COWBIRD: Coordinated Optimization without Big Resource Demands

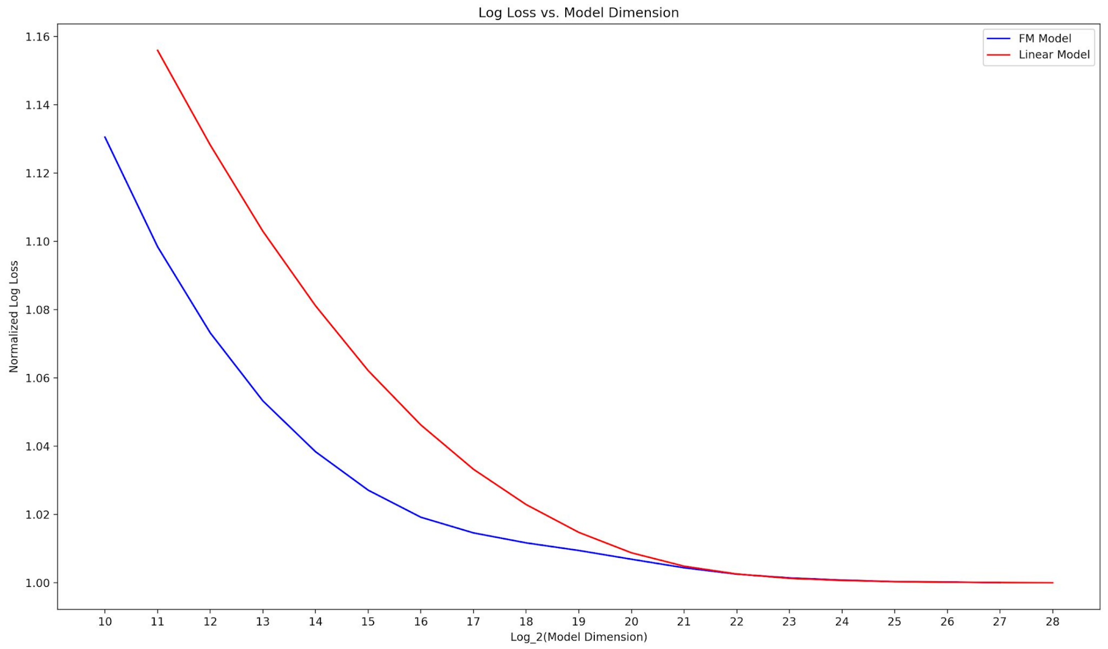
Industry Context

The ad-tech ecosystem pays publishers for the content they create while enabling companies to advertise their products. A key component of this value exchange is the ability of buyers to accurately value ad opportunities. Without this ability, there is an asymmetry of information about the value of ad opportunities that could have severe adverse effects on the efficacy of this market.

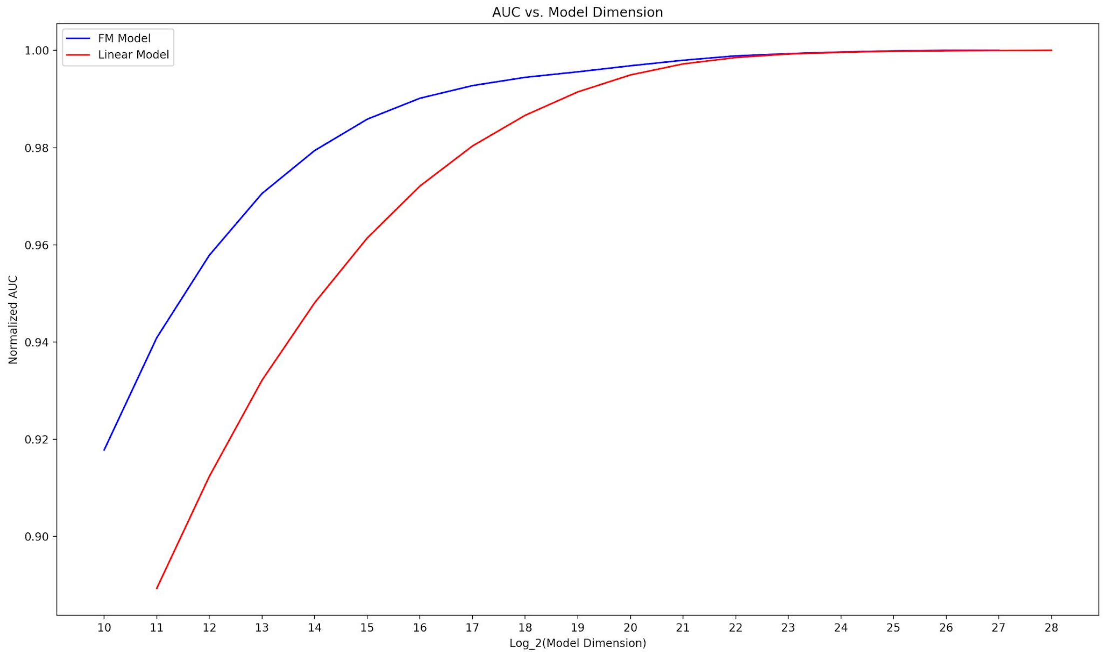
Motivating Observation

In MURRE, we saw that the sparse binary logistic regression models like those used at NextRoll can be reduced in size by several orders of magnitude without sacrificing all that much in terms of performance. We even see tiny bidding models with as few as 2^{15} weights performing reasonably well.

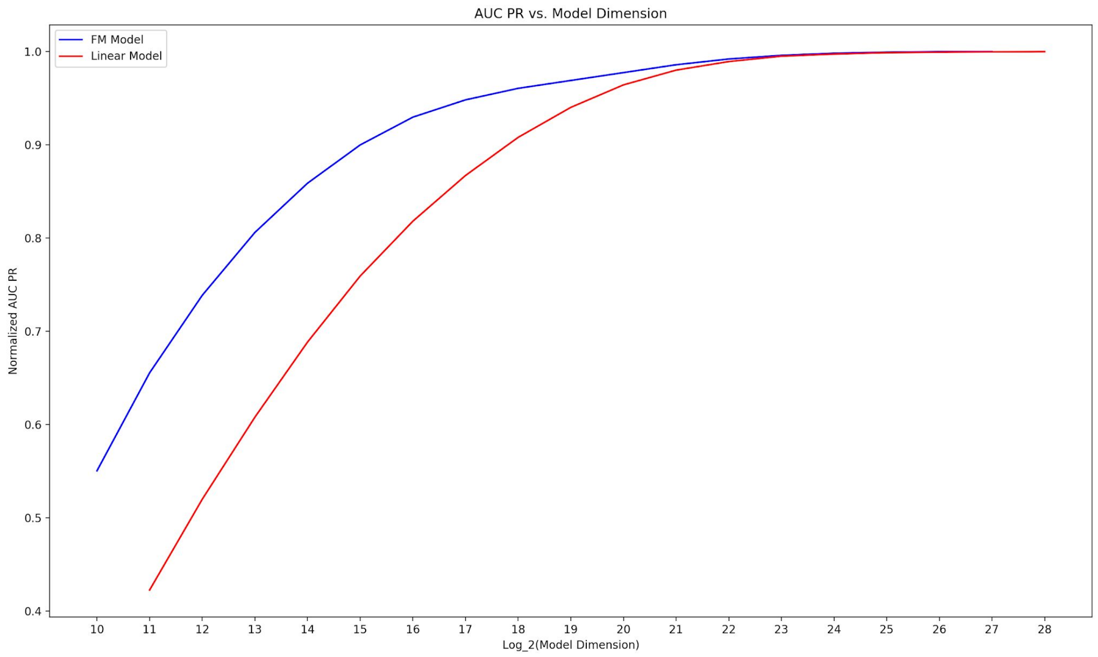
Motivating Observation



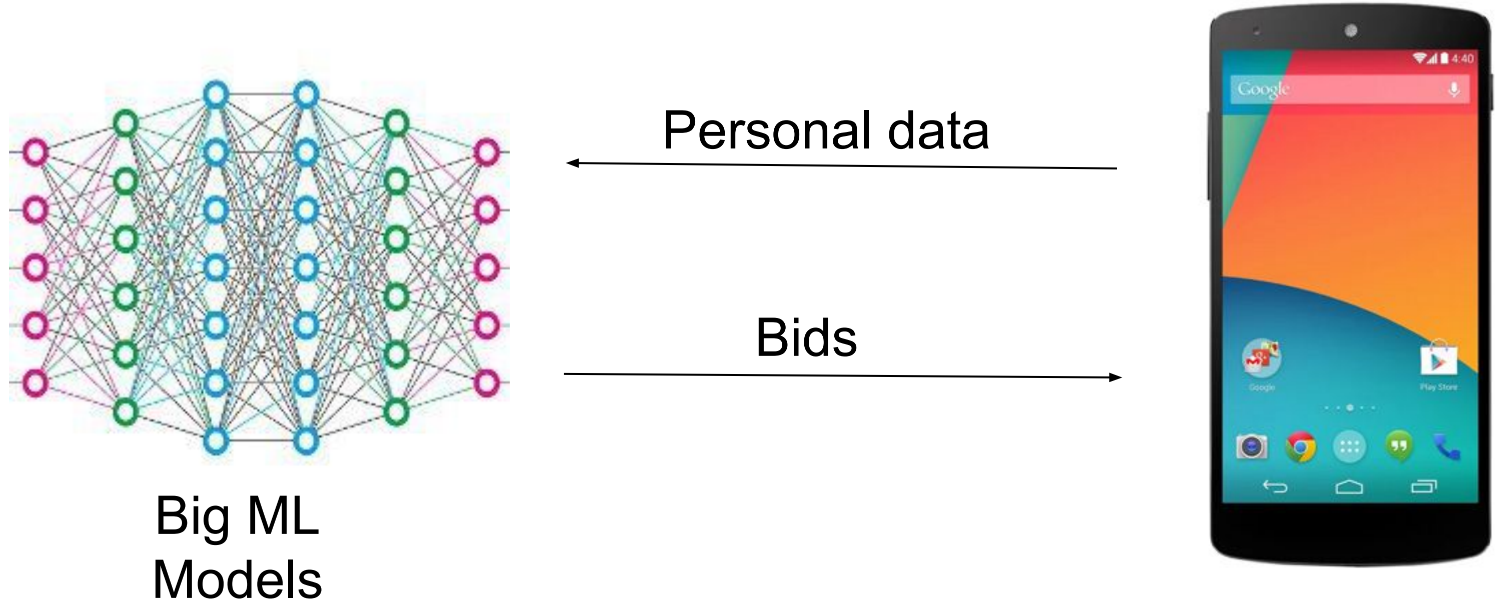
Motivating Observation



Motivating Observation



Third-Party Cookie Paradigm



Federated Learning Paradigm

Instead of running large bidding models (~1GB) in the cloud, we can imagine running tiny bidding models (~100 KB) on-device. In this paradigm, personal data would be kept entirely on-device.

The Proposal

Summary

In this proposal, the browser acts as a federated learning platform, allowing ad-tech companies to optimize toward customizable objectives with customizable models and features.

Step 1: On an Advertiser’s Site

The ad-tech company associated with advertiser’s site asks the browser to pull the following model-related information:

- Versioned model weights
- A function to compute model features given:
  - Contextual information related to the ad opportunity
  - Browser event history, as described in SPURFOWL
  - Interest group data
- A function to evaluate the model given features

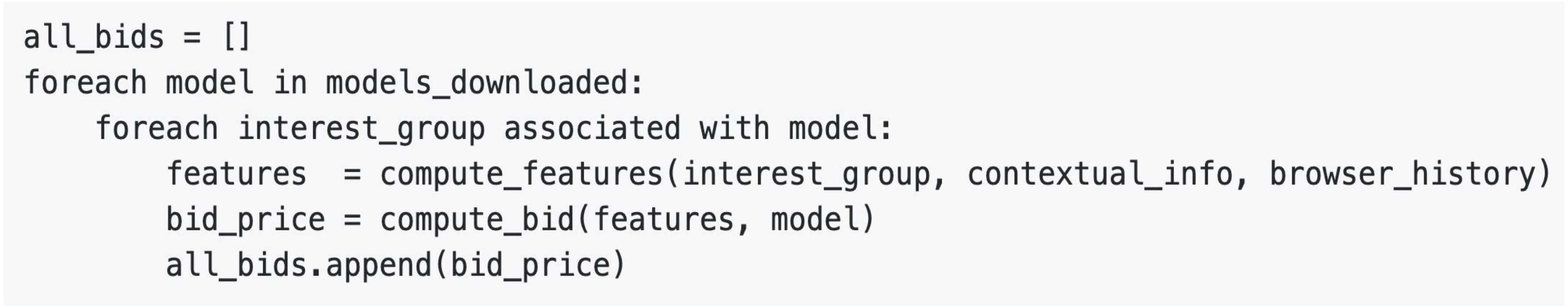
Step 1: On an Advertiser’s Site

The advertiser’s site also asks the browser to pull model the following gradient-related information:

- A function to determine observation labels given the browser event history (as described in SPURFOWL)
- A function to determine the model gradient given:
    - The label of the observation
    - The model features associated with the observation
    - The model data

Step 2: On an Publisher’s Site

On a publisher’s site, the browser itself is now able to produce bids without sending personal data off-device. These bids can be evaluated by publisher-supplied logic, or by the browser itself.



```
all_bids = []
foreach model in models_downloaded:
    foreach interest_group associated with model:
        features  = compute_features(interest_group, contextual_info, browser_history)
        bid_price = compute_bid(features, model)
        all_bids.append(bid_price)
```

Step 3: Label and Gradient Generation

At some later time, the browser can label observations on behalf of the ad-tech companies associated with the browser, and send its local gradients to the gradient aggregation service. Something like this:

```
batch_gradient = (0, ..., 0)
foreach observation in browser_history:
    if observation.dsp = current_dsp and observation is not labeled:
        label = compute_label(observation, browser_history)
        if label is not SUBSAMPLE_EVENT or RECOMPUTE_LATER:
            features = observation.saved_model_feature_state
            gradient = compute_gradient(label, features, model)
            batch_gradient += gradient
```

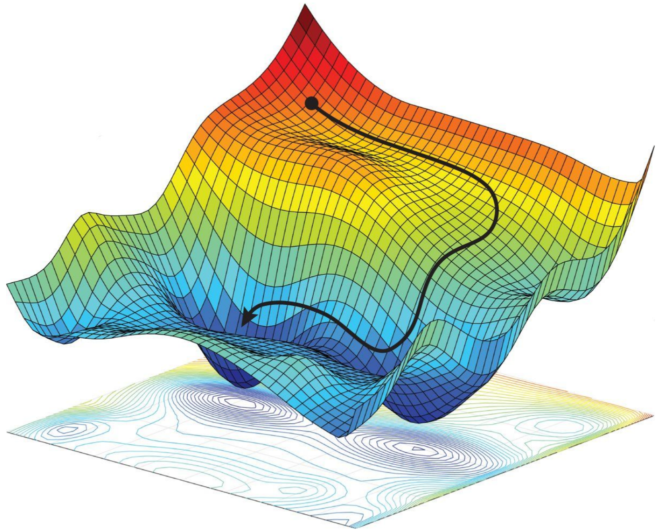
Step 4: Gradient Aggregation

The gradient aggregation service sums the gradients collected for a given ad-tech company and model version from many browsers. It then adds noise as necessary to protect privacy.

Note: Submissions to the gradient aggregation service should be keyed on model version for them to be useful because gradients are a function of the versioned model weights. Model versioning also allows A/B testing.

Step 5: Model Training and Updates

The ad-tech company receives its aggregated gradient, and uses it to improve its model. This means taking a step in the direction of the gradient, and perhaps applying regularization. The ad-tech company then releases a new version of its model, which browsers eventually pull down.



Quick Analysis

Resource Considerations

Let’s say we had a maximum of 500 models per browser:

- At 100 KB per model that means we would need at most 50 MB of memory per device.
- If models were refreshed at the start of each browsing session we would require at most 50MB of model data downloaded per session.
- If gradients are submitted for aggregation at the end of each browsing session we would require at most 50MB of gradient data uploaded per session. This would be much lower if gradients are mostly sparse.

Privacy Considerations: Attack Surface

A malicious client of this API interesting in leaking personal data has the following avenues of attack:

1. Attacks via programmed model features
2. Attacks via programmed gradient values

Privacy Considerations: Mitigations

We believe the gradient aggregation service can mitigate these attacks these attacks as follows:

1. Gradient aggregation densifies gradients, mitigating attacks via programmed gradients.
2. Gradients can be noised/distorted in a host of ways while preserving the utility of the gradient. This is because gradients just need to *roughly* point in the direction that most improves the model. This can mitigate attacks via programmed gradient values.

Known Shortcomings

Bidding Logic in the Clear

Our understanding is that each consumer of the federated learning platform proposed here would have their models and related logic available for anyone with a browser to inspect. *This is a serious defect*. We hope some way of making this code/data more private could be engineered if the rest of this proposal is well-received.

Contextual Targeting

This proposal does not support contextual targeting out-of-the-box. This is because it doesn’t include a simple way of identifying and filtering based on ad context. For this reason, the contextual requests of TURTLEDOVE-style proposals would still be useful, though they would arguably be less important under this proposal.

Tiny Models and a Lack of Centralized Data

The federated learning paradigm is not without its drawbacks.There would be a performance penalty resulting from the use of tiny bidding models. Additionally, simple things like validating model changes are much harder without a centralized dataset.