A practical view on A/B tests

Tim Zhao and Jason Ma 04/21/2024

Tim Zhao: Director of TAB - Tencent A/B Tests platform
Jason Ma: Tech Lead of TAB

A/B Test: Gold Standard for making data-driven decisions

- Randomization: This random assignment helps to ensure that the two groups are similar in all respects except for the treatment being tested. This helps to eliminate bias and confounding variables, making it easier to attribute any differences in outcomes to the treatment itself.

- Controlled Environment: A/B tests are typically conducted in a controlled environment where only the variable of interest is changed. This allows for a clear cause-and-effect relationship to be established between the variable being tested and the outcome.

- Quantitative Results: They not only provide which group is better, but also how much one group is better than the other one.

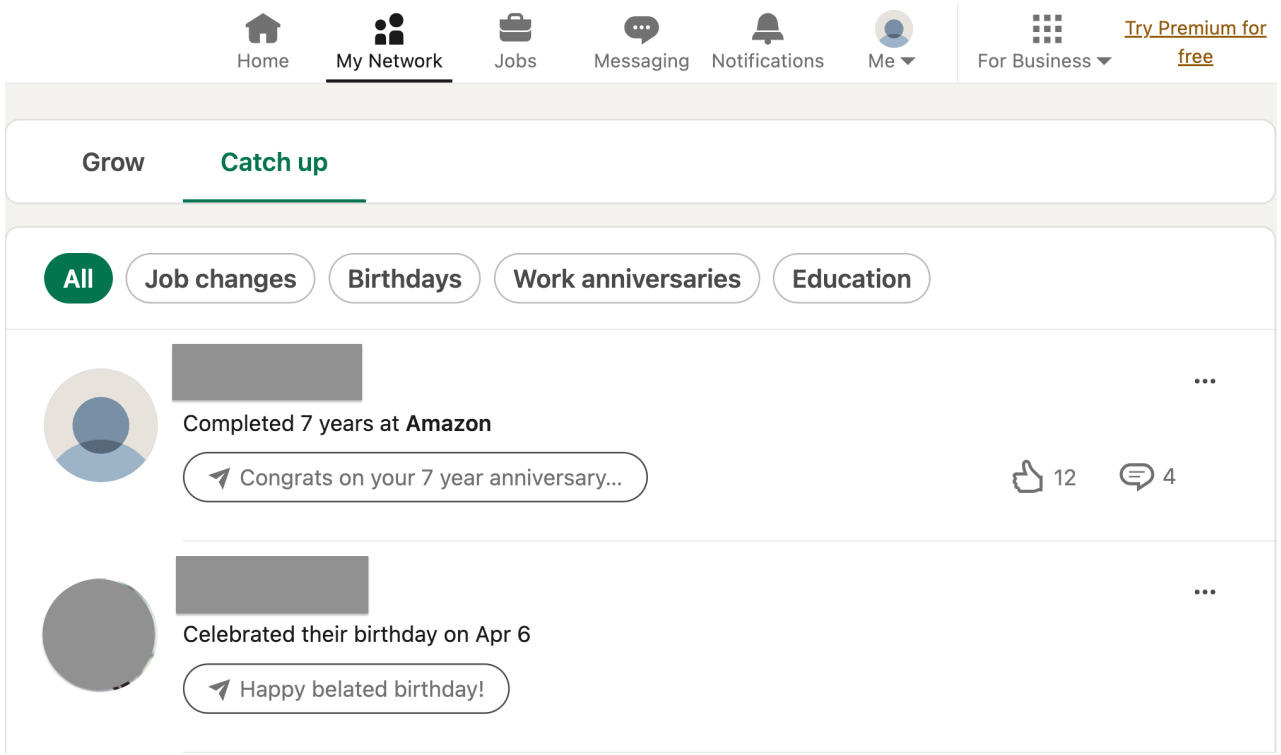
- Replicability: A/B tests can be easily replicated and validated.

- Simplicity: A/B testing is relatively simple to understand and implement.

Syllabus

- Experimentation in industry
- Decision Quality
  - Pre-experiment bias
  - SRM
  - Invalid operations
  - Backtest & Holdout
  - Metric centric view
- Decision Efficiency
  - Rapid experiment iteration
  - Parameter tuning
  - Variance reduction
  - Reduce the machine cost
- Demo

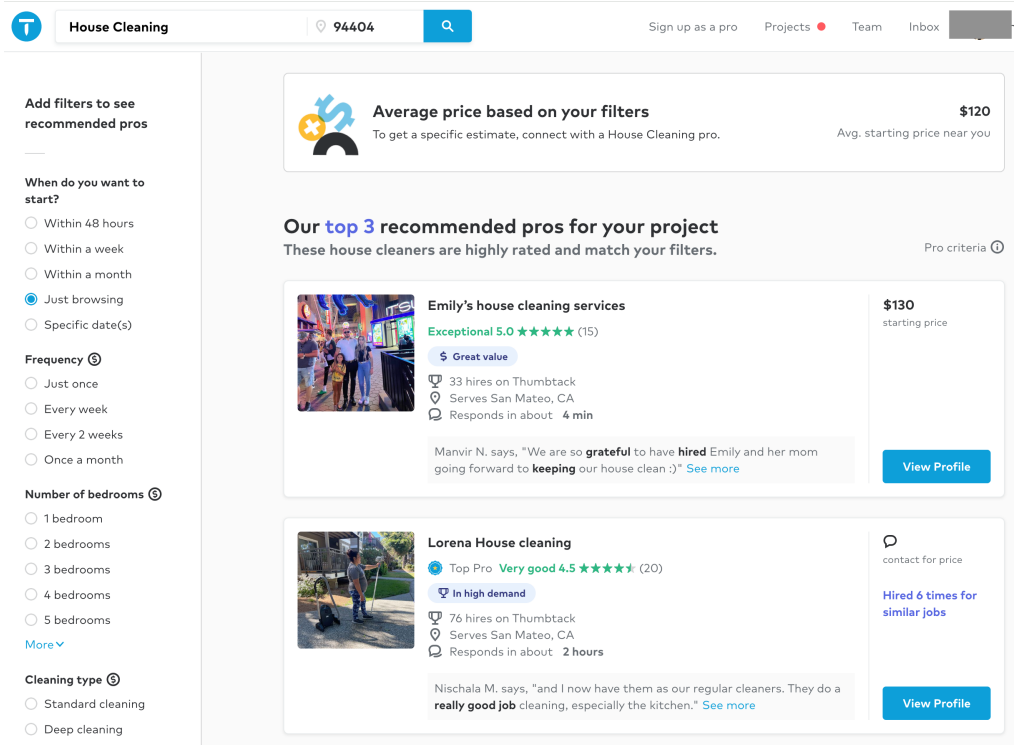
LinkedIn – Relationship matters



- Props: professional relationship opportunity
- Task: migrate the existing backend server, which is written in Python, to a new one written in Scala
- Topline: user engagement metrics
- Maintaining two different branches of code can be laborious. Given that the migration itself does not alter the ranking logic, there is a reluctance to conduct experiments.
- However, the experiment yielded a statistically significant negative result. Upon further investigation, it was discovered that some legacy code had caused a bug.

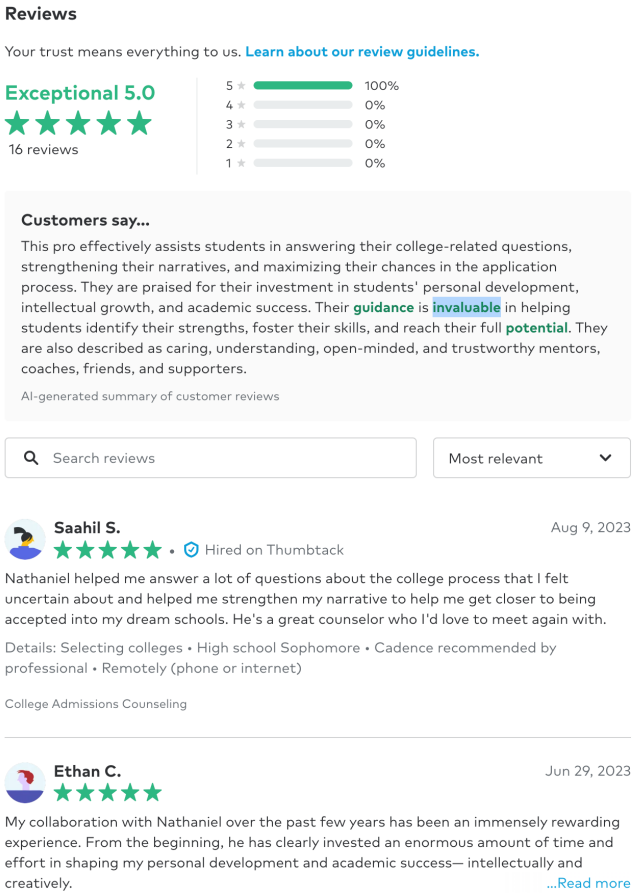
Thumbtack

- A unicorn-status online marketplace that facilitates connections between individuals and local professionals.



- Despite being a startup, the company fosters a strong data-driven culture.
- In their early stage, they only have around 100 engineers but four of whom were dedicated to building an in-house experimentation platform.
- Experiment reviews and discussions constitute the major part of weekly team meeting.

Thumbtack - Drive more reviews



- Goal metric: Is the total number of reviews a good metric?
- Guardrail metric: The quality of reviews.
- How should we drive the goal metric?
  - Conduct data analysis or learn from similar cases.
  - Formulate a reasonable hypothesis.
  - Run experiments.
  - Review the experiment, craft a well-articulated narrative for the results, and make a decision - whether to Ship, Abandon, or Iterate.
  - When the conclusion is unclear, it's necessary to delve deeper into your data from various dimensions and consider additional factors for decision making.
- In early stage, good idea can lead to substantial rewards (an increase in the two digits relative delta). Case study: Two-sided reviews.
- Became real fan of experimentation.

Meta

- Tens of thousands of experiments are conducted concurrently on a daily basis across platforms such as Facebook, Instagram, WhatsApp, Messenger, and Oculus.
- Layers and parameters are extensively utilized to accelerate the iteration of experiments.
- Holdouts are commonly adopted to assess the impact of varying organizational hierarchies every half-year.
- The quality and efficiency of experiments are crucial factors for the company's business.
- The company invests a significant amount of resources in experimentation, including new hire training during bootcamps, comprehensive classes for data scientists and product managers. Hundreds of millions of dollars are allocated annually to cover the costs associated with experimentation.

Experimentation personas

PMs: Can my idea can improve the goal metric? If so, by how much?

Data scientists: Are the experiments of high quality? What are the type I and type II error rates? How to set up metrics/domains/layers/holdouts for an org?

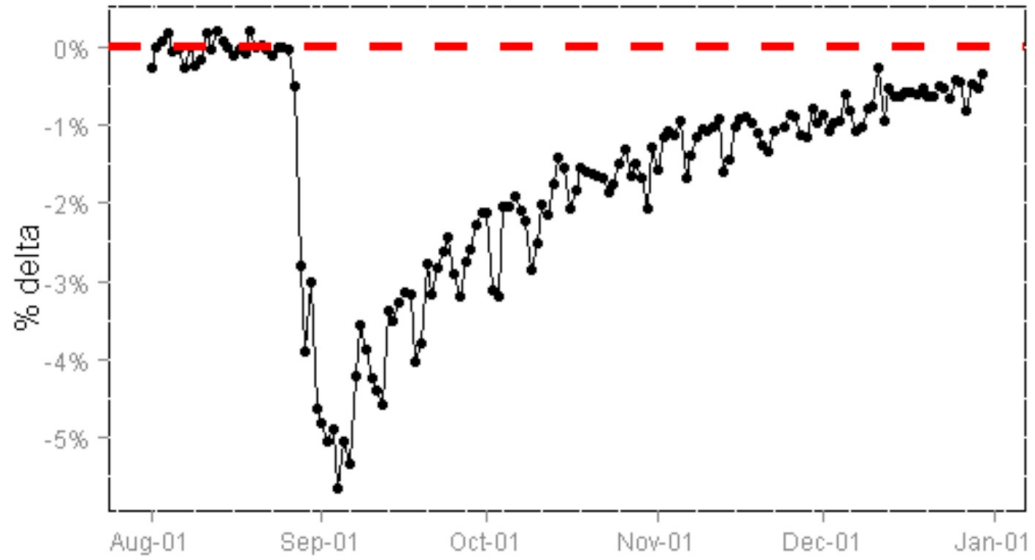
Metric defenders: Which experiments are negatively impacting my metric and how can I mitigate them?

Team leaders: How many features are shipped and what is their overall impact?

Engineers: Is it straightforward to run the experiments? Will they bring any harm to system stability or the maintainability of the codebase?

Pre-experiment bias – Carryover

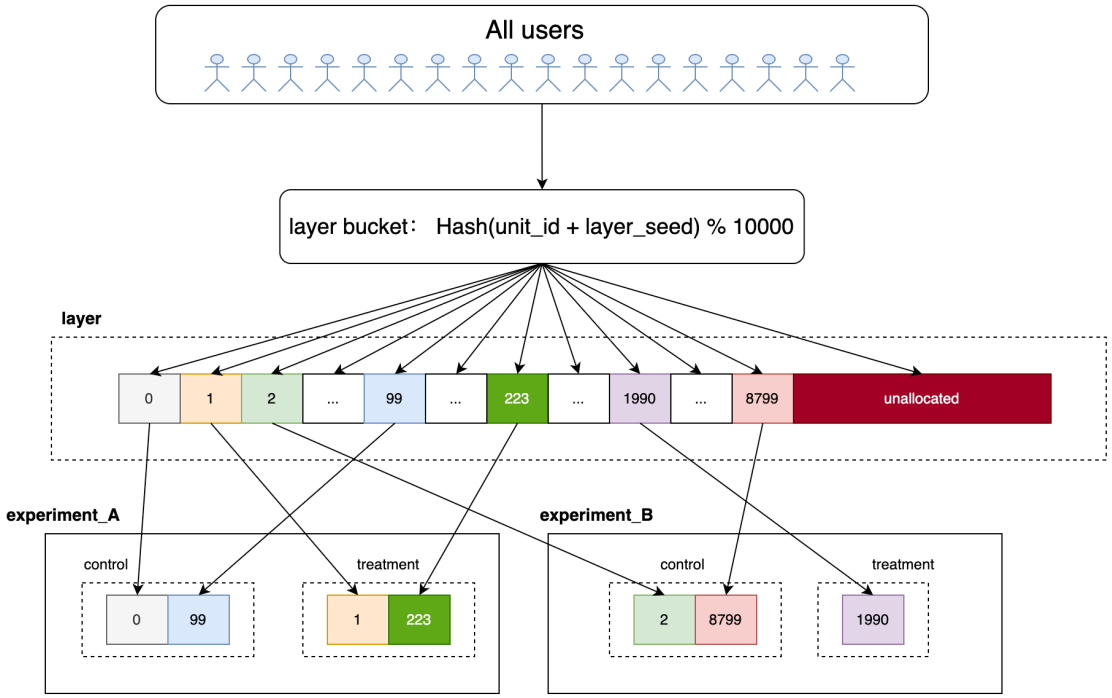
- Before the experiment is started, there is a statistically significant difference between the control and treatment groups.
- How to split the users evenly into two groups is a challenging problem for the experiment platform.
- Carryover Effect is a common factor that can cause pre-experiment bias (of course there are other factors like intrinsic user characteristics).



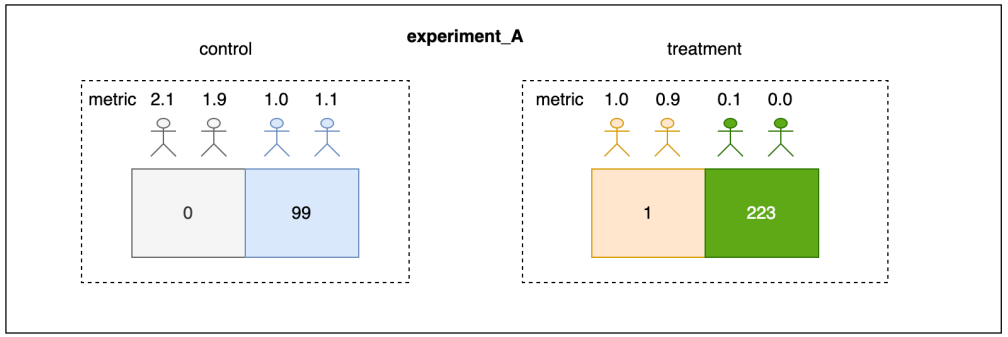
Long lasting (3 Months) Carryover effects

Trustworthy Online Controlled Experiments: Five Puzzling Outcomes Explained https://exp-platform.com/puzzling-outcomes/

Pre-experiment bias – randomization

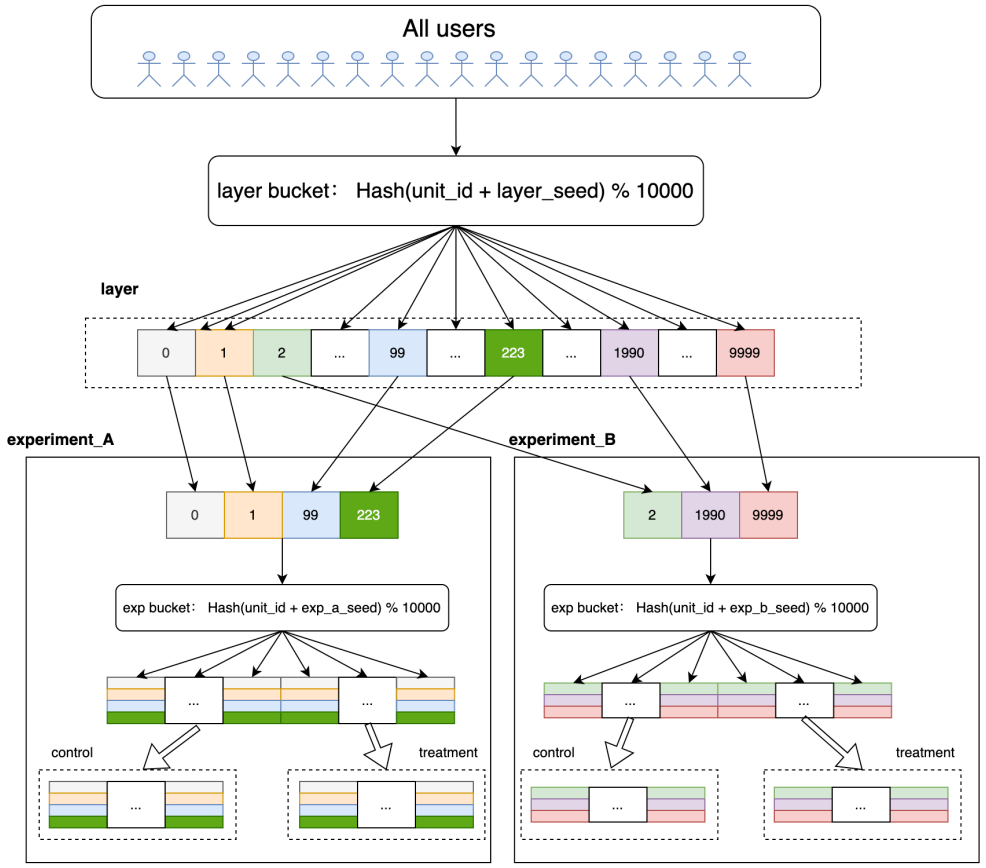


One level randomization



In a durable layer, the underlying distribution of the metric in each bucket will be different due to the carryover effect.

Pre-experiment bias – randomization

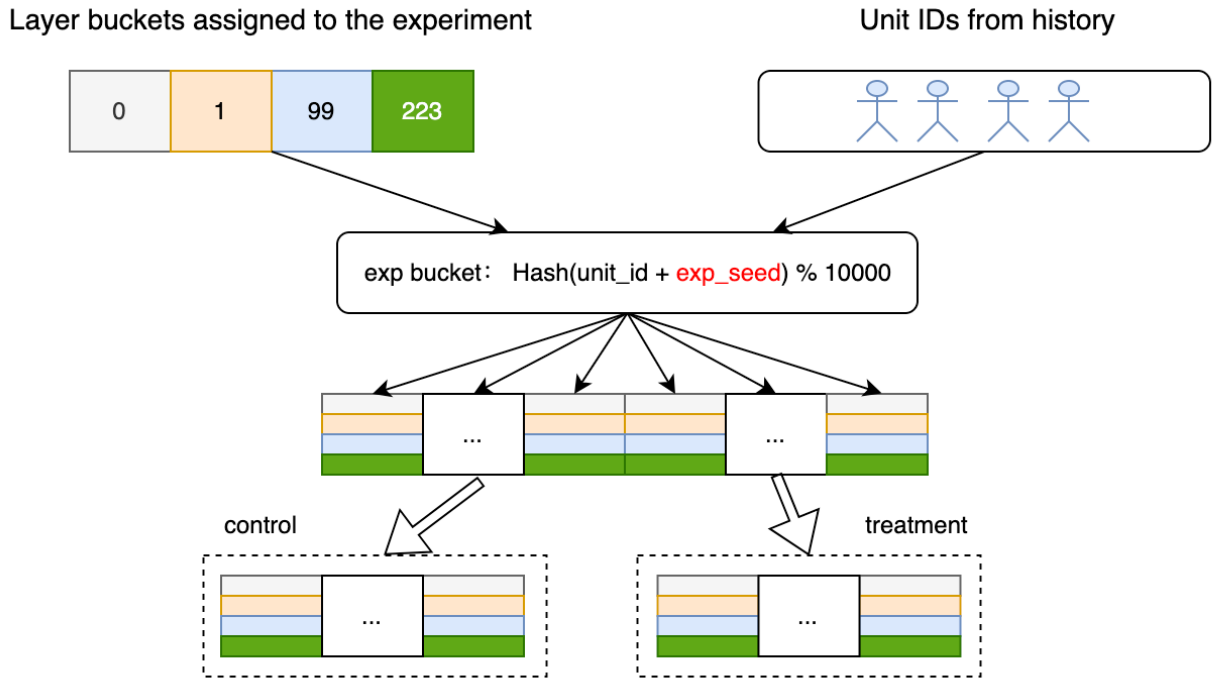


Two level randomization

carry over degree	type I error rate		
	two level randomization	one level randomization	one level with 12% frozen traffic
0%	0.0536	0.0499	0.0435
2%	0.0462	0.0661	0.0559
3%	0.0449	0.0811	0.0684
5%	0.0524	0.1266	0.1223
7%	0.0499	0.1852	0.1748
10%	0.0468	0.2489	0.2356

Type I error rate comparison using different randomization strategy, simulated result using Tencent data

Pre-experiment bias – seedfinder



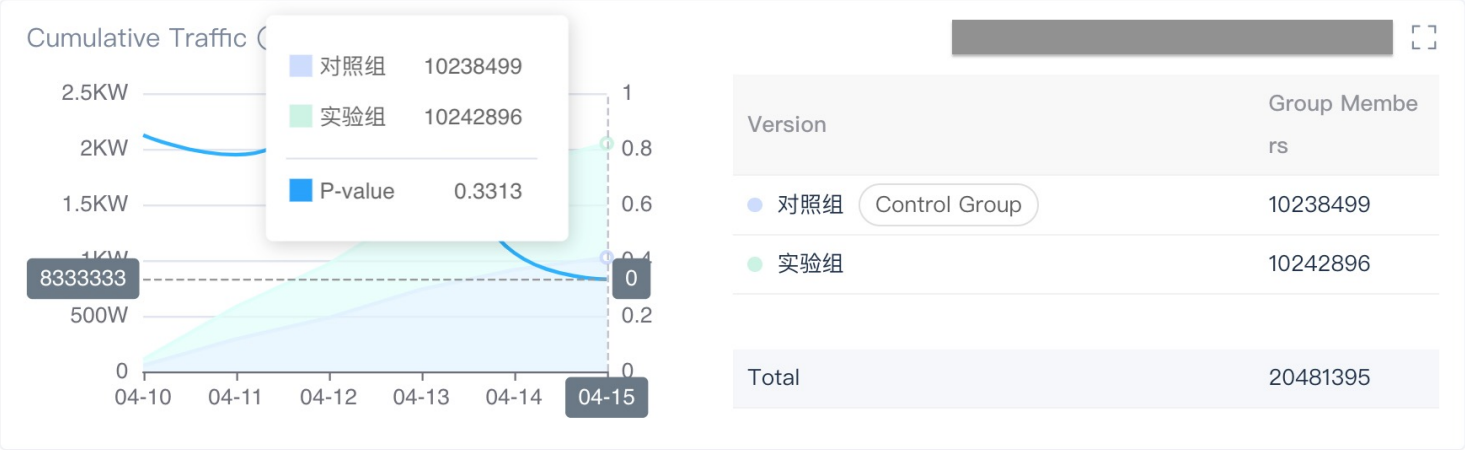
1. Compute the units that will participate in the experiment based on the buckets assigned to the experiment and historical exposure data.
2. Attempt to split these units into two groups using different experiment seeds.
3. Calculate the metrics of these units under the assignment and perform a comparison.
4. Repeat steps 2 and 3 multiple times until you find a seed that can evenly divide the units into two groups.

Pre-experiment bias

- CUPED aims to eliminate the noise introduced by pre-experiment bias by "subtracting" the pre-experiment data from the post-experiment data.
- Retrospective-AA analysis serves as a health check. After the experiment begins, compare the pre-experiment metrics for the users participating in the experiments to see if there is a statistically significant delta between the control and treatment groups. While useful, it is a lagging indicator compared to the seedfinder solution.

SRM

- Definition of sample ratio mismatch: the ratio of cumulative unique units exposed to the groups doesn’t statistically equal the ratio of the group allocations.
- Symptoms of underlying issues.
- Restart experiment after you figure out the root cause.



SRM

Suppose we have two experiments, both with a 50/50 split between control and treatment groups. Can we assert whether these two experiments have a SRM issue?

Experiment	Control	Treatment	Relative Diff	SRM?
A	1,000,000,000	1,001,000,000	0.1%	
B	10,000	10,200	2%	

SRM

Suppose we have two experiments, both with a 50/50 split between control and treatment groups. Can we assert whether these two experiments have a SRM issue?

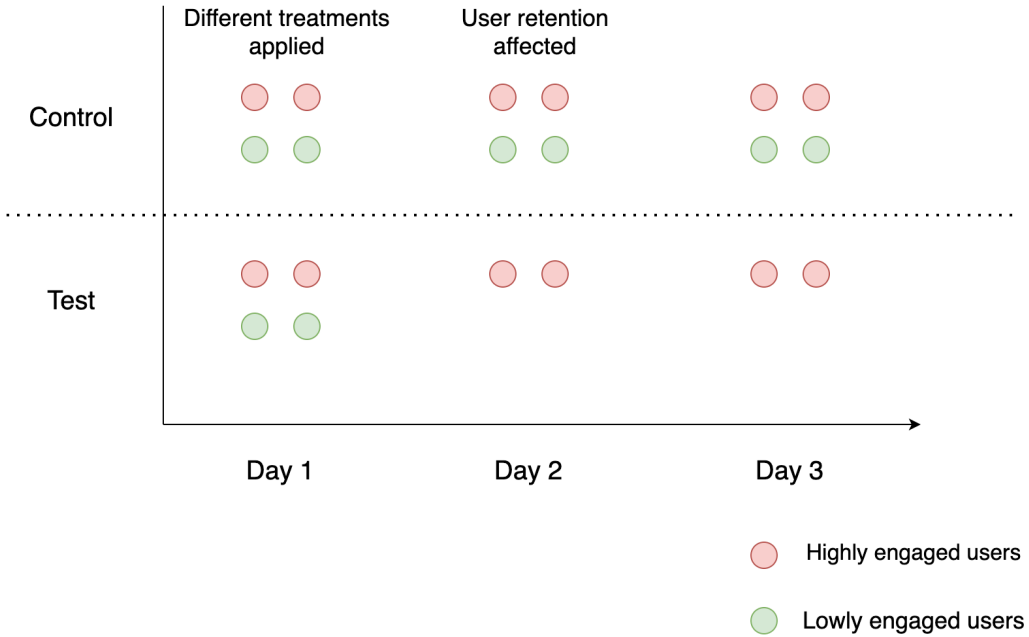
Experiment	Control	Treatment	Relative Diff	P-value	SRM?
A	1,000,000,000	1,001,000,000	0.1%	10^(-16)	yes
B	10,000	10,200	2%	0.1615	no

In practice, we choose a relatively small cutoff for p-value of SRM test, for example 0.001, to reduce the false positive rate for SRM detection.

SRM – Common causes

Root cause: The failure to collect "first exposures" when the treatment affected user retention. For example,

- The experiment was restarted without re-randomization.
- Exposure logging occurred after the treatments took effect.



What should we do to handle these cases relate to “first exposures”?

Takeaway: Always use cumulative first exposures from day 1 for metric calculation.

SRM – Common causes

Engineering issues:

- Different treatments resulted in varying server/client crash rates.
- Exposure was logged differently in different code branches, which is typically difficult to debug and requires specific tools.

```
experiment := abc.GetExperimentWithLoggingExposure("user_id", "experiment_name")
if experiment.GroupName == "Control" {
    buttonColor = "blue"
    experiment.LogExperimentExposure()
} else if experiment.GroupName == "Treatment" {
    buttonColor = "green"
}
```

Which operations are valid

1. You've conducted an experiment with a 20% allocation for 5 days, and now you wish to reduce the allocation to 10% in order to conserve traffic for another experiment.
2. You've conducted an experiment with a 50% control and 50% treatment allocation, and now you want to adjust the allocation to 80% and 20% respectively.
3. You've conducted an experiment with a 10% control and 10% treatment allocation, and now you want to increase only the treatment group to 30%, while maintaining the control group at 10%.

Which operations are valid

1. You've conducted an experiment with a 20% allocation for 5 days, and now you wish to reduce the allocation to 10% in order to conserve traffic for another experiment.

Reducing the allocation could cause some participants to stop receiving the treatment while they are still included in the experiment result calculation, which could lead to dilution.

2. You've conducted an experiment with a 50% control and 50% treatment allocation, and now you want to adjust the allocation to 80% and 20% respectively.

Changing the allocation ratio within the same experiment could cause users to switch from one group to the other, thus invalidating the results.

3. You've conducted an experiment with a 10% control and 10% treatment allocation, and now you want to increase only the treatment group to 30%, while maintaining the control group at 10%.

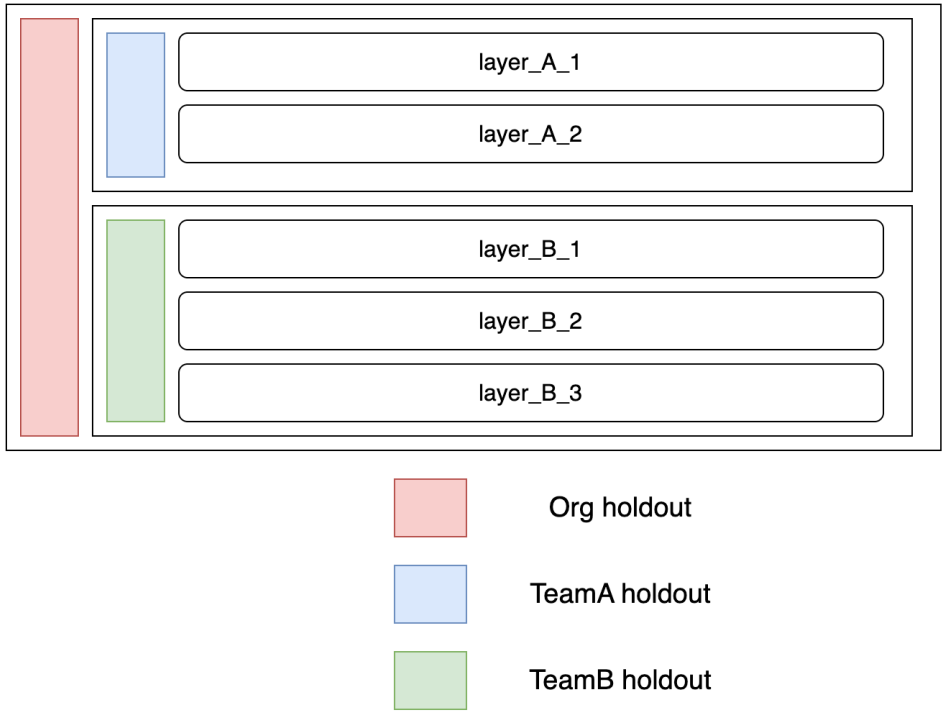
This could make comparisons difficult, as the treatment group now contains more active users, potentially leading to Simpson's paradox.

Long term effect: Backtest & holdout

- Definition:
  - A backtest is conducted after you finish an experiment (pretest). Instead of launching the winning feature to 100% of users, you retain a small percentage of users who will continue receiving the old features.
  - A holdout is a small group of users who will not participate in any experiment or feature launch.
- Difference: A backtest (in contrast with a pretest) is for single experiment, while a holdout is for aggregated results
- Similarity: Both are usually run for the long term and withhold a small segment of users.

- There are other techniques to estimate your long-term effect. The choice depends on your specific scenarios. For example, you might use user learning models or surrogate metrics.

Holdout



At the start of each half-year, holdout owners will create a holdout tree hierarchy for the entire organization. All experiments and feature launches will respect the corresponding holdouts. At the end of the half-year, the overall impact of each team or organization will be reflected in each holdout.

Holdout

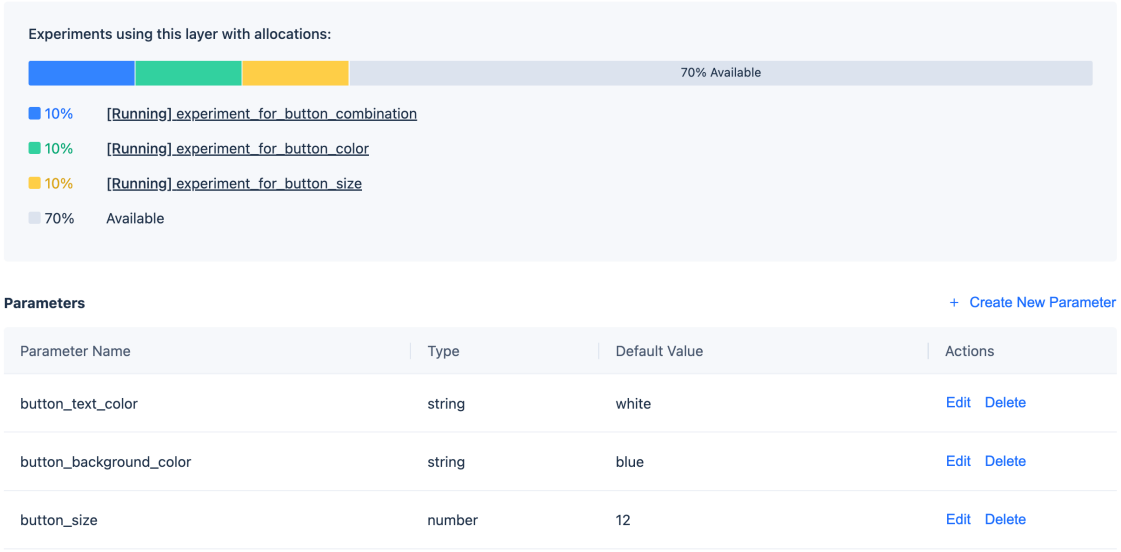
The trend curve of the holdout can reflect or monitor significant feature launches or external factors that may affect the core metrics.

Metric centric view

- Metric monitor provides a holistic view of the impact to a single metric from all experiments.
- Typically, only company critical metrics are monitored due to high computing costs.
- Challenges include:
  - The need to find a balance between recall and accuracy.
  - The multiple comparison problem, thus corrections are necessary.

Rapid experiment iteration

- Layer: a logical construct that represents the entirety of users against whom you will conduct experiments. Typically, each layer corresponds to a tangible module within your system.
- Parameters: features on each module that you need run experiment against, such as the color of a button or a ranking algorithm parameter. Features from same module are usually interdependent, thus they should belong to same layer.



https://abetterchoice.ai/docs?wj-docs=%2Fprod%2Fdocs%2Fparameters

Rapid experiment iteration

```
// Before:
experiment := abc.GetExperiment("user_id", "layer_name")
if experiment.ExperimentName == "experiment_for_button_size" && experiment.GroupName == "Control" {
  buttonSize = 12
} else if experiment.ExperimentName == "experiment_for_button_size" && experiment.GroupName == "Treatment_A" {
  buttonSize = 16
// After:
experiment := abc.GetExperiment("user_id", "layer_name")
buttonSize := experiment.GetInt64WithDefault("button_size", DEFAULT_SIZE)
```

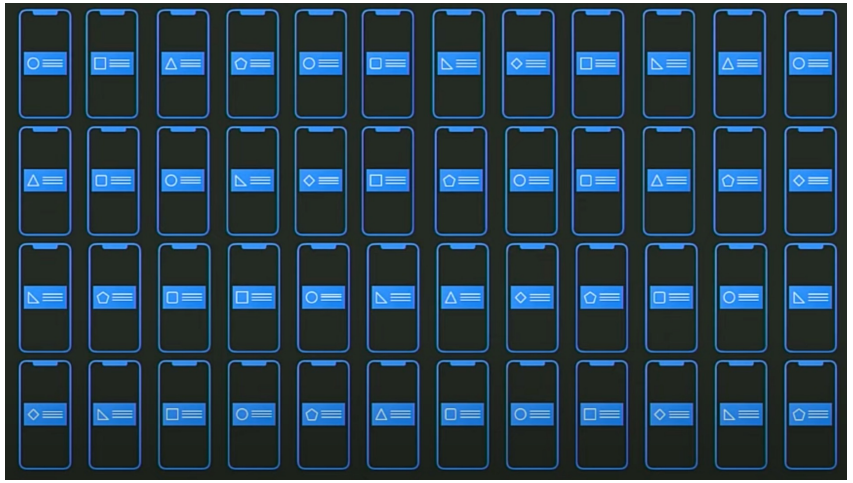
The new code is more concise and cleaner. Adding a new experiment simply involves creating a new experiment under the same layer via the UI and configuring the corresponding parameters. This allows for the execution of the new experiment without any code modifications, as the name of the layer and parameter won’t change.

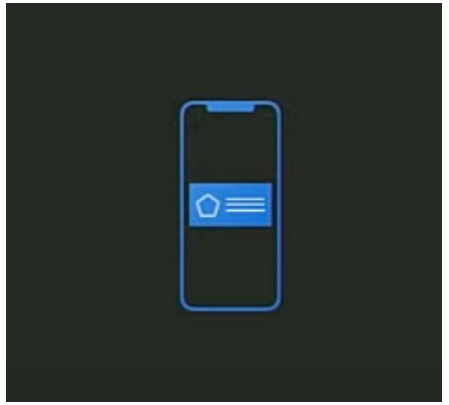
Parameter tuning

When the parameter combination space is large, for example, when there are several parameters and each parameter can choose more than 10 values, is there a more efficient way to explore?

Instead of using brute force search, we can employ more advanced techniques like Bayesian optimization to speed up this process. This method uses information from the last iteration to select the most useful next points.

Case study: Different design patterns can be used for user surveys to improve survey completion rates





F8 2019: Facebook: Product Optimization with Adaptive Experimentation
https://www.youtube.com/watch?v=2c8YX0E8Qhw

Parameter tuning

Case study: Compiler performance tuning using ax

```
#define EVALFLAGS()                                                    \
  /* F(type, name, defaultVal) */                                      \
  /*                                                                   \
   * Maximum number of elements on the VM execution stack.            \
   */                                                                  \
  F(uint64_t, VMStackElms, kEvalVMStackElmsDefault)                    \
  /*                                                                   \
   * Initial space reserved for the global variable environment (in    \
   * number of global variables).                                     \
   */                                                                  \
  F(uint32_t, VMInitialGlobalTableSize,                                \
    kEvalVMInitialGlobalTableSizeDefault)                              \
  F(bool, Jit,                          evalJitDefault())            \
  F(bool, JitEvaledCode,                true)                        \
  F(bool, JitRequireWriteLease,         false)                       \
  F(uint64_t, JitRelocationSize,        kJitRelocationSizeDefault)   \
  F(uint64_t, JitMatureSize,            125 << 20)                   \
  F(bool, JitMatureAfterWarmup,         true)                        \
  F(double, JitMaturityExponent,        1.)                          \
  F(double, LogLoadedUnitsRate,         0.0)                         \
  F(string, LogLoadedUnitsBaseDir,      "{PHP_ROOT}")                \
  F(bool, JitTimer,                     kJitTimerDefault)            \
  F(int, JitConcurrently,               1)                           \
  F(int, JitThreads,                    4)                           \
  F(int, JitWorkerThreads,              std::max(1, Process::GetCPUCount() / 2)) \
  F(int, JitWorkerThreadsForSerdes,     0)                           \
  F(int, JitWorkerArenas,               std::max(1, Process::GetCPUCount() / 4)) \
  F(bool, JitParallelDeserialize,       true)                        \
  F(int, JitLdimmqSpan,                 8)                           \
  F(int, JitPrintOptimizedIR,           0)                           \
  F(bool, RecordSubprocessTimes,        false)                       \
  F(bool, AllowHhas,                    false)                       \
  F(bool, DisassemblerSourceMapping,    true)                        \
  F(bool, GenerateDocComments,          true)                        \
  F(bool, DisassemblerDocComments,      true)                        \
  F(bool, DisassemblerPropDocComments, true)                         \
  F(bool, LoadFilepathFromUnitCache,    false)                       \
  F(bool, WarnOnCoerceBuiltinParams,    false)                       \
```

```
ax.create_experiment(
    name="hhvm_jit",
    parameters=[
        {
            "name": "HHIRMixedArrayProfileThreshold",
            "type": "range",
            "bounds": [0.5, 1.0],
        },
         ...
    ],
    objective_name="cpu",
    minimize=True,
    outcome_constraints=["peak_memory <= 0.5%"],
)
```

F8 2019: Facebook: Product Optimization with Adaptive Experimentation
https://www.youtube.com/watch?v=2c8YX0E8Qhw

Variance reduction

In practice, changes in metrics between two groups may be influenced by pre-experiment factors unrelated to the treatment itself. Therefore, pre-experiment data is often utilized to reduce noise and obtain more precise results. In statistical terms, this helps achieve a smaller sample variance.

$$N_{min} = \frac{2 \times \left(Z_{1-\frac{\alpha}{2}} + Z_{1-\beta}\right)^2}{\Delta^2} \times \sigma^2$$

α	Type I error rate (significance level)
β	Type II error rate (1- power)
σ^2	Sample variance
Δ	Difference between two groups
N_{min}	Minimum sample size required

Variance reduction

team	experiment	metric	days required to collect enough samples		days saved	days saved ratio
			before	with CUPED		
A	exp1	m1	2	1	1	50%
		m2	5	2	3	60%
		m3	8+	1	7+	87.5%+
	exp2	m1	8+	2	6+	75%+
		m2	8+	3	5+	62.5+
		m3	8+	2	6+	75%+
B	exp3	m1	3	1	2	66.7%
		m2	3	1	2	66.7%
		m3	3	1	2	66.7%
	exp4	m1	8+	3	5+	62.5%+
		m2	8+	3	5+	62.5%+
		m3	8+	3	5+	62.5%+
C	exp5	m1	4	2	2	50%
		m2	2	1	1	50%
		m3	13+	9	4+	30.8%+
	exp6	m1	5	2	3	60%
		m2	5	2	3	60%
		m3	14+	11	3+	21.4%+

It's important to note that in practice, factors such as the novelty effect should be considered when deciding the duration of an experiment.

The number of days required to collect sufficient samples to detect a change of 1% or larger.

Machine cost

Running experiments is not free. They can not only lead to inconsistent user experiences, but also incur significant machine costs.

Best practices:

1. Do not add too many metrics to one experiment (note this will also introduce multi comparison problem and increase p-packing risk!)
   - Typically, there should be no more than three goal metrics per experiment, and they should be closely related to your hypothesis.
   - Large companies might include many tracking metrics, but your decision should primarily hinge on your goal metrics.
2. Once a conclusive result is obtained, stop the experiment and clean up the code.
3. Continually improve the computational efficiency of your experimentation platform.

Summary – use experiments wisely

Experiment iteration 1 → Experiment iteration 2 → Experiment iteration 3 → Ready to GO!

Setup

- Formulate a clear and well-explained hypothesis.
- Utilize parameters to expedite the iteration of experiments.

Dogfooding

- Ensure that the correct features are visible on the product for each group.
- Make certain that each group receives at least one exposure.

Running

- Keep an eye on the movements of the guardrail metrics.
- Pay attention to health checks such as SRM.
- Ensure that you collect enough samples.
- Avoid taking invalid operations.

Review

- Interpret the data wisely, avoiding pitfalls such as p-hacking.
- Pay attention to novelty effect or primacy effects.
- Examine the confidence and p-values, not just the ATE.
- Drill down the data under different dimensions.

Decision

- Keep a record of your decision for future reference.
- Once a final conclusion has been reached, clean up the code.

Demo

Q&A