DF^2: Distribution-Free Decision-Focused Learning

Lingkai Kong[1,2] Wenhao Mu[2] Jiaming Cui[2,3] Yuchen Zhuang[2] B. Aditya Prakash[2] Bo Dai[2] Chao Zhang[2]

[1]Harvard University, Cambridge, Massachusetts, USA
[2]Georgia Institute of Technology, Atlanta, Georgia, USA
[3]Virginia Tech, Blacksburg, Virginia, USA

Abstract

Decision-focused learning (DFL), which differentiates through the KKT conditions, has recently emerged as a powerful approach for predict-then-optimize problems. However, under probabilistic settings, DFL faces three major bottlenecks: model mismatch error, sample average approximation error, and gradient approximation error. Model mismatch error stems from the misalignment between the model's parameterized predictive distribution and the true probability distribution. Sample average approximation error arises when using finite samples to approximate the expected optimization objective. Gradient approximation error occurs when the objectives are non-convex and KKT conditions cannot be directly applied. In this paper, we present DF^2 —the first *distribution-free* decision-focused learning method designed to mitigate these three bottlenecks. Rather than depending on a task-specific forecaster that requires precise model assumptions, our method directly learns the expected optimization function during training. To efficiently learn the function in a data-driven manner, we devise an attention-based model architecture inspired by the distribution-based parameterization of the expected objective. We evaluate DF^2 on two synthetic problems and three real-world problems, demonstrating the effectiveness of DF^2. Our code can be found at: `https://github.com/Lingkai-Kong/DF2`.

1 INTRODUCTION

Many decision-making problems are fundamentally optimization problems that require the minimization of a cost function, which often depends on parameters that are both *unknown* and *context-dependent*. Typically, these parameters are estimated using observed features. For instance, hedge funds regularly recalibrate their portfolios to maximize expected returns, which involves predicting the future return rates of various stocks. Similarly, in personalized medicine, the selection of treatments for individual patients must predict unique responses to ensure optimal outcomes.

Given the growing capacity to train powerful deep learning models, a common strategy for this problem is the two-stage pipeline. This approach first learns a predictive model for unknown parameters using a generic loss function (*e.g.*, negative log-likelihood) during the prediction stage, and then applies the model's outputs in a downstream optimization problem. Despite its widespread use, this pipeline implicitly assumes that better predictive accuracy—measured by the prediction loss—translates to better optimization performance. However, this assumption often breaks down, as prediction errors can have non-uniform effects on the optimization objective. To address this issue, *Decision-Focused Learning (DFL)* [Donti et al., 2017, Wilder et al., 2019, Wang et al., 2020b, Sun et al., 2023, Yan et al., 2021, Rodriguez-Diaz et al., 2024] integrates the prediction and optimization stages into a single end-to-end model. A prominent line of work leverages the implicit function theorem and the KKT conditions to differentiate through the optimization layer [Donti et al., 2017], enabling the learning process to align predictive outputs directly with decision quality. This results in models that are trained explicitly for decision-making, often framed as regret minimization.

Despite its promising results, DFL via differentiation through KKT conditions faces several critical bottlenecks in the probabilistic setting, where the predictive model outputs a distribution rather than a point estimate: (1) Model mismatch error: real-world applications often operate in highly uncertain environments and involve complex, multimodal probability distributions. In contrast, DFL by differentiating through KKT conditions requires simple parameterized distribution models for computational feasibility, leading to a mismatch. (2) Sample average approximation error: When there is no closed-form expression for the expected optimization objective, we typically draw a finite number

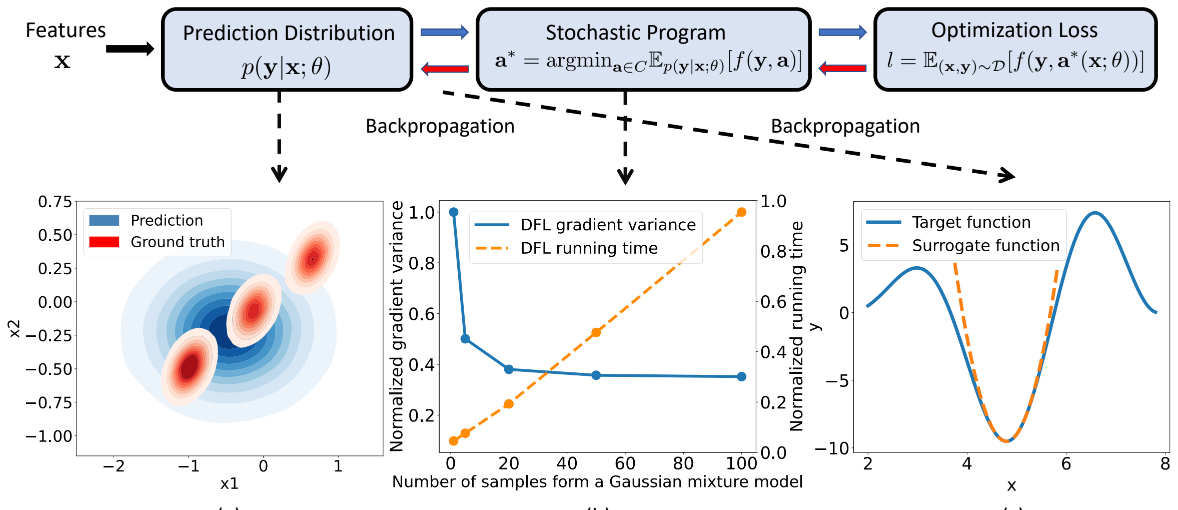

Features
$\mathbf{x}$
Prediction Distribution
$p(\mathbf{y}|\mathbf{x};\theta)$
Stochastic Program
$\mathbf{a}^* = \text{argmin}_{\mathbf{a}\in C}\mathbb{E}_{p(\mathbf{y}|\mathbf{x};\theta)}[f(\mathbf{y},\mathbf{a})]$
Optimization Loss
$l = \mathbb{E}_{(\mathbf{x},\mathbf{y})\sim\mathcal{D}}[f(\mathbf{y},\mathbf{a}^*(\mathbf{x};\theta))]$
Backpropagation
Backpropagation
Prediction
Ground truth
x1
x2
DFL gradient variance
DFL running time
Normalized gradient variance
Normalized running time
Number of samples form a Gaussian mixture model
Target function
Surrogate function
x
y
(a)
(b)
(c)

Figure 1: Decision-focused learning [Donti et al., 2017] directly optimizes the task loss and leads to better decision regret. However, it suffers from three significant bottlenecks. More illustrations are in Section 2.

of samples from the distribution for averaging, which will introduce extra statistical errors. (3) Gradient approximation error: the KKT condition is only a sufficient condition for optimal solution of convex problem, which is unable to characterize the optimal solution in non-convex setting, and thus, will lead to inaccuracies that cumulatively result in lower decision quality. Recent works [Kong et al., 2022, Shah et al., 2022, 2023] have proposed surrogate objectives to bypass the challenges of gradient computation. However, these approaches are still model-based and suffer from the other two bottlenecks. While SPO [Elmachtoub and Grigas, 2022] generally converges to a decision with optimal expected costs regardless of the distribution, it is restricted to linear objectives.

We propose DF^2, the first distribution-free decision-focused learning method, to mitigate the three bottlenecks and handle complex objectives beyond the linear class. Instead of relying on a task-specific forecaster that necessitates precise model assumptions, we propose to learn directly the expectation of the optimization objective function from the data. Upon learning, we can obtain the optimal decision by maximizing the learned expected function within the feasible space. In order to ensure that the network architecture lies within the true model class and minimize bias error, we have developed an attention-based network architecture that emulates the distribution-based parameterization of the expected objective. This attention architecture also preserves the convexity of the original optimization objective. In contrast to the two-stage model, DF^2 is decision-aware. Compared to DFL methods, DF^2 avoids model mismatch error, gradient approximation error, and sample average approximation error at test time.

Our main contributions can be summarized as follows: (1) We propose a distribution-free training objective for DFL. It mitigates the three bottlenecks of existing methods under the probabilistic setting. (2) We propose an attention-based network architecture inspired by the distribution-based parameterization to ensure the network architecture is within the true model class. (3) Experiments on two synthetic datasets and three real-world datasets show that our method can achieve better performance than existing DFL methods.

2 PRELIMINARIES

2.1 DECISION-FOCUSED LEARNING BY DIFFERENTIATING THROUGH KKT CONDITIONS

In the predict-then-optimize problem, a predictor $\mathbf{M}_\theta$ inputs features $\mathbf{x}$ and outputs a point estimate $\hat{\mathbf{y}}$. This estimate parameterizes the optimization problem $\arg\min_{\mathbf{a}\in C} f(\mathbf{y},\mathbf{a})$, where f is the cost function, $\mathbf{a}$ is the decision variable, and C is the feasible space.

However, point estimations fail to capture the uncertainty inherent in model predictions [Abdar et al., 2021] and the stochastic nature of the problem parameters [Schneider and Kirkpatrick, 2007]. To address this, we focus on a probabilistic framework, wherein the predictor's output is a probability distribution $p_\theta(\mathbf{y}|\mathbf{x})$, rather than a mere point estimate. This allows us to engage in stochastic optimization, where the objective is to find the optimal action $\mathbf{a}^*(\mathbf{x};\theta)$ that minimizes the expected cost, formalized as $\arg\min_{\mathbf{a}\in C}\mathbb{E}_{p_\theta(\mathbf{y}|\mathbf{x})}[f(\mathbf{y},\mathbf{a})]$. This method more effectively accounts for the uncertainties and variabilities present in the parameters.

Predictions are then evaluated based on the decision loss they generate, essentially the cost function's value using the true parameters $\mathbf{y}$. For a dataset $\mathcal{D} = \{\mathbf{x}_i,\mathbf{y}_i\}_{i=1}^N$, the goal is to train a model $\mathbf{M}_{\boldsymbol{\theta}}$ to minimize the decision loss:

$$\theta^* = \arg\min_\theta \frac{1}{N}\sum_{i=1}^N f(\mathbf{y}_i,\mathbf{a}^*(\mathbf{x}_i;\theta)). \qquad (1)$$

By directly optimizing the decision loss, the gradient of the model parameters can be calculated using the chain rule: $\frac{\mathrm{d}f(\mathbf{y},\mathbf{a}^*(\mathbf{x};\theta))}{\mathrm{d}\theta} = \frac{\mathrm{d}f(\mathbf{y},\mathbf{a}^*(\mathbf{x};\theta))}{\mathrm{d}\mathbf{a}^*(\mathbf{x};\theta)}\frac{\mathrm{d}\mathbf{a}^*(\mathbf{x};\theta)}{\mathrm{d}\theta}$. To compute the Jacobian $\frac{\mathrm{d}\mathbf{a}^*(\mathbf{x};\theta)}{\mathrm{d}\theta}$ for backpropagation, OptNet [Amos and Kolter, 2017] assumes quadratic optimization objectives and differentiates through the KKT conditions using the implicit function theorem. Later, cvxpylayers [Agrawal et al., 2019] extends it to more general cases of convex optimization using disciplined parameterized programming (DPP) grammar.

2.2 BOTTLENECKS UNDER THE PROBABILISTIC SETTING

Although DFL by differentiating through KKT condition can achieve better decisions compared to the two-stage learning, they have three significant bottlenecks under the probabilistic setting.

Bottleneck 1: Model Mismatch Error. Real-world applications often involve complex and multi-modal probability distributions $p(\mathbf{y}|\mathbf{x})$ [Kong et al., 2023]. One prominent example is the wind power forecasting task, where the environment exhibits high uncertainty due to the dynamic and stochastic nature of wind patterns. Factors such as changing weather conditions, terrain, and turbulence can significantly affect the true distribution of wind power, making it highly intricate and challenging to model accurately.

However, existing approaches [Donti et al., 2017, Kong et al., 2022] tend to assume simple distributions, *e.g.*, isotropic Gaussian distribution, for computational feasibility. However, this assumption can lead to considerable misalignment between the model's parameterized distribution and the true underlying distribution in tasks with high uncertainty. This mismatch results in poor approximations and reduced decision-focused learning performance. Fig. 1(a) illustrates this issue using a ground-truth distribution composed of a mixture of three Gaussians. As we can see, the performances of DFL approaches suffer due to the model mismatch error, which is particularly pronounced in tasks with highly uncertain environments.

Bottleneck 2: Sample Average Approximation Error. In complex optimization problems, closed-form expressions for expectations might be unavailable, necessitating the use of sample average approximation [Kim et al., 2015, Verweij et al., 2003, Kleywegt et al., 2002]. Although adopting a more expressive distribution, such as a mixture density network, could potentially improve performance, doing so introduces another issue—sample approximation error. As shown in Fig. 1(b), when dealing with intricate distributions, increasing the sample size reduces the gradient variance slowly but demands substantially higher computational resources and longer running times.

Bottleneck 3: Gradient Approximation Error. The KKT condition can only be applied to convex objectives. However, many real-world applications involve complicated non-convex objectives. Though Perrault et al. [2020], Wang et al. [2020a] propose to approximate the non-convex objectives by a quadratic function around a local minimum to approximate $\frac{\mathrm{d}\mathbf{a}^*}{\mathrm{d}\theta}$ (Fig. 1(c)), the inaccurate gradients may be aggregated during the training iterations and thus lead to poor decisions.

The first two errors occur during both training and testing, whereas gradient approximation errors occur only during training. Recently, several methods [Kong et al., 2022, Shah et al., 2022, 2023] have proposed surrogate losses for DFL to avoid differentiating through KKT conditions. However, they still suffer from the first two bottlenecks.

It should be noted that when the objective function is linear, the expectation of a linear function has a closed-form expression and only requires estimating the mean of a distribution. Therefore, the model does not suffer from these bottlenecks. As a result, SPO [Elmachtoub and Grigas, 2022] proves that it converges to a decision with optimal expected costs regardless of the distribution. In our paper, we consider a more complex setting where estimating the expected cost requires the entire predictive distribution.

3 DISTRIBUTION-FREE DECISION-FOCUSED LEARNING

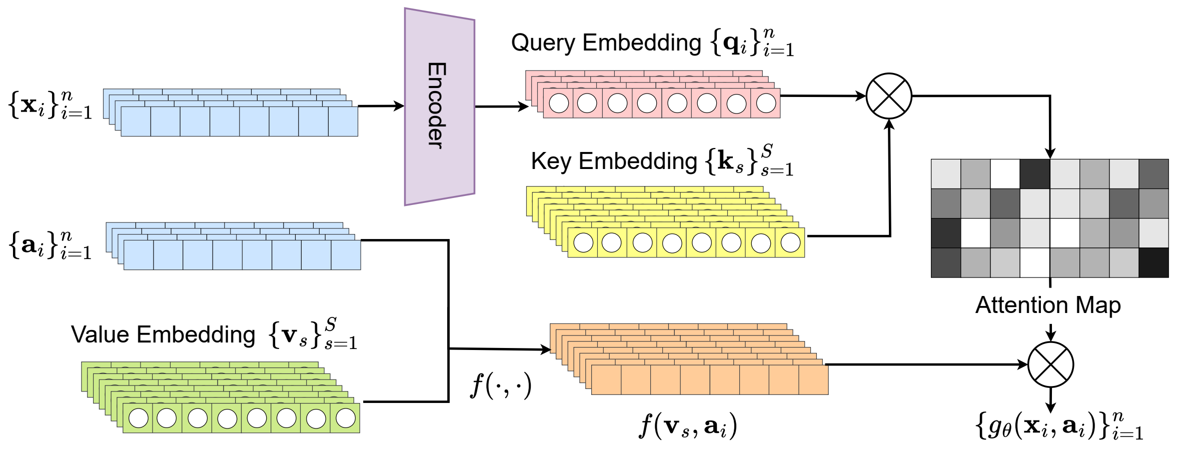
In this section, we introduce DF^2 which mitigates all the three bottlenecks within a single model. We first introduce the distribution-free training objective which transforms DFL into a function approximation problem. Then, we design an attention-based architecture inspired by the distribution-based parameterization to reduce the bias error. Finally, we discuss how to obtain the optimal decision during inference.

3.1 DISTRIBUTION-FREE TRAINING OBJECTIVE

Existing DFL methods primarily rely on a distribution-based approach. These techniques learn a forecaster that outputs probability distribution $p(\mathbf{y}|\mathbf{x})$ based on various model assumptions. However, a more straightforward approach is to estimate the expected cost function $\mathbb{E}_{p(\mathbf{y}|\mathbf{x})}[f(\mathbf{y},\mathbf{a})]$ directly from the training data $\mathcal{D} = \{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^N$.

The cornerstone of our method is the observation that the expected cost objective is only a function of $\mathbf{a}$ and $\mathbf{x}$, which is represented as $g(\mathbf{x},\mathbf{a}) = \mathbb{E}_{p(\mathbf{y}|\mathbf{x})}[f(\mathbf{y},\mathbf{a})]$. We propose a direct approach to learn a neural network with parameters θ to match the expected cost function $\mathbb{E}_{p(\mathbf{y}|\mathbf{x})}[f(\mathbf{y},\mathbf{a})]$. Our objective is to minimize the mean square error (MSE) between the fitted function $g(\mathbf{x},\mathbf{a})$ and the cost function $f(\mathbf{y},\mathbf{a})$ sampled from $p(\mathbf{x},\mathbf{y})$:

$$g^*(\mathbf{x},\mathbf{a}) = \arg\min_g \mathbb{E}_{(\mathbf{x},\mathbf{y})}\mathbb{E}_a[g(\mathbf{x},\mathbf{a}) - f(\mathbf{y},\mathbf{a})]^2. \quad (2)$$



$\{\mathbf{x}_i\}_{i=1}^n$
Encoder
Query Embedding $\{\mathbf{q}_i\}_{i=1}^n$
Key Embedding $\{\mathbf{k}_s\}_{s=1}^S$
$\{\mathbf{a}_i\}_{i=1}^n$
Attention Map
Value Embedding $\{\mathbf{v}_s\}_{s=1}^S$
$f(\cdot,\cdot)$
$f(\mathbf{v}_s, \mathbf{a}_i)$
$\{g_\theta(\mathbf{x}_i, \mathbf{a}_i)\}_{i=1}^n$

Figure 2: The proposed attention-based network architecture of DF^2. The network contains an encoder and a set of learnable attention points $\{\mathbf{k}_s, \mathbf{v}_s\}_{s+1}^S$. Given an input feature $\mathbf{x}$, the encoder first project it to query embedding space and then compute the attention weights by its dot product with the key embeddings. The final function value $g(\mathbf{x}, \mathbf{a})$ is a weighted combination of $f(\mathbf{v}, \mathbf{a})$. The designed network architecture can effectively reduce the bias error in Proposition 1.

The proposed training objective can be efficiently optimized using stochastic gradient-based methods such as ADAM [Kingma and Ba, 2015b].

In the ideal case, when we have infinite training data and model capacity, the optimal solution g^* of Eq. 2 is the ground-truth conditional expectation $\mathbb{E}_{p(\mathbf{y}|\mathbf{x})}[f(\mathbf{y}, \mathbf{a})]$. Upon learning the optimal function, the optimal action can be derived by maximizing the fitted function $\mathbf{a}^* = \arg\min_{\mathbf{a}\in C} g_\theta(\mathbf{x}, \mathbf{a})$. However, in practical situations where training data and model capacity are limited, we obtain the expected error on the test set as the following proposition.

Proposition 1. *The expected MSE of the optimal solution g^* on the test set is:*

$$MSE_{\text{test}} = \underbrace{\mathbb{E}_{\mathcal{D}'}\left[\left(g^*_{\mathcal{D}'}(\mathbf{x}, \mathbf{a}) - \mathbb{E}_{p(\mathbf{y}|\mathbf{x})}[f(\mathbf{y}, \mathbf{a})]\right)^2\right]}_{\textit{Bias}} + \underbrace{\mathbb{E}_{\mathcal{D}'}\left[\left(g^*_{\mathcal{D}'}(\mathbf{x}, \mathbf{a}) - \mathbb{E}_{\mathcal{D}'}[g^*_{\mathcal{D}'}(\mathbf{x}, \mathbf{a})]\right)^2\right]}_{\textit{Variance}},$$

where $\mathcal{D}'$ denotes the training dataset augmented with the sampled actions $\mathbf{a}$, and $g^*_{\mathcal{D}'}(\mathbf{x}, \mathbf{a})$ denotes the function fitted on the dataset $\mathcal{D}'$.

Proof. *See Appendix D for a detailed proof.*

Sampling Action from the Constrained Space. In practice, it's unnecessary to fit the true objective across the entire Euclidean space. Instead, we only need to sample from the constrained space C. There are several strategies for this. One simple approach is to sample from a relaxed version of the constrained space,such as an outer bounding box that encloses C. This allows us to sample each dimension of $\mathbf{a}$ independently from a uniform distribution. Moreover, many predict-then-optimize problems are resource allocation problems where the decision variable $\mathbf{a}$ is a simplex; for a simplex, we can directly sample from the Dirichlet distribution. Appendix B provides more illustrations on the relaxed constrained sampling. Alternatively, we can employ Markov chain Monte Carlo (MCMC) methods to uniformly sample within C, such as Ball Walk [Lovász and Simonovits, 1990] and the hit-and-run algorithm [Bélisle et al., 1993, Lovász, 1999]. However, these methods typically incur higher computational costs.

In contrast to traditional DFL, our framework effectively transforms decision-focused learning into a function approximation problem, circumventing the complexities of solving and differentiating through the optimization problem. This approach avoids both model mismatch error and gradient approximation error. While we do not claim to fully address the sample average approximation error during training, as we still rely on finite data to estimate the expected cost function, we can avoid this error at inference time, see Section 3.3.

As we can see from Proposition 1, the test MSE consists of the bias and variance terms. The variance term will be reduced by sampling more data. To ensure that the bias error term approaches zero with more training data, it is crucial to keep the network architecture within the model class. To tackle this challenge, we introduce an attention-based network architecture in the following subsection.

3.2 DISTRIBUTION-BASED PARAMETERIZATION

The key of our architecture design is to mimic the distribution-based parameterization of the expected cost function. Since our training objective bypass the need of solving and differentiating through the stochastic optimization problem, we can adopt an expressive non-parametric distribution with kernel conditional mean embedding (CME)

Variable	$\mathbf{x}$	$\mathbf{y}$
Domain	$\mathcal{X}$	$\mathcal{Y}$
Kernel	$\mathcal{R}_{\mathbf{x}}(\mathbf{x},\mathbf{x}')$	$\mathcal{R}_{\mathbf{y}}(\mathbf{y},\mathbf{y}')$
Feature map	$\mathcal{R}_{\mathbf{x}}(\mathbf{x},\cdot)$	$\mathcal{R}_{\mathbf{y}}(\mathbf{y},\cdot)$
RKHS	$\mathcal{G}$	$\mathcal{F}$

Table 1: Table of Notations

to parameterize our model. The proposed network architecture can lead to zero bias error in Proposition 1.

CME [Song et al., 2009, 2013] is a powerful tool to compute the expectation of a function in the reproducing kernel Hilbert space (RKHS), without the curse of dimensionality. Let $\mathcal{F}$ be a RKHS over the domain of $\mathbf{y}$ with kernel function $\mathcal{R}_{\mathbf{y}}(\mathbf{y},\mathbf{y}')$ and inner product $\langle\cdot,\cdot\rangle_{\mathcal{F}}$. For a particular $\mathbf{a}$, we denote the corresponding function as $f_{\mathbf{a}}(\mathbf{y})$. CME projects the conditional distribution to its expected feature map $\mu_{\mathbf{y}|\mathbf{x}} \triangleq \mathbb{E}_{p(\mathbf{y}|\mathbf{x})}[\mathcal{R}_{\mathbf{y}}(\mathbf{y},\cdot)]$ and evaluates the conditional expectation of any RKHS function, $f_{\mathbf{a}} \in \mathcal{F}$, as an inner product in $\mathcal{F}$ using the reproducing property:

$$\begin{aligned}\mathbb{E}_{p(\mathbf{y}|\mathbf{x})}[f_{\mathbf{a}}] &= \int p(\mathbf{y}|\mathbf{x})\langle\mathcal{R}_y(\mathbf{y},\cdot), f_{\mathbf{a}}\rangle_{\mathcal{F}} d\mathbf{y} \\ &= \left\langle \int p(\mathbf{y}|\mathbf{x})\mathcal{R}_y(\mathbf{y},\cdot)\mathrm{d}\mathbf{y}, f_{\mathbf{a}}\right\rangle_{\mathcal{F}} = \langle\mu_{\mathbf{y}|\mathbf{x}}, f_{\mathbf{a}}\rangle_{\mathcal{F}}.\end{aligned}$$

Assume that for all $f_{\mathbf{a}} \in \mathcal{F}$, the conditional expectation $\mathbb{E}_{p(\mathbf{y}|\mathbf{x})}[f_{\mathbf{a}}(\mathbf{y})]$ is an element of the RKHS over the domain of $\mathbf{x}$, the conditional embedding can be estimated with a finite dataset $\{\mathbf{x}_s,\mathbf{y}_s\}_{s=1}^S$ as $\hat{\mu}_{\mathbf{y}|\mathbf{x}} = \sum_{s=1}^S \beta_s(\mathbf{x})\mathcal{R}_{\mathbf{y}}(\mathbf{y}_s,\cdot)$, where β_s is a real-valued weight and can be computed with matrix calculation (see more details about this computation in Appendix C).

One advantage of CME is that $\hat{\mu}_{\mathbf{y}|\mathbf{x}}$ can converge to $\mu_{\mathbf{y}|\mathbf{x}}$ in the RKHS norm at an overall rate of $\mathcal{O}(S^{-\frac{1}{2}})$ [Song et al., 2009], which is independent of the input dimensions. This property let CME works well in the high-dimensional space. With the estimated CME, the conditional expectation can be computed by the reproducing property:

$$\begin{aligned}\mathbb{E}_{p(\mathbf{y}|\mathbf{x})}[f_{\mathbf{a}}(\mathbf{y})] = \langle\hat{\mu}_{\mathbf{y}|\mathbf{x}}, f_{\mathbf{a}}\rangle_{\mathcal{F}} &= \left\langle \sum_{s=1}^S \beta_s(\mathbf{x})\mathcal{R}_{\mathbf{y}}(\mathbf{y}_s,\cdot), f_{\mathbf{a}}\right\rangle_{\mathcal{F}} \\ &= \sum_{s=1}^S \beta_s(\mathbf{x})f_{\mathbf{a}}(\mathbf{y}_s). \end{aligned} \tag{3}$$

As shown in Eq. 3, the formulation is essentially a weighted combination of $f_{\mathbf{a}}(\mathbf{y}_s)$, where the weights are conditioned on the input features $\mathbf{x}$. This observation inspires us to leverage attention-based parameterization to represent the function $g(\mathbf{x},\mathbf{a})$. The attention mechanism forms the foundation of the transformer architecture [Vaswani et al., 2017] and has been successfully utilized across various deep learning applications [Kenton and Toutanova, 2019, Brown et al., 2020, Dosovitskiy et al., 2021].

Inspired by this, we introduce a set of learnable attention points $\{\mathbf{k}_s,\mathbf{v}_s\}_{s=1}^S$, where $\mathbf{k}$ is the key embedding and $\mathbf{v}$ is the corresponding value embedding. For an input $\mathbf{x}$, the encoder first maps it to the query embedding space $\mathbf{q}$ and compute the attention weights by its product with the key embeddings. We set the value function as $f(\mathbf{v}_s,\mathbf{a})$ and, consequently, reformulate the function $g(\mathbf{x},\mathbf{a})$ using the softmax attention mechanism [Vaswani et al., 2017]:

$$\begin{aligned} g(\mathbf{x},\mathbf{a}) =&\mathrm{Softmax}\left(\left[\frac{\mathbf{q}(\mathbf{x})^\top\mathbf{k}_1}{\sqrt{d}},\cdots,\frac{\mathbf{q}(\mathbf{x})^\top\mathbf{k}_S}{\sqrt{d}}\right]\right)^\top \\ &[f(\mathbf{v}_1,\mathbf{a}),\cdots,f(\mathbf{v}_S,\mathbf{a})],\end{aligned} \tag{4}$$

where d is the dimension size of the key embeddings and value embeddings.

Proposition 2. *It holds for any $\mathbf{x}$ and $\mathbf{a}$, the function $g(\mathbf{x},\mathbf{a})$ defined by the softmax attention in Eq. 4 $\mathbb{E}_{\hat{p}_{\mathcal{R}}(\mathbf{y}|\mathbf{x})}[f(\mathbf{y},\mathbf{a})] = g(\mathbf{x},\mathbf{a})$. Here, $\hat{p}_{\mathcal{R}}(\mathbf{y}|\mathbf{x})$ is a parameterization restriction of $p(\mathbf{y}|\mathbf{x})$.*

Proof. See Appendix E for a detailed proof.

From Proposition 2, it is evident that with the attention-based network architecture, we can guarantee that our learned expected function resides within the true model class. To speed up the training procedure, one can initialize the value embeddings of the attention points with randomly selected labels from the training dataset. This approach provides a reasonable starting point for the model and reduces the time it takes for the model to converge to a solution. The training procedure of DF^2 is given in Algorithm 1.

Remark. Our proposed attention-based network architecture represents a parameterization of $p(\mathbf{y}|\mathbf{x})$, drawing similarities with the two-stage model and DFL. Compared with the two-stage model, we learn the expected cost function to make DF^2 decision-aware. Compared with DFL, we do not have to solve the stochastic optimization problem during learning. As a result, we can adopt an expressive nonparametric distribution with CME to parameterize $p(\mathbf{y}|\mathbf{x})$ to avoid the model mismatch error.

3.3 MODEL INFERENCE

At test time, we can obtain the optimal decision by maximizing the learned expected cost $\arg\min_{\mathbf{a}\in C} g(\mathbf{x},\mathbf{a})$. The final representation of $g(\mathbf{x},\mathbf{a})$ is a weighted combination of $f(\mathbf{v}_s,\mathbf{a})$ with different value embeddings. Another benefit of the proposed attention-based network architecture is that it can preserve the convex property of the cost function.

Proposition 3. *As long as $f(\mathbf{y},\mathbf{a})$ is a convex function with respect to $\mathbf{a}$, $g(\mathbf{x},\mathbf{a})$ is a convex function with respect to $\mathbf{a}$.*

Proof: This is a direct consequence of the theorem that a convex combination of convex functions remains a convex function

When the original objective is convex, we can use any existing black-box convex solver [Diamond and Boyd, 2016, Agrawal et al., 2018, Gurobi Optimization, LLC, 2023]. For non-convex problem, we can use projected gradient descent.

Although Eq. 2 involves sampling $\mathbf{x}, \mathbf{y}$ during training, this introduces generalization error due to the finite size of the training dataset. Crucially, this generalization error is distinct from the SAA error, which arises in existing methods that require sampling from a predicted distribution (e.g., a Gaussian with learned parameters) to estimate an expected objective. In such cases, the generalization error in the predictive model is further compounded by the additional variance introduced through sampling, resulting in compounded inaccuracies.

In contrast, our method learns the expected objective $g(\mathbf{x}, \mathbf{a})$ directly and does not require sampling at inference time, thereby eliminating the additional SAA error. Nonetheless, like all learning-based methods, it remains subject to generalization error stemming from limited training data.

4 ADDITIONAL RELATED WORK

SO-EBM [Kong et al., 2022] proposes a surrogate learning objective by maximizing the likelihood of the precomputed optimal decision within an energy-based probability parameterization. LODL [Shah et al., 2022, 2023] and LANCER [Zharmagambetov et al., 2023] approximate the decision-focused loss with a quadratic function or a neural network. DF^2 is different from them: (1) They assume a deterministic setting while we assume the problem parameter $\mathbf{y}$ is a probability distribution. (2) They approximate the decision loss which is a function of the problem parameter $\mathbf{y}$. In contrast, DF^2 directly learns the expected cost function which remains independent of $\mathbf{y}$. (3) They still relies on initially learning a forecaster to infer $\mathbf{y}$ from $\mathbf{x}$. Consequently, they remain susceptible to both model mismatch error and sample average approximation error in our probabilistic setting. Recently, Bansal et al. [2023] proposes TaskMet with the motivation to simultaneously optimize predictive loss and decision loss, rather than addressing the three bottlenecks.

Several other works have focused on linear objectives, where DFL through KKT condition may encounter singular value issues. To address this, the SPO+ loss [Elmachtoub and Grigas, 2022] evaluates prediction errors relative to optimization objectives using the subgradient method. The approach by Wilder et al. [2019] incorporates a quadratic regularization term for smoothing. Meanwhile, Mandi and Guns [2020] introduces a log barrier regularizer and differentiates through the homogeneous self-dual embedding. In contrast, our method is crafted for a broader range of objectives.

When the optimization problem is discrete, differentiating through the optimization layer is even more challenging since the gradient is ill-defined in the discrete domain. Various solutions have been proposed, such as tackling the discrete challenge via interpolation [Pogančić et al., 2019], perturbation [Niepert et al., 2021, Berthet et al., 2020], subgradient methods [Mandi et al., 2020], and cutting planes [Ferber et al., 2020]. Our method is directly applicable to the discrete setting and we leave it for future exploration.

5 EXPERIMENTS

In this section, we empirically evaluate DF^2 and conduct experiments in both synthetic and real-world scenarios. Finally, we perform ablation studies to show the effect of each model design in DF^2.

5.1 SYNTHETIC PROBLEMS

To highlight the ability to learn the true expected objective, we first validate our method on a synthetic dataset where the true underlying model is known to us. To simulate the multimodal scenario in the real world, we generate 5000 feature-parameter pairs using a Gaussian mixture model with three components (3 GMM). We consider both convex and non-convex objectives. The details of the data generation process and the objectives are provided in the Appendix F.2.

Experimental Setup.

Since we know the true underlying data generation process for this synthetic setting, we compute the lower bound of the decision regret and use the gap of the model's decision regret from this lower bound as the evaluation metric. We compare with the following baselines: (1) A two-stage model trained with negative log-likelihood. (2) DFL [Donti et al., 2017]. (3) SO-EBM [Kong et al., 2022]: It uses the energy-based model as a surrogate objective to speed up DFL. (4) Policy-net: It directly maps from the input features to the decision variables by minimizing the task loss using supervised learning. (5) LODL [Shah et al., 2022]: it approximates the decision loss with a surrogate function.

For the two-stage model, DFL, SO-EBM and LODL, the forecasters use GMM with a different number of components and use 100 samples to estimate the expectation of the objective as we found that more samples bring limited performance gain but lead to longer training time. We also evaluate scenarios where the forecaster provides only a point estimate of the problem parameter, with the exception of SO-EBM, which is originally used in the probabilistic setting. For a fair comparison, we use the same backbone for the encoder of DF^2 and the forecaster of the baselines and 1000 attention points for both the convex and non-convex objectives. For the two-stage model, DFL and SO-EBM, the forecasters use GMM with a different number of components and use 100 samples to estimate the expectation of the objective as we found that more samples bring little performance gain. For a fair comparison, we use the same backbone for the encoder of DF^2 and the forecaster of the baselines and 1000 attention points for both the convex and

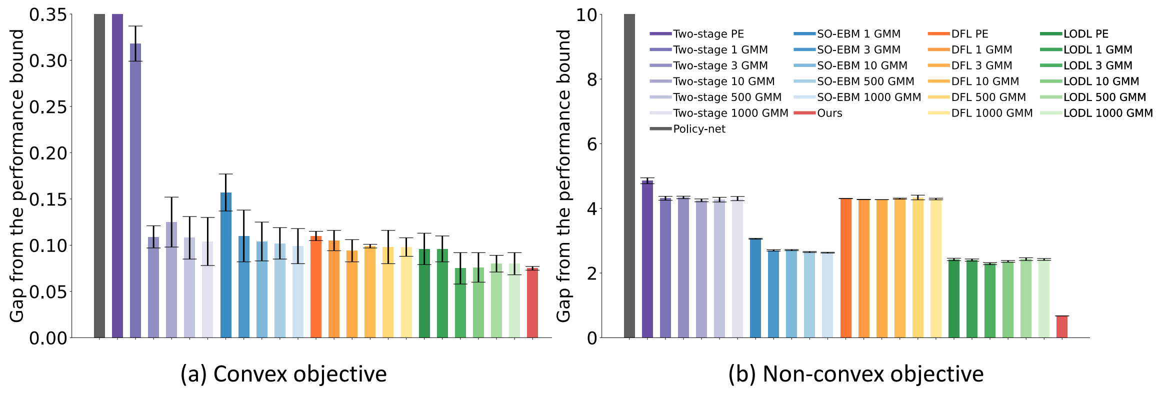


Figure 3: The gap of the model's decision regret from the lower bound of the decision regret on the synthetic data for both the convex and non-convex objectives. 'PE' denotes that the forecaster only produces a point estimate for the problem parameter.

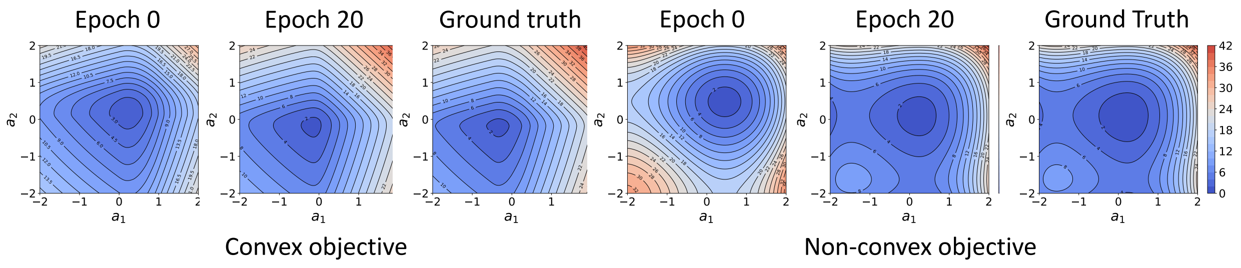


Figure 4: Randomly initialized landscape, DF^2 recovered landscape and the ground-truth landscape on the synthetic data. The landscape is conditioned on an input feature sampled from the test set.

non-convex objectives. Appendix F provides more details of the experimental setup and model parameters.

Results. Fig. 3 shows the results on both the convex and non-convex objective for all the methods. As we can see, DF^2 can outperform all the baselines. The improvement of DF^2 against the baselines becomes more significant on the non-convex objective. Specifically, DF^2 reduces the gap from the performance bound by 56.5% compared with the strongest baseline LODL. When the baseline methods utilized GMMs with a different number of components, their performance deteriorated, indicating that they suffer from model mismatch errors. In contrast, our method consistently outperformed the baselines, regardless of the number of components they used. This consistency is evidence that our approach can effectively mitigate model mismatch errors. Even when the baseline methods were aligned with the ground-truth model class, our method still outperformed them since we can also avoid the sampling average approximation error at test time.

It's important to note that when the forecaster yields only a point estimate, both existing DFL frameworks and the two-stage method show the worst performance for this imbalanced cost function. This underscores the importance of quantifying uncertainty in the forecaster's predictions, especially in risk-sensitive domains.

Fig. 4 visualizes the learned expected function and the ground truth expectation on a test sample for both objectives. We found that the DF^2 can effectively recover the landscape of the ground truth expected cost.

5.2 REAL-WORLD PROBLEMS

Next, we delve into three real-world problems encompassing both convex and non-convex objectives.

5.2.1 Experimental Setup.

Wind Power Bidding. In this task, a wind power firm engages in both energy and reserve markets, given the generated wind power $\mathbf{x} \in \mathbb{R}^{24}$ in the last 24 hours. The firm needs to decide the energy quantity $\mathbf{a}_E \in \mathbb{R}^{12}$ to bid and quantity $\mathbf{a}_R \in \mathbb{R}^{12}$ to reserve over the next 12-24 hours in advance, based on the forecasted wind power $\mathbf{y} \in \mathbb{R}^{12}$. The optimization objective is a piecewise function consisting of three segments [Sanayha and Vateekul, 2022a, Cao et al., 2020a], which is to maximize the revenue of the energy sales while minimizing the penalties for decision inaccuracies of overbidding and underbidding.

Inventory Optimization. In this task, a department store is tasked with predicting the sales $\mathbf{y} \in \mathbb{R}^7$ for the upcoming 7th-14th days based on the past 14 days' sales data $\mathbf{x} \in \mathbb{R}^{14}$ for a specific product, and accordingly, determining the best replenishment strategy $\mathbf{a} \in \mathbb{R}^7$ for each day. The optimization objective is a combination of an under-purchasing penalty, an over-purchasing penalty, and a squared loss be-

Method	Decision Regret		
	Power Bidding	Inventory Opt.	Vaccine Dist.
Policy-net	489.01 ±12.39	3.96 ±0.28	604 ±12.30
Two-stage PE	518.19 ±14.84	3.97 ±0.15	573 ±10.26
Two-stage 1-GMM	69.36 ±4.33	3.32 ±0.10	538 ±9.30
Two-stage 3-GMM	69.89 ±1.50	3.27 ±0.08	534 ±8.40
Two-stage 10-GMM	70.51 ±2.29	3.29 ±0.05	533 ±7.95
Two-stage 500-GMM	66.84 ±1.43	3.24 ±0.07	524 ±7.95
Two-stage 1000-GMM	65.83 ±1.70	3.27 ±0.05	527 ±7.65
SO-EBM 1-GMM	67.32 ±1.97	3.37 ±0.02	512 ±8.55
SO-EBM 10-GMM	66.93 ±2.45	3.26 ±0.03	513 ±7.95
SO-EBM 500-GMM	67.02 ±2.16	3.37 ±0.05	513 ±8.70
SO-EBM 1000-GMM	66.40 ±2.23	3.21 ±0.07	516 ±9.45
DFL PE	69.46 ±1.21	3.35 ±0.03	519 ±7.37
DFL 1-GMM	66.85 ±1.47	3.36 ±0.02	515 ±8.25
DFL 3-GMM	66.60 ±3.23	3.36 ±0.05	513 ±7.05
DFL 10-GMM	66.45 ±2.32	3.31 ±0.01	513 ±7.65
DFL 500-GMM	65.06 ±0.88	3.24 ±0.09	507 ±6.60
DFL 1000-GMM	64.65 ±3.70	3.21 ±0.07	513 ±7.35
LODL PE	67.92 ±1.49	3.36 ±0.06	512 ±7.01
LODL 1-GMM	66.87 ±1.36	3.34 ±0.01	508 ±6.23
LODL 3-GMM	65.75 ±1.86	3.31 ±0.06	506 ±6.84
LODL 10-GMM	65.29 ±1.23	3.26 ±0.02	504 ±6.38
LODL 500-GMM	64.24 ±1.45	3.22 ±0.05	502 ±7.02
LODL 1000-GMM	64.13 ±2.47	3.24 ±0.04	503 ±7.01
Ours	**60.90** ±0.60	**3.09** ±0.09	**492** ±7.05

Table 2: Decision regret of each method – **lower is better**. 'PE' denotes point estimate for the parameter.

tween supplies and demands.

Vaccine Distribution for COVID-19. During the COVID-19 pandemic, computing a vaccine distribution strategy is one of the most challenging problems for epidemiologists and policymakers. In practice, meta-population Ordinary Differential Equations (ODEs) based epidemiological models [Pei et al., 2020a] are widely used to predict and evaluate the outcomes of different vaccine distribution strategies. These models rely on people mobility data, such as Origin-Destination (OD) matrices, to capture the pandemic spread dynamics across diverse locations [Li et al., 2020a]. In this task, given the OD matrices $\mathbf{x} \in \mathbb{R}^{47\times47\times7}$ of last week, *i.e.*, $\mathbf{x}[i,j,t]$ represents the number of people move from region i to j on day t, we need to decide the vaccine distribution $\mathbf{a} \in \mathbb{R}^{47}$ across the 47 regions in Japan with a budget constraint ($\mathbf{a}[i]$ is the number of vaccines distributed to the region i). The optimization objective is to minimize the total number of infected people over the ODE-drived dynamics, based on the forecasted OD matrices $\mathbf{y} \in \mathbb{R}^{47\times47\times7}$ for the next week. This task is a challenging non-convex optimization problem due to the nonlinear simulation model.

Due to space limit, we provide more details of the experimental setup and the optimization objectives in Appendix F.

5.2.2 Results.

Table 2 presents the decision regret across three real-world problems, demonstrating that our method consistently outperforms all baselines. Specifically, DF^2 improves decision regret by $\{5.0\%, 3.7\%, 2.0\%\}$ compared to the strongest baseline. These three forecasting tasks are characterized by high uncertainty, making it challenging to formulate a precise model assumption. In such scenarios, it is more effective to derive the expected cost function directly from the data, eliminating the need for any parametric distribution assumptions. Moreover, it is clear that simply increasing the number of components in the GMM does not significantly enhance DFL's performance due to increased sample approximation errors. Finally, the probabilistic approach generally exhibits higher and more reliable performance than methods that rely solely on learning a point estimate forecaster.

5.3 ABLATION STUDY

In this subsection, we investigate each component of DF^2 via ablation studies on the wind power bidding.

Impact of attention-based architecture. Without the attention-based network architecture, we see a significant performance drop in Fig. 5(a). This is because, without the attention architecture, the network architecture may not be within the true model class and thus suffer from high bias error in Proposition 1.

Impact of number of attention points. Our model performance can be improved with more attention points as in Fig.5(b). We also plot the decision regret and training time of DFL. We find that when the number of attention points is over 200, DF^2 can outperform DFL in terms of the decision regret while being orders of magnitude faster.

Impact of training data size. Our method outperforms baselines constantly with different ratios of training data as shown in Fig. 5(c). The superior performance is because we use attention-based network architecture to mimic the distribution-based parameterization. Compared with the two-stage model, we are decision-aware; compared with DFL methods, we mitigate the three bottlenecks.

DF^2 *vs DFL with different number of samples.* The number of samples used to estimate the expected objective in DFL is an important hyperparameter. To investigate its impact, we compare the decision regret and training time of DF^2 with DFL using different numbers of samples. We use GMM with 1000 components in the DFL forecaster as it achieves the best performance shown in Section 5.2. As shown in Fig. 6, when the number of samples for DFL exceeds 100, the performance improvement becomes very marginal (64.52 with 100 samples vs. 64.07 with 200 samples). However, the training time increases significantly (1878 seconds/epoch with 100 samples vs. 5251 seconds/epoch with 200 samples). In contrast, DF^2 achieves significantly better decision regret (58.41) while being orders of magnitude faster (2.17 seconds/epoch).

Impact of learnable value embeddings. In DF^2, the value embeddings are initialized with randomly sampled labels from the training set and then updated during the training process. An alternative is to directly use these randomly

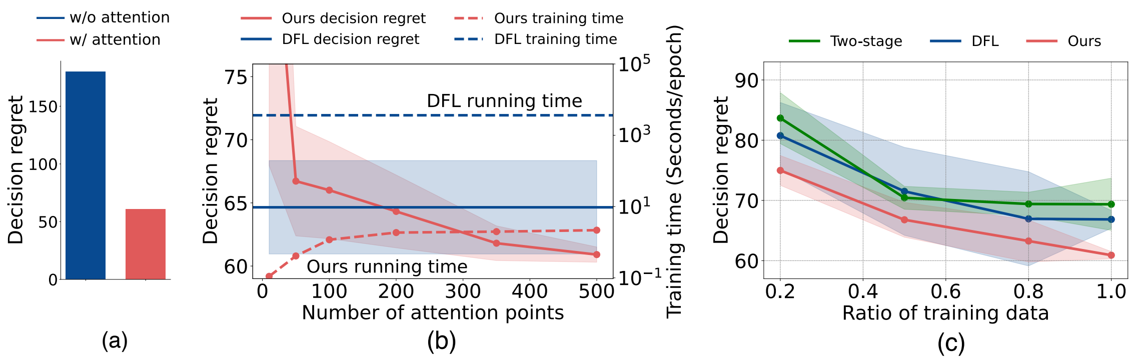


Figure 5: Ablation study on the impact of attention-based architecture, number of attention points, and training data size on the wind power bidding problem.

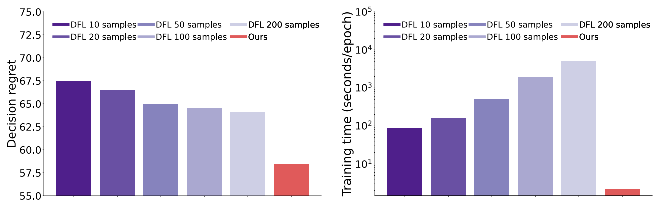


Figure 6: DF^2 vs. DFL with different numbers of samples: the left figure shows decision regret, while the right figure displays training time.

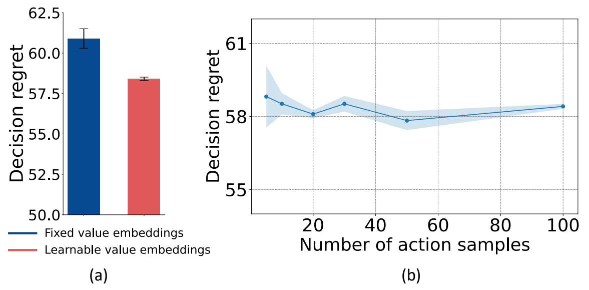


Figure 7: Impact of learnable value embeddings and number of action samples.

selected labels and keep the value embeddings fixed during the training process. We examine whether making the value embeddings learnable improves the performance. The results are shown in Fig. 7(a). As we can see, with learnable value embeddings, the decision regret of DF^2 decreases significantly compared with the fixed value embeddings.

Impact of number of action samples. In DF^2, we need to sample actions for each $(\mathbf{x}, \mathbf{y})$ pair at each training iteration to fit the function. In this study, we investigate the influence of the number of action samples on the performance. As shown in Fig. 7(b), the decision regret remains stable even for a sample size of 5. Notably, as the number of action samples increases, the variance of the decision regret across different random seeds decreases, indicating improved stability in the results.

6 CONCLUSION AND LIMITATIONS

We focus on mitigating the three bottlenecks of DFL by differentiating through KKT conditions under the probabilistic setting: (1) model mismatch error, (2) sample average approximation error, and (3) gradient approximation error. To this end, we propose DF^2– the first distribution-free DFL method which does not require any model assumption. DF^2 adopts a distribution-free training objective that directly learns the expected cost function from the data. To reduce the bias error, we design an attention-based network architecture, drawing inspiration from the distribution-based parameterization of the expected cost function. Empirically, we demonstrate that DF^2 is effective in a wide range of stochastic optimization problems with either convex or non-convex objectives.

Limitations. In our work, we focus on the probabilistic setting where the predictive distribution of the forecasting task has high uncertainty. In this setting, both model mismatch error and sample average approximation error are significant. However, if the forecasting task is relatively straightforward, a simple Gaussian distribution might suffice. For certain objective functions, the expectation under a Gaussian distribution has a closed-form expression. In such cases, existing model-based DFL methods may still be a better choice.

When the number of attention points is large, scalability may become an issue at inference time. This challenge can potentially be alleviated by employing fast attention mechanisms, such as sparse attention (*e.g.*, Longformer; [Beltagy et al., 2020]) or low-rank approximations (*e.g.*, Linformer; [Wang et al., 2020c]).

ACKNOWLEDGMENTS

This work was supported in part by the following grants and awards:

- NSF IIS-2008334, IIS-2106961, IIS-2403240, CAREER IIS-2028586, CAREER IIS-2144338, RAPID IIS-2027862, Medium IIS-1955883, Medium IIS-2106961, Medium IIS-2403240, Expeditions CCF-

1918770, PIPP CCF-2200269, ECCS-2401391

- ONR N000142512173
- Centers for Disease Control and Prevention Modeling Infectious Diseases In Healthcare program
- Dolby faculty research award
- Meta faculty gift, and funds/computing resources from Georgia Tech and GTRI.
- PPP DA 2025 Flash Funding.

REFERENCES

Moloud Abdar, Farhad Pourpanah, Sadiq Hussain, Dana Rezazadegan, Li Liu, Mohammad Ghavamzadeh, Paul Fieguth, Xiaochun Cao, Abbas Khosravi, U Rajendra Acharya, et al. A review of uncertainty quantification in deep learning: Techniques, applications and challenges. *Information fusion*, 76:243–297, 2021.

Akshay Agrawal, Robin Verschueren, Steven Diamond, and Stephen Boyd. A rewriting system for convex optimization problems. *Journal of Control and Decision*, 5(1): 42–60, 2018.

Akshay Agrawal, Brandon Amos, Shane Barratt, Stephen Boyd, Steven Diamond, and J Zico Kolter. Differentiable convex optimization layers. *Advances in neural information processing systems*, 32, 2019.

Brandon Amos and J Zico Kolter. Optnet: Differentiable optimization as a layer in neural networks. In *International Conference on Machine Learning*, pages 136–145. PMLR, 2017.

Dishank Bansal, Ricky TQ Chen, Mustafa Mukadam, and Brandon Amos. Taskmet: Task-driven metric learning for model learning. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.

Amir Beck and Marc Teboulle. Mirror descent and nonlinear projected subgradient methods for convex optimization. *Operations Research Letters*, 31(3):167–175, 2003.

Claude JP Bélisle, H Edwin Romeijn, and Robert L Smith. Hit-and-run algorithms for generating multivariate distributions. *Mathematics of Operations Research*, 18(2): 255–266, 1993.

Iz Beltagy, Matthew E Peters, and Arman Cohan. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*, 2020.

Quentin Berthet, Mathieu Blondel, Olivier Teboul, Marco Cuturi, Jean-Philippe Vert, and Francis Bach. Learning with differentiable pertubed optimizers. *Advances in neural information processing systems*, 33:9508–9519, 2020.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

Di Cao, Weihao Hu, Xiao Xu, Tomislav Dragicevic, Qi Huang, Zhou Liu, Zhe Chen, and Frede Blaabjerg. Bidding strategy for trading wind energy and purchasing reserve of wind power producer – a drl based approach. *International Journal of Electrical Power & Energy Systems*, 117:105648, 2020a.

Di Cao, Weihao Hu, Xiao Xu, Tomislav Dragicevic, Qi Huang, Zhou Liu, Zhe Chen, and Frede Blaabjerg. Bidding strategy for trading wind energy and purchasing reserve of wind power producer – a drl based approach. *International Journal of Electrical Power & Energy Systems*, 117:105648, 2020b.

Steven Diamond and Stephen Boyd. CVXPY: A Python-embedded modeling language for convex optimization. *Journal of Machine Learning Research*, 17(83):1–5, 2016.

Priya Donti, Brandon Amos, and J Zico Kolter. Task-based end-to-end model learning in stochastic optimization. *Advances in neural information processing systems*, 30, 2017.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=YicbFdNTTy.

Adam N Elmachtoub and Paul Grigas. Smart "predict, then optimize". *Management Science*, 68(1):9–26, 2022.

Aaron Ferber, Bryan Wilder, Bistra Dilkina, and Milind Tambe. Mipaal: Mixed integer program as a layer. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 1504–1511, 2020.

Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2023. URL https://www.gurobi.com.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016.

Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. In *International Conference on Learning Representations*,

2017. URL https://openreview.net/forum?id=rkE3y85ee.

Jacob Devlin Ming-Wei Chang Kenton and Lee Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of NAACL-HLT*, pages 4171–4186, 2019.

Sujin Kim, Raghu Pasupathy, and Shane G Henderson. A guide to sample average approximation. *Handbook of simulation optimization*, pages 207–243, 2015.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *International Conference on Representation Learning*, 2015a.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Representation Learning*, 2015b.

Anton J Kleywegt, Alexander Shapiro, and Tito Homem-de Mello. The sample average approximation method for stochastic discrete optimization. *SIAM Journal on Optimization*, 12(2):479–502, 2002.

Lingkai Kong, Jiaming Cui, Yuchen Zhuang, Rui Feng, B Aditya Prakash, and Chao Zhang. End-to-end stochastic optimization with energy-based model. In *Advances in Neural Information Processing Systems*, 2022.

Lingkai Kong, Harshavardhan Kamarthi, Peng Chen, B Aditya Prakash, and Chao Zhang. Uncertainty quantification in deep learning. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 5809–5810, 2023.

Ruiyun Li, Sen Pei, Bin Chen, Yimeng Song, Tao Zhang, Wan Yang, and Jeffrey Shaman. Substantial undocumented infection facilitates the rapid dissemination of novel coronavirus (sars-cov-2). *Science*, 368(6490):489–493, 2020a.

Ruiyun Li, Sen Pei, Bin Chen, Yimeng Song, Tao Zhang, Wan Yang, and Jeffrey Shaman. Substantial undocumented infection facilitates the rapid dissemination of novel coronavirus (sars-cov-2). *Science*, 368(6490):489–493, 2020b.

Yaguang Li, Rose Yu, Cyrus Shahabi, and Yan Liu. Diffusion convolutional recurrent neural network: Data-driven traffic forecasting. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=SJiHXGWAZ.

László Lovász. Hit-and-run mixes fast. *Mathematical programming*, 86:443–461, 1999.

László Lovász and Miklós Simonovits. The mixing rate of markov chains, an isoperimetric inequality, and computing the volume. In *Proceedings [1990] 31st annual symposium on foundations of computer science*, pages 346–354. IEEE, 1990.

Jayanta Mandi and Tias Guns. Interior point solving for lp-based prediction+ optimisation. *Advances in Neural Information Processing Systems*, 33:7272–7282, 2020.

Jayanta Mandi, Peter J Stuckey, Tias Guns, et al. Smart predict-and-optimize for hard combinatorial optimization problems. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 1603–1610, 2020.

Mathias Niepert, Pasquale Minervini, and Luca Franceschi. Implicit mle: backpropagating through discrete exponential family distributions. *Advances in Neural Information Processing Systems*, 34:14567–14579, 2021.

Sen Pei, Sasikiran Kandula, and Jeffrey Shaman. Differential effects of intervention timing on covid-19 spread in the united states. *Science advances*, 6(49):eabd6370, 2020a.

Sen Pei, Sasikiran Kandula, and Jeffrey Shaman. Differential effects of intervention timing on covid-19 spread in the united states. *Science advances*, 6(49):eabd6370, 2020b.

Andrew Perrault, Bryan Wilder, Eric Ewing, Aditya Mate, Bistra Dilkina, and Milind Tambe. End-to-end game-focused learning of adversary behavior in security games. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 1378–1386, 2020.

Marin Vlastelica Pogančić, Anselm Paulus, Vit Musil, Georg Martius, and Michal Rolinek. Differentiation of blackbox combinatorial solvers. In *International Conference on Learning Representations*, 2019.

Paula Rodriguez-Diaz, Lingkai Kong, Kai Wang, David Alvarez-Melis, and Milind Tambe. What is the right notion of distance between predict-then-optimize tasks? *arXiv preprint arXiv:2409.06997*, 2024.

Manassakan Sanayha and Peerapon Vateekul. Model-based deep reinforcement learning for wind energy bidding. *International Journal of Electrical Power & Energy Systems*, 136:107625, 2022a.

Manassakan Sanayha and Peerapon Vateekul. Model-based deep reinforcement learning for wind energy bidding. *International Journal of Electrical Power & Energy Systems*, 136:107625, 2022b.

Johannes Schneider and Scott Kirkpatrick. *Stochastic optimization*. Springer Science & Business Media, 2007.

Sanket Shah, Bryan Wilder, Andrew Perrault, and Milind Tambe. Learning (local) surrogate loss functions for predict-then-optimize problems. *Advances in Neural Information Processing Systems*, 2022.

Sanket Shah, Andrew Perrault, Bryan Wilder, and Milind Tambe. Leaving the nest: Going beyond local loss functions for predict-then-optimize. *arXiv preprint arXiv:2305.16830*, 2023.

Le Song, Jonathan Huang, Alex Smola, and Kenji Fukumizu. Hilbert space embeddings of conditional distributions with applications to dynamical systems. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 961–968, 2009.

Le Song, Kenji Fukumizu, and Arthur Gretton. Kernel embeddings of conditional distributions: A unified kernel framework for nonparametric inference in graphical models. *IEEE Signal Processing Magazine*, 30(4):98–111, 2013.

Haixiang Sun, Ye Shi, Jingya Wang, Hoang Duong Tuan, H. Vincent Poor, and Dacheng Tao. Alternating differentiation for optimization layers. In *The Eleventh International Conference on Learning Representations*, 2023. URL `https://openreview.net/forum?id=KKBMz-EL4tD`.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Bram Verweij, Shabbir Ahmed, Anton J Kleywegt, George Nemhauser, and Alexander Shapiro. The sample average approximation method applied to stochastic routing problems: a computational study. *Computational optimization and applications*, 24(2):289–333, 2003.

Kai Wang, Andrew Perrault, Aditya Mate, and Milind Tambe. Scalable game-focused learning of adversary models: Data-to-decisions in network security games. In *AAMAS*, pages 1449–1457, 2020a.

Kai Wang, Bryan Wilder, Andrew Perrault, and Milind Tambe. Automatically learning compact quality-aware surrogates for optimization problems. *Advances in Neural Information Processing Systems*, 33:9586–9596, 2020b.

Sinong Wang, Belinda Z Li, Madian Khabsa, Han Fang, and Hao Ma. Linformer: Self-attention with linear complexity. *arXiv preprint arXiv:2006.04768*, 2020c.

Bryan Wilder, Bistra Dilkina, and Milind Tambe. Melding the data-decisions pipeline: Decision-focused learning for combinatorial optimization. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 1658–1665, 2019.

Kai Yan, Jie Yan, Chuan Luo, Liting Chen, Qingwei Lin, and Dongmei Zhang. A surrogate objective framework for prediction+ programming with soft constraints. *Advances in Neural Information Processing Systems*, 34:21520–21532, 2021.

Arman Zharmagambetov, Brandon Amos, Aaron M Ferber, Taoan Huang, Bistra Dilkina, and Yuandong Tian. Landscape surrogate: Learning decision losses for mathematical optimization under partial information. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL `https://openreview.net/forum?id=qyEm4tF2p1`.

Appendix for DF^2: Distribution-Free Decision-Focused Learning

Lingkai Kong[1,2] **Wenhao Mu**[2] **Jiaming Cui**[2,3] **Yuchen Zhuang**[2] **B. Aditya Prakash**[2] **Bo Dai**[2] **Chao Zhang**[2]

[1]Harvard University, Cambridge, Massachusetts, USA
[2]Georgia Institute of Technology, Atlanta, Georgia, USA
[3]Virginia Tech, Blacksburg, Virginia, USA

A TRAINING ALGORITHM

The full training procedure of DF^2 is given in Algorithm 1.

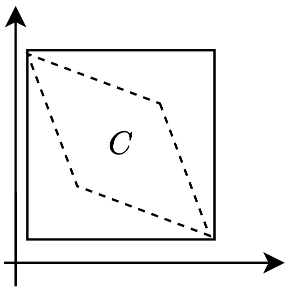


Figure 8: Relaxed constrained sampling. We can sample from the encompassing outer box of the original constrained space.

Algorithm 1 Training Procedure of DF^2

```
Require: Objective function $f$, feasible set $\mathcal{C}$, training dataset $\mathcal{D}$
Ensure: Learned encoder, value embeddings, and key embeddings
 1: Initialize encoder, value embeddings, and key embeddings
 2: for $t = 1$ to $T$ do
 3:     Sample a mini-batch $B = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^{|B|}$ from $\mathcal{D}$
 4:     for each $(\mathbf{x}_i, \mathbf{y}_i)$ in $B$ do
 5:         Sample actions $\{\mathbf{a}_i^j\}_{j=1}^J$ from the feasible set $C$
 6:         Compute $g(\mathbf{x}_i, \mathbf{a}_i^j)$ for all $j$, as defined in Eq. 4.
 7:         Compute the MSE loss for the $i$-th sample:
                $$L_i = \frac{1}{J}\sum_{j=1}^{J}\left(f(\mathbf{x}_i, \mathbf{y}_i) - g(\mathbf{x}_i, \mathbf{a}_i^j)\right)^2$$
 8:     end for
 9:     Update encoder, value embeddings, and key embeddings using the aggregated loss $\sum_i L_i$
10: end for
```

B CONSTRAINED SAMPLING

In practice, it's unnecessary to fit the true objective across the entire Euclidean space. Instead, we only need to sample from the constrained space C. There are several strategies for this. First, we can employ Markov chain Monte Carlo (MCMC) methods to uniformly sample within C, such as Ball Walk Lovász and Simonovits [1990] and the hit-and-run algorithm Bélisle et al. [1993], Lovász [1999]. Alternatively. we can sample from a relaxed constrained space, such as the encompassing outer box of the original constrained space. This allows us to sample each dimension of $\mathbf{a}$ independently from a uniform distribution. Figure 8 gives an illustration.

Consider the following convex constraints:

$$\mathbf{A}\mathbf{a} = \mathbf{b}, \quad \mathbf{G}\mathbf{a} \preceq \mathbf{h}. \tag{5}$$

In particular, instead of directly sampling the full-dimensional decision vector $\mathbf{a}$, we initially output a subset of the variables $a_1, \cdots, a_d$, and then deduce the remaining variables by resolving the given set of equations.

To sample from the relaxed constraint space, we initially determine the maximum and minimum values for $a_1, \cdots, a_d$, guided by the given inequality constraints. These boundary points can be effortlessly acquired utilizing the Python SciPy package. Following this, we execute uniform sampling between these extremal values for each variable in the set $a_1, \cdots, a_d$. Essentially, we transform the polyhedron into a box, simplifying the uniform sampling process.

Furthermore, many predict-then-optimize problems manifest as resource allocation issues wherein the decision variable $\mathbf{a}$ embodies a simplex; in such cases, we can directly sample from the Dirichlet distribution.

Variable	$\mathbf{x}$	$\mathbf{y}$
Domain	$\mathcal{X}$	$\mathcal{Y}$
Kernel	$\mathcal{R}_{\mathbf{x}}(\mathbf{x},\mathbf{x}')$	$\mathcal{R}_{\mathbf{y}}(\mathbf{y},\mathbf{y}')$
Feature map	$\phi(\mathbf{x})/\mathcal{R}_{\mathbf{x}}(\mathbf{x},\cdot)$	$\varphi(\mathbf{y})/\mathcal{R}_{\mathbf{y}}(\mathbf{y},\cdot)$
Feature matrix	$\Upsilon = (\phi(\mathbf{x}_1),\cdots,\phi(\mathbf{x}_s))$	$\Phi = (\varphi(\mathbf{y}_1),\cdots,\varphi(\mathbf{y}_s))$
Kernel matrix	$\mathbf{K} = \Upsilon^\top\Upsilon$	$\mathbf{L} = \Phi^\top\Phi$
RKHS	$\mathcal{G}$	$\mathcal{F}$

Table 3: Table of Notations

C ADDITIONAL BACKGROUND ON CONDITIONAL MEAN EMBEDDING

We provide more details about how to compute conditional mean embedding (CME) in this subsection. Table 3 presents the notations related to CME.

Let $\mathcal{F}$ be a reproducing kernel Hilbert space (RKHS) over the domain of $\mathbf{y}$ with kernel function $\mathcal{R}_{\mathbf{y}}(\mathbf{y},\mathbf{y}')$ and inner product $\langle\cdot,\cdot\rangle_{\mathcal{F}}$. Its inner product $\langle\cdot,\cdot\rangle_{\mathcal{F}}$ satisfies the reproducing property:

$$\langle f(\cdot), \mathcal{R}_{\mathbf{y}}(\mathbf{y},\cdot)\rangle_{\mathcal{F}} = f(\mathbf{y}),$$

meaning that we can view the evaluation of a function $f \in \mathcal{F}$ at any point $\mathbf{y}$ as an inner product and the linear evaluation operator is given by $\mathcal{R}_{\mathbf{y}}(\mathbf{y},\cdot)$, *i.e.*, the kernel function. Alternatively, $\mathcal{R}_{\mathbf{y}}(\mathbf{y},\cdot)$ can also be viewed as a feature map $\varphi(\mathbf{y})$ where $\mathcal{R}_{\mathbf{y}}(\mathbf{y},\mathbf{y}') = \langle\varphi(\mathbf{y}),\varphi(\mathbf{y}')\rangle_{\mathcal{F}}$. Similarly, we can define the RKHS $\mathcal{G}$ over the domain of $\mathbf{x}$ with kernel function $\mathcal{R}_{\mathbf{x}}(\mathbf{x},\mathbf{x}')$.

For a particular $\mathbf{a}$, we denote the corresponding function with respect to $\mathbf{y}$ as $f_{\mathbf{a}}(\mathbf{y})$. CME projects the conditional distribution to its expected feature map $\mu_{\mathbf{y}|\mathbf{x}} \triangleq \mathbb{E}_{p(\mathbf{y}|\mathbf{x})}[\mathcal{R}_{\mathbf{y}}(\mathbf{y},\cdot)]$ and evaluates the conditional expectation of any RKHS function, $f_{\mathbf{a}} \in \mathcal{F}$, as an inner product in $\mathcal{F}$ using the reproducing property:

$$\begin{aligned}\mathbb{E}_{p(\mathbf{y}|\mathbf{x})}[f_{\mathbf{a}}] &= \int p(\mathbf{y}|\mathbf{x})\langle\mathcal{R}_{\mathbf{y}}(\mathbf{y},\cdot), f_{\mathbf{a}}\rangle_{\mathcal{F}} d\mathbf{y} \\ &= \left\langle \int p(\mathbf{y}|\mathbf{x})\mathcal{R}_{\mathbf{y}}(\mathbf{y},\cdot)\mathrm{d}\mathbf{y}, f_{\mathbf{a}} \right\rangle_{\mathcal{F}} \\ &= \langle\mu_{\mathbf{y}|\mathbf{x}}, f_{\mathbf{a}}\rangle_{\mathcal{F}}. \end{aligned} \tag{6}$$

Assume that for all $f_{\mathbf{a}} \in \mathcal{F}$, the conditional expectation $\mathbb{E}_{p(\mathbf{y}|\mathbf{x})}[f_{\mathbf{a}}(\mathbf{y})]$ is an element of $\mathcal{G}$, the conditional embedding can be estimated with a finite dataset $\{\mathbf{x}_s,\mathbf{y}_s\}_{s=1}^S$ as Song et al. [2013, 2009]:

$$\hat{\mu}_{\mathbf{y}|\mathbf{x}} = \Phi(\mathbf{K}+\lambda\mathbf{I})^{-1}\Upsilon^\top\phi(\mathbf{x}) = \sum_{s=1}^{S}\beta_s(\mathbf{x})\mathcal{R}_{\mathbf{y}}(\mathbf{y}_s,\cdot), \tag{7}$$

where $\Phi = (\mathcal{R}_{\mathbf{y}}(\mathbf{y}_1,\cdot),\cdots,\mathcal{R}_{\mathbf{y}}(\mathbf{y}_S,\cdot))$ is the feature matrix; $\mathbf{K} = \Upsilon^\top\Upsilon$ is the Gram matrix for samples from variable $\mathbf{x}$ with $\Upsilon = (\mathcal{R}_{\mathbf{x}}(\mathbf{x}_1,\cdot),\cdots,\mathcal{R}_{\mathbf{x}}(\mathbf{x}_S,\cdot))$; λ is the additional regularization parameter to avoid overfitting. Though the assumption $\mathbb{E}_{p(\mathbf{y}|\mathbf{x})}[f_{\mathbf{a}}(\mathbf{y})] \in \mathcal{G}$ is not necessarily true for continuous domains, existing works treat the expression as an approximation Song et al. [2009] and works well in practice.

One advantage of CME is that $\hat{\mu}_{\mathbf{y}|\mathbf{x}}$ can converge to $\mu_{\mathbf{y}|\mathbf{x}}$ in the RKHS norm at an overall rate of $\mathcal{O}(S^{-\frac{1}{2}})$ Song et al. [2009], which is independent of the input dimensions. This property let CME works well in the high-dimensional space.

As we can see from Eq. 7, the empirical estimator of CME, $\hat{\mu}_{\mathbf{y}|\mathbf{x}}$, applies non-uniform weights, β_s, on observations which are, in turn, determined by the conditioning variable $\mathbf{x}$.

D PROOF OF PROPOSITION 1

Proposition 1. *The expected MSE of the optimal solution g^* on the test set is:*

$$MSE_{\text{test}} = \underbrace{\mathbb{E}_{\mathcal{D}'}\left[\left(g^*_{\mathcal{D}'}(\mathbf{x},\mathbf{a}) - \mathbb{E}_{p(\mathbf{y}|\mathbf{x})}[f(\mathbf{y},\mathbf{a})]\right)^2\right]}_{\textit{Bias}} + \underbrace{\mathbb{E}_{\mathcal{D}'}\left[\left(g^*_{\mathcal{D}'}(\mathbf{x},\mathbf{a}) - \mathbb{E}_{\mathcal{D}'}[g^*_{\mathcal{D}'}(\mathbf{x},\mathbf{a})]\right)^2\right]}_{\textit{Variance}},$$

where $\mathcal{D}'$ denotes the training dataset augmented with the sampled actions $\mathbf{a}$.

Proof. The training set consists of the given $\mathcal{D} = \{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^N$ augmented with the sampled actions $\mathbf{a}$. We denote the augmented dataset as $\mathcal{D}'$. We assume the fitted function is in a hypothesis $g^*(\mathbf{x},\mathbf{a})$. Let $g^*_{\mathcal{D}'}(\mathbf{x},\mathbf{a})$ denote the function fitted on the dataset $\mathcal{D}'$. The expectation of the mean squared error (MSE) for a given unseen test sample, over all possible learning sets, is:

$$\begin{aligned}
&\mathbb{E}_{\mathcal{D}'}[(\mathbb{E}_{p(\mathbf{y}|\mathbf{x})}[f(\mathbf{y},\mathbf{a})] - g^*_{\mathcal{D}'}(\mathbf{x},\mathbf{a}))^2] \\
&= \mathbb{E}_{\mathcal{D}'}[(\underbrace{\mathbb{E}_{p(\mathbf{y}|\mathbf{x}^*)}[f(\mathbf{y},\mathbf{a})] - \mathbb{E}_{\mathcal{D}'}[g^*_{\mathcal{D}'}(\mathbf{x},\mathbf{a})]}_{a} \\
&+ \underbrace{\mathbb{E}_{\mathcal{D}'}[g^*_{\mathcal{D}'}(\mathbf{x},\mathbf{a})] - g^*_{\mathcal{D}'}(\mathbf{x},\mathbf{a})}_{b})^2] \\
&= \mathbb{E}_{\mathcal{D}'}[(a+b)^2] \\
&= \mathbb{E}_{\mathcal{D}'}[a^2] + \mathbb{E}_{\mathcal{D}'}[b^2] + \mathbb{E}_{\mathcal{D}'}[2ab]
\end{aligned}$$

The first two terms represent the bias and variance errors respectively:

$$\mathbb{E}_{\mathcal{D}'}[a^2] = \mathbb{E}_{\mathcal{D}'}\left[\left(g^*_{\mathcal{D}'}(\mathbf{x},\mathbf{a}) - \mathbb{E}_{p(\mathbf{y}|\mathbf{x})}[f(\mathbf{y},\mathbf{a})]\right)^2\right] = \text{Bias}^2(g^*).$$

$$\mathbb{E}_{\mathcal{D}'}[b^2] = \mathbb{E}_{\mathcal{D}'}\left[\left(g^*_{\mathcal{D}'}(\mathbf{x},\mathbf{a}) - \mathbb{E}_{\mathcal{D}'}[g^*_{\mathcal{D}'}(\mathbf{x},\mathbf{a})]\right)^2\right] = \text{Variance}(g^*),$$

Next, we prove the cross-term $\mathbb{E}_{\mathcal{D}'}[2ab] = 0$. To simplify the notation, let $\overline{g}$ denote $\mathbb{E}_{\mathcal{D}'}[g^*_{\mathcal{D}'}(\mathbf{x},\mathbf{a})]$; g denote $g^*_{\mathcal{D}'}(\mathbf{x},\mathbf{a})$; $\tilde{f}$ denote $\mathbb{E}_{p(\mathbf{y}|\mathbf{x})}[f(\mathbf{y},\mathbf{a})]$. Then we can obtain:

$$\begin{aligned}
&\mathbb{E}_{\mathcal{D}'}\left[2\left(g-\overline{g}\right)\left(\overline{g}-\tilde{f}\right)\right] \\
&= 2 \cdot \mathbb{E}_{\mathcal{D}'}[g \cdot \overline{g} - g \cdot \tilde{f} - \overline{g} \cdot \overline{g} + \overline{g} \cdot \tilde{f}] \\
&= 2 \cdot \mathbb{E}_{\mathcal{D}'}[g] \cdot \overline{g} - 2 \cdot \mathbb{E}_{\mathcal{D}'}[g] \cdot \tilde{f} - 2 \cdot \mathbb{E}_{\mathcal{D}'}[\overline{g}^2] + 2 \cdot \tilde{f} \cdot \mathbb{E}_{\mathcal{D}'}[\overline{g}] \\
&= 2 \cdot \overline{g}^2 - 2 \cdot \overline{g} \cdot \tilde{f} - 2 \cdot \overline{g}^2 + 2 \cdot \tilde{f} \cdot \overline{g} \\
&= 0
\end{aligned}$$

Hence, the expectation of the MSE for a given test sample $\mathbf{x}^*$ is expressed as:

$$\begin{aligned}
\text{MSE}_{\text{test}} &= \mathbb{E}_{\mathcal{D}'}[(\mathbb{E}_{p(\mathbf{y}|\mathbf{x})}[f(\mathbf{y},\mathbf{a})] - g^*_{\mathcal{D}'}(\mathbf{x},\mathbf{a}))^2] \\
&= \underbrace{\mathbb{E}_{\mathcal{D}'}\left[\left(g^*_{\mathcal{D}'}(\mathbf{x},\mathbf{a}) - \mathbb{E}_{p(\mathbf{y}|\mathbf{x})}[f(\mathbf{y},\mathbf{a})]\right)^2\right]}_{\text{Bias}} \\
&+ \underbrace{\mathbb{E}_{\mathcal{D}'}\left[\left(g^*_{\mathcal{D}'}(\mathbf{x},\mathbf{a}) - \mathbb{E}_{\mathcal{D}'}[g^*_{\mathcal{D}'}(\mathbf{x},\mathbf{a})]\right)^2\right]}_{\text{Variance}}
\end{aligned} \tag{8}$$

Since the training dataset consists of $\mathbf{x}, \mathbf{y}, \mathbf{a}$, and each $\mathbf{y}$ corresponds to a specific $\mathbf{x}$ from $\mathcal{D}$, we can replace the expectation $\mathbb{E}_{\mathcal{D}'}[\cdot]$ in Eq. 8 with $\mathbb{E}_{\mathbf{x},\mathbf{a}}[\cdot]$ and recover Proposition 1.

□

E PROOF OF PROPOSITION 2

Proposition 2. *It holds for any* $\mathbf{x}$ *and* $\mathbf{a}$*, the function* $g(\mathbf{x}, \mathbf{a})$ *defined by the softmax attention in Eq. 4* $\mathbb{E}_{\hat{p}_{\mathcal{R}}(\mathbf{y}|\mathbf{x})}[f(\mathbf{y}, \mathbf{a})] = g(\mathbf{x}, \mathbf{a})$*. Here,* $\hat{p}_{\mathcal{R}}(\mathbf{y}|\mathbf{x})$ *is a parameterization restriction of* $p(\mathbf{y}|\mathbf{x})$.

Proof. In order to ensure that $\hat{p}_{\mathcal{R}}(\mathbf{y}|\mathbf{x})$ is a valid parameterization of $p(\mathbf{y}|\mathbf{x})$, we define it as a conditional kernel density estimator (KDE) as follows,

$$\hat{p}_{\mathcal{R}}(\mathbf{y}|\mathbf{x}) = \frac{\sum_{s=1}^{S} \mathcal{R}_{\mathbf{x}}(\mathbf{k}_s, \mathbf{q}(\mathbf{x}))\mathcal{R}_{\mathbf{y}}(\mathbf{y}_s, \mathbf{y})}{\sum_{s=1}^{S} \mathcal{R}_{\mathbf{x}}(\mathbf{k}_s, \mathbf{q}(\mathbf{x}))}, \tag{9}$$

Then, we can obtain

$$\begin{aligned}
\mathbb{E}_{\hat{p}_{\mathcal{R}}(\mathbf{y}|\mathbf{x})}[f(\mathbf{y}, \mathbf{a})] &= \int \frac{\sum_{s=1}^{S} \mathcal{R}_{\mathbf{x}}(\mathbf{k}_s, \mathbf{q}(\mathbf{x}))\mathcal{R}_{\mathbf{y}}(\mathbf{y}_s, \mathbf{y})}{\sum_{s=1}^{S} \mathcal{R}_{\mathbf{x}}(\mathbf{k}_s, \mathbf{q}(\mathbf{x}))} f(\mathbf{y}, \mathbf{a})\mathrm{d}\mathbf{y} \\
&= \frac{\sum_{s=1}^{S} \mathcal{R}_{\mathbf{x}}(\mathbf{k}_s, \mathbf{q}) \int \mathcal{R}_{\mathbf{y}}(\mathbf{y}_s, \mathbf{y}) f(\mathbf{y}, \mathbf{a})\mathrm{d}\mathbf{y}}{\sum_{s=1}^{S} \mathcal{R}_{\mathbf{x}}(\mathbf{k}_s, \mathbf{q}(\mathbf{x}))} \\
&= \frac{\sum_{s=1}^{S} \mathcal{R}_{\mathbf{x}}(\mathbf{k}_s, \mathbf{q}) \int \mathcal{R}_{\mathbf{z}}(f(\mathbf{y}_s, \mathbf{a}), \mathbf{z})\mathbf{z}\mathrm{d}\mathbf{z}}{\sum_{s=1}^{S} \mathcal{R}_{\mathbf{x}}(\mathbf{k}_s, \mathbf{q}(\mathbf{x}))} \\
&= \frac{\sum_{s=1}^{S} \mathcal{R}_{\mathbf{x}}(\mathbf{k}_s, \mathbf{q}) f(\mathbf{y}_s, \mathbf{a})}{\sum_{s=1}^{S} \mathcal{R}_{\mathbf{x}}(\mathbf{k}_s, \mathbf{q}(\mathbf{x}))}
\end{aligned} \tag{10}$$

The second last equation comes from the result of the change of variable by setting $\mathbf{z} = f(\mathbf{y}, \mathbf{a})$. The last equation comes from the assumption that $\mathcal{R}_{\mathbf{z}}(\mathbf{z}_s, \mathbf{z})$ is symmetric.

When $\mathcal{R}_{\mathbf{x}}(\mathbf{k}, \mathbf{q})$ is an exponential kernel. *i.e.*, $\mathcal{R}_{\mathbf{x}}(\mathbf{k}, \mathbf{q}) = \exp(\frac{\mathbf{q}^\top \mathbf{k}}{\sqrt{d}})$, we can obtain

$$\begin{aligned}
\mathbb{E}_{\hat{p}_{\mathcal{R}}(\mathbf{y}|\mathbf{x})}[f(\mathbf{y}, \mathbf{a})] &= \frac{\sum_{s=1}^{S} \mathcal{R}_{\mathbf{x}}(\mathbf{k}_s, \mathbf{q}(\mathbf{x})) f(\mathbf{y}_s, \mathbf{a})}{\sum_{s=1}^{S} \mathcal{R}_{\mathbf{x}}(\mathbf{k}_s, \mathbf{q}(\mathbf{x}))} \\
&= \frac{\sum_{s=1}^{S} \exp\left(\frac{\mathbf{q}(\mathbf{x})^\top \mathbf{k}_s}{\sqrt{d}}\right) f(\mathbf{y}_s, \mathbf{a})}{\sum_{s=1}^{S} \exp\left(\frac{\mathbf{q}(\mathbf{x})^\top \mathbf{k}_s}{\sqrt{d}}\right)} \\
&= \text{Softmax}\left(\left[\frac{\mathbf{q}(\mathbf{x})^\top \mathbf{k}_1}{\sqrt{d}}, \cdots, \frac{\mathbf{q}(\mathbf{x})^\top \mathbf{k}_S}{\sqrt{d}}\right]\right)^\top \\
&\quad [f(\mathbf{v}_1, \mathbf{a}), \cdots, f(\mathbf{v}_S, \mathbf{a})] \\
&= g(\mathbf{x}, \mathbf{a}).
\end{aligned} \tag{11}$$

The second last equation comes from the definition of the softmax function and replacing the notation $\mathbf{y}$ with $\mathbf{v}$ which is commonly used in the existing literature.

□

F EXPERIMENTAL DETAILS

F.1 COMPUTING INFRASTRUCTURE

System: Ubuntu 18.04.6 LTS; Python 3.9; Pytorch 1.11. CPU: Intel(R) Xeon(R) Silver 4214 CPU @ 2.20GHz. GPU: GeForce GTX 2080 Ti.

F.2 SYNTHETIC DATA

Data generation process: We generate the synthetic dataset following a mixture of three Gaussians:

$$\begin{aligned} &\mathbf{x} \sim \mathcal{U}^2[-1,1], \\ &\mathbf{y} \sim 0.3\mathcal{N}(\mathbf{A}_1\mathbf{x}, 0.1\cdot\mathbf{I}) + 0.3\mathcal{N}(\mathbf{A}_2\mathbf{x}, 0.1\cdot\mathbf{I}) + 0.4\mathcal{N}(\mathbf{A}_3\mathbf{x}, 0.1\cdot\mathbf{I}), \end{aligned} \tag{12}$$

where the elements of $\mathbf{A}_1, \mathbf{A}_2, \mathbf{A}_3 \in \mathbb{R}^{2\times 2}$ are uniformly sampled from $\mathcal{U}[0,1]$.

We generate 5000 $(\mathbf{x}, \mathbf{y})$ pairs, randomly dividing them into a training set (70%, 3500 pairs), and equal validation and testing sets (15% each, 750 pairs).

Optimization objective: We consider both the convex and non-convex objectives.

Convex objective:

$$\begin{aligned} &\text{minimize}_{\mathbf{a}\in\mathbb{R}^2}\mathbb{E}_{p(\mathbf{y}|\mathbf{x})}\left[\sum_{i=1}^{2}\left(5(\mathbf{y}[i]-\mathbf{a}[i])_+ + 20(\mathbf{a}[i]-\mathbf{y}[i])_+ + 0.5(\mathbf{y}[i]-\mathbf{a}[i])_+^2 + 0.2(\mathbf{a}[i]-\mathbf{y}[i])_+^2\right)\right] \\ &\text{subject to} \quad -1 \le \mathbf{a}[i] \le 1, \forall i. \end{aligned}$$

Non-convex objective:

$$\begin{aligned} &\text{minimize}_{\mathbf{a}\in\mathbb{R}^2}\mathbb{E}_{p(\mathbf{y}|\mathbf{x})}\sum_{i=1}^{2}\left[10(\mathbf{y}[i]-\mathbf{a}[i])_+^2 + 2(\mathbf{a}[i]-\mathbf{y}[i])_+^2 + 4\mathbf{a}[i]^3\right] \\ &\text{subject to} \quad -2 \le \mathbf{a}[i] \le 2, \forall i, \end{aligned}$$

where $(v)_+$ denote $\max\{v, 0\}$.

Solver at test time: At test time, for a fair comparison, we use the same optimization solver for all the methods. Specifically, we use projected gradient descent and the gradient update step adopts the Adam Kingma and Ba [2015a] optimizer. The learning rate is 0.01 and we repeat 500 iterations. We empirically found that this solver solves this optimization problem very well.

Model Hyperparameters: For the two-stage model, DFL, LODL and SO-EBM, the forecaster uses GMM with a different number of components and use 100 samples to estimate the expectation of the objective as we found that more samples bring little performance gain. The forecaster uses a neural network with one hidden layer as the feature extractor which is further stacked by a linear layer. This network has a hidden size of 128, employing ReLU as the nonlinear activation function. The forecaster outputs the mean, log variance, and weight for each GMM component. During training, we sample from the GMM using the Gumbel softmax trick Jang et al. [2017] to make the sampling process differentiable. SO-EBM draws 512 samples from the proposal distribution to estimate the gradient of the model parameters. The proposal distribution is a mixture of Gaussians with 3 components where the variances are $\{0.01, 0.02, 0.05\}$.

For a fair comparison, DF^2 uses the same feature extractor for the encoder. The attention architecture uses 1000 attention points for both the convex and non-convex objectives. During training, DF^2 samples 100 actions $\mathbf{a}$ uniformly from the constrained space, *i.e.*, the box, for each $(\mathbf{x}, \mathbf{y})$ pair at each iteration for function fitting.

Model Optimization: We use the Adam Kingma and Ba [2015a] algorithm for model optimization. The number of training epochs is 50. The learning rate for all the methods is 10^{-3}. DFL, LODL and SO-EBM use the two-stage model as the pre-trained model for faster training convergence.

F.3 WIND POWER BIDDING

Optimization objective: In this task, a wind power firm engages in both energy and reserve markets, given the generated wind power $\mathbf{x} \in \mathbb{R}^{24}$ in the last 24 hours. The firm needs to decide the energy quantity $\mathbf{a}_E \in \mathbb{R}^{12}$ to bid and quantity $\mathbf{a}_R \in \mathbb{R}^{12}$ to reserve over the upcoming 12th-24th hours in advance, based on the forecasted wind power $\mathbf{y} \in \mathbb{R}^{12}$. The optimization objective is to maximize the profit which is a piecewise function consisting of three segments Cao et al. [2020b], Sanayha and Vateekul [2022b]:

$$\text{maximize}_{\mathbf{a}_E \in \mathbb{R}^{12}, \mathbf{a}_R \in \mathbb{R}^{12}} \mathbb{E}_{p(\mathbf{y}|\mathbf{x})} \sum_{i=1}^{12} P\mathbf{y}[i] - \nu \mathbf{a}_R[i]$$

$$+ \begin{cases} -\Delta P_{\text{up},1}(\mathbf{a}_E[i] - \mathbf{a}_R[i] - \mathbf{y}[i]) - \Delta P_{\text{up},2}(\mathbf{a}_E[i] - \mathbf{a}_R[i] - \mathbf{y}[i])^2 \\ -\mu \mathbf{a}_R[i] - F, \text{if } \mathbf{y}[i] < \mathbf{a}_E[i] - \mathbf{a}_R[i] \\ -\mu(\mathbf{a}_E[i] - \mathbf{y}[i]), \text{if } \mathbf{a}_E[i] - \mathbf{a}_R[i] \leq \mathbf{y}[i] \leq \mathbf{a}_E[i] \\ -\Delta P_{\text{down}}(\mathbf{y}[i] - \mathbf{a}_E[i]), \text{if } \mathbf{y}[i] > \mathbf{a}_E[i] \end{cases}$$

$$\text{subject to} \quad E_{\min} \leq \mathbf{a}_E[i] \leq E_{\max}, R_{\min} \leq \mathbf{a}_R[i] \leq R_{\max}, \quad \forall i.$$

P is the regular price of the wind energy sold, $\mathbf{y}[i]$ is the energy generated during period i, $\mathbf{a}_E[i]$ and $\mathbf{a}_R[i]$ are the bid and up reserve energy volumes for period i, respectively. ν corresponds to the opportunity cost when the company participates in the reserve markets, and μ is the deploy price of the reserved energy. This structure encapsulates three market participation scenarios. In the scenario where $\mathbf{y}[i] < \mathbf{a}_E[i] - \mathbf{a}_R[i]$, the company overbids, consequently deploying all reserved energy and facing a linear overbidding penalty, a quadratic overbidding penalty and a constant penalty determined by coefficients $\Delta P_{\text{up},1}$, $\Delta P_{\text{up},2}$, and F. If $\mathbf{a}_E[i] - \mathbf{a}_R[i] \leq \mathbf{y}[i] \leq \mathbf{a}_E[i]$, the company meets its bid by deploying reserve market energy, thereby avoiding penalties. In this case, the company only needs to pay the deployment fee for the reserved energy. However, when $\mathbf{y}[i] > \mathbf{a}_E[i]$, the company underbids, resulting in the selling of surplus electricity at a discount and incurring losses defined by the coefficient ΔP_{down}. We set P as 100, according to the average bidding price obtained from Nord Pool, a European power exchange. ν and μ are 20 and 110 respectively, as a general setting Cao et al. [2020b], Sanayha and Vateekul [2022b]. The value of $\Delta P_{\text{up},1}$, $\Delta P_{\text{up},2}$, ΔP_{down} and F are set to 200, 100, 20 and 10, to ensure an effective penalty. $E_{\min} = 0$, $R_{\min} = 0.15$, and $E_{\max} = R_{\max} = 4$.

According to the optimality condition, the optimal $\mathbf{a}_R[i]$ is always equal to $R_{\min}$ for all i. Therefore, we only need to determine the decision variable $\mathbf{a}_E$.

We use the wind power generation dataset of the German energy company TenneT during 08/23/2019 to 09/22/2020 [1]. The split ratio of the training dataset, validation dataset, and test datset are 64%, 16%, 20%, respectively.

Solver at test time: At test time, for a fair comparison, we use the same optimization solver for all the methods. Specifically, we use projected gradient descent and the gradient update step adopts the Adam Kingma and Ba [2015b] optimizer. The learning rate is 0.1 and we repeat 500 iterations. We empirically found that this solver solves this optimization problem very well.

Model Hyperparameters: For the two-stage model, DFL and SO-EBM, the forecaster uses GMM with a different number of components and use 100 samples to estimate the expectation of the objective as we found that more samples bring little performance gain. The forecaster uses a two-layer long short-term memory network (LSTM) as the feature extractor which is further stacked by a linear layer. The network has a hidden size of 256. It takes the historical wind power in the last 24 hours as input features and outputs the forecasted wind power for the 12th to 24th hours in the future. The forecaster outputs the mean, log variance, and weight for each GMM component. During training, we sample from the GMM using the Gumbel softmax trick Jang et al. [2016] to make the sampling process differentiable. SO-EBM draws 512 samples from the proposal distribution to estimate the gradient of the model parameters. The proposal distribution is a mixture of Gaussians with 3 components where the variances are $\{0.02, 0.05, 0.1\}$.

For a fair comparison, DF^2 uses the same LSTM architecture as the encoder and 500 attention points. During training, we sample 100 actions $\mathbf{a}$ uniformly from the constrained space for each $(\mathbf{x}, \mathbf{y})$ pair at each iteration.

Model Optimization: We use the Adam Kingma and Ba [2015b] algorithm for model optimization. The number of training epochs is 200. The learning rate for all the methods is 10^{-3}. DFL and SO-EBM use the two-stage model as the pre-trained model for faster training convergence.

F.4 COVID-19 VACCINE DISTRIBUTION

Optimization objective: In this task, given the OD matrices $\mathbf{x} \in \mathbb{R}^{47\times47\times7}$ of last week, *i.e.*, $\mathbf{x}[i,j,t]$ represents the number of people move from region i to j on day t, we need to decide the vaccine distribution $\mathbf{a} \in \mathbb{R}^{47}$ across the 47 regions in Japan with a budget constraint ($\mathbf{a}[i]$ is the number of vaccines distributed to the region i). The optimization objective

[1]The dataset is available at: `https://www.kaggle.com/datasets/jorgesandoval/wind-power-generation?select=TransnetBW.csv`

is to minimize the total number of infected people over the ODE-drived dynamics, based on the forecasted OD matrices $\mathbf{y} \in \mathbb{R}^{47\times 47\times 7}$ for the next week.

We want to distribute the vaccine over each county to minimize the number of infected cases. The number of infected cases is given by a metapopulation SEIRV model Li et al. [2020b], Pei et al. [2020b], denoted by Simulator$(\cdot,\cdot)$:

$$\begin{aligned} &\arg\min_{\mathbf{a}\in\mathbb{R}^{47}} \mathbb{E}_{p(\mathbf{y}|\mathbf{x})}[\text{Simulator}(\mathbf{y},\mathbf{a})], \\ &\text{Subject to} \quad \sum_i \mathbf{a}[i] \leq \text{Budget}, \mathbf{a}[i] \geq 0. \end{aligned}$$

We use the OD matrices dataset of Japan [2] during 04/01/2020 to 02/28/2021. The split ratio of the training dataset, validation dataset, and test datset are 64%, 16%, 20%, respectively. We set the budget as 5×10^6.

Details of the simulator: The SEIRV model is an epidemiological model used to predict and understand the spread of infectious diseases. It divides the population into five compartments: Susceptible (S), Exposed (E), Infectious (I), Recovered (R) and Vaccined (V). The model is defined by a set of differential equations that describe the transitions between these compartments. There are four hyperparameters in the SEIRV model:

- β - Transmission rate: Represents the average number of contacts per person per unit of time multiplied by the probability of disease transmission in a contact between a susceptible and an infectious individual.
- σ - Latent rate (or the inverse of the incubation period): The rate at which exposed individuals progress to the infectious state. The incubation period is the time it takes for an individual to become infectious after exposure.
- γ - Recovery rate (or the inverse of the infectious period): The rate at which infectious individuals recover or die and transition to the recovered state. The infectious period is the time during which an infected individual can transmit the disease.
- N - Total population: The sum of individuals in all compartments (S, E, I, R, V).

When considering mobility flow among different regions, we need to adapt the SEIRV model to account for the movement of individuals between regions. In this case, the model becomes a spatially explicit, multi-region SEIRV model. Each region will have its own SEIRV model, and the flow of individuals between regions will affect the dynamics of the compartments. Specifically, for each region $k = 1, \cdots, K$, we have:

$$\begin{aligned} \frac{\mathrm{d}\mathbf{S}[k]}{\mathrm{d}t} &= -\boldsymbol{\beta}[k]\frac{\mathbf{S}[k]\cdot\mathbf{I}[k]}{\mathbf{N}[k]} - \frac{\mathbf{S}[k]}{\mathbf{S}[k]+\mathbf{E}[k]}\cdot\frac{\mathbf{a}[k]}{T} \\ &\quad + \sum_{i\neq k}\tilde{\mathbf{y}}[i,k,t]\cdot\mathbf{S}[i] - \sum_{j\neq k}\tilde{\mathbf{y}}[k,j,t]\cdot\mathbf{S}[k], \\ \frac{\mathrm{d}\mathbf{E}[k]}{\mathrm{d}t} &= \boldsymbol{\beta}[k]\frac{\mathbf{S}[k]\cdot\mathbf{I}[k]}{\mathbf{N}[k]} - \boldsymbol{\sigma}[k]\cdot\mathbf{E}[k] - \frac{\mathbf{E}[k]}{\mathbf{S}[k]+\mathbf{E}[k]}\cdot\frac{\mathbf{a}[k]}{T} \\ &\quad + \sum_{i\neq k}\tilde{\mathbf{y}}[i,k,t]\cdot\mathbf{E}[i] - \sum_{j\neq k}\tilde{\mathbf{y}}[k,j,t]\cdot\mathbf{E}[k], \\ \frac{\mathrm{d}\mathbf{I}[k]}{\mathrm{d}t} &= \boldsymbol{\sigma}[k]\cdot\mathbf{E}[k] - \boldsymbol{\gamma}[k]\cdot\mathbf{I}[k] \\ &\quad + \sum_{i\neq k}\tilde{\mathbf{y}}[i,k,t]\cdot\mathbf{I}[i] - \sum_{j\neq k}\tilde{\mathbf{y}}[k,j,t]\cdot\mathbf{I}[k], \\ \frac{\mathrm{d}\mathbf{R}[k]}{\mathrm{d}t} &= \boldsymbol{\gamma}[k]\cdot\mathbf{I}[k] + \sum_{i\neq k}\tilde{\mathbf{y}}[i,k,t]\cdot\mathbf{R}[i] - \sum_{j\neq k}\tilde{\mathbf{y}}[k,j,t]\cdot\mathbf{R}[k], \\ \frac{\mathrm{d}\mathbf{V}[k]}{\mathrm{d}t} &= \frac{\mathbf{a}[k]}{T} + \sum_{i\neq k}\tilde{\mathbf{y}}[i,k,t]\cdot\mathbf{V}[i] - \sum_{j\neq k}\tilde{\mathbf{y}}[k,j,t]\cdot\mathbf{V}[k], \end{aligned} \tag{13}$$

where $\boldsymbol{\beta}[k]$, $\boldsymbol{\gamma}[k]$, and $\boldsymbol{\sigma}[k]$ are hyper-parameter for region k. These hyperparameters are fitted on the dataset using maximum likelihood estimation. $\tilde{\mathbf{y}}$ is the normalized OD matrix.

[2] The dataset is available at `https://github.com/deepkashiwa20/ODCRN/tree/main/data`

Finally, the simulator will output the total number of newly infected people across all the regions and we aim to minimize this value.

Solver at test time: At test time, for a fair comparison, we use the same optimization solver for all the methods. Specifically, we use mirror descent Beck and Teboulle [2003] so that the updated decision variable will still variable satisfy the constraints. Specifically, the update rule takes the following form at t-th iteration:

$$\mathbf{a}_{t+1}[i] = \text{Budget} \cdot \frac{\mathbf{a}_t[i]\exp(-\gamma \nabla_i f(\mathbf{a}_t))}{\sum_{j=1}^{n} \mathbf{a}_t[i]\exp(-\gamma \nabla_j f(\mathbf{a}_t))}, \tag{14}$$

where γ is the learning rate. We set the learning rate as 0.01 and repeat 500 iterations. We empirically found that this solver solves this optimization problem very well.

Model Hyperparameters: For the two-stage model, DFL, LODL and SO-EBM, the forecaster uses GMM with a different number of components and use 100 samples to estimate the expectation of the objective as we found that more samples bring little performance gain. The forecaster is a DC-RNN Li et al. [2018] which adopts an encoder-decoder architecture. The encoder and decoder both have two hidden layers with a hidden size of 128. The forecaster takes the OD matrices of last week as input features and predicts the OD matrices of next week. The forecaster outputs the mean, log variance, and weight for each GMM component. During training, we sample from the GMM using the Gumbel softmax trick Jang et al. [2017] to make the sampling process differentiable. Since the decision variable is a simplex, we train SO-EBM with projected Langevin dynamics. Specifically, at each iteration of the Langevin dynamics, we project the decision variable into the simplex. The number of iterations of the Langevin dynamics is 100 and the step size is 0.05.

For a fair comparison, DF^2 employs the same encoder as the DC-RNN architecture and uses 100 attention points. During training, DF^2 samples 100 actions $\mathbf{a}$ uniformly from the constrained space, *i.e.*, the simplex, for each $(\mathbf{x}, \mathbf{y})$ pair at each iteration for function fitting. To uniformly sample from the simplex, we sample from the Dirichlet distribution where all parameters are 1.

Model Optimization: We use the Adam Kingma and Ba [2015a] algorithm for model optimization. The number of training epochs is 50. The learning rate for all the methods is 10^{-4}. DFL, LODL and SO-EBM use the two-stage model as the pre-trained model for faster training convergence.

F.5 INVENTORY OPTIMIZATION

Optimization objective In this task, a department store is tasked with predicting the sales $\mathbf{y} \in \mathbb{R}^7$ for the upcoming 7th-14th days based on the past 14 days' sales data $\mathbf{x} \in \mathbb{R}^{14}$ for a specific product, and accordingly, determining the best replenishment strategy $\mathbf{a} \in \mathbb{R}^7$ for each day. The optimization objective is a combination of an under-purchasing penalty, an over-purchasing penalty, and a squared loss between supplies and demands:

$$\begin{aligned}\text{minimize}_{\mathbf{a}\in\mathbb{R}^7} \mathbb{E}_{p(\mathbf{y}|\mathbf{x})} \sum_{i=1}^{7} & [20(\mathbf{y}[i] - \mathbf{a}[i])_+ + 5(\mathbf{a}[i] - \mathbf{y}[i])_+ \\ & + (\mathbf{a}[i] - \mathbf{y}[i])^2] \\ & \text{subject to} \quad 0 \le \mathbf{a}[i] \le 3, \forall i,\end{aligned}$$

where $(v)_+$ denote $\max\{v, 0\}$.

Solver at test time: At test time, for a fair comparison, we use the same optimization solver for all the methods. Specifically, we use projected gradient descent and the gradient update step adopts the Adam Kingma and Ba [2015a] optimizer. The learning rate is 0.1 and we repeat 500 iterations. We empirically found that this solver solves this optimization problem very well.

Model Hyperparameters: The forecaster of the two-stage model, DFL, LODL and SO-EBM uses a two-layer long short-term memory network (LSTM) Hochreiter and Schmidhuber [1997] as a feature extractor which is further stacked by a linear layer. The forecaster takes the historical item sales in the last 14 days as input features and outputs the forecasted item sales for the 7th to 14th days in the future. The network has a hidden size of 128. SO-EBM draws 512 samples from the proposal distribution to estimate the gradient of the model parameters. The proposal distribution is a mixture of Gaussians with 3 components where the variances are $\{0.05, 0.1, 0.2\}$.

For a fair comparison, DF^2 uses the same LSTM architecture as the encoder and 230 attention points. During training, the two-stage model, DFL, LODL and SO-EBM use 100 samples to estimate the expected objective as more samples provide little performance gain.

Model Optimization: We use the Adam Kingma and Ba [2015a] algorithm for model optimization. The number of training epochs is 200. The learning rate for all the methods is 10^{-3}. DFL, LODL and SO-EBM use the two-stage model as the pre-trained model for faster training convergence.