A/B/C/D variant of a feature) or bi-variable designs.***

At Microsoft, teams are encouraged to deploy new code quickly and use experiments to control the rollout, starting with 1% increments.

## 7. HAVE ENOUGH USERS`

You need big enough samples! The authors provide a rule of thumb for assessing the number of users needed for a distribution of the mean to well approximate a normal distribution (a common assumption in experiment methodologies, per the central limit theorem). I'm out of space to do it justice, so if you're interested, do check out the full paper!

# THE EVOLUTION OF CONTINUOUS EXPERIMENTATION IN SOFTWARE PRODUCT DEVELOPMENT

**If you've been thinking, "we should probably do more A/B testing in my company," then "The Evolution of Continuous Experimentation in Software Product Development" (ICSE 2017) is for you. (An author's personal version can be found here.)**

*ICSE 2017*

Anchored in experiences at Microsoft, Fabijan et al. describe a tried and tested roadmap for gradually introducing a more data-driven culture in an organisation.

*...Despite having data, the number of companies that successfully transform into data-driven organizations stays low and how this transformation is done in practice is little studied.* ***(All quotes from Fabijan et al. 2017.)***

If we can get there though, the benefits can be large: "The impact of scaling out the experimentation platform across Microsoft is in hundreds of millions of dollars of additional revenue annually." (Though, to put this in context, in 2016 Microsoft generated $85.3 billion in revenue.)

It's interesting to see the perspective of the authors, who consider continuous experimentation to be the next logical step in the evolution of software-product-development practices. I'm reminded of Martin Fowler's "You must be this tall to use microservices."

***At first [companies] inherit the agile principles within the development part of the organization and expand them to other departments. Next, companies focus on various lean concepts such as eliminating waste, removing constraints in the development pipeline, and advancing towards continuous integration and continuous deployment of software functionality. Continuous deployment, however, is characterized by a bidirectional channel that enables companies not only to send data to their customers to rapidly prototype with them, but also to receive feedback data from products in the field.***

It's a great observation that continuous development (CD) facilitates this two-way flow of information. Once we get to CD though, the journey is not over; in fact, in a sense it's only just beginning:

***Controlled experimentation is becoming the norm in advanced software companies for reliably evaluating ideas with customers in order to correctly prioritize product-development activities.***

As we'll soon see, when we reach the upper levels of controlled-experimentation maturity, we're essentially doing continuous experimentation (which I'm going to christen "CE", so that we can go on a journey from CI to CD to CE!).

## THE EXPERIMENTATION EVOLUTION MODEL

The paper contains some background information on how the experimentation roadmap (the "experimentation evolution model") was arrived at. It combines historical data points collected over two years, a series of semi-structured interviews, and, most importantly, an internal model used by the Analysis and Experimentation team at Microsoft "to illustrate and compare progress of different product teams on their path towards data-driven development at scale". Even though it's based primarily in experiences at Microsoft (some of the expert participants also had experiences elsewhere), the authors believe it should have more general applicability, and that certainly seems credible to me. We can use it to benchmark our own company's sophistication and plan the next steps on our improvement journey.

***...We present the transition process model of moving from a situation with ad hoc data analysis towards continuous controlled experimentation at scale. We name this process the "experimentation evolution model."***

Most of what we need to know is encapsulated in this handy chart,

which shows maturity levels of crawl, walk, run, and fly across three dimensions: technical, organisational, and business.

| | Category/ Phase | Crawl | Walk | Run | Fly |
|---|---|---|---|---|---|
| Technical Evolution | Technical focus of product dev. Activities | (1) Logging of signals<br>(2) Work on data quality issues<br>(3) Manual analysis of experiments<br>Transitioning from the debugging logs to a format that can be used for data-driven development. | (1) Setting-up a reliable pipeline<br>(2) Creation of simple metrics<br>Combining signals with analysis units. Four types of metrics are created: debug metrics (largest group), success metrics, guardrail metrics and data quality metrics. | (1) Learning experiments<br>(2) Comprehensive metrics<br>Creation of comprehensive set of metrics using the knowledge from the learning experiments. | (1) Standardized process for metric design and evaluation, and OEC improvement |
| | Experimentation platform complexity | No experimentation platform<br>An initial experiment can be coded manually (ad-hoc). | Platform is required<br>3rd party platform can be used or internally developed. The following two features are required:<br>• Power Analysis<br>• Pre-Experiment A/A testing | New platform features<br>The experimentation platform should be extended with the following features:<br>• Alerting<br>• Control of carry-over effect<br>• Experiment iteration support | Advanced platform features<br>The following features are needed:<br>• Interaction control and detection<br>• Near real-time detection and automatic shutdown of harmful experiments<br>• Institutional memory |
| | Experimentation pervasiveness | Generating management support<br>Experimenting with e.g. design options for which it's not a priori clear which one is better. To generate management support to move to the next stage. | Experiment on individual feature level<br>Broadening the types of experiments run on a limited set of features (design to performance, from performance to infrastructure experiments) | Expanding to (1) more features and (2) other products<br>Experiment on most new features and most products. | Experiment with every minor change to portfolio<br>Experiment with any change on all products in the portfolio. Even to e.g. small bug fixes on feature level. |
| Organizational Evolution | Engineering team self-sufficiency | Limited understanding<br>External Data Scientist knowledge is needed in order to set-up, execute and analyse a controlled experiment. | Creation and set-up of experiments<br>Creating the experiment (instrumentation, A/A testing, assigning traffic) is managed by the local Experiment Owners. Data scientists responsible for the platform supervise Experiment Owners and correct errors. | Creation and execution of experiments<br>Includes monitoring for bad experiments, making ramp-up and shut-down decisions, designing and deploying experiment-specific metrics. | Creation, execution and analyses of experiments<br>Scorecards showing the experiment results are intuitive for interpretation and conclusion making. |
| | Experimentation team organization | Standalone<br>Fully centralized data science team. In product teams, however, no or very little data science skills. The standalone team needs to train the local product teams on experimentation. We introduce the role of Experiment Owner (EO). | Embedded<br>Data science team that implemented the platform supports different product teams and their Experiment Owners. Product teams do not have their own data scientists that would analyse experiments independently. | Partnership<br>Product teams hire their own data scientists that create a strong unity with business. Learning between the teams is limited to their communication. | Partnership<br>Small data science teams in each of the product teams.<br>Learnings from experiments are shared automatically across organization via the institutional memory features. |
| Business Evolution | Overall Evaluation Criteria (OEC) | OEC is **defined** for the first set of experiments with a few key signals that will help ground expectations and evaluation of the experiment results. | **OEC evolves** from a few key signals to a structured set of metrics consisting of Success, Guardrail and Data Quality metrics. Debug metrics are not a part of OEC. | OEC is **tailored** with the findings from the learning experiments. Single metric as a weighted combination of others is desired. | OEC is **stable**, only periodic changes allowed (e.g. 1 per year). It is also used for setting the performance goals for teams within the organization. |

## BEFORE WE GET STARTED: PREREQUISITES

Most of what we need to learn, we can pick up as we grow in experience, but there are two foundations we need in place right at the very beginning: a base level of statistical understanding and the ability to access product instrumentation data.

***To evaluate the product statistics, skills that are typically possessed by data scientists are required within the company. Here, we specifically emphasize the understanding of hypothesis testing, randomization, sample-size determination, and confidence-interval calculation with multiple testing.***

The Hypothesis Kit can guide us through some of the basics.

Regarding instrumentation data, we can add the necessary instrumentation as we go along, but what we really need to consider up front are the policies that allow experimenters access to the data, which in some domains "is a serious concern and needs to be addressed both on legal and technical levels."

## CRAWL

Welcome to our first experiment. With a policy in place, we're going to need some data to work with. It's time to start thinking a little differently about our logging system!

***In non-data-driven companies, logging exists for the purpose of debugging product features. This is usually very limited and not useful for analyzing how users interact with the products. Logging procedures in the organization need to be updated by creating a centralized catalog of events in the form of class and enumeration, and implemented in the product telemetry.***

At Microsoft, the raw data collected or sent from a product or feature (clicks, swipes, time spent loading a feature, and so on) are called "signals". From the signals, an analyst should be able to reconstruct the interactions a user had with the product.

We don't need an experimentation platform to run our first experiment; we can simply split our users between two versions of the same product and measure how the signals differ between the versions. For guidance on calculating the statistics behind a controlled experiment, the authors recommend "Controlled experiments on the web: survey and practical guide".

Since product teams may not have the necessary expertise, they'll require training and help from a standalone data-scientist team (or at least the MVP version: one data scientist).

***The main purpose of the first experiments is to gain traction and evangelize the results to obtain the necessary funding needed to develop an experimentation platform and culture within the company.***

In the crawl phase, we'll also define the first version of our overall evaluation criteria (OEC). Teams should be informed that this will develop over time.

## WALK

In the crawl phase, we had basic signals. But in the walk phase, we now define metrics based on those signals:

***Metrics are functions that take signals as an input and output a number per unit.... Microsoft recognizes three classes of signals for their products: action signals (e.g., clicks, page views, visits etc.), time signals (minutes per session, total time on site, page load time, etc.), and value signals (revenue, units purchased, ads clicked, etc.).***

Units of analysis vary depending on context, but could be, for example, per user, per session, per user-day, and per experiment.

*A popular research contribution from Google provides practical guidance on the creation of these metrics for measuring user experience on a large scale. ["**Measuring the User Experience on a Large Scale: User-Centered Metrics for Web Applications**".]*

While we introduce metrics, we'll want to start working with a basic experimentation platform. We can develop our own or use a commercial offering.

> ***" Controlled experimentation is becoming the norm in advanced software companies for reliably evaluating ideas with customers in order to correctly prioritize product development activities."***

***Regardless of the decision, the experimentation platform should have two essential features integrated in this phase: (1) power analysis and (2) pre-experiment A/A testing.***

Power analysis lets us determine the minimal sample size and A/A testing gives us confidence that the basic setup is working.

Experiment ownership moves the product manager, but execution, monitoring, and analysis is still done by data scientists. These data scientists may be embedded within the product teams, and they communicate and work with a central data-science team that's responsible for the platform. (I guess a lighter-weight version, if using a commercial experimentation platform, would be to form a guild).

***In contrast to the "crawl" phase, the OEC will evolve from a few key signals to a structured set of metrics consisting of success metrics (the ones we intend to improve), guardrail metrics (constraints that are not allowed to be changed), and data-quality metrics (the metrics that ensure that the experiments were set up correctly and results can be trusted).***

(I looked at these different metric types in the first review of this collection.)

## RUN

In the run phase, the number of experiments starts to really ramp up — let's say a hundred or so experiments a year. Metrics become more sophisticated, evolving from counting signals to more abstract concepts such as loyalty and success.

***To evaluate the metrics, product teams should start running learning experiments where a small degradation in user experience is intentionally introduced for learning purposes (e.g., degradation of results, slowdown of a feature). With such learning experiments, teams will have a better understanding of the importance of certain features and the effect that changes have on the metrics.***

To support all this activity, the experimentation platform also needs to evolve. At this stage, we probably want to add alerting, control of carry-over effects, and support for experiment iteration.

Alerting lets us divert traffic to the control if an emergency situation occurs.

***The naive approach to alerting on any statistically significant negative metric changes will lead to an unacceptable number of false alerts.***

This is just the other side of the coin of the early stopping problem. Guidance on avoiding this situation and developing useful alerting can be found in "Online Controlled Experiments at Large Scale" (see section 4.3).

Carry-over effects are when results from a prior experiment may colour a future one. A feature that re-randomizes the pop-

ulation between experiments can address this. Experiment iteration allows experiments to start on a small percentage of traffic and gradually expand.

***Initially, experiments in this phase should start on a small percentage of traffic (e.g., 0.5% of users assigned to treatment).... Over time, the percentage should automatically increase... if no alerts on guardrail metrics were triggered beforehand.***

This helps to minimise the risk of user harm from experiments that have a negative impact.

At the run phase, experimentation is simply "what we do".

***Product teams should be experimenting with every increment to their products (e.g., introduction of new features, algorithm changes, etc.). Experimenting should be the norm for identifying the value of new features as well as for identifying the impact of smaller changes to existing features.***

Data scientists are now employed directly by the product teams, and are trained by the central-platform data-science team to become local operational data scientists.

## FLY

In the fly phase, we can't move without running an experiment, it seems!

***Controlled experiments are the norm for every change to any product in the company's portfolio.... Every small change to any product in the portfolio (e.g., a minor bug fix) should be supported by data from a controlled experiment.***

(Minor bug fixes? Really? I guess the notion here is that any change, even a bug fix, could unintentionally impact metrics. For example, suppose the fix introduces a small performance delay....)

At this stage, the OEC is stable and well defined. Product teams invest in standardising metric design and evaluation practices, and schedule activities to update the existing OEC when needed.

With thousands of experiments simultaneously active, we'll need a bit more sophistication in our platform, too:

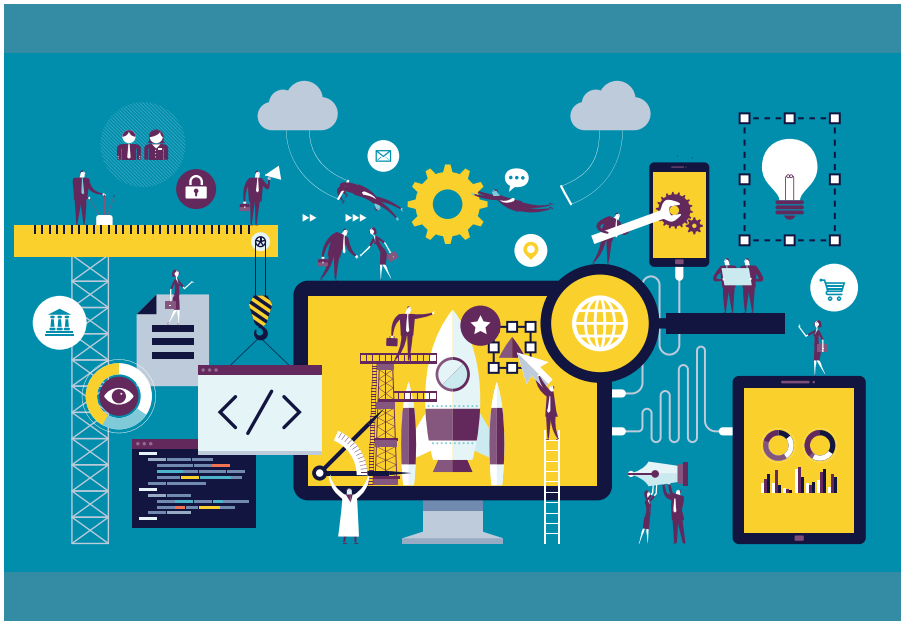
- Interaction control and detection help us to figure out when experiments with conflicting outcomes are run on the same set of users.
- The alerting on guardrail metrics that happened based on periodic evaluation in the run phase needs to become near-real time, and the emergency shutdown functionality is automatic.
- We should introduce some way to capture experiments that have been run and their results as an institutional memory.

Where does your company fit on the experimentation evolution model?

# GOOGLE VIZIER: A SERVICE FOR BLACK-BOX OPTIMISATION

**We just looked at the role of an internal (or external) experimentation platform. In "Google Vizier: A Service for Black-Box Optimization" (KDD 2017), Golovin et al. from Google remind us that such experimentation is just one form of optimisation.**

*Golovin et al. (KDD 2017)*



Google Vizier is an internal Google service for optimising pretty much anything. It is a scalable, state-of-the-art internal service for black-box optimization within Google. And if you use, for example, Google's public HyperTune system (part of the Google Cloud Machine Learning Engine service), then you're using Vizier under the covers.

The very opening sentences of the abstract are worth dwelling on for a moment:

> *Any sufficiently complex system acts as a black box when it becomes easier to experiment with than to understand. Hence, black-box optimization has become increasingly important as systems have become more complex.* ***(All quotes from Golovin et al. 2017.)***

This is recognising that something important is happening all around us: we've made systems of such emergent complexity that we don't truly understand them anymore. Or perhaps we could understand them with enough effort, but that barrier is so high that it's easier just to try a few things and see what happens.

At the tail end of the paper, we find something else very much of our times, and so very Google. If you ever find yourself in the Google cafeteria, try the cookies. Vizier optimised the recipe

for those cookies, with trials sent to chefs to bake and feedback on the trials obtained by Google workers eating the results.

***It has already proven to be a valuable platform for research and development, and we expect it will only grow more so as the area of black-box optimization grows in importance. Also, it designs excellent cookies, which is a very rare capability among computational systems.***

## BLACK-BOX OPTIMISATION

The delightful thing about black-box optimisation and Vizier is that, on the outside, it's so simple.

***Black-box optimization is the task of optimizing an objective function / with a limited budget for evaluations. The adjective "black-box" means that while we can evaluate / for any /, we have no access to any other information about /, such as gradients or the Hessian.***

There's a certain cost to performing each evaluation, so we want to be smart about how we explore the space, with the overall goal of finding the best operating parameters $x$ as quickly as possible. All we need to get started is a system whose performance $f(x)$ can be measured as a function of adjustable parameters.

A trial is a list of parameter values, $x$ (I keep wanting that to be $\mathbf{x}$, but I'll stick with the notation in the paper) that will lead to a single evaluation of $f(x)$.

A study is a single optimisation run over a feasible space. We describe the space when setting up the study. It is assumed that $f(x)$ does not change in the course of a study.

A worker is a process responsible for evaluating a pending trial and calculating its objective value.

There are many different approaches to optimisation (we've looked at several in previous editions of The Morning Paper), including Bayesian optimisation, derivative-free optimisation, sequential experimental design, and variations on multi-armed bandits. Vizier supports multiple different algorithms under the cover, with a default of batched Gaussian-process bandits.

## SAMPLE USE CASES AT GOOGLE

**Hyper-parameter tuning of machine learning models:** "Our implementation scales to service the entire hyper-parameter tuning workload across Alphabet, which is extensive." (No kidding!) In fact, the authors estimate that over a billion people have benefited from better user experiences via improvements to production models made by Vizier.

**Automated A/B testing of Google web properties:** Google tunes user-interface parameters as well as traffic-serving parameters (e.g., "How should the search results returned from Google Maps trade search relevance for distance from the user?")

> ***" If you ever find yourself in the Google cafeteria, try the cookies. The recipe for those cookies was optimised by Vizier, with trials sent to chefs to bake, and feedback on the trials obtained by Googler's eating the results."***

**Physical design and logistical problems:** Details of the cookie optimisation are in section 5.3, and bring out a few interesting details such as the ability to mark a given trial as infeasible.

## VIZIER DESIGN GOALS AND USER INTERFACE

Vizier is designed to operate at large scale (it's Google!) and to be easy to use, requiring minimal user configuration and setup. It operates as a service with a minimal RPC interface (there are dashboards etc. too, of course).

Users provide a study configuration that specifies the set of parameters in the search space, along with the feasible sets for each. Parameter types can be doubles, integers, discrete, or categorical. Vizier takes care of normalisation behind the scenes. With that, we're off to the races. A typical usage of Vizier looks like this:

```
# Register this client with the Study, creating it if
# necessary.
client.LoadStudy(study_config, worker_handle)
while (not client.StudyIsDone()):
  # Obtain a trial to evaluate.
  trial = client.GetSuggestion()
  # Evaluate the objective function at the trial parameters.
  metrics = RunTrial(trial)
  # Report back the results.
  client.CompleteTrial(trial, metrics)
```

The most important operations in the Vizier interface are:

- `CreateStudy`, which which we've just seen;
- `SuggestTrials`, which kicks off a background process to generate a set of suggested trials to be run;
- `AddMeasurementToTrial`, which allows clients to provide intermediate metrics during the evaluation of a trial (used by automated stopping rules);
- CompleteTrial, which changes a trial's status to completed and provides the objective value; and
- `ShouldTrialStop` , which kicks off a background process to figure out whether a trial should be stopped early.

Sophisticated end users can plug in their own algorithms via the Vizier "playground".

The web dashboard can be used to monitor and interact with Vizier studies. Common use cases are tracking progress, interactive visualisations, study management (CRUD), requesting new suggestions, early stopping, and so on.


**Figure 3: A section of the dashboard for tracking the progress of Trials and the corresponding objective function values. Note also, the presence of actions buttons such as Get Suggestion for manually requesting suggestions.**

The interactive visualisations are based on the visualisation technique of parallel coordinates.

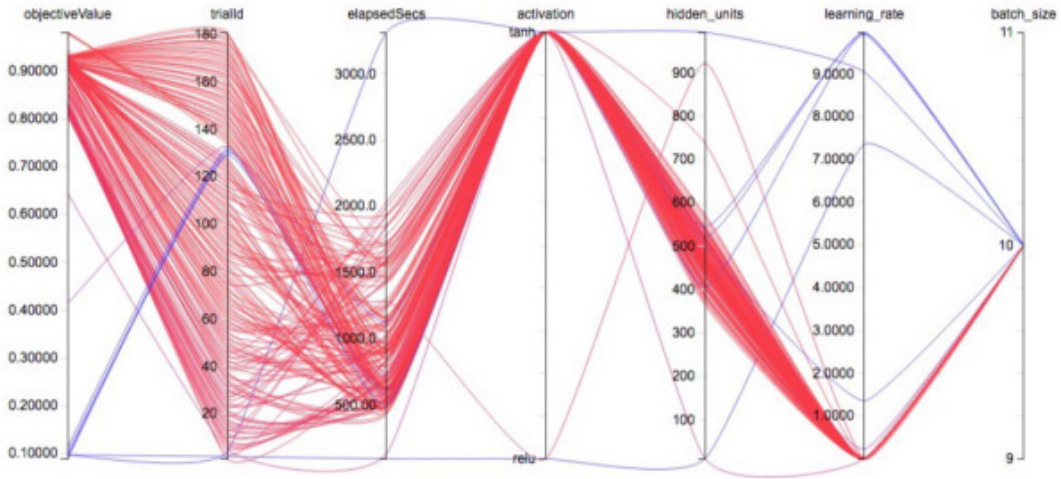



**FIgure 4: The Parallel Coordinates visualisaion [18] is used for examining results from different Vizier runs. It has the benefit of scaling to high dimensional spaces (~15 dimensions) and works with both numerical and categorical paramenters. Additionaly, it is interactive and allows various models of slicing and ficing data.**

## VIZIER ON THE INSIDE

The Vizier suggestion service is partitioned across multiple Google datacentres.

***Each instance of the suggestion service potentially can generate suggestions for several studies in parallel, giving us a massively scalable suggestion infrastructure.***

For studies under 1,000 trials, Vizier defaults to using batched Gaussian-process bandits. For studies with tens of thousands of trials or more, Vizier supports `RandomSearch` and `GridSearch` as first-class choices, and many other published algorithms can be used through the playground. Data normalisation is handled before trials are presented to the trial suggestion algorithms, and its suggestions are transparently mapped back to the user-specified scaling.

***In some important applications of black-box optimization, information related to the performance of a trial may become available during trial evaluation. Perhaps the best example of such a performance curve occurs when tuning machine-learning hyper-parameters for models trained progressively (e.g., via some version of stochastic gradient descent). In this case, the model typically becomes more accurate as it trains on more data, and the accuracy of the model is available at the end of each training epoch.***

In these cases, Vizier supports automated early stopping using one of two rules: a performance-curve stopping rule, and a median stopping rule. (The paper makes no mention of guidance given to users or suitability of early stopping in the A/B-test use case). The performance-curve stopping rule uses regression on performance curves to predict the final objective value of a trial. The median stopping rules stop a pending trial if its best objective value is strictly worse than the median value of the running averages of all completed trials' objectives.

Vizier also supports transfer learning, using outcomes of previous optimisations to inform future ones:

***...Our strategy is to build a stack of Gaussian-process regressors, where each regressor is associated with a study, and where each level is trained on the residuals relative to the regressor below it. Our model is that the studies were performed in a linear sequence, each study using the studies before it as priors.***

Consider as an example a production machine-learning system that is expensive to train, so the number of trials that can be run for hyper-parameter tuning is limited. However, the system may be worked on year after year:

***Over time, the total number of trials spanning several small hyper-parameter tuning runs can be quite informative. Our transfer-learning scheme is particularly well suited to this case.***

## HOW WELL DOES VIZIER WORK?

Section 4 of the paper provides brief evaluation details. In Figure 6 we can see the effectiveness of

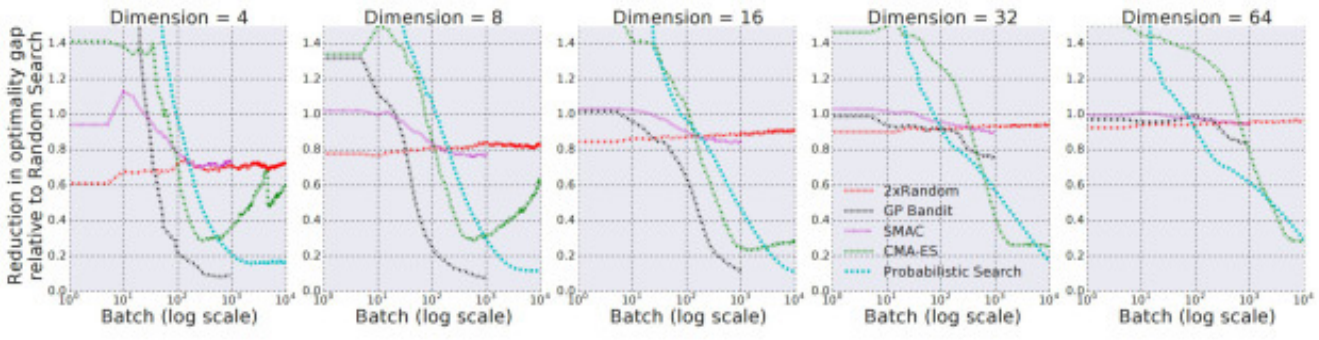



**FIgure 6: Ratio of the average optimality gap of each optimizer to that of Random Search at a given number of samples. The 2 x Random Search is a Random Search allowed to sample two points at every step (as opposed to a single point for the other algorithms).**

the default Gaussian-process bandit algorithm compared against a baseline of random search and a number of other search algorithms.

And here we see the benefits of transfer learning over time:

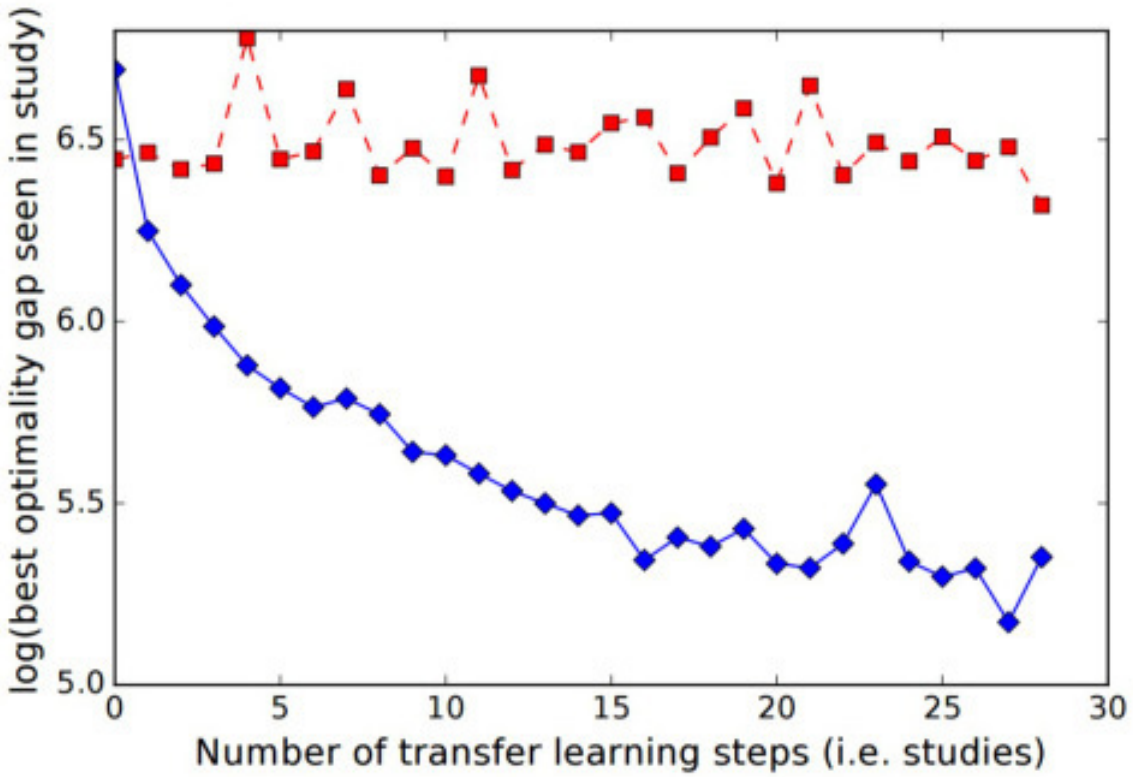



**FIgure 7: Convergence of transfer learining in a 10 dimensional space. This shows a sequence of studies with progressive transfer learining for both GP Bandit (blue diamonds) and Random Search (Red sqares) optimizers. The X-axis sgows the index of the study, i.e. the number of times that transfer learning has been applied; the Y-axis shows the log of the best mean optimality gap seen in the study (see Section 4.1). Each study contains six trials; for the GP Bandit-based optimizer the previous studies are used as priors for transfer learning. Note that the GP bandits shows a consistent improvement in optimality gap from study to study, thus demonstrating an effective transfer of knowledge from earlier trials; Random Search does not do transfer learning.**

# TFX: A TENSORFLOW-BASED PRODUCTION-SCALE MACHINE-LEARNING PLATFORM

**What world-class looks like in online product and service development has been undergoing quite the revolution over the last few years. The series of papers we've been looking at recently can help us to understand where the bar is (it will have moved on again by the time most companies get there, of course!)**

The new baseline: so far we've embraced automated testing, continuous integration, continuous delivery, and perhaps continuous deployment, and we have the sophistication to roll out new changes in a gradual manner, monitor behaviour, and stop or roll back when a problem is detected.

On top of this, we've put in place **a sophisticated metrics system and a continuous experimentation platform**.

Due to the increasing complexity of systems, we might also need to extend this to a general-purpose **black-box optimization platform**.

But we're not done yet! All those machine-learning models we've been optimising need to be trained, validated, and served somehow. We need a machine-learning *platform*. That's the topic of Baylor et al.'s "**TFX: A TensorFlow-Based Production Scale Machine Learning Platform**" (KDD 2017), which describes the machine-learning platform inside Google.

(In a similar manner, if anyone knows of a good paper capturing at a high level what a state-of-the-art security platform looks like, please let me know).

Just to be clear here, I'm not saying we necessarily need to build our own versions of all of these platforms. It's more about embracing their use as part of everyday practices, and for many organisations that can do that by renting, that's going to be the best choice.

## WHY DO WE NEED A MACHINE-LEARNING PLATFORM?

The code that implements our machine-learning model is only a tiny part of

what goes into using machine learning in production systems. We looked at some of the lessons learned in deploying machine learning at scale previously in "Machine Learning: The High-Interest Credit-Card of Technical Debt". Data needs to be transformed and validated, model validation needs to be coupled with data validation so that bad (yet validated) models don't reach production, and we need a scalable serving infrastructure. The key components of a machine-learning platform (i.e., TFX) are illustrated in the figure below:

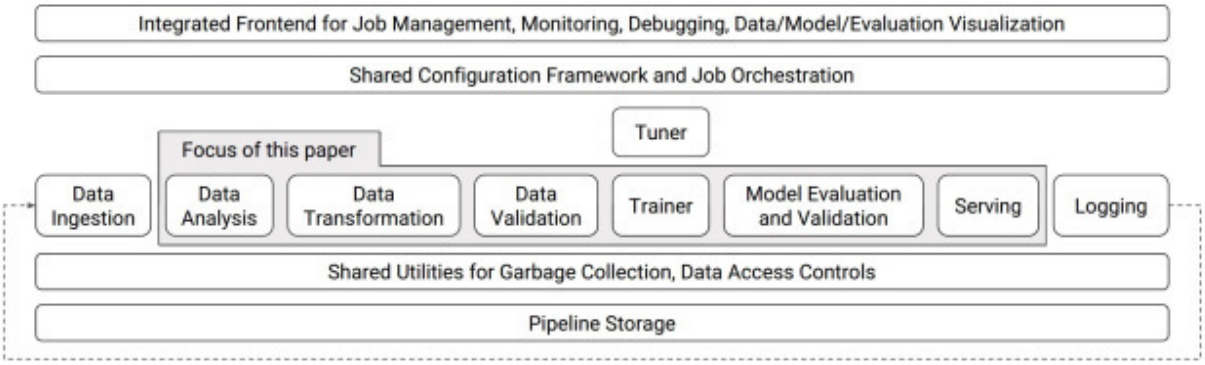



**FIgure 1: High-level component of a machine learning platform.**

*By integrating the aforementioned components into one platform, we were able to standardize the components, simplify the platform configuration, and* ***reduce the time to production from the order of months to weeks****, while providing platform stability that minimizes disruptions.* ***(All quotes from Baylor et al. 2017, and emphasis mine.)***

## DATA ANALYSIS, TRANSFORMATION, AND VALIDATION

We're only as good as our data. Capturing data anomalies early, before they propagate downstream, saves a lot of wasted time later on.

***Small bugs in the data can significantly degrade model quality over a period of time in a way that is hard to detect and diagnose (unlike catastrophic bugs that cause spectacular failures and are thus easy to track down), so constant data vigilance should be a part of any long running development of a machine-learning platform.***

To establish a baseline and monitor for changes, TFX generates a set of descriptive statistics for each dataset fed to the system. These include feature presence and values, cross-feature statistics, and configurable slices. These statistics need to be computed efficiently at scale and can be expensive to compute exactly on large training data. In this case, distributed streaming algorithms that give approximate results can be used.

TFX also includes a suite of data transformations that support feature wrangling. As an example, TFX can generate feature-to-integer mappings, known as "vocabularies". It's easy to mess things up when transformations differ in subtle ways between training and serving. TFX automatically exports any data transformations as part of the trained model to help avoid these issues.

To perform validation, TFX relies on a schema that provides a versioned description of the expected data. The schema describes features and their expected types, valency, and domain. TFX can help users to generate the first version of their schema automatically.

TFX takes care to provide useful and informative anomaly alerts so that users don't become blind to them.

***We want the user to treat data errors with the same rigor and care***

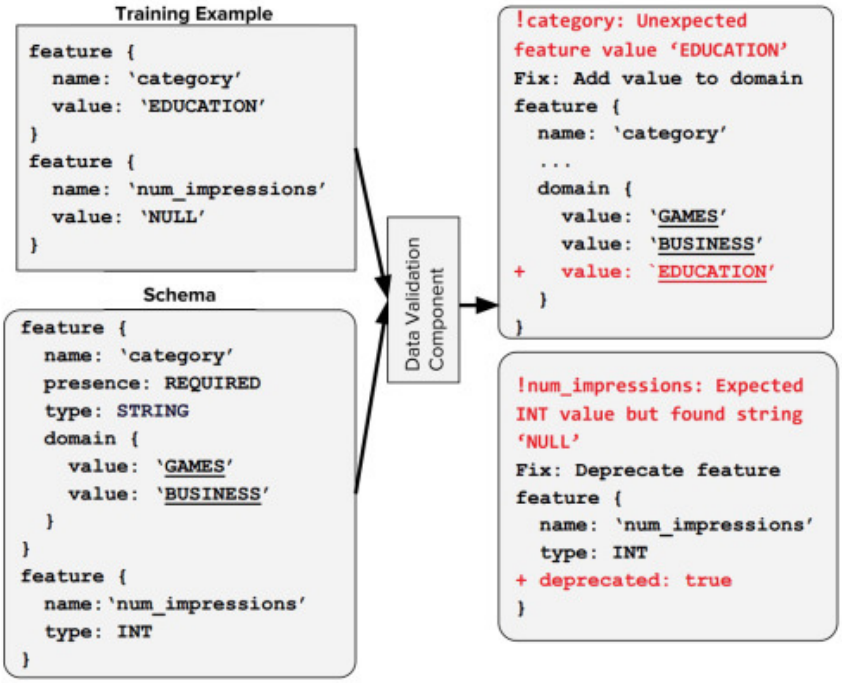



**FIgure 2: Sample validation of an example against a simple schema for an app store application. The schema indicates that the expected type for the 'category' feature is STRING and that the 'num_impressions' feature is INT. Furthermore, the category feature must be present in all example and assume values from the specified domain. On validating the example against this scehma, the module detects two anomalies with simple explanations as well as suggested schema modifications. The first suggestion reflects a schema change to account for an evolution of the data (the appearance of a new value). In contrast, the second suggestion reflects the fact that there is an underlying data problem that needs to be fixed, so the feature should be marked as problematic while the problem is being investigated.**

***that they deal with bugs in code. To promote this practice, we allow anomalies to be filed just like any software bug where they are documented, tracked, and eventually resolved.***

## TRAINING

Once our modelling code (written in TensorFlow, of course) is integrated with TFX, we can easily switch learning algorithms. Continuously training and exporting machine-learning models is a common production use case, yet in many scenarios it is too time and resource intensive to retrain models from scratch each time.

***For many production use cases, freshness of machine-learning models is critical.... A lot of such use cases also have huge training sets (O(100B) data points) which may take hours (or days in some cases) of training.... This results in a trade-off between model quality and model freshness. Warm-starting is a practical technique to offset this trade-off and, when used correctly, can result in models of [the] same quality as one would obtain after training for many hours in much less time and fewer resources.***

Warm-starting is built into TFX, and the ability to selectively warm-start features of a network has been implemented and open-sourced in TensorFlow. When training a new version of a network using warm-starting, the parameters corresponding to warm-start features are initialised from a previously trained version of

the model, and fine tuning begins from there.

## EVALUATION AND VALIDATION

TFX contains an evaluation and validation component designed to ensure that models are "good" before serving them to users.

***Machine-learned models are often parts of complex systems comprising a large number of data sources and interacting components, which are commonly entangled together. This creates large surfaces on which bugs can grow and unexpected interactions can develop, potentially to the detriment of end-user experiences via the degradation of the machine-learned model.***

How are new models rolled out to production? Via A/B testing, of course! Models are first evaluated on held-out data to determine if they are promising enough for a live test, with TFX providing proxy metrics that can approximate business metrics. For models that pass this test, teams progress to product-specific A/B experiments to determine how the models actually perform on live traffic and relevant business metrics.

***Once a model is launched to production and is continuously being updated, automated validation is used to ensure that the updated models are good. We validate that a model is safe to serve with a simple canary process. We evaluate prediction quality by comparing the model quality against a fixed threshold as well as against a baseline model (e.g., the current production model).***

If a model update fails validation, it is not pushed to serving, and an alert is generated for the owning product team. Initially, teams weren't sure whether they needed or wanted the validation functionality, but after encountering real issues in production that validation could have prevented, they all of a sudden became very keen to switch it on!

## SERVING

TensorFlow Serving provides a complete serving solution for deploying machine-learned models to production environments. Production serving requires, among other things, low latency and high efficiency. For most models, a common TensorFlow data format is used, but a specialized protocol buffer parser was built with lazy parsing for data-intensive (versus CPU-intensive) networks such as linear models.

***While implementing this system, extreme care was taken to minimize data copying. This was especially challenging for sparse data configurations. The application of the specialized protocol buffer parser resulted in a speedup of 2-5 times on benchmarked datasets.***

## TFX AND GOOGLE PLAY

One of the first teams within Google to move onto the TFX platform was Google Play, who use it for their recommender system. This system recommends relevant Android apps to Play app users when they visit the store homepage. The training dataset has hundreds of billions of examples, and in production the system must handle thousands of queries per second with strict latency requirements (tens of milliseconds).

***As we moved the Google Play ranking system from its previous version to TFX, we saw an increased velocity of iterating on new experiments, reduced technical debt, and improved model quality.***

## NEXT STEP: EXPLANATIONS?

***...As machine learning becomes more prevalent, there is a strong need for understandability, where a model can explain its decision and actions to users [and, I should note, that this can be a legal requirement in many cases with new legislation such as GDPR]. We believe the lessons we learned from deploying TFX provide a basis for building an interactive platform that provides deeper insights to users.***

# HERE IS WHAT WE'VE COVERED IN THE PREVIOUS ISSUES



## GRAY FAILURE: THE ACHILLES' HEEL OF CLOUD-SCALE SYSTEMS

## TRAJECTORY RECOVERY FROM ASH: USER PRIVACY IS NOT PRESERVED IN AGGREGATED MOBILITY DATA

## IOT GOES NUCLEAR: CREATING A ZIGBEE CHAIN REACTION

## SYSTEM PROGRAMMING IN RUST: BEYOND SAFETY

## MOSAIC: PROCESSING A TRILLION-EDGE GRAPH ON A SINGLE MACHINE

## TOWARD SUSTAINABLE INSIGHTS, OR WHY POLYGAMY IS BAD FOR YOU

## WHEN DNNS GO WRONG - ADVERSARIAL EXAMPLES AND WHAT WE CAN LEARN FROM THEM

## THOU SHALT NOT DEPEND ON ME: ANALYSING THE USE OF OUTDATED JAVASCRIPT LIBRARIES ON THE WEB

## REDUNDANCY DOES NOT IMPLY FAULT TOLERANCE: ANALYSIS OF DISTRIBUTED STORAGE REACTIONS TO SINGLE ERRORS AND CORRUPTIONS

## THE CURIOUS CASE OF THE PDF CONVERTER THAT LIKES MOZART