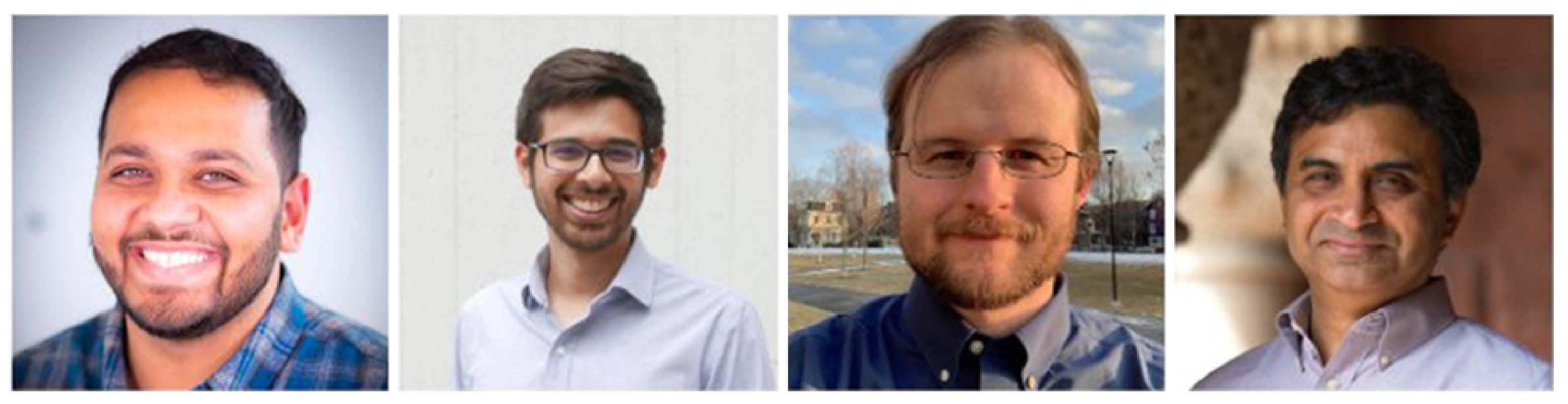
Leaving the Nest
Going Beyond Local Loss Functions for Predict-Then-Optimize

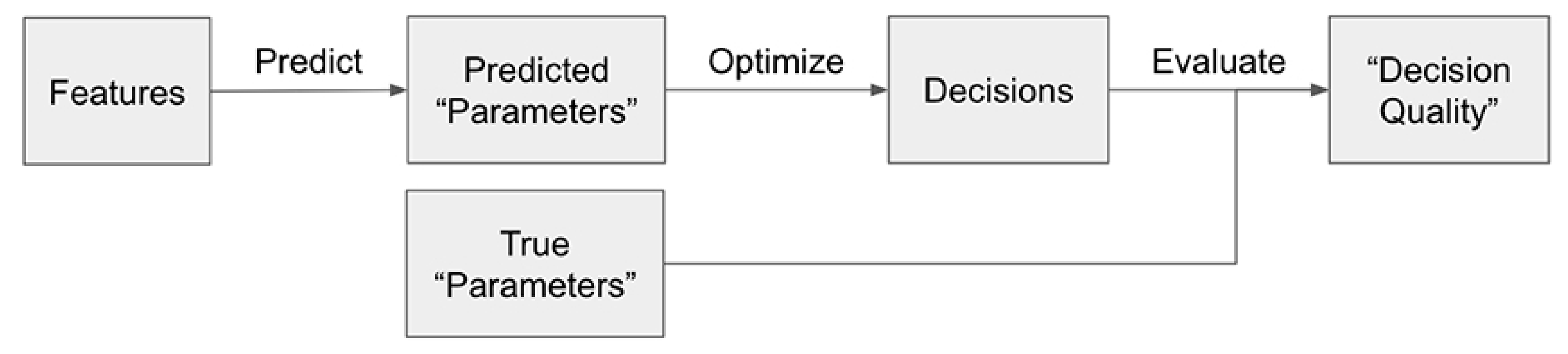
Sanket Shah[1], Bryan Wilder[2], Andrew Perrault[3], Milind Tambe[1]



TL;DR: We propose two improvements to learning loss functions for Predict-then-Optimize that make them more computationally efficient to learn and deploy.

Setting

There are three steps in Predict-then-Optimize (PtO). In the **Predict** step, a predictive model is used to make predictions about "parameters" based on some features. Next, in the **Optimize** step, these predictions are used to parameterize an optimization problem, whose solution yields a "decision". Finally, in the **Evaluate** step, the produced decision is evaluated based on how well it performs on the "true parameters".



Example: Web Advertising

	a	b	c	d
1	0.1	0.2	0.9	0.8
2	0.4	0.4	0.4	0.4
3	0.9	0.8	0.2	0.1

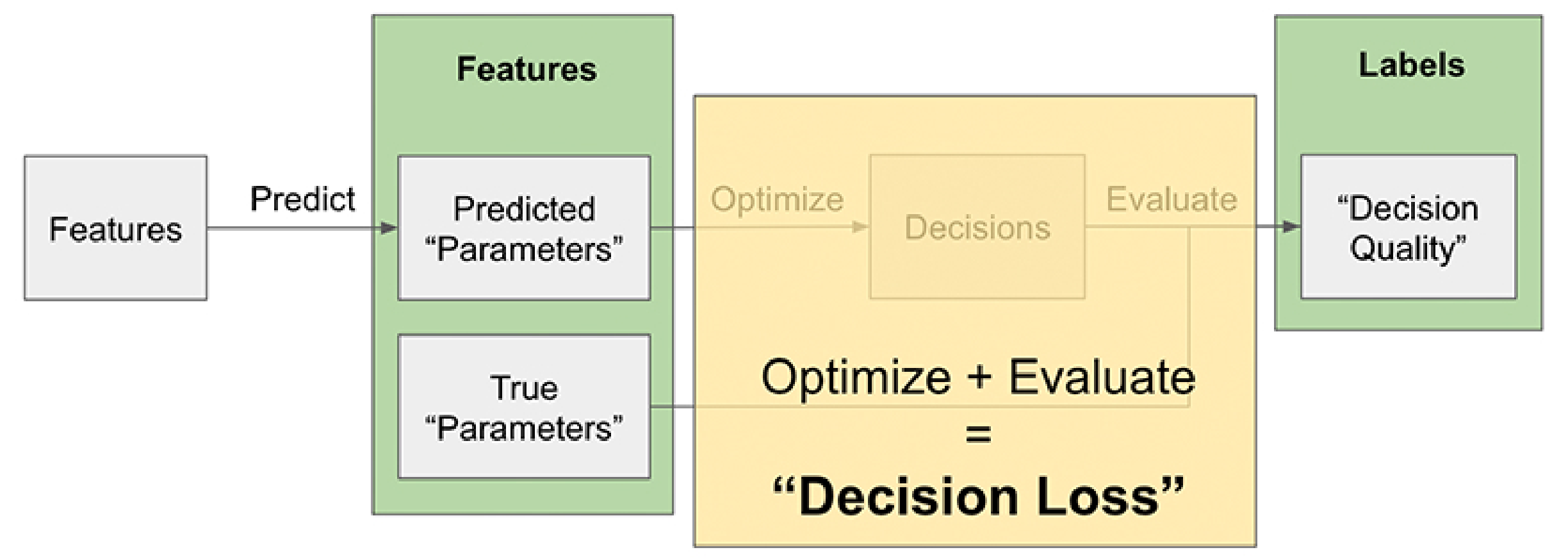
Predict: Click-through rates for different (website, demographic) combinations from website metadata.

Optimize: Choose the subset of websites that maximize the number of people that click on the ad at least once.

Evaluate: How many people would have actually clicked on the ad if we advertised on websites 1 and 3?

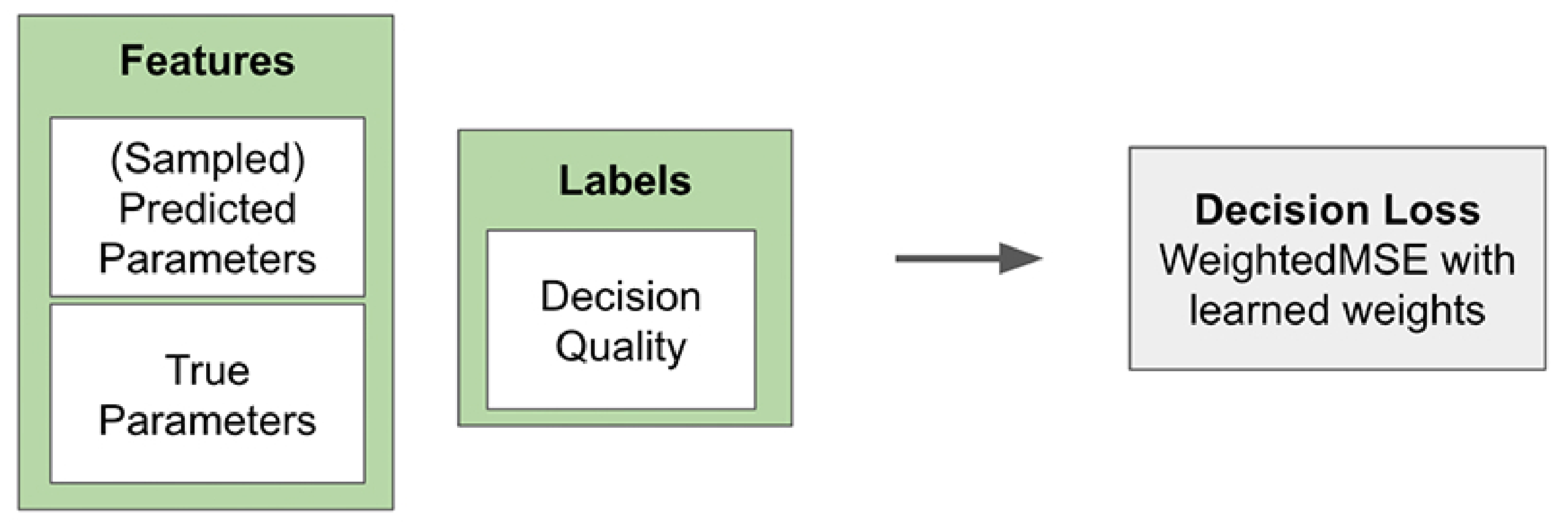
Decision Loss

The Predict-Then-Optimize pipeline can be interpreted as being a loss in itself. While the actual form of the **Decision Loss (DL)** is complex, we use **supervised learning** to approximate this mapping.



1 2 3

Learning Decision Losses

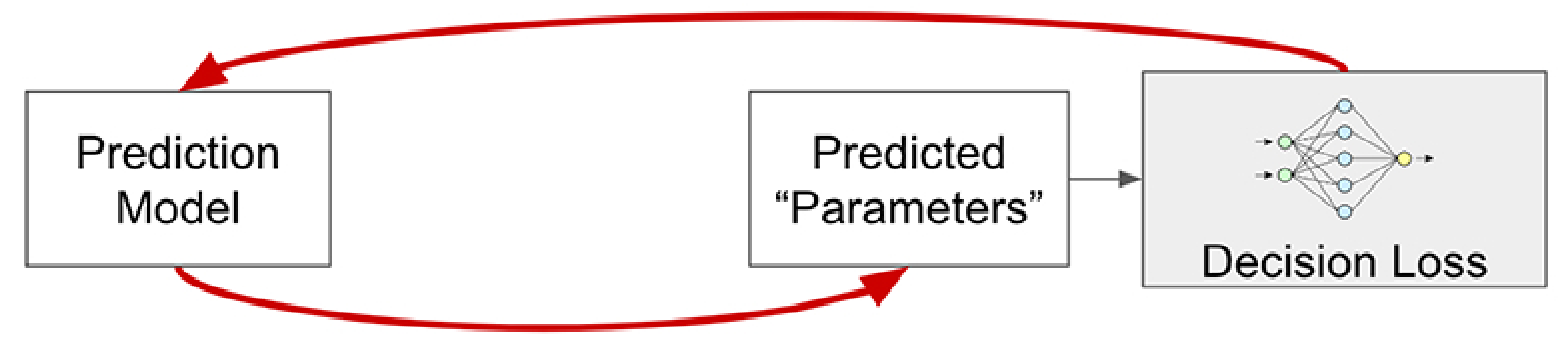

Features
(Sampled) Predicted Parameters
True Parameters
Labels
Decision Quality
Decision Loss
WeightedMSE with learned weights

Step 1: Generate samples of possible predictions
Step 2: Calculate the "Decision Quality" for sampled predictions
Step 3: Fit a "decision loss" for each decision-making instance, i.e., for each set of "true parameters"

Challenge: Computational Cost

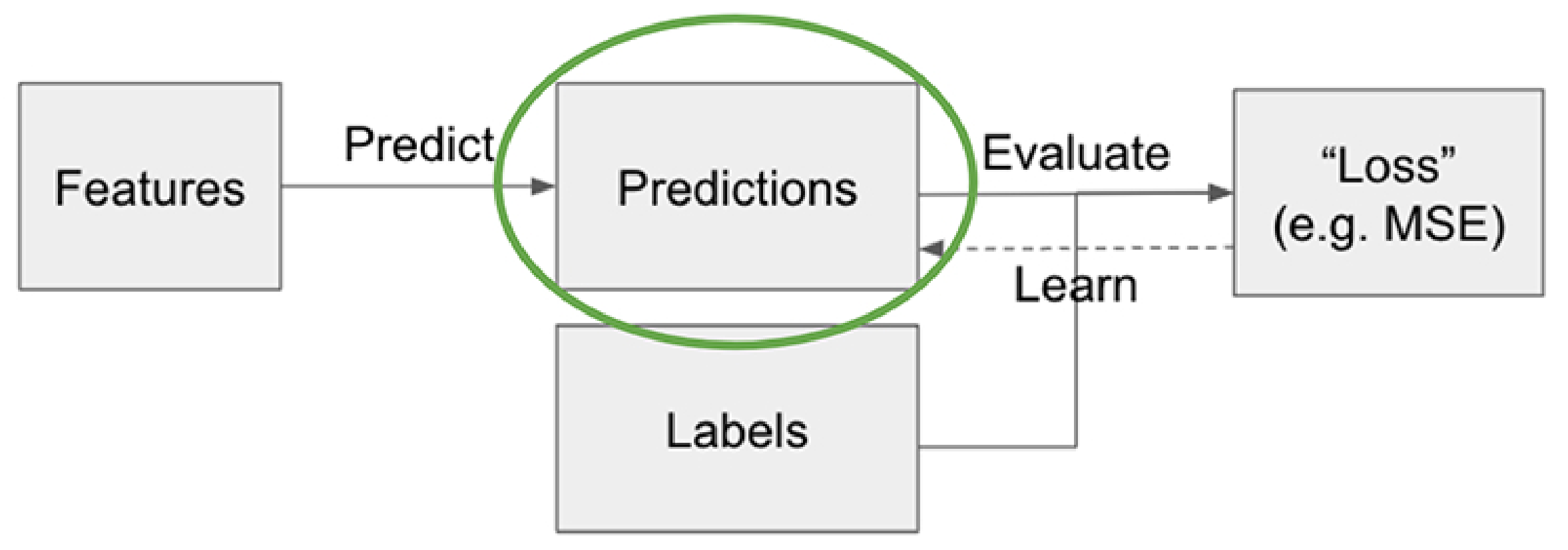
Learning these decision losses can be expensive and most of the time is typically taken by Step 2, in which we need to make calls to some optimization solver to compute the "decision quality" for our sampled predictions. However, reducing the cost of solving arbitrary optimization problems is difficult. Instead, we modify steps 1 and 3 to be more sample efficient, so we need to make fewer calls to the optimization solver for similar performance.

Contribution 1: Model-Based Sampling



There is a circular dependency involved in sampling predictions in Step 1 of learning the decision loss. The desired predictions depend on the predictive model, but the predictive model in turn depends on the loss function which uses these predictions as input. However, despite this, it is important to generate realistic predictions; the underlying decision loss is complex and we're unlikely to be able to accurately model it for arbitrary predictions.

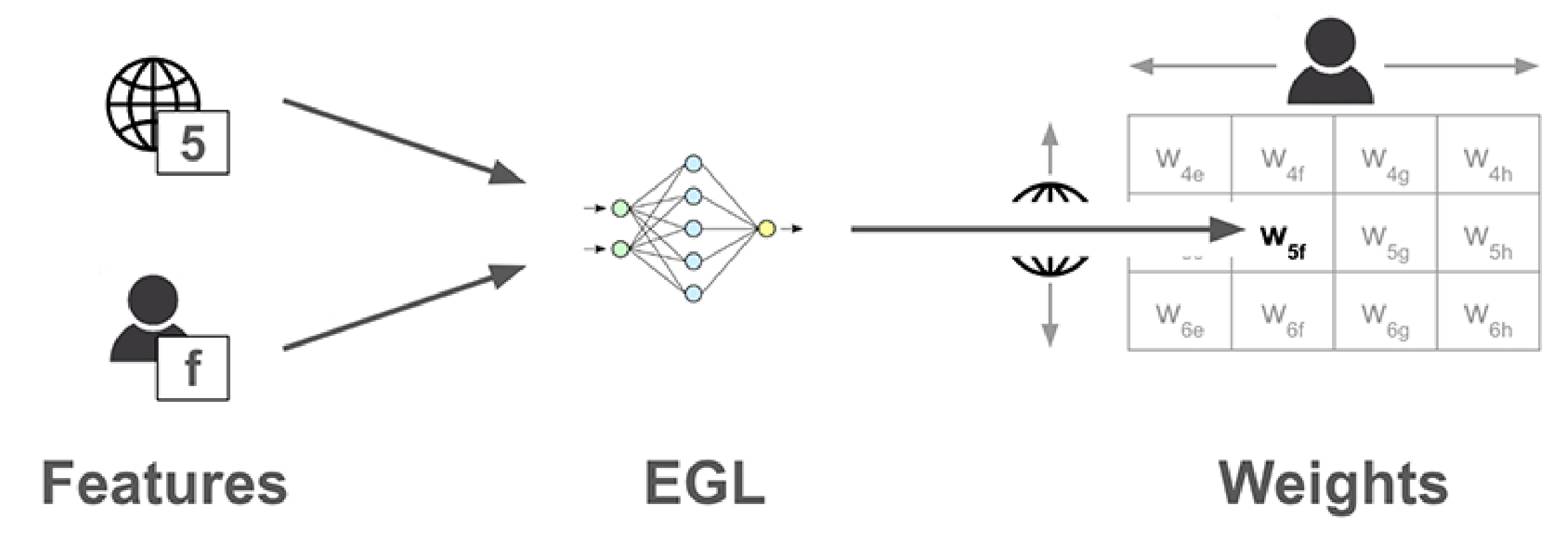
To generate realistic predictions, we sample predictions from the output of predictive models trained on the MSE loss across (a) different random initializations, and (b) different points in their training trajectory. While this does not generate the exact predictions that will be encountered while training the predictive model, we believe that it does a good job of characterizing the "space" of realistic predictions.



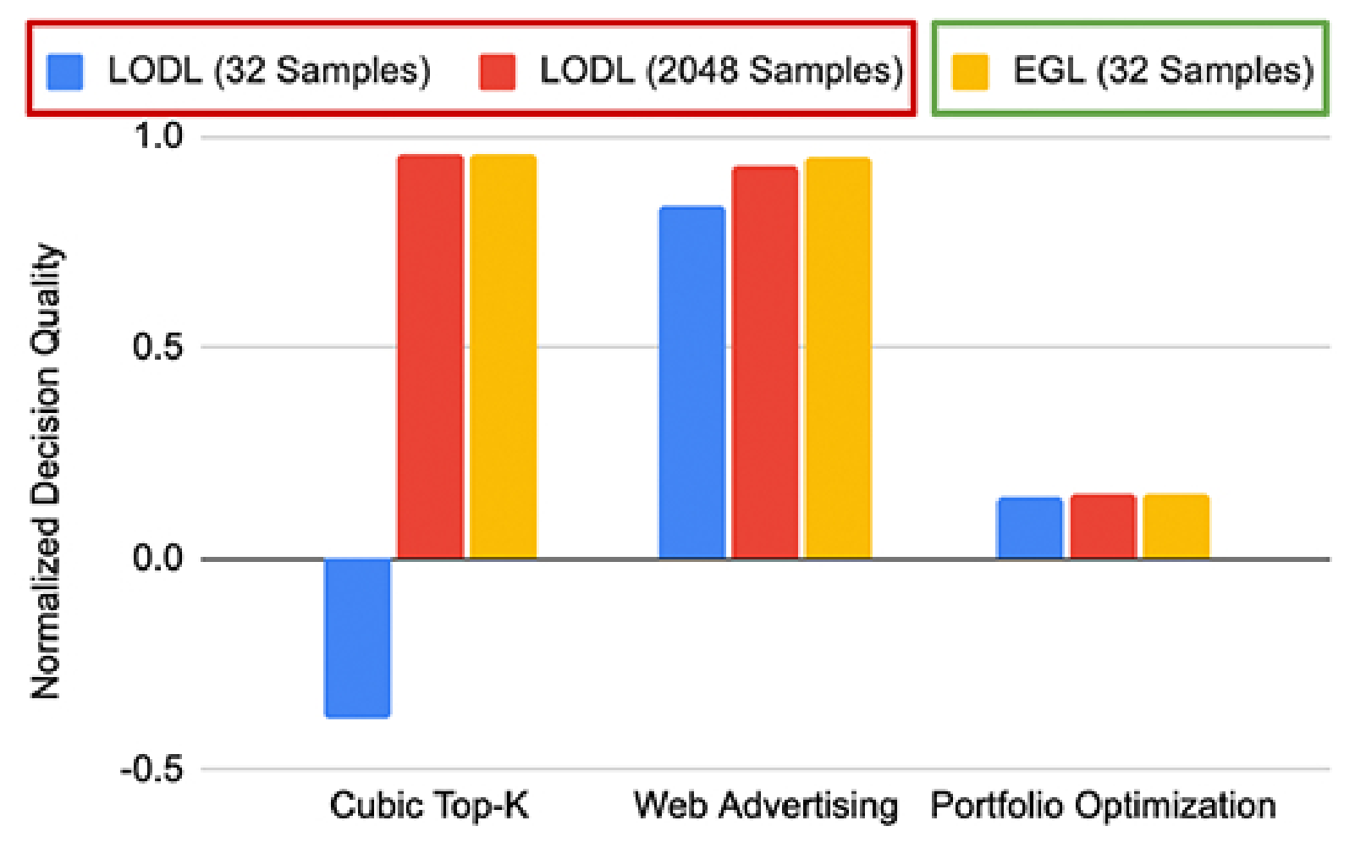
Contribution 2: Feature-Based Parameterization

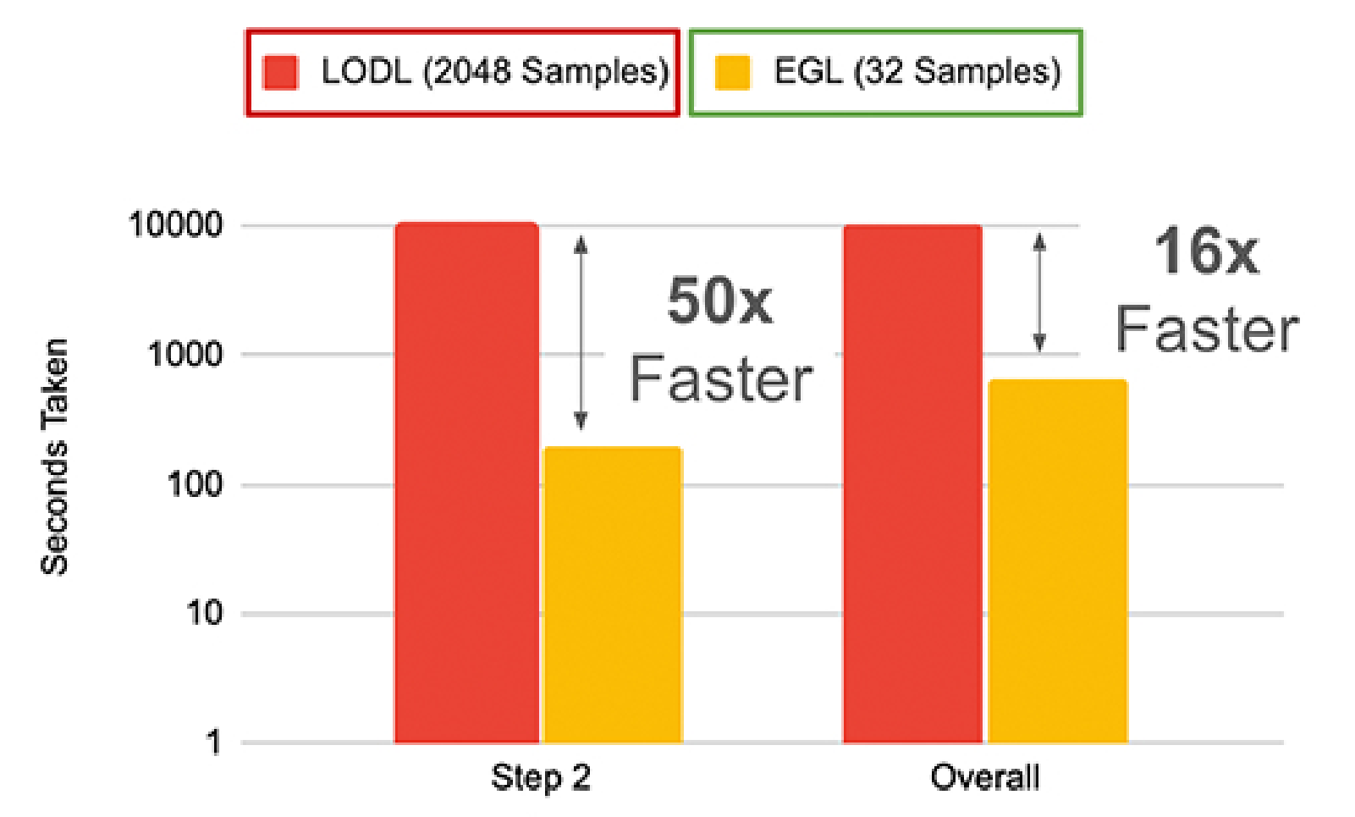
In Step 3, we have to learn a loss function, i.e., the decision quality induced by an arbitrary prediction. For a WeightedMSE type loss function, this involves learning the weights associated with each parameter such that the resulting loss function approximates the (regret in) decision quality. So, the higher the weight, the more an error in this parameter leads to a change in decision quality.

Past work tries to learn all these weights independently, which is easy from an optimization perspective but sample inefficient. Instead, in this paper, we learn a **single** mapping (e.g., neural network) from the features associated with a parameter to its corresponding weight.



Results





We evaluate the performance of our approach on the same three domains as our closest related work (LODLs).

In terms of **performance**, we find that our approach (EGLs) can perform on par with (or better than) LODLs with an order of magnitude fewer samples (32 vs 2048).

In terms of **computational cost**, this translates into a 16x speedup in learning loss functions over past work.

We believe that these improvements bring DFL one step closer to being accessible in practice.

Acknowledgements

Sanket Shah was supported by the ARO under Grant Number: W911NF-18-1-0208. Bryan Wilder was supported by NSF (Award Number 2229881). The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the NSF, ARO or the U.S. Government.