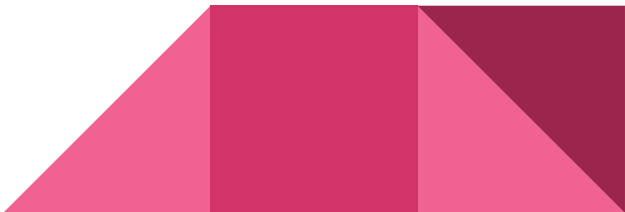




Managing the ML Lifecycle

Giulio Zhou

Background and Context

- In the past few decades, we've seen an explosion of ML applications - generating untold quantities of data - with real-time demands for scalable serving and learning
 - Most of the attention/hype goes to the ML algorithms themselves (decision trees, logistic regression, deep neural networks, etc) rather than ML systems design and implementation
 - In 2012, Google's ad networks served 29,741,270,774 ads per day
 - Google Technical Debt paper lists some problems of interest, does not discuss potential solutions in great detail
- 

What defines a “good” ML system?

- Good question
- A good ML system *should*:
 - Serve predictions quickly and correctly across all levels of load
 - Utilize the right combination of contextual and historical data
 - Rapidly incorporate feedback into models
 - Be flexible and easy to extend to new machine learning models
- Definition may vary for companies and machine learning algorithms at different scales (i.e. MapReduce is great for Google, but only if you're Google, Facebook, Amazon, etc)

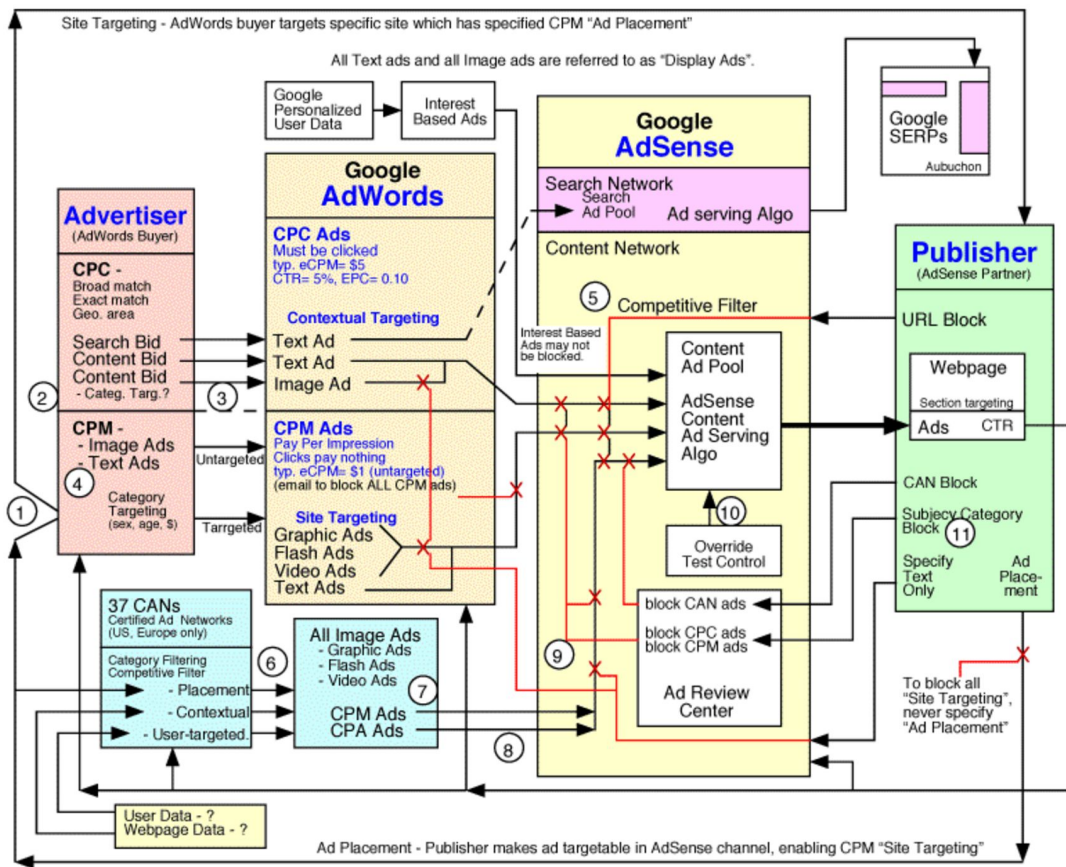


Ideal ML Pipeline




Expectations

Reality



Problems and Motivation

- By the nature of machine learning models, “the desired behavior cannot be effectively implemented in software logic without dependency on external data” (Google, et al)
 - Traditional SW design principles of abstraction and isolation are not always enforceable with data
 - Specifically:
 - Data comes in a multitude of forms
 - At large scales, the generation and consumption of signals are decoupled
 - Feedback loops are necessary but can have unintended consequences
 - Need to balance rate of innovation with long-term maintainability
 - “Technical Debt”
- 

Technical Debt: When to refactor?



Sparsity is Challenging

- Bag of Words text models can have dimensionality on the order of billions with only a few hundred non-zero values
- In many applications, features are often missing and/or noisy
- Training data is typically unbalanced; in the case of ad serving, there are usually far more no-click examples than clicked examples
- Cannot apply familiar techniques such as dropout - a computer vision trick to occasionally turn off activations to learn robust models - in training
- Important to take these lessons into account to:
 - Optimize the performance of ML systems
 - Ensure that good models can be learned in a real-time setting



The Brittleness of ML Systems



Entanglement (CACE)

Hidden/Unintended
Feedback loops


Data Dependencies

Glue code &
Pipeline jungles

Dead code paths,
Configuration Bloat



The Brittleness of ML Systems (cont.)

- *Changing Anything Changes Everything*
 - ML systems combine signals and features together, cannot be considered truly independent
 - Changes in feature usage or logging can dramatically affect various slices of the distribution
 - Enforcing modularity may help, but adds a different kind of complexity
 - Hidden and Unintended Feedback Loops
 - Undeclared consumers of signals
 - Feedback loops operate on longer timescale, unnoticeable in initial reviews
 - Data dependencies
 - Glue code and pipeline jungles
 - Dead code paths and configuration bloat
- 

Ex: Data Dependencies

- Underutilized data dependencies: Legacy features, bundled features, epsilon-features
 - Features could be removed with little to no reduction in accuracy, but the model assigns them non-negative weight since they're present
- Model unintentionally learns correlations between features, what happens if those features suddenly diverge (perhaps because of some “improvement”)?
- Infeasible to support static analysis of data dependencies at large scales



Ex: Unspecified Consumers (Font size feedback)

- An “intelligent” font size module takes in CTR (click-through rate) as an input signal
- Larger font size make ads more noticeable but also susceptible to unintentional clicking, increasing CTR but creating an undesired feedback loop

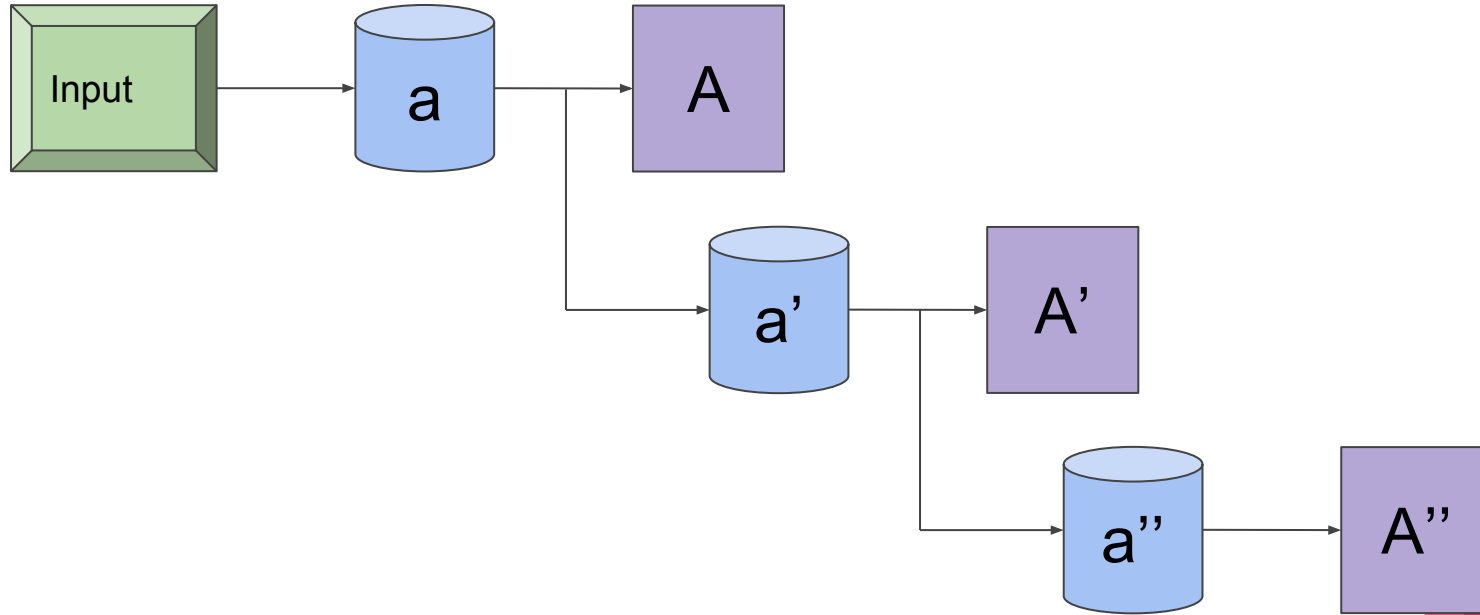
Font size

Font size

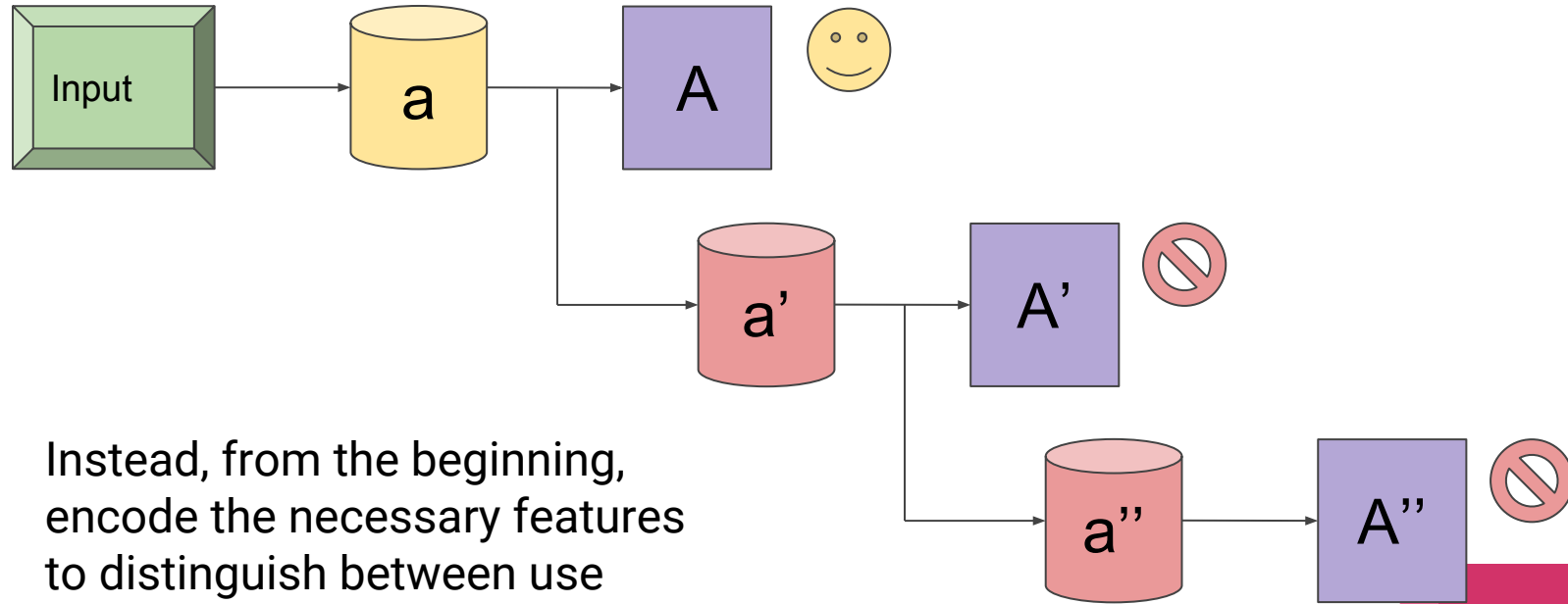
Font size



Ex: Correction Policy Cascade



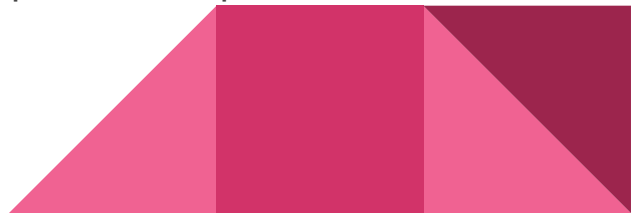
Ex: Correction Policy Cascade (cont.)



Instead, from the beginning,
encode the necessary features
to distinguish between use
cases A, A' and A''

Ex: Glue Code/Pipeline Jungles

- To build upon existing ML frameworks and ensure interoperability between systems all written in different languages
 - Most of the code goes into formatting data and outputs into intermediate forms
 - Freezes the system to the peculiarities of specific ML packages, stifling innovation
- As mentioned previously, data comes in a multitude of forms and must be wrangled into a usable form
 - “Pipeline jungles” can emerge from too much black-boxing, and the resulting decoupling of production/consumption of features
 - The result is an endless array of scrapes, joins, resampling steps, is failure-prone and must be maintained with expensive end-to-end integration tests



Ex: Dead codepaths, Knight Capital



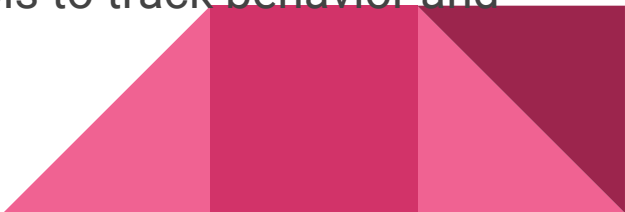
- Technician forgets to copy the new Retail Liquidity Program (RLP) code to one of servers, execution with repurposed flag leads to cascade of unusual stock moves at NYSE
- In just 45 minutes, Knight Capital lost \$465 million

Brief counter-argument

- “All he would say was that sometimes you have to burn it down and start over.” - Kelly Braffet, Last Seen Leaving
- Many of the previously mentioned points apply to large, mature systems (typical of those found at Google)
- Only doable if (1) you can hire entire teams of teams dedicated to this task or (2) if the system is sufficiently small
- Static analysis of data dependencies is potentially feasible in smaller settings
- What if you can't afford to reimplement a ML system from scratch?

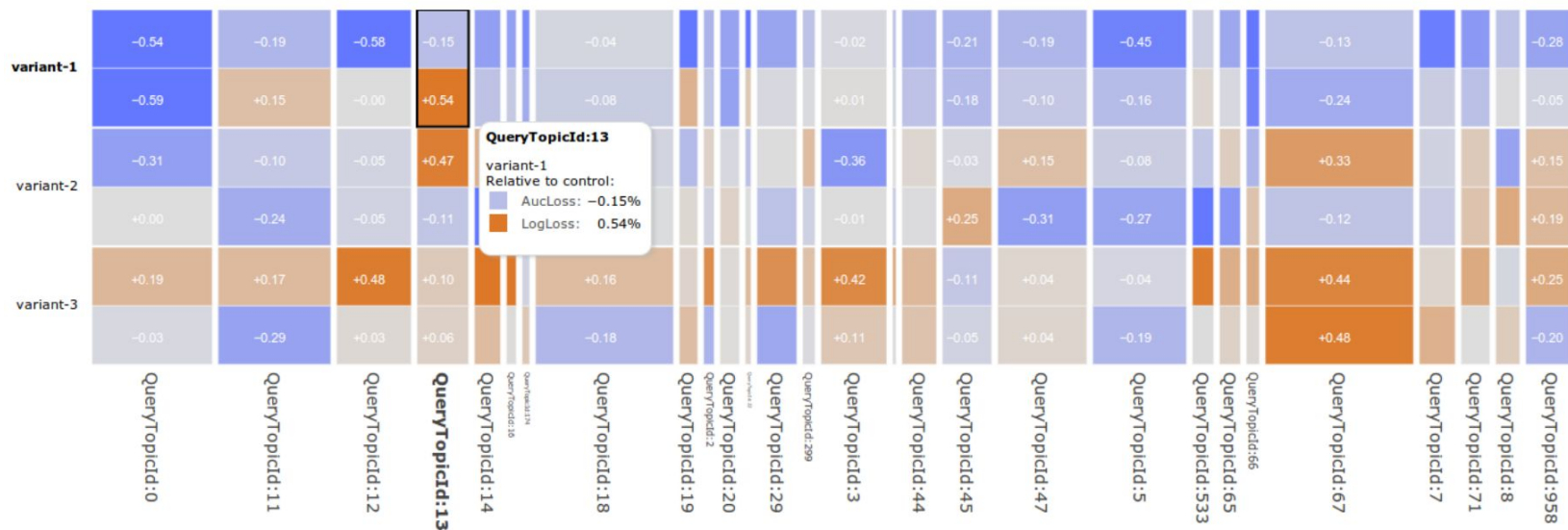


(Somewhat) Successful Solutions

- Online learning algorithms (streaming per-coordinate descent, learning rates) to more quickly adapt to feedback
 - Facebook and Google ad serving systems discuss this
 - Google metadata index
 - Internal tool for tracking and maintaining signals, earmark old signals for deprecation or vet new ones for introduction
 - Eliminate unimportant and redundant features using L1 regularization, boosting to get sparse solutions
 - Develop model monitoring and data visualization tools to track behavior and uncover data dependencies
- 

Ex: Understanding through Visualization

- Visualize beyond aggregate metrics via per-country or per-topic slicing



Lessons/How to better innovate

- Design principles: what are some takeaways when designing scalable machine learning systems?
 - Important to move fast, but keep in mind the tradeoff between quick wins and long-term maintainability of the system
 - It's impossible to build a perfect system from the beginning; you should try anyway, and then proceed with these lessons in mind
- Lessons for Clipper and RISE system design
 - Lightweight online layer for correction and personalization
 - Monitoring and visualization modules for tracking model performance
 - (Joey) Any other ways we're connecting this with our new RISE projects?



Questions?

