Accounting for Heterogeneous Parameters in Decision-Focused Learning

Anonymous authors
Paper under double-blind review

Abstract

Decision-focused learning (DFL) is a recent machine learning paradigm aimed at tackling predict-then-optimize problems, where the task is to predict the parameter values of a parametric optimization problem from features. Instead of maximizing predictive accuracy, DFL maximizes downstream decision quality, by training the model to avoid specifically those errors that most negatively impact decision-making. In this work, we systematically investigate an understudied aspect of DFL: that for different parameters, the way prediction errors affect downstream decision quality may differ, and that the conventional model architecture cannot account for these differences. We first formalize this issue and provide a theoretical characterization of when it arises. We then show that significantly better decision quality can often be achieved by equipping models with the ability to learn parameter-specific predictive mappings – even when the true underlying mappings are identical. To this end, we investigate three architectural alterations to the predictive model, and propose a data augmentation scheme to enhance data efficiency. We extensively evaluate the impact of these changes across several dimensions, using linear and nonlinear predictive models, and optimization problems of different complexities. Our findings show that significant performance gains can be realized through such architectural alterations and data augmentation scheme, across different problem types.

1 Introduction

Decision-making processes are commonly modeled as constrained optimization problems, which are solved by optimizing an objective function, subject to constraints. However, in many applications, some of the problem's parameters are not known at solving time. For example, a delivery company must make its high-level routing decisions with respect to uncertain future traffic conditions. These traffic conditions must be predicted before a solution can be computed.

Such problems can be framed as *predict-then-optimize problems* (Mandi et al., 2024), consisting of a prediction and an optimization stage. In the prediction stage, a machine learning model is used to predict values for the unknown parameters from known contextual features. In the optimization stage, the resulting optimization problem is solved. The quality of the produced solutions thus hinges on the predictive model used in the first stage, making its training procedure highly important. Two paradigms can be distinguished.

In *prediction-focused learning* (PFL), the predictive model is trained to maximize the accuracy of the predictions, using traditional losses like the mean squared error or negative log-likelihood. Though simple and efficient, this generally leads to suboptimal decision-making. This is because, while perfect predictions lead to perfect decisions, different kinds of imperfect predictions affect downstream decision-making in different ways (Shah et al., 2024).

In *decision-focused learning* (DFL), the predictive model is instead trained with a loss function that reflects downstream decision quality. In other words, DFL trains the model to make predictions that maximize the quality of the solutions obtained after the optimization stage. As numerous works have shown, models trained with DFL generally lead to better decisions than models trained with PFL (Amos & Kolter, 2017;

Donti et al., 2017; Elmachtoub & Grigas, 2022; Mandi et al., 2022; Mulamba et al., 2021; Pogančić et al., 2020; Wilder et al., 2019).

However, while the loss functions used in DFL are designed to maximize decision quality, little attention is paid to model architecture. In existing work, it is typically chosen based on whether the different parameters of the optimization problem depend on their features in the same way or not. In most works, each parameter depends on its contextual features in the same way. Consequently, a single predictive model is *shared* across the parameters, and each parameter's value is produced independently through a separate invocation of this model (Shah et al., 2022; 2024; Wilder et al., 2019; Mandi et al., 2022; Mulamba et al., 2021; Mandi & Guns, 2020; Ferber et al., 2020). In rare cases where each parameter depends on features in a distinct way, a *separate* model is trained for each parameter instead, such as in the synthetic benchmark used in (Elmachtoub & Grigas, 2022) and (Tang & Khalil, 2022a).

In this work, we show that choosing the architecture based solely on whether the parameters share or differ in their dependence on features, is suboptimal. We demonstrate that *even when all parameters depend on their features in exactly the same way*, superior decision quality can often be achieved by learning parameter-specific mappings. This stems from the fact that, often, the parameters influence decision-making in different ways.

Consider, for example, a knapsack problem where the item values are parameterized and must be predicted, while the item weights are known. In this setting, the exact same error in the prediction of an item's value can affect the optimal decision differently depending on whether it is made on a high-weight or a low-weight item. For instance, overestimating the value of a heavy item might lead it to be selected at the expense of multiple lighter, more valuable items, reducing decision quality. Conversely, the same error, made in the value prediction of a light item, might not affect decision quality at all, because the item would be selected either way. In this example, the impact of prediction errors on downstream decision quality varies across the parameters. We refer to such parameters as *heterogeneous*.

Because DFL accounts for how prediction errors impact decision quality, and because this impact differs between heterogeneous parameters, our study shows that a model that can predict in distinct ways for heterogeneous parameters is better equipped to maximize decision quality than a model that cannot. We contribute the following:

1. We show that in predict-then-optimize problems, prediction errors can influence downstream decision quality in distinct ways for different parameters. We formalize this phenomenon, and theoretically characterize when and why it occurs.
2. We investigate three alterations to the model architecture that allow it to learn parameter-specific mappings. To improve data efficiency, we also propose a form of data augmentation that permutes the parameters at training time.
3. We conduct an extensive experimental evaluation across several dimensions, using linear and non-linear predictive models, on optimization problems of varying complexity. Our results show that the ability to learn parameter-specific mappings can significantly improve decision quality. They also highlight the effectiveness of the data augmentation scheme, especially in the low-data regime, or when the problem contains many parameters.

2 Background

The DFL paradigm aims to solve predict-then-optimize problems, which involve the prediction of unknown objective parameters y of a parametric constrained optimization problem of the following form:

$$z^\star(y) = \arg\min_z \; f(z; y) \;\; \text{s.t.} \;\; z \in \Omega \tag{1}$$

For any given parameter vector $y \in \mathbb{R}^n$, solving the optimization problem involves finding a solution $z^\star(y) \in \mathbb{R}^m$ in non-empty feasible space Ω that minimizes objective function $f(z; y)$.

We assume that the unknown objective parameters $y = [y_1\ y_2\ \dots\ y_n]$ are correlated with known feature vectors $x = [x_1\ x_2\ \dots\ x_n]$ according to some distribution P, wherein each parameter $y_i \in \mathbb{R}$ depends on its features $x_i \in \mathbb{R}^f$ according to the same underlying mapping. Though distribution P is not known, we assume access to a training set of examples $D = \{(x^{(i)}, y^{(i)})\}_{i=1}^{d}$, drawn from P, where each $x^{(i)}$ is a vector of feature vectors and each $y^{(i)}$ is the vector of corresponding parameters.

This training set is used to train a model m_ω with learnable weights ω – usually a linear regressor or a neural network – which makes predictions $\hat{y} = [m_\omega(x_1)\ m_\omega(x_2)\ \dots\ m_\omega(x_n)]$. With slight abuse of notation, we will simply denote this as $\hat{y} = m_\omega(x)$.

Unlike conventional regression, the objective in training is *not* to optimize the accuracy of $\hat{y}$. Rather, one aims to make predictions $\hat{y}$ that lead to good decisions $z^\star(\hat{y})$ with respect to the real parameters y. This is measured by the *regret*, which expresses the suboptimality of the made decisions $z^\star(\hat{y})$ with respect to the ground-truth parameters y.

$$\textit{Regret}(\hat{y}, y) = f(z^\star(\hat{y}); y) - f(z^\star(y); y) \tag{2}$$

Thus, the aim is to minimize the *expected regret*:

$$\min_\omega \mathbb{E}_{x,y\sim P}[\textit{Regret}(m_\omega(x), y)]. \tag{3}$$

In practice, since distribution P is unknown, the expected regret is approximated using the training set D via empirical risk minimization (ERM):

$$\min_\omega \ \frac{1}{d}\sum_{i=1}^{d} \textit{Regret}\big(m_\omega(x^{(i)}),\, y^{(i)}\big). \tag{4}$$

3 Related work

The aim of DFL is to train predictive models that maximize downstream decision quality, rather than predictive accuracy. Methodologically, it is closely related to differentiable optimization, which offers a way of differentiating through any kind of continuous convex optimization problem (Amos & Kolter, 2017; Agrawal et al., 2019).

Most work in DFL, however, focuses on *combinatorial* optimization problems, which pose an additional challenge due to their discrete nature, making the gradient of the regret zero almost everywhere. Existing work has circumvented this obstacle in different ways. A first approach is based on analytical smoothing of the optimization problem, leading to non-zero gradients of the regret (Wilder et al., 2019; Mandi & Guns, 2020). Another approach focuses on computing informative weight updates by perturbing the predicted parameters (Pogančić et al., 2020; Berthet et al., 2020; Niepert et al., 2021). Finally, surrogate loss functions can be used, like the seminal SPO+ loss (Elmachtoub & Grigas, 2022), various learnable losses (Shah et al., 2022; 2024; Bansal et al., 2024), losses based on noise-contrastive estimation (Mulamba et al., 2021) or learning to rank (Mandi et al., 2022), and losses based on the geometry of the feasible space (Tang & Khalil, 2024; Berden et al., 2025). For a comprehensive overview, we refer to (Mandi et al., 2024).

Another direction within DFL has instead focused on improving scalability. Because DFL involves repeatedly solving optimization problems, training can be significantly slower than in PFL. To improve scalability, prior work has investigated the use of problem relaxations and warm-starting techniques Mandi et al. (2020), computationally cheaper solution caches in place of exact solvers Mulamba et al. (2021), and surrogate losses that bypass expensive optimization entirely Tang & Khalil (2024); Berden et al. (2025).

As highlighted in this section, existing work has primarily focused on creating learning signals and improving scalability. The focus of our work is quite different. We systematically study the fact that for different parameters, prediction errors can affect downstream decision quality in different ways, and how this motivates alterations to the model's architecture.

4 Heterogeneous parameters

One of the central concepts of this paper is that in predict-then-optimize problems, prediction errors do not always affect downstream decision quality in the same way for the different parameters. Before formalizing this and theoretically characterizing exactly when it happens, we give an illustrative example.

Example 1. Consider a parametric knapsack problem:

$$\begin{aligned} z^\star(y) = \arg\max \;\; & y_1 z_1 + y_2 z_2 + y_3 z_3 \\ \text{s.t.} \;\; & z_1 + z_2 + 2z_3 \leq 2 \\ & z_1, z_2, z_3 \in \{0, 1\} \end{aligned}$$

In this problem, prediction errors on parameters y_1 and y_3 do not affect downstream decision-making in the same way, due to the differing coefficients associated with z_1 and z_3 in the constraint. We call y_1 and y_3 *heterogeneous* parameters. Take, for example, $y = [7\ 4\ 4]$ and $\hat{y} = [9\ 4\ 4]$. Even though value 7 is mispredicted as 9, parameter vector $\hat{y}$ still leads to correct decision $z^\star(\hat{y}) = [1\ 1\ 0]$, and thus to zero regret. However, if this same error were made on the third parameter instead (i.e., for $y = [4\ 4\ 7]$ and $\hat{y} = [4\ 4\ 9]$), then solution $z^\star(\hat{y}) = [0\ 0\ 1]$ would wrongfully put the third item in the knapsack, leading to a regret of 1.

On the other hand, prediction errors on parameters y_1 and y_2 affect downstream decision quality in the same way. We call them *homogeneous* parameters. For any y and $\hat{y}$, the regret remains the same before and after interchanging y_1 and y_2 as well as $\hat{y}_1$ and $\hat{y}_2$.

We now formalize the difference between homogeneous and heterogeneous parameters as follows.

Definition 1. *Let σ_{ij} be the transposition that interchanges the values of parameters y_i and y_j. Then, parameters y_i and y_j are homogeneous if and only if, for all $\hat{y}$ and y, the regret remains invariant under transposition σ_{ij}, i.e., Regret$(\hat{y}, y)$ = Regret$(\sigma_{ij}(\hat{y}), \sigma_{ij}(y))$. Parameters y_i and y_j are heterogeneous when they are not homogeneous.*

This definition expresses that for heterogeneous parameters, the impact of a parameter misprediction $\hat{y}_k$ is not only characterized by the *values* of $\hat{y}_k$ and y_k, but also by k, i.e., by *which* parameter the prediction error is made on.

In Example 1, one could directly identify the homogeneous and heterogeneous parameters. Homogeneous parameters were associated with items of identical weight, while heterogeneous parameters were associated with items of differing weight. However, the distinction is not always as easily observable. It is possible to construct pathological cases in which the parameters appear heterogeneous but are, in fact, homogeneous. For instance, two items whose weights both exceed the capacity have homogeneous objective parameters, even if their weights differ.

This observation motivates the need for a precise understanding, which we now develop. Before presenting our main theoretical result, we introduce the necessary definitions and supporting lemmas.

Definition 2. *A feasible solution $z \in \Omega$ is* supported *if and only if there exists a parameter vector $y \in \mathbb{R}^n$ for which $z = z^\star(y)$.*

The following lemma establishes a structural constraint that any pair of homogeneous parameters must satisfy.

Lemma 1. *Let y_i and y_j be two homogeneous parameters. Then, there do not exist any two parameter vectors y and y' for which $z^\star(y) = z^\star(y')$ but $z^\star(\sigma_{ij}(y)) \neq z^\star(\sigma_{ij}(y'))$.*

We now introduce a transformation on supported solutions that will serve as an important tool in characterizing homogeneity.

Lemma 2. *Let $Z^\star$ be the set of supported solutions. Let $\tau : Z^\star \to Z^\star$ be a transformation over solutions vectors such that, τ is an involution (i.e., for all $z \in Z^\star$, $\tau(\tau(z)) = z$), and:*

$$\forall y \in \mathbb{R}^n \; \exists b \in \mathbb{R} \; \forall z \in Z^\star : f(z; y) = f(\tau(z); \sigma_{ij}(y)) + b$$

Then, it must hold that:

$$\forall y \in \mathbb{R}^n : z^\star(\sigma_{ij}(y)) = \tau(z^\star(y))$$

Proofs of Lemmas 1 and 2 are provided in Appendix A. Together, these two lemmas allow us to derive an exact characterization of when two parameters behave homogeneously.

Two parameters y_i and y_j are homogeneous if and only if, through a transformation τ, the quality of every supported solution z, relative to the optimal solution, is maintained when interchanging y_i and y_j. This means that objective function f is invariant (modulo a constant) to the interchanging of values y_i and y_j, and the simultaneous modification of solution z by transformation τ. We formalize this as follows.

Proposition 3. *Let $Z^\star$ be the set of supported solutions. Then, parameters y_i and y_j are homogeneous if and only if there exists a transformation $\tau : Z^\star \to Z^\star$ over solutions z such that τ is an involution (i.e., $\forall z \in Z^\star : \tau(\tau(z)) = z$), and such that:*

$$\forall y \in \mathbb{R}^n \; \exists b \in \mathbb{R} \; \forall z \in Z^\star : f(z; y) = f(\tau(z); \sigma_{ij}(y)) + b$$

Proof. We prove the proposition by proving both directions of the equivalence separately.

(⇒) We start by assuming that y_i and y_j are homogeneous, and proving that there exists a $\tau : Z^\star \to Z^\star$ such that $\forall z \in Z^\star : \tau(\tau(z)) = z$ and $\forall y \in \mathbb{R}^n \; \exists b \in \mathbb{R} \; \forall z \in Z^\star : f(z; y) + b = f(\tau(z); \sigma_{ij}(y))$.

Since y_i and y_j are homogeneous, the following holds:

$$\begin{aligned}
&\forall \hat{y}, y \in \mathbb{R}^n : \mathit{Regret}(\hat{y}, y) = \mathit{Regret}(\sigma_{ij}(\hat{y}), \sigma_{ij}(y)) \\
\Leftrightarrow\; &\forall \hat{y}, y \in \mathbb{R}^n : f(z^\star(\hat{y}); y) - f(z^\star(y); y) = f(z^\star(\sigma_{ij}(\hat{y})); \sigma_{ij}(y)) - f(z^\star(\sigma_{ij}(y)); \sigma_{ij}(y)) \\
\Leftrightarrow\; &\forall \hat{y}, y \in \mathbb{R}^n : f(z^\star(\hat{y}); y) = f(z^\star(\sigma_{ij}(\hat{y})); \sigma_{ij}(y)) + f(z^\star(y); y) - f(z^\star(\sigma_{ij}(y)); \sigma_{ij}(y)) \qquad (6)
\end{aligned}$$

In the right-hand side of equation 6, the term $f(z^\star(y); y) - f(z^\star(\sigma_{ij}(y)); \sigma_{ij}(y))$ is a constant that depends on y. We introduce symbol b to denote this constant.

$$\forall \hat{y}, y \in \mathbb{R}^n \; \exists b \in \mathbb{R} : f(z^\star(\hat{y}); y) = f(z^\star(\sigma_{ij}(\hat{y})); \sigma_{ij}(y)) + b$$

We now define transformation $\tau : Z^\star \to Z^\star$ as follows: $\tau(z^\star(y)) = z^\star(\sigma_{ij}(y))$. This transformation is only a valid function if each element of its domain is mapped onto a unique element of its codomain. In other words, there may not exist a $\hat{y}$ and y for which $z^\star(\hat{y}) = z^\star(y)$ but $z^\star(\sigma_{ij}(\hat{y})) \neq z^\star(\sigma_{ij}(y))$. Since y_i and y_j are homogeneous, such a $\hat{y}$ and y do not exist (Lemma 1), making τ a valid function. Thus:

$$\exists \tau : Z^\star \to Z^\star, \; \forall \hat{y}, y \in \mathbb{R}^n \; \exists b \in \mathbb{R} : f(z^\star(\hat{y}); y) = f(z^\star(\sigma_{ij}(\hat{y})); \sigma_{ij}(y)) + b$$

Since $Z^\star = \{z \in \Omega : \exists y \in \mathbb{R}^n : z = z^\star(y)\}$, and since b depends only on y (and not on $\hat{y}$):

$$\exists \tau : Z^\star \to Z^\star, \; \forall y \in \mathbb{R}^n \; \exists b \in \mathbb{R} \; \forall z \in Z^\star : f(z; y) = f(\tau(z); \sigma_{ij}(y)) + b$$

Also note that, by construction, τ is an involution, i.e., $\forall z \in Z^\star : \tau(\tau(z)) = z$. This can be shown as follows: $\forall y \in \mathbb{R}^n : \tau(\tau(z^\star(y))) = \tau(z^\star(\sigma_{ij}(y))) = z^\star(\sigma_{ij}(\sigma_{ij}(y))) = z^\star(y)$. This proves the first direction of the equivalence expressed in Proposition 3.

(⇐) We now prove the other direction. In other words, we assume that there exists a $\tau : Z^\star \to Z^\star$ such that $\forall z \in Z^\star : \tau(\tau(z)) = z$ and $\forall y \in \mathbb{R}^n \; \exists b \in \mathbb{R} \; \forall z \in Z^\star : f(z; y) = f(\tau(z); \sigma_{ij}(y)) + b$, and we prove that parameters y_i and y_j are homogeneous.

The assumption expresses that

$$\forall y \in \mathbb{R}^n \; \exists b \in \mathbb{R} \; \forall z \in Z^\star : f(z; y) = f(\tau(z); \sigma_{ij}(y)) + b$$

Since $Z^\star = \{z \in \Omega : \exists y \in \mathbb{R}^n : z = z^\star(y)\}$, it follows that:

$$\forall y, \hat{y} \in \mathbb{R}^n \; \exists b \in \mathbb{R} : f(z^\star(\hat{y}); y) = f(\tau(z^\star(\hat{y})); \sigma_{ij}(y)) + b$$

By Lemma 2, it holds that $\forall y \in \mathbb{R}^n : z^\star(\sigma_{ij}(y)) = \tau(z^\star(y))$, and thus:

$$\forall y, \hat{y} \in \mathbb{R}^n \; \exists b \in \mathbb{R} : f(z^\star(\hat{y}); y) = f(z^\star(\sigma_{ij}(\hat{y})); \sigma_{ij}(y)) + b$$

From this, homogeneity of y_i and y_j follows:

$$\forall y, \hat{y} \in \mathbb{R}^n \; \exists b \in \mathbb{R} : f(z^\star(\hat{y}); y) = f(z^\star(\sigma_{ij}(\hat{y})); \sigma_{ij}(y)) + b \tag{7}$$

$$\Rightarrow \forall y, \hat{y} \in \mathbb{R}^n \; \exists b \in \mathbb{R} : f(z^\star(\hat{y}); y) - f(z^\star(y); y) = f(z^\star(\sigma_{ij}(\hat{y})); \sigma_{ij}(y)) + b - f(z^\star(y); y) \tag{8}$$

$$\Rightarrow \forall y, \hat{y} \in \mathbb{R}^n \; \exists b \in \mathbb{R} : f(z^\star(\hat{y}); y) - f(z^\star(y); y) = f(z^\star(\sigma_{ij}(\hat{y})); \sigma_{ij}(y)) + b - f(\tau(z^\star(y)); \sigma_{ij}(y)) - b \tag{9}$$

$$\Rightarrow \forall y, \hat{y} \in \mathbb{R}^n : f(z^\star(\hat{y}); y) - f(z^\star(y); y) = f(z^\star(\sigma_{ij}(\hat{y})); \sigma_{ij}(y)) - f(z^\star(\sigma_{ij}(y)); \sigma_{ij}(y)) \tag{10}$$

$$\Rightarrow \forall y, \hat{y} \in \mathbb{R}^n : \mathit{Regret}(\hat{y}, y) = \mathit{Regret}(\sigma_{ij}(\hat{y}), \sigma_{ij}(y)) \tag{11}$$

Equation 8 is obtained by subtracting $f(z^\star(\hat{y}); y)$ from both sides of equality 7. Equation 9 follows by applying equation 4 to the last term of equation 8. Equation 10 follows by Lemma 2, and by cancelling out b and $-b$. Finally, equation 11 is obtained, by applying the definition of regret (equation 2).

This proves the second direction of the equivalence, thereby proving the proposition. □

The proposition reveals several important insights. First, it rules out some alternative understandings of the source of homogeneous and heterogeneous parameters. For instance, when the optimal value of the problem is invariant to transposition σ_{ij} (i.e., $\forall y : f(z^\star(y); y) = f(z^\star(\sigma_{ij}(y)); \sigma_{ij})$), this does not directly imply homogeneity of parameters y_i and y_j. Similarly, even if there exists a transformation τ that preserves the optimal solution with respect to σ_{ij} (i.e., $\forall y : z^\star(y) = \tau(z^\star(\sigma_{ij}(y)))$), this still does not imply homogeneity. We elaborate on this discussion in Appendix B, where we also provide additional examples of homogeneous parameters, and the corresponding transformation τ and choices of b.

Second, the proposition shows that two parameters are homogeneous *only* under strict conditions. Many realistic problems instead contain heterogeneous parameters. In Example 1, this heterogeneity was caused by varying item weights in the constraints. Similarly, problems that contain other types of varying quantities – such as demands or capacities – typically contain heterogeneous parameters. Examples include blending problems with varying material restrictions, lot sizing problems with varying startup or holding costs and facility location problems with varying customer demands.

5 Accounting for heterogeneous parameters

In this section, we show that in DFL for problems with heterogeneous parameters, models that can predict in parameter-specific ways have an edge over models that cannot. This is because DFL accounts for how prediction errors impact decision quality, and, as we have shown, this impact differs between heterogeneous parameters. As a result, a model that can predict in different ways for the different parameters is better equipped to make predictions that maximize decision quality.

> **Example 2.** We generate a knapsack problem with 10 items. The total capacity of the knapsack is 5. The item weights are known, and are evenly spaced between 0.5 and 1.5. Each item value y_i is unknown but correlated with a single known feature x_i as follows: $y_i = 10x_i^3 - 6.5x_i + 3$ (taken from (Shah et al., 2022; 2024)). This cubic relationship is approximated using linear regression, resulting in a misspecified model that cannot fully capture the true underlying mapping. As was shown in (Elmachtoub & Grigas, 2022), DFL generally outperforms PFL under such misspecification.

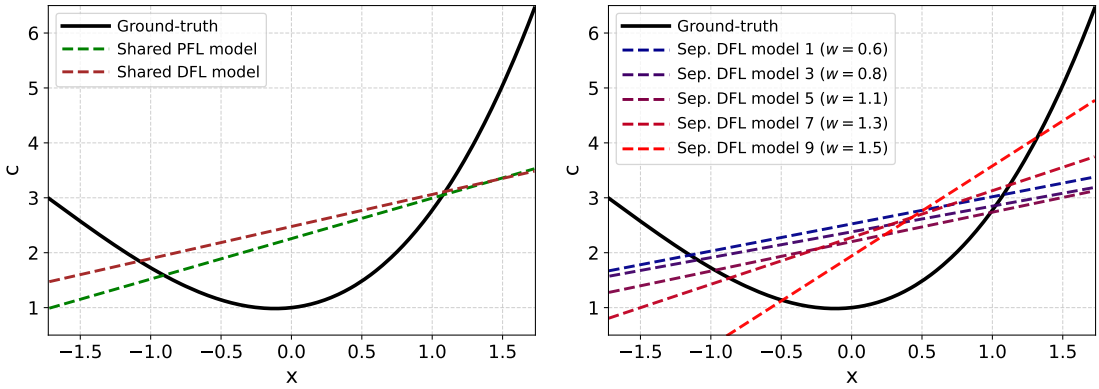

Figure 1: Learned models for a predict-then-optimize knapsack problem. Each item's value y_i is unknown but correlated with a known feature x_i according to a cubic relationship, which is the same for all items, and which is approximated using linear regression. Because the underlying relationship is identical for each parameter, a shared model is typically trained and used across all parameters (left panel). However, when separate parameter-specific models are trained instead (right panel), they learn a distinct mapping for each parameter, whose slopes are correlated with the associated item weights. This occurs because the parameters are heterogeneous (due to differing item weights in the constraints), and results in improved decision quality.

> Conventionally, one would train a single model that is shared across all parameters, because each parameter depends on its feature according to the same cubic mapping. We train such a shared model with PFL and DFL. Additionally, because the parameters are heterogeneous, we use DFL to train an architecture that consists of 10 parameter-specific models.
>
> The learned models are shown in Figure 1. On the left panel, we see that the shared DFL model has learned a slightly different slope and intercept than the PFL model, resulting in marginally improved regret (8.32% vs. 8.99%). On the right panel, we show 5 of the 10 learned parameter-specific models. These models have clearly learned distinct mappings for the different parameters, with the slope of the regressor increasing as the weight of the associated item increases. This has an intuitive explanation: high-weight items should only be chosen in the knapsack solution when they have a high value, so the corresponding model mostly tries to capture the high-value region of the underlying function. This led to significantly better downstream performance, achieving a regret of 6.89%.

In what follows, we introduce three distinct architectural changes that can account for parameter heterogeneity (Definition 1), as well as a data augmentation scheme to improve data efficiency. We design these architectural alterations and the data augmentation scheme under the assumption that each parameter depends on its features according to the same true underlying mapping. This is the most common setting in prior work, and the primary setting of interest to this paper. After all, if each parameter depended on its features in a unique way, the decision to train a separate model per parameter would be the default choice, regardless of whether the parameters are heterogeneous or not.

5.1 Architectures

We will study three distinct architectural adaptations, shown in Figure 2, to learn parameter-specific mappings.

Encoding the parameter index. One approach to allow the model to predict in parameter-specific ways is to include the parameter index as part of the input provided to the model, alongside the contextual features. To avoid introducing an inductive bias that treats nearby parameters more similarly than distant ones, we use a one-hot encoding for the parameter index instead of a direct numerical encoding.

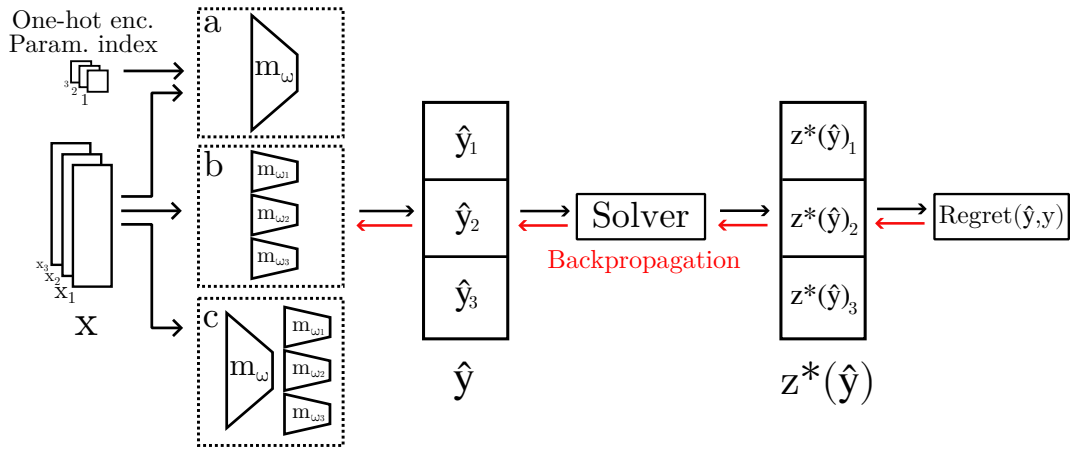

Figure 2: The three proposed parameter-aware architectures of the predictive model: (a) a shared model which we give the parameter index, (b) a separate model per parameter, and (c) a shared backbone with a separate head per parameter.

Parameter-specific models. An alternative approach is to train a separate model for each parameter, instead of one model that is shared across all parameters. In this architecture, each parameter's features are given to the associated model, i.e., $\hat{y} = [m_{\omega_1}(x_1)\ m_{\omega_2}(x_2)\ ...\ m_{\omega_n}(x_n)]$. This way, each parameter's model can learn to predict in a way that accounts for how errors on that specific parameter affect decision making. A downside, however, is that each model is only trained on a subset of the training data. If there are d examples coming from the true mapping in the training set, and n parameters in the optimization problem, then each model is trained on only d/n examples.

Shared backbone with parameter-specific heads. To keep the benefit of parameter-specific models, while reducing their downside, we also consider a multi-headed architecture consisting of a shared backbone, followed by n parameter-specific heads, as is common in multi-task learning (Caruana, 1997; Crawshaw, 2020; Tang & Khalil, 2022b). Each parameter prediction first passes through the shared backbone, and then through the associated head. This way, the backbone can utilize all available training data to learn the underlying mapping from features to parameter values, while each head can learn to adjust the predictions in a parameter-specific way to account for how errors impact decision quality.

5.2 Data augmentation

To further improve on the drawback that parameter-specific components only get to see segments of the training data, we can make use of the fact that the parameters depend on their features identically. Because of this, any value that is associated with any parameter y_i in a historical instance of the optimization problem, could also have been associated with any of the other parameters. Hence we can interchange available feature-value pairs across the parameters, to create additional optimization problem instances that are still in-distribution. We thus propose a data augmentation scheme (Algorithm 1) that *permutes the parameter vectors* at training time, to obtain new optimization problem instances during learning. This scheme can make up for the loss in data efficiency, because each component gets to see many more samples from the true underlying mapping.

This data augmentation scheme should not be confused with ordinary data shuffling during training. While data shuffling only alters the order in which instances are presented to the model, our data augmentation scheme alters the order of parameters *within each instance*. When the parameters are heterogeneous, this transformation can substantially alter the instance itself, along with its associated optimal solution.

Algorithm 1 Decision-focused learning using gradient descent with data augmentation

Input: Training data $D \equiv \{(x^{(i)}, y^{(i)})\}_{i=1}^{N}$;
Hyperparams: α: learning rate, *epochs*: number of epochs
Output: Predictive model m_ω

```
Initialize m_ω
for each epoch do
    for each instance (x, y) do
        Permute the parameters in y
        Permute the feature vectors in x accordingly
        ŷ = [m_ω(x_1) ... m_ω(x_n)]
        z*(ŷ) = arg min_{z∈Ω} f(z; ŷ)
        z*(y) = arg min_{z∈Ω} f(z; y)
        Compute Regret(ŷ, y) using z*(ŷ) and z*(y)
        ω ← ω − α dRegret/dω
    end for
end for
```

6 Experiments

We extensively evaluate the gains in decision quality realized by using parameter-aware architectures and data augmentation, and how they are impacted by:

1. The amount of training data
2. Homogeneity vs. heterogeneity of the parameters
3. The number of parameters

We investigate the impact of each of these dimensions on a different domain, in order to also evidence the general applicability of our findings. Our experiments involve both synthetic and real data. They involve an integer linear program (ILP), a mixed-integer linear program (MILP) and a continuous nonlinear optimization problem. They include linear and nonlinear predictive models. And finally, they include parameter heterogeneity caused by the constraints, by the objective function, and by both.

6.1 Experimental setup

To perform our evaluation, we train various models with differing architectures on several benchmarks (which we introduce below). The PFL models are trained to minimize the mean squared error. For continuous smooth problems, the DFL models are trained using the regret as loss function directly (Agrawal et al., 2019). For combinatorial problems, they are trained using the surrogate SPO+ loss function (Elmachtoub & Grigas, 2022). We opt for this surrogate loss because it has shown great performance across the board in comparative analyses (Tang & Khalil, 2022a; Mandi et al., 2024). However, our findings generalize to the use of other DFL loss functions, as we evidence in Appendix D. Models are trained for a fixed number of epochs, which varies for the different benchmarks, but is always chosen large enough for all models to converge. All learning rates are tuned on the validation set. We report the average regret over the test set in our results. When evaluating on the test set, we always use the model weights that, throughout training, led to the best performance on an independent validation set. Reported results are always the mean and standard error of the mean, taken over 50 independent runs with different train-validation-test splits. With the exception of Table 3, we always report the average *relative* regret:

$$\textit{Rel. Regret}(z^\star(\hat{y}), y) = \frac{f(z^\star(\hat{y}); y) - f(z^\star(y); y)}{f(z^\star(y); y)} \tag{12}$$

The relative regret is the regret equation 2 divided by true optimal value $f(z^\star(y); y)$, and expresses suboptimality as a percentage. For instance, a relative regret of 2% means that the solution produced with the

predicted parameters is 2% worse than the true optimal solution. For each training method, we put the result of the best performing architecture in **bold**. Additionally, we also bold the results of architectures whose performance does not differ from the best-performing architecture with statistical significance. Further details (e.g., tuning details, architectural details, train-validation-test splits) are provided in Appendix C. All code is implemented in Python, using PyTorch (Paszke et al., 2019) and Gurobi (Gurobi Optimization, LLC, 2024). All code and data will be made available upon acceptance.

6.2 Benchmarks

In our evaluation, we use the following predict-then-optimize benchmarks.

6.2.1 Facility location over polynomial mapping

Predict: The ground-truth mapping is a synthetic degree 6 polynomial that is approximated using linear regression to simulate model misspecification. This is common practice in DFL evaluations (Elmachtoub & Grigas, 2022; Mandi et al., 2022; Schutte et al., 2023; Tang & Khalil, 2022a; 2024; Mandi et al., 2024), and is especially relevant when training interpretable models (Futoma et al., 2020; Hughes et al., 2018; Sharma et al., 2021).

Concretely, the data generating process is the following. First, we generate the parameters of the true model as a vector $m \in \mathbb{R}^5$, wherein each entry is a real number uniformly sampled from $[0, 1]$. We then generate features-target pairs as follows. First, each element of feature vector $x_i \in \mathbb{R}^5$ is sampled from standard normal distribution $\mathcal{N}(0, 1)$. Then, the corresponding target value $y_i \in \mathbb{R}$ is generated as $y_i = \frac{1}{3.5^6}\left(1 + (\frac{m^\top x}{5} + 3)^6\right)$.

Optimize: Given a set of potential facility locations $\mathcal{I}$ and a set of customers $\mathcal{J}$, the capacitated facility location problem asks to minimize the total operation cost while satisfying the customer demands of a single product. Let d_j be the demand of customer j, f_i be the fixed cost of opening a facility at location i, y_{ij} be the transportation cost for one unit of product from a facility at location i to customer j, and u_i be the capacity of a facility located at i. Also, let z_i be a binary decision variable that denotes whether a facility is opened at location i, and let z_{ij} be a continuous decision variable denoting the fraction of demand of customer $j \in \mathcal{J}$ served from a facility located at $i \in \mathcal{I}$. Then, the capacitated facility location problem can be formalized as follows:

$$\min \quad \sum_{i \in \mathcal{I}} \sum_{j \in \mathcal{J}} y_{ij} d_j z_{ij} + \sum_{i \in \mathcal{I}} f_i z_i \tag{13}$$

$$\text{s.t.} \quad \sum_{i \in \mathcal{I}} z_{ij} = 1 \text{ for all } j \in \mathcal{J} \tag{14}$$

$$\sum_{j \in \mathcal{J}} d_j z_{ij} \leq u_i z_i \text{ for all } i \in \mathcal{I} \tag{15}$$

$$z_i \in \{0, 1\}, z_{ij} \geq 0 \text{ for all } i \in \mathcal{I} \text{ and } j \in \mathcal{J}. \tag{16}$$

The objective (13) minimizes the total cost of transportation and the fixed costs for the opened facilities. Constraint (14) ensures that the demand of each customer is satisfied. Constraint (15) guarantees that facility capacities are respected and that only the opened facilities are used. Constraint (16) is the domain constraint for the decision variables.

In our experiments, we use 5 facilities and 10 customers. The transportation costs y_{ij} are unknown and to be predicted by the predictive model. The other quantities in the optimization problem are known, and generated as follows. The capacities u_i are evenly spaced between 5 and 15. Each demand d_j is set to $max(0, D)$, where D is sampled (independently for each customer's demand) from $\mathcal{N}(2.5, 1.25)$, which is the normal distribution with mean 2.5 and standard deviation 1.25. Each setup cost f_i is sampled from $U[3, 10]$, where U denotes the uniform distribution.

6.2.2 Knapsack over California housing prices

Predict: This prediction task is taken from a common machine learning benchmark (Pace & Barry, 1997), which is offered as a benchmark in the scikit-learn Python package (Pedregosa et al., 2011). In this benchmark, the median house prices of districts in California must be predicted from 8 correlated local features, including spatial features, features about the districts' populations and other aggregate housing statistics.

Optimize: Given that there is a set of items $\mathcal{I} = \{1, \ldots, n\}$ where item $i \in \mathcal{I}$ has a weight w_i and value y_i, the knapsack problem asks to find the subset of all items with maximum total value without exceeding the knapsack capacity W. Let z_i be a binary decision variable that denotes whether item i is selected. The knapsack problem is then formalized as follows:

$$\max \quad \sum_{i \in \mathcal{I}} y_i z_i \tag{17}$$

$$\text{s.t.} \quad \sum_{i \in \mathcal{I}} w_i z_i \leq W \tag{18}$$

$$z_i \in \{0, 1\} \text{ for all } i \in \mathcal{I}. \tag{19}$$

The objective function (17) maximizes the total value of selected items. Constraint (18) ensures that the knapsack capacity is not exceeded. Constraint (19) is the domain constraint for the decision variables.

In our experiments, the problem contains 40 items. The total capacity is 20. The item values y_i are unknown and to be predicted by the predictive model. The item weights are evenly spaced over a range which we vary to control the degree of heterogeneity. In the homogeneous parameters setting, each weight is set to 1. In the heterogeneous parameters setting with a small range of weights, the weights are evenly spaced between 0.5 and 1.5. In the heterogeneous parameters setting with a large range of weights, the weights are evenly spaced between 0 and 2.

6.2.3 Multi-item newsvendor over Friedman data

Predict: The synthetic Friedman #1 regression dataset was first described in (Friedman, 1991). A generator is provided in scikit-learn (Pedregosa et al., 2011). Each features-target pair is generated as follows. First, 5 features are sampled uniformly from the range $[0, 1]$. Then, the target value y_i follows from its features through the nonlinear mapping $y_i = 10 \sin(\pi x_1 x_2) + 20(x_3 - 0.5)^2 + 10x_3 + 5x_4 + 2\epsilon$, where ϵ is sampled from $N(0, 1)$

Optimize: There is a set of items $\mathcal{I} = \{1, \ldots, n\}$. The problem involves deciding how much of each item to produce. There is an overall budget, limiting the total production. Overproduction and underproduction have distinct costs in the objective, and the ratio between these costs differs from item to item. Let z_i be the produced quantity of item i, let y_i be the demand of item i, and let u_i and o_i be the costs of underproducing and overproducing item i (with respect to demand y_i), respectively. Then, the multi-item newsvendor problem is formalized as follows:

$$\min \quad \sum_{i \in \mathcal{I}} [u_i \max(y_i - z_i, 0) + o_i \max(z_i - y_i, 0)] \tag{20}$$

$$\text{s.t.} \quad \sum_{i \in \mathcal{I}} z_i \leq W \tag{21}$$

$$z_i \geq 0 \text{ for all } i \in \mathcal{I}. \tag{22}$$

The objective function (20) minimizes the total cost of underproduction and overproduction of the various items. Constraint (21) ensures that the total production capacity is not exceeded. Constraint (22) ensures that production cannot be negative.

In our experiments, we used the multi-item newsvendor problem to investigate the effect of the number of parameters in the optimization problem. These parameters represented the item demands. We varied

Table 1: The effect of the training set size, evaluated on a facility location problem using a polynomial mapping

Method	Architecture	Relative regret (%)			
		10 instances	20 instances	100 instances	200 instances
PFL	Shared Model	**3.83 ± 0.32**	**3.93 ± 0.30**	**3.95 ± 0.30**	**4.11 ± 0.31**
	Shared + Param. Index	**3.59 ± 0.22**	**3.76 ± 0.22**	**4.00 ± 0.27**	4.28 ± 0.33
	Separate Models	12.14 ± 0.59	9.92 ± 0.60	5.88 ± 0.41	4.53 ± 0.33
	Separate Heads	7.61 ± 0.51	7.63 ± 0.48	5.56 ± 0.42	4.58 ± 0.36
DFL	Shared Model	**2.81 ± 0.19**	**2.77 ± 0.19**	2.87 ± 0.19	2.91 ± 0.19
	Shared + Param. Index	3.83 ± 0.20	**2.89 ± 0.16**	**2.19 ± 0.12**	2.22 ± 0.13
	Separate Models	25.64 ± 0.91	12.38 ± 0.36	2.96 ± 0.14	2.21 ± 0.12
	Separate Heads	13.48 ± 0.70	5.39 ± 0.25	**2.30 ± 0.15**	**1.96 ± 0.13**
DFL with data augmentation	Shared Model	2.67 ± 0.19	2.73 ± 0.19	2.77 ± 0.18	2.82 ± 0.19
	Shared + Param. Index	2.10 ± 0.12	2.05 ± 0.12	2.12 ± 0.13	2.13 ± 0.13
	Separate Models	**1.72 ± 0.10**	**1.71 ± 0.10**	**1.72 ± 0.11**	1.71 ± 0.10
	Separate Heads	**1.72 ± 0.12**	**1.71 ± 0.11**	**1.67 ± 0.11**	**1.62 ± 0.11**

the number of parameters, and thus items, by considering problems with 10, 50 and 100 items. The total production capacity was set such that, on average, about 75% of the demand in a problem instance could be met. The underproduction unit costs u_i were evenly spaced between 1 and 10, while the overproduction unit costs o_i were evenly spaced over the same range, but in reverse order, i.e., they were evenly spaced between 10 and 1.

6.3 Results and discussion

Amount of training data. We investigate how our proposed architectures compare to the conventional use of a shared model, in function of the amount of training data available. We do so using the facility location problem over a polynomial mapping, as the synthetic nature of this mapping allows us to vary the amount of available training data arbitrarily. As is conventional when using this mapping in DFL evaluations, the predictive models are fully linear, introducing model misspecification. The results can be found in Table 1.

The first four rows of this table (as well as of later tables) show that our proposed architectures do not have a significant positive impact when the model is trained to maximize accuracy using PFL. This is expected, since each parameter depends on features according to the same true underlying mapping, meaning that learning parameter-specific mappings does not benefit predictive accuracy.

The next four rows show that better decision quality (i.e., lower regret) can be achieved by training with DFL instead of PFL. They also show that, when enough training data is available, all three of our proposed architectures outperform the conventional shared one, with the architecture consisting of a shared backbone and separate parameter-specific heads performing best.

The final four rows show that further improvements can be realized by the use of our data augmentation scheme, in which the parameter vectors available in the training data are permuted in random ways. As expected, this is especially important in the low data regime, although it also leads to improvements when more data is available.

Homogeneous vs. heterogeneous parameters We investigate the difference in performance between the architectures for problems with homogeneous parameters and for problems with heterogeneous parameters, using knapsack problems over housing districts in California. We compare the results on a problem with homogeneous parameters (in which all item weights are equal), with two problems with heterogeneous parameters. In the first, the weights are taken from a small interval. In the second, they are taken from a larger interval. We again use linear predictive models. The results can be found in Table 2.

Table 2: The effect of homogeneous vs. heterogeneous parameters, evaluated on a knapsack problem using housing data

Method	Architecture	Relative regret (%)		
		Homogeneous	Heterogeneous	
			Small range	Large range
PFL	Shared Model	**5.31 ± 0.05**	**4.48 ± 0.03**	**3.15 ± 0.03**
	Shared + Param. Index	5.75 ± 0.06	4.72 ± 0.04	3.30 ± 0.03
	Separate Models	**5.32 ± 0.04**	4.62 ± 0.03	3.32 ± 0.03
	Separate Heads	**5.35 ± 0.05**	4.62 ± 0.04	3.33 ± 0.03
DFL	Shared Model	**4.38 ± 0.04**	4.03 ± 0.04	3.03 ± 0.03
	Shared + Param. Index	**4.41 ± 0.04**	4.07 ± 0.04	2.96 ± 0.03
	Separate Models	4.63 ± 0.03	3.94 ± 0.03	2.82 ± 0.02
	Separate Heads	4.47 ± 0.03	**3.80 ± 0.02**	**2.71 ± 0.02**
DFL with data augmentation	Shared Model	**4.37 ± 0.04**	4.04 ± 0.03	3.02 ± 0.03
	Shared + Param. Index	**4.39 ± 0.04**	4.07 ± 0.03	2.92 ± 0.03
	Separate Models	4.61 ± 0.03	3.89 ± 0.03	**2.70 ± 0.02**
	Separate Heads	4.42 ± 0.04	**3.83 ± 0.03**	**2.69 ± 0.03**

Again, the first three rows PFL results evidence the fact that each parameter depends on its features in the same way. The middle three rows show the results for models trained with DFL. Again, DFL improves the regret, regardless of the architecture used. When the parameters are homogeneous, a conventional shared model performs best. However, when there is some heterogeneity in the parameters, separate models and a shared backbone with separate parameter-specific heads are superior, although the difference this makes is more modest than in the previous domain. Increasing the heterogeneity further also increases this gap in performance. These results are in line with our expectation: it is the heterogeneity of the parameters that motivates the introduction of parameters-specific components in the model.

On this domain, data augmentation did not appear to improve performance all that much, indicating that there is already enough data available as is.

The simplicity of the knapsack problem also allows us to get some further insight into what these components have specifically learned. To this end, we inspect the way the prediction errors are distributed for the different items, by computing the Pearson correlation coefficient between the prediction errors (on the test set) and the weights (i.e., constraint coefficients) of the items for which the predictions are respectively made. For the models trained with DFL in the slightly heterogeneous setting, this correlation coefficient is 0.001 for the shared architecture, -0.402 for the separate models, and -0.220 for the shared backbone with separate heads. The negative correlation coefficients for our proposed architectures mean that these models have learned to, on average, *underestimate* more the relative value of high-weight items, and *overestimate* more the relative value of low-weight items. This makes sense: high-weight items are more costly, and thus chosen less often on average. Using our proposed architectures, the models have learned to distribute their prediction errors in parameter-specific ways.

Number of parameters. We investigate how the number of parameters in the optimization problem affects performance. We do so using a multi-item newsvendor problem over the Friedman #1 dataset. When changing the number of parameters, we fix the total number of samples from the true underlying mapping that are available in the training set, so as to isolate the specific effect of the number of parameters. In other words, when we increase the number of parameters, this does not alter the amount of training data the model has access to. Also note that the true optimal values are quite small in this domain, making the denominator in the *relative* regret computation close to zero. For this reason, we instead directly report the regret equation 2 on this benchmark. The predictive models are neural networks with a single hidden layer of 32 neurons, utilizing ReLU activation. The results are shown in Table 3.

Table 3: The effect of the number of parameters, evaluated using on a multi-item newsvendor problem using Friedman #1 data

Method	**Architecture**	**Regret**		
		10 parameters	**50 parameters**	**100 parameters**
PFL	Shared Model	$\mathbf{64.48 \pm 0.29}$	$\mathbf{333.29 \pm 1.16}$	$\mathbf{699.22 \pm 4.05}$
	Shared + Param. Index	66.41 ± 0.37	367.91 ± 3.40	760.39 ± 4.53
	Separate Models	72.91 ± 0.44	476.69 ± 1.31	1217.38 ± 2.22
	Separate Heads	66.74 ± 0.32	387.34 ± 1.35	956.41 ± 6.77
DFL	Shared Model	63.91 ± 0.21	321.92 ± 0.48	$\mathbf{651.92 \pm 2.60}$
	Shared + Param. Index	$\mathbf{58.11 \pm 0.18}$	$\mathbf{309.22 \pm 1.69}$	$\mathbf{649.56 \pm 1.68}$
	Separate Models	67.62 ± 0.32	426.18 ± 2.50	1055.38 ± 3.77
	Separate Heads	59.02 ± 0.18	342.60 ± 1.98	785.89 ± 3.38
DFL with data augmentation	Shared Model	63.49 ± 0.21	321.12 ± 0.54	647.80 ± 1.40
	Shared + Param. Index	$\mathbf{56.06 \pm 0.18}$	$\mathbf{280.48 \pm 0.43}$	$\mathbf{570.07 \pm 1.11}$
	Separate Models	63.82 ± 0.26	349.87 ± 2.23	839.40 ± 4.36
	Separate Heads	59.08 ± 0.25	352.25 ± 4.99	753.44 ± 7.10

For the smallest number of parameters, the results are in line with what was discussed before: training with PFL does not benefit from learning parameter-specific mappings, training with DFL outperforms PFL, and training with DFL *does* benefit from parameter-specific mappings. Data augmentation improves performance further.

When the number of parameters increases, the importance of data augmentation grows. And for 100 parameters, data augmentation is even required in order to outperform the conventional shared model. This is because increasing the number of parameters also shrinks the segments of the dataset that are used to train the parameter-specific mappings with. The use of data augmentation makes up for this.

Unlike the previous two domains, in this domain the best performance is achieved by the model that is given a one-hot encoding of the parameter index, which becomes especially apparent as the number of parameters increases. We attribute this to the fact that, in this benchmark, the predictive models are more expressive than the ones used in the previous benchmarks. The models are thus more capable of appropriately processing the parameter index.

7 Conclusion and future work

In this paper, we investigated an understudied aspect of DFL: that mispredictions can affect downstream decision quality in different ways for different parameters. We formalized this by distinguishing between homogeneous and heterogeneous parameters, and provided a theoretical result that characterizes exactly when parameters are heterogeneous. This characterization shows that homogeneity requires a strong symmetry property of the optimization problem: there must exist an involutive transformation on supported solutions under which the objective is invariant, up to an additive constant, when parameters are permuted. As a consequence, homogeneous parameters arise only under restrictive structural conditions, whereas heterogeneous parameters are the norm in realistic optimization problems that involve non-uniform constraints, costs, demands, or capacities.

We then showed that accounting for heterogeneous parameters in the architecture of the predictive model can significantly improve decision quality. To this end, we investigated three architectural adaptations, and a data augmentation scheme to improve data efficiency. Our experiments confirm that, in the presence of parameter heterogeneity, parameter-aware architectures outperform a conventional shared model. The results suggest that, when the predictive model is linear – as is often the case in prior work on DFL – one should opt for a shared backbone with parameter-specific heads. When the model is more expressive (e.g., a multi-layer neural network), better performance can be reached by using a shared architecture that is given an encoding of the parameter index. The results also showed that – under parameter heterogeneity – data

augmentation leads to further improvements, especially when little data is available, or when the problem contains many parameters.

A promising direction for future work is the extension to a setting in which the parametric optimization problem itself varies across instances, for example when constraint coefficients or feasible regions are instance-dependent. Another direction is a broader exploration of the role data augmentation can play within DFL. For example, when only few samples from the true mapping are, different combinations of these samples within the optimization problem may offer additional signal to the predictive model.

References

Akshay Agrawal, Brandon Amos, Shane Barratt, Stephen Boyd, Steven Diamond, and J. Zico Kolter. Differentiable convex optimization layers. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL `https://proceedings.neurips.cc/paper_files/paper/2019/file/9ce3c52fc54362e22053399d3181c638-Paper.pdf`.

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

Dishank Bansal, Ricky TQ Chen, Mustafa Mukadam, and Brandon Amos. Taskmet: Task-driven metric learning for model learning. *Advances in Neural Information Processing Systems*, 36, 2024.

Senne Berden, Ali İrfan Mahmutoğulları, Dimos Tsouros, and Tias Guns. Solver-free decision-focused learning for linear optimization problems. *arXiv preprint arXiv:2505.22224*, 2025.

Quentin Berthet, Mathieu Blondel, Olivier Teboul, Marco Cuturi, Jean-Philippe Vert, and Francis Bach. Learning with differentiable pertubed optimizers. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 9508–9519. Curran Associates, Inc., 2020. URL `https://proceedings.neurips.cc/paper_files/paper/2020/file/6bb56208f672af0dd65451f869fedfd9-Paper.pdf`.

Rich Caruana. Multitask learning. *Machine learning*, 28:41–75, 1997.

Michael Crawshaw. Multi-task learning with deep neural networks: A survey. *arXiv preprint arXiv:2009.09796*, 2020.

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.

Adam N. Elmachtoub and Paul Grigas. Smart "predict, then optimize". *Management Science*, 68(1):9–26, 2022. doi: 10.1287/mnsc.2020.3922. URL `https://doi.org/10.1287/mnsc.2020.3922`.

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, pp. 1504–1511, 2020.

Jerome H Friedman. Multivariate adaptive regression splines. *The annals of statistics*, 19(1):1–67, 1991.

Joseph Futoma, Michael C Hughes, and Finale Doshi-Velez. Popcorn: Partially observed prediction constrained reinforcement learning. *arXiv preprint arXiv:2001.04032*, 2020.

Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2024. URL `https://www.gurobi.com`.

Michael C Hughes, Gabriel Hope, Leah Weiner, Thomas H McCoy Jr, Roy H Perlis, Erik B Sudderth, and Finale Doshi-Velez. Semi-supervised prediction-constrained topic models. In *AISTATS*, pp. 1067–1076, 2018.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Jayanta Mandi and Tias Guns. Interior point solving for lp-based prediction+optimisation. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 7272–7282. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/51311013e51adebc3c34d2cc591fefee-Paper.pdf.

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, pp. 1603–1610, 2020.

Jayanta Mandi, Víctor Bucarey, Maxime Mulamba Ke Tchomba, and Tias Guns. Decision-focused learning: Through the lens of learning to rank. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato (eds.), *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 14935–14947. PMLR, 17–23 Jul 2022. URL https://proceedings.mlr.press/v162/mandi22a.html.

Jayanta Mandi, James Kotary, Senne Berden, Maxime Mulamba, Victor Bucarey, Tias Guns, and Ferdinando Fioretto. Decision-focused learning: Foundations, state of the art, benchmark and future opportunities. *Journal of Artificial Intelligence Research*, 80:1623–1701, 2024.

Maxime Mulamba, Jayanta Mandi, Michelangelo Diligenti, Michele Lombardi, Victor Bucarey, and Tias Guns. Contrastive losses and solution caching for predict-and-optimize. In Zhi-Hua Zhou (ed.), *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*, pp. 2833–2840. International Joint Conferences on Artificial Intelligence Organization, 8 2021. doi: 10.24963/ijcai.2021/390. URL https://doi.org/10.24963/ijcai.2021/390. Main Track.

Mathias Niepert, Pasquale Minervini, and Luca Franceschi. Implicit mle: Backpropagating through discrete exponential family distributions. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems*, volume 34, pp. 14567–14579. Curran Associates, Inc., 2021. URL https://proceedings.neurips.cc/paper_files/paper/2021/file/7a430339c10c642c4b2251756fd1b484-Paper.pdf.

R Kelley Pace and Ronald Barry. Sparse spatial autoregressions. *Statistics & Probability Letters*, 33(3): 291–297, 1997.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Marin Vlastelica Pogančić, Anselm Paulus, Vit Musil, Georg Martius, and Michal Rolinek. Differentiation of blackbox combinatorial solvers. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=BkevoJSYPB.

Subham Sekhar Sahoo, Anselm Paulus, Marin Vlastelica, Vít Musil, Volodymyr Kuleshov, and Georg Martius. Backpropagation through combinatorial algorithms: Identity with projection works. *arXiv preprint arXiv:2205.15213*, 2022.

Noah Schutte, Krzysztof Postek, and Neil Yorke-Smith. Robust losses for decision-focused learning, 2023.

Sanket Shah, Kai Wang, Bryan Wilder, Andrew Perrault, and Milind Tambe. Decision-focused learning without differentiable optimization: Learning locally optimized decision losses. *arXiv preprint arXiv:2203.16067*, 2022.

Sanket Shah, Bryan Wilder, Andrew Perrault, and Milind Tambe. Leaving the nest: Going beyond local loss functions for predict-then-optimize. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 14902–14909, 2024.

Abhishek Sharma, Catherine Zeng, Sanjana Narayanan, Sonali Parbhoo, and Finale Doshi-Velez. On learning prediction-focused mixtures. *arXiv preprint arXiv:2110.13221*, 2021.

Bo Tang and Elias B Khalil. Pyepo: A pytorch-based end-to-end predict-then-optimize library with linear objective function. In *OPT 2022: Optimization for Machine Learning (NeurIPS 2022 Workshop)*, 2022a.

Bo Tang and Elias B Khalil. Multi-task predict-then-optimize. *arXiv preprint arXiv:2212.05403*, 2022b.

Bo Tang and Elias B. Khalil. Cave: A cone-aligned approach for fast predict-then-optimize with binary linear programs, 2024.

Bryan Wilder, Bistra Dilkina, and Milind Tambe. Melding the data-decisions pipeline: Decision-focused learning for combinatorial optimization. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33 (01):1658–1665, Jul. 2019. doi: 10.1609/aaai.v33i01.33011658. URL `https://ojs.aaai.org/index.php/AAAI/article/view/3982`.

A Proofs of lemmas

In this section, we prove the two supporting lemmas. For clarity, we assume that $z^\star(y)$ is uniquely defined for each y, though the proofs can be readily extended to cases with multiple optimal solutions.

Lemma 4. *Let y_i and y_j be two homogeneous parameters. Then, there do not exist any two parameter vectors y and y' for which $z^\star(y) = z^\star(y')$ but $z^\star(\sigma_{ij}(y)) \neq z^\star(\sigma_{ij}(y'))$.*

Proof. (By contradiction) Assume that parameters y_i and y_j are homogeneous, and that there *do* exist two parameter vectors y and y' for which $z^\star(y) = z^\star(y')$ but $z^\star(\sigma_{ij}(y)) \neq z^\star(\sigma_{ij}(y'))$

Then, since $z^\star(\sigma_{ij}(y)) \neq z^\star(\sigma_{ij}(y'))$, it must hold that

$$f(z^\star(\sigma_{ij}(y'));\sigma_{ij}(y')) < f(z^\star(\sigma_{ij}(y));\sigma_{ij}(y')) \tag{23}$$

Also, since y_i and y_j are homogeneous, the following holds:

$$\begin{aligned}
\textit{Regret}(y, y') &= \textit{Regret}(\sigma_{ij}(y), \sigma_{ij}(y')) && (24)\\
\Rightarrow f(z^\star(y); y') - f(z^\star(y'); y') &= f(z^\star(\sigma_{ij}(y)); \sigma_{ij}(y')) - f(z^\star(\sigma_{ij}(y')); \sigma_{ij}(y')) && (25)\\
\Rightarrow f(z^\star(y'); y') - f(z^\star(y'); y') &= f(z^\star(\sigma_{ij}(y)); \sigma_{ij}(y')) - f(z^\star(\sigma_{ij}(y')); \sigma_{ij}(y')) && (26)\\
\Rightarrow f(z^\star(\sigma_{ij}(y)); \sigma_{ij}(y')) &= f(z^\star(\sigma_{ij}(y')); \sigma_{ij}(y')) && (27)
\end{aligned}$$

Equation 24 follows from the definition of homogeneous parameters. Equation 25 follows from the definition of regret. Equation 26 follows from the assumption that $z^\star(y) = z^\star(y')$. Finally, equation 27 is derived by rearranging terms.

Equation 27 is in contradiction with inequality 23. This contradiction arises from our initial assumption when y_i and y_j are homogeneous, there can exist parameter vectors y and y' such that $z^\star(y) = z^\star(y')$ but $z^\star(\sigma_{ij}(y)) \neq z^\star(\sigma_{ij}(y'))$. Therefore, the assumption is false, and the lemma is proved. □

Lemma 5. *Let $Z^\star$ be the set of* supported *solutions. Let $\tau : Z^\star \to Z^\star$ be a transformation over solutions vectors such that, τ is an involution (i.e., for all $z \in Z^\star$, $\tau(\tau(z)) = z$), and:*

$$\forall y \in \mathbb{R}^n\ \exists b \in \mathbb{R}\ \forall z \in Z^\star : f(z; y) = f(\tau(z); \sigma_{ij}(y)) + b \tag{28}$$

Then, it must hold that:

$$\forall y \in \mathbb{R}^n : z^\star(\sigma_{ij}(y)) = \tau(z^\star(y)) \tag{29}$$

Proof. (By contradiction)

Assume that $\exists y : z^\star(\sigma_{ij}(y)) \neq \tau(z^\star(y))$. Then, $\tau(z^\star(y))$ must be suboptimal with respect to $\sigma_{ij}(y)$:

$$f(z^\star(\sigma_{ij}(y)); \sigma_{ij}(y)) < f(\tau(z^\star(y)); \sigma_{ij}(y)) \tag{30}$$

$$\Rightarrow f(\tau(z^\star(\sigma_{ij}(y))); \sigma_{ij}(\sigma_{ij}(y))) + b < f(\tau(\tau(z^\star(y))); \sigma_{ij}(\sigma_{ij}(y))) + b \tag{31}$$

$$\Rightarrow f(\tau(z^\star(\sigma_{ij}(y))); y) < f(z^\star(y); y) \tag{32}$$

Inequality 31 is obtained by applying assumption 28 to both sides of equation 31. Inequality 32 follows from the fact that $\tau(\tau(z)) = z$ and $\sigma_{ij}(\sigma_{ij}(y)) = y$.

However, inequality 32 contradicts the optimality of $z^\star(y)$, so assumption $z^\star(\sigma_{ij}(y)) \neq \tau(z^\star(y))$ must be false. Therefore, $z^\star(\sigma_{ij}(y)) = \tau(z^\star(y))$. □

B Additional discussion of theoretical results

We start this discussion by illustrating how Proposition 3 applies to Example 1. We repeat the proposition here:

Proposition 6. *Let $Z^\star = \{z \in \Omega : \exists y \in \mathbb{R}^n : z = z^\star(y)\}$ be the set of* supported *solutions. Then, parameters y_i and y_j are homogeneous if and only if there exists a transformation $\tau : Z^\star \to Z^\star$ over solutions z such that τ is an involution (i.e., $\forall z \in Z^\star : \tau(\tau(z)) = z$), and such that:*

$$\forall y \in \mathbb{R}^n \; \exists b \in \mathbb{R} \; \forall z \in Z^\star : f(z; y) = f(\tau(z); \sigma_{ij}(y)) + b$$

Now consider again the parameteric optimization problem from Example 1:

$$\begin{aligned} z^\star(y) = \arg\max \quad & y_1 z_1 + y_2 z_2 + y_3 z_3 \\ \text{s.t.} \quad & z_1 + z_2 + 2z_3 \leq 2 \\ & z_1, z_2, z_3 \in \{0, 1\} \end{aligned}$$

In this problem, parameters y_1 and y_2 are homogeneous. Here, τ is simply the transposition that interchanges the values of z_1 and z_2, while $b = 0$ for all y. Then, for example, for $y = [1\ 2\ 3]$, and $z = [0\ 1\ 0]$: $f(z; y) = [0\ 1\ 0][1\ 2\ 3]^\top = 2$, while $f(\tau(z); \sigma_{ij}(y)) + b = [1\ 0\ 0][2\ 1\ 3]^\top + 0 = 2$.

In Example 1, the homogeneous parameters were associated with items of identical weight, while heterogeneous parameters were associated with items of differing weight. However, the distinction is not always as easily observable. It is possible to construct pathological cases in which the parameters appear heterogeneous but are, in fact, homogeneous. Take, for example, the following problem:

$$\begin{aligned} z^\star(y) = \arg\max \quad & y_1 z_1 + y_2 z_2 + y_3 z_3 \\ \text{s.t.} \quad & 3z_1 + 2z_2 + z_3 \leq 1 \\ & z_1, z_2, z_3 \in \{0, 1\} \end{aligned}$$

Here, y_1 and y_2 are homogeneous (with the same τ and choice of b as before), despite having differing weights in the constraint. This intuitively makes sense: since z_1 and z_2 both must be 0, it does not make a difference to the regret whether a certain prediction error is made on y_1 or on y_2.

While in the previous two examples, additive term b was always 0; this term may not be left out of the proposition. We show this by means of the following example:

$$\begin{aligned} z^\star(y) = \arg\max \quad & y_1 z_1 + y_2 z_2 \\ \text{s.t.} \quad & z_1 = 1 \\ & z_2 = 0 \end{aligned}$$

Parameters y_1 and y_2 are clearly homogeneous, since the only feasible solution is $z = [1\ 0]$, and thus the regret is always 0. However, to apply the proposition, τ is the identity function, while $\forall y : b = y_1 - y_2$. For instance, take $y = [3\ 1]$, for which $b = 2$. Then, $f(z; y) = 3$, while $f(\tau(z); \sigma_{12}(y)) + b = 1 + 2 = 3$

Finally, since the proposition captures the necessary and sufficient conditions for homogeneity to occur, it also rules out several competing explanations for homogeneity. For instance, when the optimal value of the problem is invariant to transposition σ_{ij} (i.e., $\forall y : f(z^\star(y) = f(z^\star(\sigma_{ij}(y))))$, this does not directly imply homogeneity of parameters y_i and y_j. Similarly, even if there exists a transformation τ that preserves the optimal solution with respect to σ_{ij} (i.e., $\forall y : z^\star(y) = \tau(z^\star(\sigma_{ij}(y))))$, this still does not imply homogeneity. We illustrate both using the following unconstrained optimization problem:

$$z^\star(y) = \arg\min \quad |y_1 - z_1| + 2|y_2 - z_2|$$

In this problem, both $\forall y : f(z^\star(y); y) = f(z^\star(\sigma_{ij}(y)); \sigma_{ij})$ and $\exists \tau : \forall y : z^\star(y) = \tau(z^\star(\sigma_{ij}(y)))$ hold. The former holds because for every y, the optimal value is 0. The latter holds for τ equal to the transposition that interchanges the values of z_1 and z_2, because for any y, the optimal solution $z^\star(y) = [y_1\ y_2]$. Thus, after applying σ_{12}, the optimal solution becomes $\tau(z^\star(y)) = [y_2\ y_1]$.

Still, parameters y_1 and y_2 are heterogeneous. Take, for example, $y = [1\ 1]$ and $\hat{y} = [2\ 3]$. Now, $Regret(\hat{y}, y) = 5$, while $Regret(\sigma_{12}(y), \sigma_{12}(\hat{y})) = 4$.

C Detailed experimental setup

In this section, we provide additional details about our experimental setup. We begin with general considerations that apply to all three experiments before discussing specifics for each one.

To perform our evaluation, we trained various models with differing architectures on several benchmarks. All models were trained using the Adam (Kingma & Ba, 2014) optimizer. The learning rates were tuned separately for each method-architecture-benchmark combination, where validation set regret was used to decide which learning rate performed best. Considered values were $0.001, 0.01, 0.1$ and 1. The best-performing learning rate turned out to be 0.01 in each experiment. All models were trained for a fixed number of epochs, which varies for the different benchmarks, but was always chosen large enough for all models to converge. The batch size was chosen to be 16. Note that this refers to the number of optimization problem instances. Thus, for an optimization problem containing 10 variables, a batch size of 16 means that 160 samples from the ground-truth mapping from features to parameters are involved in one weight update.

Reported results are always the mean test set regret, and the standard error of the mean, taken over 50 independent runs with different train-validation-test splits. When evaluating on the test set, we always use the model that, throughout training, performed best on the validation set. In the tables of results, for each training method, we put the result of the best performing architecture in **bold**. Additionally, we also bold the results of architectures whose performance does not differ significantly from the best-performing architecture. To assess whether an architecture performs significantly worse than the best architecture, we conduct paired t-tests on the test regrets across the 50 dataset seeds. The base significance level is set to $\alpha = 0.05$, to which we apply the Holm-Bonferroni correction to account for multiple comparisons.

C.1 Experiment 1: Effect training set size

This experiment was performed using the polynomial mapping as ground-truth features-parameters mapping, and using the facility location problem as optimization problem. Each instance of the optimization problem contained 5 facilities and 10 customers, meaning that 50 samples from the ground-truth mapping were used per instance. Each model was trained for 200 epochs. The training set sizes were varied in the experiment, and are reported in Table 1. The validation set always contained 250 instances of the optimization problem, while the test set always contained 1000 instances. The models were linear models, containing one hidden layer. Although the hidden layer does not add to the representational capacity of the model, it allows for the introduction of variable-specific heads. The hidden layer contained 5 hidden units. In other words, the first layer was a $\mathbb{R}^5 \to \mathbb{R}^5$ mapping and the second layer was a $\mathbb{R}^5 \to \mathbb{R}$ mapping. For the shared architecture,

both layers are shared for all parameters. For the separate models architecture, each parameter has its own dedicated two-layer model. For the architecture consisting of a shared backbone and separate heads, the first layer is shared for all parameters, while each parameter has its own dedicated second layer.

C.2 Experiment 2: Effect homogeneous vs. heterogeneous parameters

This experiment was performed using the California housing prices mapping as ground-truth features-parameters mapping, and using the knapsack problem as optimization problem. Each instance of the knapsack problem contained 40 items. Each model was trained for 100 epochs. All training sets contained 250 instances of the optimization problem (i.e., $40 \times 250 = 10320$ features-parameter pairs). All validation and test sets contained 129 instances of the optimization problem. The models were linear models, this time with a hidden layer size of 8, to match the 8 input features.

C.3 Experiment 3: Effect number of variables

This experiment was performed using the Friedman #1 mapping as ground-truth features-parameters mapping, and using the multi-item newsvendor problem as optimization problem. The number of parameters in the optimization problem was varied in the experiment between 10, 50 and 100. Each model was trained for 500 epochs. Because of the varying number of decision variables, the total number of optimization problem instances in the training set also had to be varied, in order to keep the total number of samples from the ground-truth mapping in the training set constant. For 10 parameters, the training set contained 500 instances. For 50 parameters, it contained 100 instances. For 100 parameters, it contained 50 instances. The remaining instances out of a total of 1000 were divided equally between the validation and test set. The models were nonlinear models with one hidden layer containing 32 hidden units and ReLU activation. The architecture with a shared backbone, the first $5 \to 32$ layer was shared, and the subsequent $32 \to 1$ layers were parameter-specific.

D Additional results for other DFL losses

To strengthen the evidence for the generality of our conclusions, we provide some additional results including other DFL losses below. We use the parametric knapsack problem in combination with a ground-truth polynomial mapping of degree 6. The knapsack weights are evenly spaced between 0 and 2, leading to parameter heterogeneity in the objective. We average the results over 10 seeds. Additional included DFL methods are the PFYL method (Berthet et al., 2020) and the Identity method (Sahoo et al., 2022).

Table 4: Results for some additional DFL losses

Loss	**Architecture**	**Relative regret (%)**
MSE	Shared Model	$\mathbf{0.26 \pm 0.01}$
	Shared + Param. Index	$\mathbf{0.28 \pm 0.01}$
	Separate Models	0.37 ± 0.01
	Separate Heads	0.32 ± 0.01
SPO+	Shared Model	0.25 ± 0.01
	Shared + Param. Index	0.11 ± 0.01
	Separate Models	0.10 ± 0.01
	Separate Heads	$\mathbf{0.03 \pm 0.01}$
PFYL	Shared Model	0.30 ± 0.01
	Shared + Param. Index	0.13 ± 0.01
	Separate Models	0.12 ± 0.01
	Separate Heads	$\mathbf{0.04 \pm 0.01}$
Identity	Shared Model	0.36 ± 0.03
	Shared + Param. Index	0.15 ± 0.01
	Separate Models	0.17 ± 0.01
	Separate Heads	$\mathbf{0.09 \pm 0.01}$