LEARNING THROUGH THE HARD PROBLEM: DECISION-FOCUSED LEARNING WITH A FULL MILP COMBINATORIAL LAYER

Anonymous authors
Paper under double-blind review

ABSTRACT

While real-world applications frequently require solving problems of mixed-integer linear programs (MILPs) with shared structural properties, commercial solvers process each new instance independently from scratch. Graph neural networks (GNNs) offer a promising alternative by predicting optimal binary decisions to guide solvers through variable fixing or warm-starting. However, hard fixing based on thresholded predictions can render the restricted problem infeasible or substantial suboptimal when the GNN is wrong, and training the predictor with a standard classification loss does not account for how prediction errors propagate into decision quality. We investigate three ways of injecting GNN predictions into a MILP solver: hard fixing, big-M penalization, and a new bounded linear soft-penalization surrogate. We show that the soft surrogate can be trained end-to-end with decision-focused losses (SPO+, Fenchel-Young) without introducing the discontinuities that make hard-thresholded restrictions unstable. We evaluate this framework on an energy-aware lot-sizing problem, both in deterministic and two-stage stochastic settings, and show that the proposed methodology achieves favorable and tunable trade-offs between computational time and solution quality.

1 INTRODUCTION

Optimization models play a central role in modern decision-making, supporting critical tasks in production planning, logistics, energy management, and resource allocation. In many real-world applications, similar instances of a hard decision problem must be solved repeatedly under changing contextual conditions, such as fluctuating demands, prices, or renewable availability. Classical optimization algorithms typically solve each new instance independently, without explicitly learning from structural patterns shared across previously solved instances. Machine learning (ML) offers a promising way to learn and generalize these shared patterns to accelerate optimization (Bengio et al., 2021; Yang et al., 2022; Zhang et al., 2023).

While standard ML models are trained to minimize statistical prediction error, Decision-Focused Learning (DFL) (Mandi et al., 2024) aligns model training directly with operational objectives. Although DFL can be realized through direct task-loss surrogates or implicit gradient estimators, integrating complex downstream decision-making often involves embedding an explicit combinatorial optimization (CO) layer into the learning pipeline. To ensure computational tractability during training, a substantial part of the DFL literature has relied on continuous relaxations (Wilder et al., 2019; Ferber et al., 2020), regularized or perturbed optimization mappings(Blondel et al., 2020; Berthet et al., 2020), tractable surrogate losses(Elmachtoub & Grigas, 2022), or simplified CO layers (Parmentier, 2025) (see Appendix A for a detailed review). While recent approaches incorporate richer combinatorial structures, a fundamental question remains:

To what extent is it necessary to simplify the original problem in order to obtain a CO layer that is sufficiently efficient for decision-focused training?

In this work, we investigate a different direction. We introduce a generic, problem-independent CO-layer formulation for mathematical optimization problems involving binary decision variables that preserves the original feasible region. Rather than simplifying or restricting the feasible region, we modify the objective by introducing a linear soft-penalization term derived from the model's

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

predictions. This term guides the CO layer toward the predicted decisions, with the aim of making the resulting problem computationally easier to solve, while preserving all original constraints and allowing the optimizer to deviate whenever beneficial.

The soft penalization is controlled by a parameter M. It acts as a guidance-strength parameter rather than as a constant required to enforce a hard logical implication. Combined with the predictions of the trained model, M determines the strength of the guidance imposed on the optimization problem and, consequently, the trade-off between solution quality and computational time. Importantly, *this trade-off can be adjusted at inference time* by changing M, without altering the feasible region or retraining the predictive model. This allows the desired balance between relying on the learned predictions and optimizing the original objective to be selected a posteriori.

We evaluate the proposed framework on an energy-aware lot-sizing problem, both in deterministic and two-stage stochastic settings. Industrial production planning increasingly requires balancing traditional operational objectives with volatile renewable availability and time-varying energy tariffs (Wichmann et al., 2019; Rodoplu et al., 2019; Liao & Gicquel, 2023; Guo et al., 2024). The resulting models provide a challenging MILP setting in which production, setup, inventory, and energy decisions are tightly coupled. We use Graph Neural Networks (GNNs) to capture the shared structure across problem instances (Nair et al., 2020; Khalil et al., 2022) and DFL to train the predictor through the proposed soft-guided MILP layer, and study the resulting trade-off between solution quality and computational time.

Contributions. The main contributions of this work can be summarized as follows.

- We introduce a generic soft-guidance framework for optimization problems involving binary variables. The proposed approach is problem-independent, preserves the original feasible region, and is evaluated on an energy-aware multi-product lot-sizing problem and its two-stage stochastic extension, two challenging MILP settings for which scalability is a central computational issue.
- We characterize theoretically the role of the guidance parameter M, showing how it controls the balance between following the learned predictions and optimizing the original objective. In contrast to conventional hard penalization schemes, the proposed formulation uses bounded, prediction-dependent objective coefficients, providing a simple and interpretable mechanism for calibrating the strength of the guidance.
- We integrate the proposed CO layer into decision-focused learning pipelines, and compare it with alternative ways of exploiting learned binary decisions, including hard fixing and standard penalization. The experiments show that soft guidance provides favorable trade-offs between solution quality and computational time in both predict-then-optimize and decision-focused settings.
- To test generalizability, we evaluate how well the learned guidance transfers to unseen instances, specifically testing its performance on larger-scale problem instances beyond the training distribution. The results show that the guidance remains effective without retraining, providing substantial computational gains while maintaining competitive solution quality across the tested instance distributions. These findings also motivate going beyond the common small-to-large view of generalization toward richer distributional characterizations of mathematical-programming instances, which could support more effective adaptation of the guidance at inference time.

2 Problem Formulation

Consider a mixed integer linear programming problem P of the following form:

$$P: \quad \min_{x \in \mathcal{X}, y} \; f(x,y) = c_1^T x + c_2^T y \tag{1a}$$

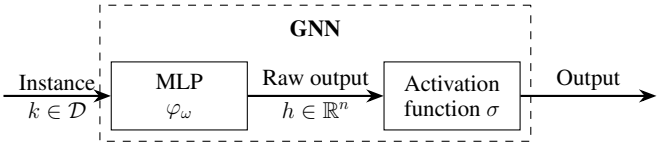
$$s.t. \; g(x,y) \geqslant 0, \tag{1b}$$

$$y \in \{0,1\}^n, \tag{1c}$$

where x, y are sets of continuous and binary variables, respectively, with $\mathcal{X}$ being a subset of a Euclidean space. We focus on mixed-integer linear programs where the combinatorial complexity is

primarily driven by the choice of the discrete variables. In these problems, fixing the binary decisions y yields a tractable continuous linear program (LP) over x. We therefore direct our predictive model toward these structural binary variables. Our framework is valid for an arbitrary vector-valued function $g(x, y)$, but in this work we will focus on the case where it represents a set of linear constraints. We denote by z^* the optimal value and by (x^*, y^*) an optimal solution of P.

MILP representation. We use a bipartite graph $\mathcal{G}$ to represent the problem in equation 1, consisting of two disjoint sets of variable nodes U, corresponding to individual decision variables and constraint nodes V, corresponding to individual constraints. An edge exists only between a variable node $u \in U$ and a constraint node $v \in V$ if the decision variable u is involved in the constraint v (Nair et al., 2020).



Model description. Next, we let φ_ω be a parameterized GNN on $\mathcal{G}$ with weights ω, acting as a *predictor* of the binary decisions y. We denote the raw output (logits) of the final layer as $h \in \mathbb{R}^n$. To map these latent values to the probability interval $[0\ 1]$, we apply an element-wise activation function σ (e.g., the sigmoid function), such that:

$$\hat{p}_i = \sigma(h_i), \quad \forall i \in \{1, \ldots, n\}. \tag{2}$$

In this formulation, each $\hat{p}_i$ represents the predicted likelihood that the binary decision variable y_i equals 1 in the optimal solution to problem P.

Prediction-Guided Heuristics. There can be various strategies to incorporate the predicted probabilities $\hat{p}_i$ in the MILP problem. Two of the more common strategies are *variable fixing*, where variables with high-confidence predictions are fixed to their predicted values, and *variable penalization*, where the solver assigns a significant penalty to deviations from the predicted values, thereby guiding the solver with high-probability assignments.

The variable-fixing heuristic transforms P into a restricted surrogate problem denoted by $P_{\text{Fix}}(\tau, \hat{p})$. We predefine a fixed threshold $\tau \in [0, 1]$ to partition the decision space: if $\hat{p}_i \geqslant \tau$, the variable is fixed to $y_i = 1$; otherwise, the variable remains free. By maintaining all original constraints while reducing the number of discrete decisions, the resulting restricted MILP $P_{\text{Fix}}(\tau, \hat{p})$ is formulated as:

$$P_{\text{Fix}}(\tau, \hat{p}): \quad \min_{x \in \mathcal{X}, y} f^{\text{Fix}}(x, y, \hat{p}) = c_1^T x + c_2^T y \tag{3a}$$

$$s.t.\ \ g(x, y) \geqslant 0, \tag{3b}$$

$$y_i = 1, \qquad \forall i \in \mathcal{I}_\tau, \tag{3c}$$

$$y_i \in \{0, 1\}, \qquad \forall i \notin \mathcal{I}_\tau. \tag{3d}$$

where $\mathcal{I}_\tau$ is defined by $\mathcal{I}_\tau = \{i \in \{1, \ldots, n\} \mid \hat{p}_i \geqslant \tau\}$. We denote by $z^*_{\text{Fix}}(\tau, \hat{p})$ the optimal value and by $(x^*_{\text{Fix},\tau,\hat{p}}, y^*_{\text{Fix},\tau,\hat{p}})$ an optimal solution of $P_{\text{Fix}}(\tau, \hat{p})$.

Since we are fixing part of the binary variables, the computational burden of solving restricted problem $P_{\text{Fix}}(\tau, \hat{p})$ is reduced compared to the original problem. However, we risk making bad decisions with high objective values or wrong decisions that make the problem infeasible. By setting a high value for τ, we restrict the fixing strategy to only those variables where the GNN is mostly confident about. This approach reduces the probability of fixing a variable to 1 that should ideally be 0. While this maintains a high degree of solution accuracy, it leaves a larger number of binary variables for the solver to process, resulting in a smaller reduction in computation time.

A slightly more sophisticated approach is to add a penalization term to the objective function instead of directly fixing the variables. Accordingly, we consider a second heuristic that adds an additional penalization term in the objective function but keeps the constraints unchanged. We denote the

problem as $P_{\text{Pen}}(\tau, \hat{p}; M)$. It is formulated as:

$$P_{\text{Pen}}(\tau, \hat{p}; M): \quad \min_{x \in \mathcal{X}, y} f^{\text{Pen}}(x, y, \hat{p}; M) = c_1^T x + c_2^T y + M \sum_{i \in \mathcal{I}_\tau} (1 - y_i) \tag{4a}$$

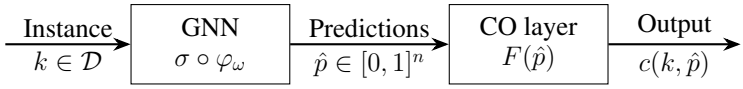
$$s.t. \;\; g(x, y) \geqslant 0, \tag{4b}$$

$$y \in \{0, 1\}^n, \tag{4c}$$

where M is a positive constant. $\mathcal{I}_\tau$ is defined in the same way as in the fixing strategy. We denote by $z^*_{\text{Pen}}(\tau, \hat{p}; M)$ the optimal value and by $(x^*_{\text{Pen},\tau,\hat{p},M}, y^*_{\text{Pen},\tau,\hat{p},M})$ an optimal solution of $P_{\text{Pen}}(\tau, \hat{p}; M)$. On the other hand, it is not straightforward to decide a good value for the penalization term M, as it requires balancing the trade-off between encouraging the solver to follow the predictions and maintaining flexibility to find feasible and high-quality solutions.

It should be pointed out that the penalizing strategy is more flexible than the benchmark fixing strategy, since it allows to readjust the values of variables y_i even if it is in the 'confident' set $\mathcal{I}_\tau$. If the original feasible region of P is non-empty and z^* is finite, then $z^*_{\text{Pen}}(\tau, \hat{p}; M)$ exists and is finite as well. The infeasibility challenge in the fixing strategy does not exist using the penalizing strategy.

Decision-Focused Learning. The goal of the decision-focused learning is to move from "Predict-then-Optimize" sequential paradigm to an integrated "End-to-End Learning" where GNN learns through the solver. The overall structure is shown in the following figure:



Our proposed framework consists of three main steps: the GNN predictor, the combinatorial layer(CO layer), and the feedback loop for training.

Standard DFL architectures usually rely on fast oracles, such as shortest path algorithms, sorting routines, or simple continuous LP relaxations (see Appendix A for detailed review). While these oracles can be computed fast during training, they may fail to handle the complex integer requirements or coupling constraints present in the original MILP, producing decisions that are infeasible. Training the predictor end-to-end within this loop, however, requires solving the CO layer at every gradient step under the current, generally imperfect predictions, and the two heuristics described above are ill-suited to this role. The fixing heuristic P_{Fix} can render the restricted problem infeasible whenever the prediction is wrong, so it cannot be safely embedded inside the training loop; the penalizing heuristic P_{Pen} avoids infeasibility, but its confident set $\mathcal{I}_\tau$ still changes discontinuously with $\hat{p}$ at the threshold τ, so the resulting problem is not smooth in the prediction being learned. This raises a natural question:

Can a CO layer preserve feasibility under arbitrarily complicating constraints, unlike hard fixing, while remaining cheap enough to solve at every training iteration and smooth enough in $\hat{p}$ to train through, unlike a thresholded penalty or a full unrestricted MILP solve?

Section 3 answers this with a soft-guidance CO layer that meets both requirements.

3 Soft Guidance for the Combinatorial Layer

3.1 Soft guidance formulation

We propose to have a soft guidance mechanism for the combinatorial layer, which allows the optimizer to smoothly balance between following the predictive model and optimizing the original objective. The idea is to balance two competing objectives: solving the problem on its own terms, trusting the optimizer's nominal objective $f(x, y)$, and following the predictor's guidance, trusting the prediction $\hat{p}$. Benchmark fixing and benchmark penalization both resolve this balance through a hard threshold τ. This discrete, per-variable allocation of trust is what makes both heuristics unstable: small variations in $\hat{p}$ near τ flip individual variables from one regime to the other. To address this issue, we propose a *soft* integration of predictive information: a weighted-sum scalarization of the same two objectives, in which trust is allocated smoothly and globally through a single continuous weight, rather than variable by variable through a hard threshold.

To follow the idea of soft integration, we introduce a new formulation called the *soft guidance* problem for the CO layer. The problem is denoted as $P_{\text{Soft}}(\hat{p}; M)$:

$$P_{\text{Soft}}(\hat{p}; M): \quad \min_{x \in \mathcal{X}, y} \; f^{\text{Soft}}(x, y, \hat{p}; M) = c_1^T x + c_2^T y + M \sum_{i=1}^{n} ((1 - \hat{p}_i) y_i + \hat{p}_i (1 - y_i)) \tag{5a}$$

$$\text{s.t.} \quad g(x, y) \geqslant 0, \tag{5b}$$

$$y \in \{0, 1\}^n. \tag{5c}$$

We denote by $z^*_{\text{Soft}}(\hat{p}; M)$ the optimal value and by $(x^*_{\text{Soft},\hat{p},M}, y^*_{\text{Soft},\hat{p},M})$ an optimal solution of $P_{\text{Soft}}(\hat{p}; M)$. The additional term,

$$D_{\hat{p}}(y) := \sum_{i=1}^{n} \big[(1 - \hat{p}_i) y_i + \hat{p}_i (1 - y_i)\big] = \|\hat{p} - y\|_1, \tag{6}$$

can be interpreted as a weighted disagreement between the decision y_i and the predicted probability $\hat{p}_i$. For binary variables, it simplifies to a linear expression in y_i, with coefficient $(1 - 2\hat{p}_i)$, which remains bounded in $[-1, 1]$. This has several important advantages. First, the formulation is robust to prediction errors and miscalibration. Unlike cross-entropy-based penalties(see Appendix F), the induced coefficients do not explode when $\hat{p}_i$ approaches 0 or 1, thus avoiding overly aggressive or irreversible decisions. Second, the penalization provides a smooth and controlled trade-off between following the predictive model and optimizing the original objective. The parameter M directly governs this balance and remains interpretable across instances. Third, the formulation preserves feasibility, as no additional constraints are introduced. The optimization problem remains structurally identical to the original one, ensuring that a feasible solution always exists regardless of the quality of the predictions. Overall, this soft penalization can be seen as a linear and optimization-friendly surrogate of the classification mismatch, which provides a stable and effective way to inject predictive information into the decision process.

Having introduced the soft-guidance layer, the first question to address is how far its decisions can be from the true optimum for a *fixed* guidance strength M: does soft guidance, unlike hard fixing, degrade gracefully with prediction error rather than catastrophically? The following proposition answers this directly in terms of how wrong the prediction is. (See Appendix B for the proofs).

Proposition 1. *For any $\hat{p} \in [0, 1]^n$ and $M \geqslant 0$, we have $0 \leqslant z^*_{\text{Soft}}(\hat{p}; M) - z^* \leqslant M\|\hat{p} - y^*\|_1$.*

Proposition 1 can be used to derive several insights into the relationship between P and P_{Soft}. One such insight is that the more accurate the prediction is, the smaller the difference between the optimal values of the two problems will be. Since we intend to treat M as a tunable guidance strength rather than an arbitrary fixed constant, we next ask how the behavior evolves as M varies, does more guidance predictably move the decision toward the prediction, or can it behave erratically as M changes?

Proposition 2 (Monotonicity in the guidance strength)**.** *For $0 \leq M_1 < M_2$, then*

$$\|\hat{p} - y^*_{\text{Soft},\hat{p},M_2}\|_1 \leqslant \|\hat{p} - y^*_{\text{Soft},\hat{p},M_1}\|_1 \quad \textit{and} \quad f(x^*_{\text{Soft},\hat{p},M_2}, y^*_{\text{Soft},\hat{p},M_2}) \geqslant f(x^*_{\text{Soft},\hat{p},M_1}, y^*_{\text{Soft},\hat{p},M_1}).$$

Proposition 2 proves that increasing the penalty coefficient can only move the decision closer to the prediction, at the expense of the original objective. With this monotonicity in hand, we can sharpen Proposition 1 into a bound that is both tighter and explicitly monotonic in M, making precise the intuition that stronger guidance exposes the decision to more risk, but only in proportion to how wrong the prediction actually is.

Corollary 1 (Refined trust-risk bound)**.** *Under any $M \geqslant 0$ we have:*

$$0 \leqslant f(x^*_{\text{Soft},\hat{p},M}, y^*_{\text{Soft},\hat{p},M}) - z^* \leqslant M\big[\|\hat{p} - y^*\|_1 - \|\hat{p} - y^*_{\text{Soft},\hat{p},M}\|_1\big],$$

and the middle bound is non-decreasing in M: stronger guidance can only widen, never narrow, the possible gap to the true optimum.

Corollary 1 directly bounds the deterioration of the solution returned by P_{Soft} when evaluated under the original objective f. It makes explicit the interaction between guidance strength and prediction quality: M determines how strongly the optimizer follows the predictor, while $\|\hat{p} - y^*\|_1$ controls

the risk associated with inaccurate predictions. The refined term $\|\hat{p} - y^*\|_1 - \|\hat{p} - y^*_{\text{Soft},\hat{p},M}\|_1$ further accounts for how much of this guidance is actually followed by the optimizer. Thus, stronger guidance increases the potential cost of prediction errors, motivating the use of M as a tunable parameter controlling the quality–guidance trade-off.

The two extreme regimes clarify this interpretation. At $M = 0$, $P_{\text{Soft}}(\hat{p}; 0)$ reduces to P. For sufficiently large M, the optimizer first minimizes disagreement with $\hat{p}$ and then, among minimum-disagreement solutions, minimizes the original objective. These results assume that P_{Soft} is solved exactly. If instead a feasible solution $(\tilde{x}, \tilde{y})$ is returned with additive error $\delta \geq 0$ on the augmented objective, then

$$0 \leq f(\tilde{x}, \tilde{y}) - z^* \leq M\big[\|\hat{p} - y^*\|_1 - \|\hat{p} - \tilde{y}\|_1\big] + \delta \leq M\|\hat{p} - y^*\|_1 + \delta.$$

3.2 Learning Objective

Under the DFL framework, we minimize the decision regret, which measures the suboptimality of decision $y^*_{\text{Soft},\hat{p},M}$ evaluated under the true predictions $\bar{p} := y^*$. The main challenge of applying this DFL framework is the non-differentiability of the CO layer, because the optimal decision $y^*_{\text{Soft},\hat{p},M}$ is a piecewise constant function of predictions $\hat{p}$. Consequently, the gradient is almost zero everywhere, preventing the use of standard backpropagation. In the literature, to overcome the vanishing gradient problem, researchers have proposed different techniques including constructing surrogate convex loss function, objective or using the perturbed solver. We introduce two typical losses in this subsection: the Smart Predict-then-Optimize loss (SPO+) loss Elmachtoub & Grigas (2022) and the perturbed Fenchel-Young (FY) lossBerthet et al. (2020).

Both loss objectives provide explicit surrogate gradients w.r.t the cost vector $\hat{c}_p = M(1 - 2\hat{p})$:

- **SPO+ Loss:** Evaluates a shifted cost vector $c_{\text{shifted}} = 2\hat{c}_p - \bar{c}_p$ to yield gradient $\nabla_{\hat{c}_p}\mathcal{L}_{SPO+} = 2(y^* - y_{\text{shifted}})$, where $y_{\text{shifted}} \in \arg z^*_{\text{Soft}}(2\hat{p} - \bar{p}; M)$.
- **Perturbed FY Loss:** Introduces noise perturbations Z scaled by intensity $\epsilon > 0$ to construct gradient $\nabla_{\hat{c}_p}\mathcal{L}_{FY} = y^* - \hat{y}_\epsilon$, where $\hat{y}_\epsilon = \mathbb{E}_Z\left[\arg z^*_{\text{Soft}}(\hat{p} + \epsilon Z; M)\right]$ represents expected solver decisions under noise.

Backpropagation Flow. The backpropagation flows through the CO layer and GNN, the parameters of the GNN is updated using the chain rule $\frac{\partial \mathcal{L}}{\partial \omega} = \frac{\partial \mathcal{L}}{\partial \hat{c}_p} \cdot \frac{\partial \hat{c}_p}{\partial \hat{p}} \cdot \frac{\partial \hat{p}}{\partial \omega}$. The second term can be easily computed: $\frac{\partial \hat{c}_p}{\partial \hat{p}} = \frac{\partial M(1-2\hat{p})}{\partial \hat{p}} = -2M$. The third term is computed via standard automatic differentiation. Both SPO+ and FY treat the solver as a black box, requiring only optimal decision outputs under modified cost vectors. Detailed loss formulations and gradient derivations are provided in Appendix C.

4 Application case study: Energy-aware lot-sizing

In the case study, we consider a real-world energy-aware lot-sizing problem under a single-machine multi-product setting. The industrial plant is equipped with a single production machine and a local energy system (photovoltaic panels and batteries) supplying the manufacturing process. We evaluate both the deterministic formulation and its two-stage stochastic extension (detailed in Appendix D for the mathematical formulations and Appendix E for the numerical settings).

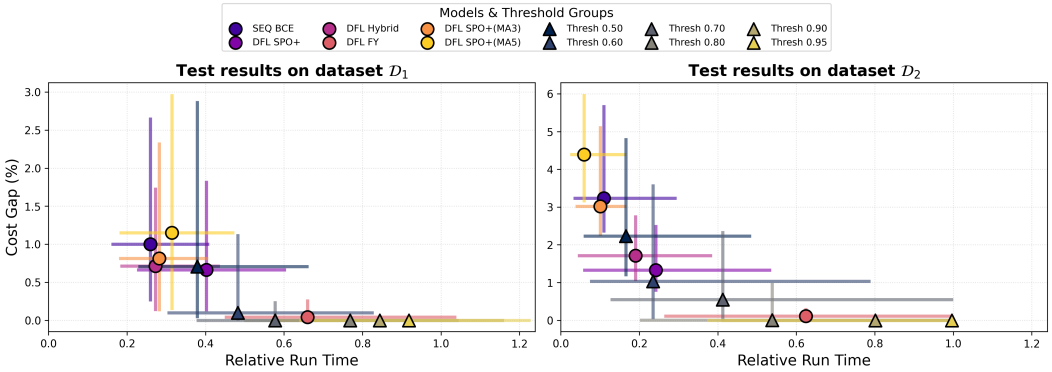
In the deterministic model, the number of binary variables is proportional to the product count multiplied by the micro-periods; predicting these variables via a GNN provides information for the optimization. To train and evaluate the models, we generate three datasets: $\mathcal{D}$ for training, and $\mathcal{D}_1, \mathcal{D}_2$ for testing (along with stochastic counterparts $\mathcal{DS}$ for training and $\mathcal{DS}_1, \mathcal{DS}_2$ for testing).

4.1 Performance of Variant Models

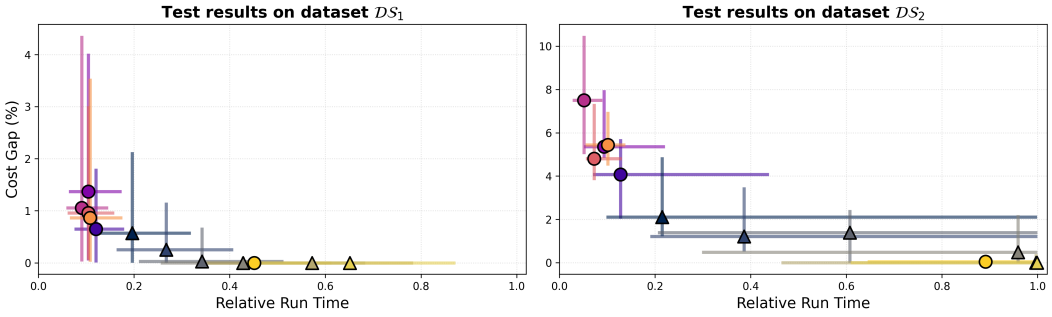
For the DFL framework, we employ a soft penalization-based surrogate in the CO layer to avoid rigid thresholding, whereas classical benchmark fixing and penalization approaches are restricted to the predict-then-optimize baseline (reported in Appendix G). Specifically, we evaluate the proposed soft-penalized models that are represented by circular markers against classical benchmark

thresholding strategies represented by triangular markers. The proposed soft-penalized configurations include the sequential predict-then-optimize baseline trained via Binary Cross-Entropy (SEQ BCE), Decision-Focused Learning models trained with SPO+ loss (DFL SPO+), a composite objective combining SPO+ and BCE loss (DFL Hybrid, $0.5\mathcal{L}_{\text{SPO+}} + 0.5\mathcal{L}_{\text{BCE}}$), the Fenchel-Young loss (DFL FY), and moving-average target smoothing variants averaged over 3 and 5 micro-periods (DFL SPO+(MA3/MA5)). In contrast, the benchmark thresholding strategies (Thresh τ, with $\tau \in \{0.50, 0.60, 0.70, 0.80, 0.90, 0.95\}$), represent sequential BCE model evaluated via the benchmark penalization across various threshold values τ.

Figure 1 summarizes performance trade-offs across deterministic ($\mathcal{D}_1, \mathcal{D}_2$ in Figure 1a and stochastic settings ($\mathcal{DS}_1, \mathcal{DS}_2$ in Figure 1b). The y axis (Cost Gap %) measures solution quality degradation relative to the exact optimal solution. The x axis (Relative Run Time) quantifies solver effort normalized against an unguided exact MILP solve. Each marker indicates empirical mean performance, while solid horizontal and vertical error bar lines depict variability across instances for runtime and cost gap, respectively.



(a) Deterministic setting ($\mathcal{D}_1$ and $\mathcal{D}_2$)



(b) Stochastic setting ($\mathcal{DS}_1$ and $\mathcal{DS}_2$)

Figure 1: Comparative evaluation of cost gap (%) versus relative run time across deterministic ($\mathcal{D}_1$, $\mathcal{D}_2$) and stochastic ($\mathcal{DS}_1$, $\mathcal{DS}_2$) test datasets.

Observation 1 On instances drawn from the same distribution as the training data, the proposed soft penalization approach consistently outperforms the benchmark penalization methods.

When evaluated on in-distribution datasets ($\mathcal{D}_1$ and $\mathcal{DS}_1$), the proposed soft penalization models establish a strictly dominant Pareto front over the benchmark thresholding strategies. As seen in the left subpanels of Figure 1a and 1b, soft-penalized models such as DFL Hybrid and DFL SPO+ cluster near the lower-left origin, achieving relative run times of $\approx 0.10 - 0.50$ (with the mean around 0.3) while maintaining low cost gaps ($\leq 3.0\%$). In contrast, to achieve comparable cost gaps under the benchmark penalization framework, the solver requires thresholds of (Thresh 0.60 through Thresh 0.95, grey to yellow triangles), which pushes relative run times up to $0.45 - 1.20$. Conversely, benchmark configurations that match the fast run time of soft-penalized models (e.g., Thresh 0.50) incur substantially wider cost gap distribution. Soft penalization effectively avoids this

rigid trade-off by guiding the solver continuously without over restricting the search space. We also notice that by averaging training targets across adjacent periods, moving-average target smoothing (MA3 and MA5) stabilizes the learning signal, resulting in faster execution profiles than the standard DFL SPO+ while maintaining controlled optimality gaps.

Observation 2 On instances larger than those in the training set, the proposed method achieves average performance (in terms of relative run time and cost gap) comparable to the benchmark penalization. Moreover, its performance distribution is substantially tighter, demonstrating greater stability with fewer extreme outliers around the mean.

On out-of-distribution instances featuring larger problem scales ($\mathcal{D}_2$ and $\mathcal{DS}_2$), both approaches shift upward and rightward due to increased combinatorial complexity. As illustrated in the right subpanels of Figure 1a and 1b, high-threshold benchmark methods (Thresh 0.80–0.95) achieve near-zero mean cost gaps but exhibit extremely wide horizontal error bars that extend across nearly the entire time domain (0.1 to 1.0). This reveals high variance: while some instances solve quickly, others suffer severe solver slowdowns. In contrast, soft-penalized DFL models (such as DFL Hybrid and DFL SPO+) cluster tightly along both axes with noticeably shorter error bar lines. This confirms that while average speed and accuracy remain competitive with high-threshold benchmarks on larger scales, soft penalization provides superior operational reliability by preventing catastrophic solve times and extreme performance outliers.

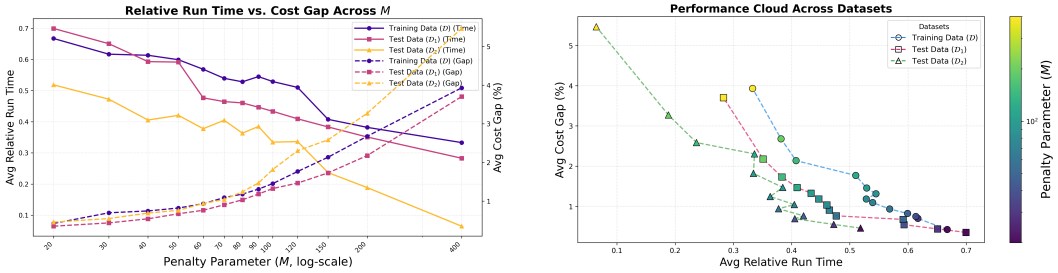
4.2 Gap and Time Evolution With Respect To M

In this section, we analyze the evolution of the average relative runtime and cost gap across varying penalty parameter values M, using the soft penalized DFL SPO+ model evaluated on different datasets (Figures 2 and 3). They both have two subfigures:

- **Subfigure (a) — Dual-Axis Line Plots:** Illustrates the trajectory of metrics as M increases along the horizontal axis (log-scaled, $M \in [20, 400]$). Solid lines mapped to the primary left vertical axis track *Avg Relative Run Time*, showing a monotonic decline as M grows. Dashed lines with marker symbols mapped to the secondary right vertical axis track *Avg Cost Gap (%)*, reflecting an upward trend with M.
- **Subfigure (b) — Efficiency-Accuracy Trade-Off Clouds:** Directly visualizes the Pareto frontier by plotting *Avg Relative Run Time* on the horizontal axis against *Avg Cost Gap (%)* on the vertical axis. Points represent evaluated configurations across training data (circles, blue dashed trajectory), in-distribution test data (squares, red dashed trajectory), and out-of-distribution test data (triangles, green dashed trajectory). Marker colors encode the value of M via a continuous colorbar ranging from dark purple ($M = 20$) to bright yellow ($M = 400$).

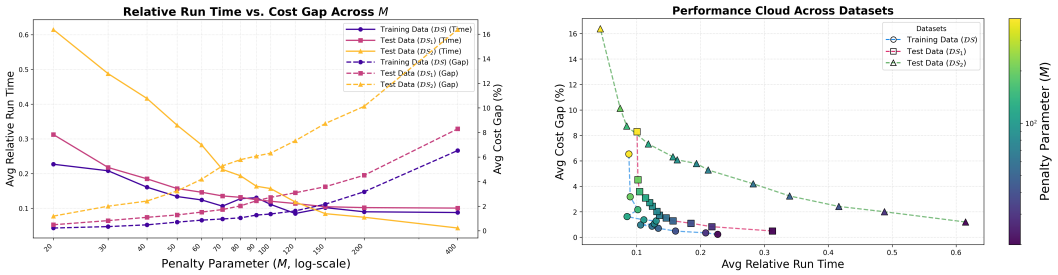
Across all instance scales, the penalty parameter M establishes a clear computational trade-off between solver speedup and final solution quality. As observed in Figure 2a, small penalty values ($M \leq 30$, purple markers) allow the solver to explore the search space with minimal soft-penalization constraint, maintaining low cost gaps (below 1%) at the expense of higher execution times (relative solve times between 0.5 and 0.7). As M increases to 400 (yellow markers), relative runtime decreases significantly. It falls to around $0.3 - 0.35$ for datasets $\mathcal{D}$ and $\mathcal{D}_1$, and drops below 0.1 for dataset $\mathcal{D}_2$. Across this entire range ($M \in [20, 400]$), the cost gap increases moderately but remains well controlled under 5.5% for deterministic instances. The range $M \in [100, 200]$ represents a favorable operational regime, effectively balancing substantial speedups with low cost gaps (under 3%). For the stochastic setting, Figure 3 confirms that similar trade-off dynamics hold across stochastic settings.

Observation 3 M can be viewed as the *guidance strength*: it governs how much weight the CO layer places on agreeing with the prediction relative to the original objective. Informally, M can also be read as expressing how much the optimizer relies on, or "trusts," the predictor; we use this reading only as intuition, since no result below establishes that M is calibrated to, or a statistically valid measure of, the predictor's actual reliability.



(a) Evolution of run time and cost gap over M (b) Efficiency-accuracy tradeoff cloud

Figure 2: Impact of penalty parameter M on average relative run time and cost gap across training ($\mathcal{D}$) and test datasets ($\mathcal{D}_1, \mathcal{D}_2$).



(a) Evolution of run time and cost gap over M (b) Efficiency-accuracy tradeoff cloud

Figure 3: Impact of penalty parameter M on average relative run time and cost gap across training ($\mathcal{DS}$) and test datasets ($\mathcal{DS}_1, \mathcal{DS}_2$).

Observation 4 On test instances drawn from the same distribution as the training data, the evolution of the cost gap and relative runtime as M varies closely mirrors the behavior observed on the training set. This suggests that the trade-offs observed during training can be used at inference time to select a value of M that provides the desired balance between solution quality and computational time. On larger instances, the predictions still effectively guide the solver and provide substantial speedups, but the transfer is less stable: as M increases, the average cost gap grows more rapidly than on the training distribution.

5 Conclusion

In this work, we investigated whether decision-focused learning for MILPs requires simplifying the original optimization problem to obtain a tractable CO layer. We proposed a generic soft-guidance formulation for problems with binary decisions that preserves the original feasible region. The guidance parameter M controls the balance between following these predictions and optimizing the original objective. We characterized this role theoretically, relating the effect of M to prediction error, while allowing the guidance strength to be adjusted at inference time without retraining the model. Experiments show favorable and tunable trade-offs between solution quality and computational time in both predict-then-optimize and decision-focused settings.

Our experiments also provide clear evidence of the generalization capability of the proposed approach. The learned guidance transfers effectively to unseen instances without retraining, including instances substantially larger than those used during training. Moreover, M remains an effective inference-time mechanism for adjusting the quality–computation trade-off across these datasets. These results motivate a broader investigation of how learned optimization guidance transfers across distributions of mathematical-programming instances. In particular, richer notions of similarity or distance between instance distributions, based on statistical, structural, or optimization-related characteristics, could help characterize this transfer and ultimately support an instance- or distribution-adaptive choice of M at inference time.

AI USE STATEMENT

In this work, we used generative AI tools for assist in the writing of proofs. We have not used generative AI tools for formulate mathematical claims, provide critical ingredients for proving mathematical claims, help develop theoretical models or conceptual frameworks, propose or refine hypotheses, design or provide feedback on research methodology or experiments, implement methods, clean and reformat dataset, generate synthetic data sets, support qualitative and thematic data analysis, interpret results. and assist with translation are not applicable to this work.

Additionally, we used generative AI tools for identify relevant literature, format references, sourcing/searching for information, edit a research paper to improve readability, summarize or analyze existing literature. We have reviewed all AI-assisted work. LLM-assisted prose and literature summaries were manually verified against primary sources to eliminate hallucinations, correct technical terminology, and ensure accurate citations. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

To ensure the reproducibility of our theoretical, methodological, and experimental results, we have taken several key measures throughout the paper and appendices:

- Methodology and Mathematical Formulations: Mathematical derivations of the soft-penalized CO layer, the DFL loss surrogates (SPO+ and perturbed FY), and their corresponding gradient backpropagation steps are detailed in Section 3 and further elaborated in Appendix C.
- Datasets and Problem Settings: A detailed description of the energy-aware lot-sizing problem is provided in Appendix D. Experimental setups, parameter distributions, energy constraint specifications (battery and PV), and synthetic data generation protocols for deterministic and stochastic cases are provided in Appendix E.
- Source Code: Complete implementation code, including the GNN architecture, soft-penalized solver layers, baseline predict-then-optimize pipelines, evaluation scripts, and plotting figures, will be made publicly available on GitHub upon acceptance to allow full replication of all figures and benchmark results.

REFERENCES

Nasser Alkhulaifi, Ismail Gokay Dogan, Timothy R. Cargan, Alexander L. Bowler, Direnc Pekaslan, Nicholas J. Watson, and Isaac Triguero. Decision-focused learning enhanced by automated feature engineering for energy storage optimisation. *Expert Systems with Applications*, 302: 130554, 2026. URL https://eprints.whiterose.ac.uk/id/eprint/238066/1/DecisionFocusedLearning.pdf.

Yoshua Bengio, Andrea Lodi, and Antoine Prouvost. Machine learning for combinatorial optimization: a methodological tour d'horizon. *European Journal of Operational Research*, 290(2): 405–421, 2021.

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

Abhinav Bhatia, Justin Svegliato, Samer B. Nashed, and Shlomo Zilberstein. Tuning the hyperparameters of anytime planning: A metareasoning approach with deep reinforcement learning. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 32, pp. 556–564, 2022. URL https://ojs.aaai.org/index.php/ICAPS/article/view/19842.

Mathieu Blondel, André FT Martins, and Vlad Niculae. Learning with fenchel-young losses. *Journal of Machine Learning Research*, 21(35):1–69, 2020.

Aymeric Blot, Holger H. Hoos, Laetitia Jourdan, Marie-Eléonore Kessaci-Marmion, and Heike Trautmann. MO-ParamILS: A multi-objective automatic algorithm configuration framework. In *Learning and Intelligent Optimization*, volume 10079 of *Lecture Notes in Computer Science*, pp. 32–47, 2016. URL https://ada.liacs.nl/papers/BloEtAl16.pdf.

Stefan Clarke and Bartolomeo Stellato. Conformal prediction for early stopping in mixed integer optimization, 2026. URL https://arxiv.org/abs/2602.01476.

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, pp. 1504–1511, 2020.

Ziqing Guo, Jingxing Gao, Yachao Dong, and Jian Du. Integrated batch production planning and scheduling with machine-learning based production capacity region considering wind energy system. *Journal of Cleaner Production*, 477:143804, 2024.

Qingyu Han, Linxin Yang, Qian Chen, Xiang Zhou, Dong Zhang, Akang Wang, Ruoyu Sun, and Xiaodong Luo. A gnn-guided predict-and-search framework for mixed-integer linear programming. *arXiv preprint arXiv:2302.05636*, 2023.

Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *Learning and Intelligent Optimization*, volume 6683 of *Lecture Notes in Computer Science*, pp. 507–523, 2011. URL https://www.cs.ubc.ca/~hoos/Publ/HutEtAl11.pdf.

Frank Hutter, Lin Xu, Holger H. Hoos, and Kevin Leyton-Brown. Algorithm runtime prediction: Methods & evaluation. *Artificial Intelligence*, 206:79–111, 2014. URL https://arxiv.org/abs/1211.0906.

Elias B Khalil, Christopher Morris, and Andrea Lodi. Mip-gnn: A data-driven framework for guiding combinatorial solvers. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pp. 10219–10227, 2022.

Ruiwen Liao and Céline Gicquel. Integrated lot-sizing and energy supply planning with onsite generation of intermittent renewable energy. *International Journal of Production Research*, pp. 1–21, 2023.

Defeng Liu, Matteo Fischetti, and Andrea Lodi. Learning to search in local branching. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pp. 3796–3803, 2022a. URL https://ojs.aaai.org/index.php/AAAI/article/view/20294.

Defeng Liu, Matteo Fischetti, and Andrea Lodi. Revisiting local branching with a machine learning lens, 2022b. URL https://arxiv.org/abs/2112.02195.

Haoyang Liu, Jie Wang, Zijie Geng, Xijun Li, Yuxuan Zong, Fangzhou Zhu, Jianye Hao, and Feng Wu. Apollo-milp: An alternating prediction-correction neural solving framework for mixed-integer linear programming. In *International Conference on Learning Representations*, volume 2025, pp. 94513–94540, 2025.

Manuel López-Ibáñez and Thomas Stützle. Automatically improving the anytime behaviour of optimisation algorithms. *European Journal of Operational Research*, 235(3):569–582, 2014. URL https://lopez-ibanez.eu/doc/LopStu2014ejor.pdf.

Jayanta Mandi, Emir Demirović, Peter J Stuckey, and Tias Guns. 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, 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.

Jayanta Mandi, Ali İrfan Mahmutoğulları, Senne Berden, and Tias Guns. Minimizing surrogate losses for decision-focused learning using differentiable optimization. In *ECAI 2025*, 2025. URL https://arxiv.org/abs/2508.11365.

Maxime Mulamba, Jayanta Mandi, Michelangelo Diligenti, Michele Lombardi, Victor Bucarey, and Tias Guns. Contrastive losses and solution caching for predict-and-optimize. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 2021. URL https://arxiv.org/abs/2011.05354.

Vinod Nair, Sergey Bartunov, Felix Gimeno, Ingrid Von Glehn, Pawel Lichocki, Ivan Lobov, Brendan O'Donoghue, Nicolas Sonnerat, Christian Tjandraatmadja, Pengming Wang, et al. Solving mixed integer programs using neural networks. *arXiv preprint arXiv:2012.13349*, 2020.

Axel Parmentier. Learning structured approximations of combinatorial optimization problems. *Open Journal of Mathematical Optimization*, 6:1–27, 2025.

Max Paulus and Andreas Krause. Learning to dive in branch and bound. *Advances in Neural Information Processing Systems*, 36:34260–34277, 2023.

Melek Rodoplu, Taha Arbaoui, and Alice Yalaoui. Single item lot sizing problem under renewable energy uncertainty. *IFAC-PapersOnLine*, 52(13):18–23, 2019.

Yunzhuang Shen, Yuan Sun, Andrew Eberhard, and Xiaodong Li. Learning primal heuristics for mixed integer programs. In *2021 international joint conference on neural networks (ijcnn)*, pp. 1–8. IEEE, 2021.

Bo Tang and Elias B. Khalil. PyEPO: A PyTorch-based end-to-end predict-then-optimize library for linear and integer programming. *Mathematical Programming Computation*, 16:297–335, 2024. URL https://arxiv.org/abs/2206.14234.

Marin Vlastelica, Anselm Paulus, Vít Musil, Georg Martius, and Michal Rolínek. Differentiation of blackbox combinatorial solvers. In *International Conference on Learning Representations*, 2020. URL https://arxiv.org/abs/1912.02175.

Matthias Gerhard Wichmann, Christoph Johannes, and Thomas Stefan Spengler. Energy-oriented lot-sizing and scheduling considering energy storages. *International Journal of Production Economics*, 216:204–214, 2019.

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, pp. 1658–1665, 2019.

Xinyi Yang, Ziyi Wang, Hengxi Zhang, Nan Ma, Ning Yang, Hualin Liu, Haifeng Zhang, and Lei Yang. A review: machine learning for combinatorial optimization problems in energy areas. *Algorithms*, 15(6):205, 2022.

Taehyun Yoon, Jinwon Choi, Hyokun Yun, and Sungbin Lim. Threshold-aware learning to generate feasible solutions for mixed integer programs, 2023. URL https://arxiv.org/abs/2308.00327. Preprint; final journal publication status not verified.

Jiayi Zhang, Chang Liu, Xijun Li, Hui-Ling Zhen, Mingxuan Yuan, Yawen Li, and Junchi Yan. A survey for solving mixed integer programming via machine learning. *Neurocomputing*, 519: 205–217, 2023.

Shlomo Zilberstein. Optimizing decision quality with contract algorithms. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 1995. URL https://www.ijcai.org/Proceedings/95-2/Papers/072.pdf.

A Related Works

Graph Neural Networks for MILPs Recent research has extensively explored using GNNs to predict optimal values for binary decision variables in MILPs. It relies on the idea of mapping variable-constraint matrices with bipartite graph structures. The foundational Neural Diving framework Nair et al. (2020) demonstrated that a GNN could be used for predicting binary variable assignments and prune the branch-and-bound search space. However, this hard-fixing approach introduced a feasibility bottleneck, where slight predictive inaccuracies often yielded infeasible sub-MILPs. To address this, subsequent studies explored different ideas including e.g. solver guiding branching rules, warm starting, or duality-guided diving. Shen et al. (2021) utilized variable-only linkage graphs to guide the branching process, Khalil et al. (2022) formulated statistical variable expectations as soft hints for warm-starting, and Paulus & Krause (2023) leveraged LP dual information alongside Farkas proofs to validate diving paths or prune unpromising ones. Most recently, closed-loop frameworks such as Apollo-MILP Liu et al. (2025) address prediction errors through iterative correction loops that refine GNN suggestions via localized trust-region searches. Beyond hard fixing, several methods already make the *strength* of reliance on predictions adjustable rather than all-or-nothing: Han et al. (2023) predict marginal variable probabilities and restrict the search to an explicit trust-region neighborhood around them; Liu et al. (2022a;b) learn how large a local-branching neighborhood, and how much time, to allocate per instance from a weighted combination of objective value and computing time; and Yoon et al. (2023) study how a coverage/threshold parameter for partial assignments transfers across instance sizes. Our contribution differs from all of these by entering through a soft objective preference over the entire, unrestricted feasible set, rather than a hard neighborhood, coverage, or partial-assignment choice.

Decision-Focused Learning with CO layer While DFL was initially formulated for continuous quadratic or convex programs, extending end-to-end task-based learning to discrete domains requires embedding Combinatorial Optimization (CO) layers directly into deep learning pipelines. Since combinatorial optimization exhibits piecewise-constant objective landscapes, yielding zero gradients almost everywhere, a central focus of DFL research has been developing differentiable surrogates for discrete oracles. To tackle the non-differentiability and zero-gradient challenges, various continuous surrogates have been proposed. These include subgradient estimates through e.g. implicit differentiation over continuous relaxations Ferber et al. (2020), smoothed convex surrogate objectives via Fenchel-Young loss Blondel et al. (2020), gradient smoothing via perturbed optimizer Berthet et al. (2020), and convex surrogate loss functions such as Smart Predict-than-Optimize (SPO+) Elmachtoub & Grigas (2022). Parmentier (2025) extends the literature by combining Gaussian perturbation with structural CO surrogates. Unlike standard DFL methods that require explicit ground-truth targets or full downstream simulation, this approach optimizes risk directly on unlabeled instances. These relaxation-based surrogates are not the only way DFL has handled discrete oracles: Mandi et al. (2020) explicitly train through the full integer program and compare it against relaxed and approximate oracles, and Vlastelica et al. (2020) differentiate through a black-box combinatorial solver, including a MILP application, via a finite-difference backward construction. Computational cost, rather than an inherent incompatibility between DFL and exact integer oracles, is the main practical obstacle to solving a full MILP at every training step, motivating solution caching Mulamba et al. (2021), efficient surrogate optimization Mandi et al. (2025), reusable predict-then-optimize libraries Tang & Khalil (2024), and applied full-MILP DFL pipelines Alkhulaifi et al. (2026).

Learned control of computational effort A separate line of work learns how much computational effort a solver should spend, rather than what to predict for the solver's inputs. Hutter et al. (2011) and Hutter et al. (2014) use empirical performance models for algorithm configuration and runtime prediction, including for CPLEX; Zilberstein (1995), López-Ibáñez & Stützle (2014), and Blot et al. (2016) study quality/time preferences and multi-objective tuning of anytime behavior, and Bhatia et al. (2022) learn stopping control for anytime planning via reinforcement learning. Most closely related to this work, Clarke & Stellato (2026) calibrate a learned early-stopping criterion for MILP solving to a quality tolerance and error probability using conformal prediction, giving a distributional coverage guarantee that an ordinary mean-response estimate does not. This literature establishes that learned, request-dependent control of computational effort is not itself new; our contribution is to

study this control specifically for the strength of decision-focused guidance, paired with an explicit, solver-independent account of its effect on the original objective (Section 3).

B PROOF OF THE PROPOSITIONS

Proof of Proposition 1

Proof. $P_{\text{Soft}}(\hat{p}; M)$ and problem P share the same feasible region since they contain exactly the same constraints. When evaluating the objective of $P_{\text{Soft}}(\hat{p}; M)$ at the true optimal y^* of the original problem P, which is a feasible but possibly suboptimal for $P_{\text{Soft}}(\hat{p}; M)$, the penalty term equals $M \sum_i |\hat{p}_i - y_i^*|$. Thus, we have $z^*_{\text{Soft}}(\hat{p}; M) \leqslant z^* + M \sum_i |\hat{p}_i - y_i^*| = z^* + M\|\hat{p} - y^*\|_1$. □

Proof of Proposition 2

Proof. Throughout this proof, let $(x_j, y_j) := (x^*_{\text{Soft},\hat{p},M_j}, y^*_{\text{Soft},\hat{p},M_j})$ for $j = 1, 2$ be shorthand for the optimal solutions of $P_{\text{Soft}}(\hat{p}; M_1)$ and $P_{\text{Soft}}(\hat{p}; M_2)$ respectively. Let $g_j = f(x_j, y_j)$ and $D_j = \|\hat{p} - y_j\|_1$ for $j = 1, 2$. Optimality gives $g_1 + M_1 D_1 \leqslant g_2 + M_1 D_2$ and $g_2 + M_2 D_2 \leqslant g_1 + M_2 D_1$. Adding yields $(M_2 - M_1)(D_2 - D_1) \leqslant 0$, so $D_2 \leqslant D_1$ since $M_2 > M_1$; substituting back gives $g_2 - g_1 \geqslant M_1(D_1 - D_2) \geqslant 0$, proving the pairwise claim. □

Proof of Corollary 1

Proof. Let $(x_M, y_M) := (x^*_{\text{Soft},\hat{p},M}, y^*_{\text{Soft},\hat{p},M})$. Optimality of (x_M, y_M) against the feasible point (x^*, y^*) gives $f(x_M, y_M) + M\|\hat{p} - y_M\|_1 \leqslant z^* + M\|\hat{p} - y^*\|_1$, i.e., $f(x_M, y_M) - z^* \leqslant M[\|\hat{p} - y^*\|_1 - \|\hat{p} - y_M\|_1]$; the left inequality holds since $z^* \leqslant f(x_M, y_M)$, and the last inequality drops the non-negative term $\|\hat{p} - y_M\|_1$. At $M = 0$ this bound is 0; for $0 < M_1 < M_2$, Proposition 2 shows $\|\hat{p} - y_M\|_1$ is non-increasing, so the bracket $\|\hat{p} - y^*\|_1 - \|\hat{p} - y_M\|_1$ is non-decreasing, and its product with M is therefore non-decreasing as well. □

C DETAILED FORMULATIONS OF DFL LOSS FUNCTIONS

In this appendix, we provide the full mathematical definitions and gradient computations for the Decision-Focused Learning (DFL) loss surrogates utilized in Section 3.

C.1 SMART PREDICT-THEN-OPTIMIZE (SPO+) LOSS

The Smart Predict-then-Optimize (SPO+) loss provides a convex surrogate for the regret, making it possible to generate a gradient when the solver's output is discrete. For each input $\hat{p}$, the associated cost or the coefficient of the corresponding variables y is denoted as $\hat{c}_p = M(1 - 2\hat{p})$. Let $\bar{c}_p = M(1 - 2\bar{p})$ be the true cost. The effectiveness of SPO+ lies in the use of a shifted cost $c_{shifted} = 2\hat{c}_p - \bar{c}_p$ (double the prediction minus the ground truth). By using the shifted cost, it provides a clear signal of where the model fails. The decision regret amplifies when the predictions are leaning towards a wrong direction. The SPO+ loss function is defined as:

$$\mathcal{L}_{SPO+}(\hat{p}, \bar{p}) = c_1^T x^* + c_2^T y^* + (2\hat{c}_p - \bar{c}_p)^T y^* + M \sum_i (2\hat{p}_i - \bar{p}_i) - z^*_{\text{Soft}}(2\hat{p} - \bar{p}; M). \quad (7)$$

The gradient of the SPO+ loss is a clean formula that equals to the difference between two schedules. It is written as:

$$\nabla_{\hat{c}_p} \mathcal{L}_{SPO+} = 2(y^* - y_{shifted}), \quad (8)$$

where $y_{shifted}$ represents the optimal decisions of the solver evaluated as the shifted cost $c_{shifted}$. It is actually the optimal solutions obtained by solving the problem $P_{\text{Soft}}(2\hat{p} - \bar{p}; M)$.

C.2 Perturbed Fenchel–Young Loss

The Fenchel–Young (FY) loss smooths non-differentiable optimization landscapes to construct informative gradients for end-to-end learning. While regularized FY formulations introduce penalty terms (e.g., quadratic or entropic) that often yield non-linear problems requiring specialized solvers, perturbed FY loss relies on cost-vector perturbations. We focus here on the perturbed FY loss. It measures the gap between the expected solution and the truth. The perturbed FY loss is defined as:

$$\mathcal{L}_{FY}(\hat{p}, \bar{p}) = z^*_{\text{Soft}}(\bar{p}; M) - \mathbb{E}_Z \left[z^*_{\text{Soft}}(\hat{p} + \epsilon Z; M) \right], \tag{9}$$

where Z is a zero-mean noise vector (e.g., standard Normal or Gumbel distribution) and $\epsilon > 0$ controls perturbation intensity. The gradient with respect to predicted costs evaluates to:

$$\nabla_{\hat{c}_p} \mathcal{L}_{FY} = y^* - \hat{y}_\epsilon, \tag{10}$$

where $\hat{y}_\epsilon = \mathbb{E}_Z \left[\arg z^*_{\text{Soft}}(\hat{p} + \epsilon Z; M) \right]$ represents the average solution. This gradient is exact whenever $\sum_i \hat{p}_i$ does not vary with ω, as is the case in the numerical case study (Appendix E), where $\hat{p}$ is produced by a per-group softmax with $\sum_j \hat{p}_{j,r} \equiv 1$; for predictions produced independently per coordinate (e.g., via a coordinate-wise sigmoid), an additional constant term analogous to the one applied to the SPO+ loss above should be included.

D Mathematical Formulations

D.1 Deterministic Energy-aware Lot-sizing Problem

This integrated production and energy supply problem addresses the operational planning of a manufacturing plant that must balance item production with volatile energy procurement. The system operates on two synchronized time scales: macro-periods for tracking item demand and inventory, and micro-periods to manage high-frequency energy price fluctuations and renewable generation. The primary goal is to minimize the total cost of startups, inventory holding, grid energy interactions, and penalties for any lost sales (unmet item demand) over a finite horizon.

The production system consists of a single resource that transitions between idle, setup, and manufacturing states to satisfy the demand for items. This resource requires energy for both the startup phase and the active processing of items. These energy requirements are met through a hybrid energy system that draws from the main grid and on-site renewable sources. A battery storage system is integrated to manage residual green energy and grid purchases, accounting for energy losses during grid transformation and battery cycles. Ultimately, the plan must simultaneously determine lot sizes, startup timings, and inventory or lost sale levels alongside energy procurement and storage setpoints.

Parameters and decision variables are given in Tables 1 and 2 respectively.

D.2 MILP Formulation

The problem is formulated as an MILP and is denoted as (LSP).

Objective function

$$\min \sum_{r=1}^{R} \sum_{j=1}^{J} f_j X_{j,r} + \sum_{t=1}^{T} \sum_{j=1}^{J} (a_j^H I_{j,t} + a_j^L L_{j,t}) + \sum_{r=1}^{R} \pi_r P_r \tag{11}$$

Production-oriented constraints

$$\sum_{j=0}^{J} Y_{j,r} = 1 \qquad \forall r \in \mathcal{R}, \tag{12}$$

$$X_{j,r} \geqslant Y_{j,r} - Y_{j,r-1} \qquad \forall j \in \mathcal{J}, r \in \mathcal{R}, \tag{13}$$

$$I_{j,t} = I_{j,t-1} + \sum_{r \in \mathcal{R}_t} Q_{j,r} - (d_{j,t} - L_{j,t}) \qquad \forall j \in \mathcal{J}, t \in \mathcal{T}, \tag{14}$$

$\mathcal{J}$	The set of products $\mathcal{J} = \{1, \ldots, J\}$; let $j = 0$ denote the dummy product corresponding to the idle state
$\mathcal{T}$	The set of macro-periods $\mathcal{T} = \{1, \ldots, T\}$
$\mathcal{R}$	The set of micro-periods $\mathcal{R} = \{1, \ldots, R\}$
$\mathcal{R}_t$	The set of micro-periods comprised in macro-period t
l	Length of a micro-period
$d_{j,t}$	Demand for item j at the end of macro-period t
f_j	Fixed startup cost for item j
a_j^H	Unit inventory holding cost for item j
a_j^L	Unit lost sales cost for item j
k_j	Unit production time for item j
e_j^F	Energy consumed during a startup for item j
e_j^M	Energy consumed for unit production of item j
q_r	PV availability in micro-period r
η^G, η^C, η^D	Efficiency of grid transformer / battery charging / battery discharging
u^B	Battery capacity
u^C, u^D	Battery charging / discharging limit within a micro-period

Table 1: Notations and parameters

$I_{j,t}$	Inventory of product j at the end of macro-period t
$L_{j,t}$	Lost sales of product j at the end of macro-period t
$Y_{j,r}$	Setup state of the resource at the end of micro-period r
$X_{j,r}$	Startup indicator: $X_{j,r} = 1$ if there is a startup for item j during r, $X_{j,r} = 0$ otherwise
$Q_{j,r}$	Quantity of item j produced in energy-oriented micro-period r
U_r	Energy used by the industrial plant in micro-period r
P_r	Electricity purchased from the grid in micro-period r
C_r	Electricity input to the battery within micro-period r
D_r	Electricity discharged from the battery within micro-period r
B_r	Electricity stored in the battery at the end of micro-period r

Table 2: Decision variables

$$k_j Q_{j,r} \leqslant l(Y_{j,r-1} + Y_{j,r}) \qquad \forall j \in \mathcal{J}, r \in \mathcal{R}, \tag{15}$$

$$\sum_{j=1}^{J} k_j Q_{j,r} \leqslant l \qquad \forall r \in \mathcal{R}, \tag{16}$$

$$Y_{j,0} = 0 \qquad \forall j \in \mathcal{J}, \tag{17}$$

$$I_{j,0} = \tilde{I}_{j,0} \qquad \forall j \in \mathcal{J}, \tag{18}$$

$$I_{j,T} \geqslant I_{j,0} \qquad \forall j \in \mathcal{J}, \tag{19}$$

We introduce here the family of production-oriented constraints equation 14-equation 19. Equations equation 12 ensure the machine is set up for at most one item at any time. It should be pointed out that the equality holds because we include the product 0, which corresponds to the machine's idle state. The relationship between setup and startup is captured by equation 13, triggering a startup if the setup state changes between periods. The inventory balance is maintained by equation 14. Inequalities equation 15 ensure that item j can be produced in micro-period r only if the machine is set up for j at the beginning or/and at the end of the period, which, combined with equation 15,

limits production to a maximum of two items per micro-period. Resource capacity is enforced by equation 16. Constraints equation 17-equation 19 define the initial and final conditions of the production system.

Energy-oriented constraints

$$U_r = \sum_{j=1}^{J} e_j^F X_{j,r} + \sum_{j=1}^{J} e_j^M Q_{j,r} \qquad \forall r \in \mathcal{R}, \tag{20}$$

$$U_r \leqslant \eta^G P_r + q_r + \eta^D D_r - \frac{C_r}{\eta^C} \qquad \forall r \in \mathcal{R}, \tag{21}$$

$$B_r = B_{r-1} + C_r - D_r \qquad \forall r \in \mathcal{R}, \tag{22}$$

$$B_r \leqslant u^B \qquad \forall r \in \mathcal{R}, \tag{23}$$

$$C_r \leqslant u^C \qquad \forall r \in \mathcal{R}, \tag{24}$$

$$D_r \leqslant u^D \qquad \forall r \in \mathcal{R}, \tag{25}$$

$$B_0 = \tilde{B}_0. \tag{26}$$

The second family of constraints are energy-oriented. Energy consumption, comprising startup and manufacturing requirements, is defined by equation 20. The micro-period energy balance is enforced by equation 21, which ensures that the plant's demand is met by a combination of grid purchases, renewable generation, and battery discharge, while accounting for losses. The battery state of charge is governed by equation 22, tracking the energy storage across micro-periods. Physical limitations are maintained through capacity constraints equation 23, charging/discharging rate limits equation 24-equation 25, and the initial state of charge definition equation 26.

Domain definition

$$X_{j,r}, Y_{j,r} \in \{0,1\} \qquad \forall j \in \mathcal{J}, r \in \mathcal{R}, \tag{27}$$

$$I_{j,t}, L_{j,t} \geqslant 0 \qquad \forall j \in \mathcal{J}, t \in \mathcal{T}, \tag{28}$$

$$Q_{j,r} \geqslant 0 \qquad \forall j \in \mathcal{J}, r \in \mathcal{R}, \tag{29}$$

$$C_r, D_r, P_r, B_r, U_r \geqslant 0 \qquad \forall r \in \mathcal{R}. \tag{30}$$

Finally, the definition domain of the decision variables are given by Constraints equation 27-equation 30.

D.3 Two-stage Stochastic Case

In this subsection, we extend the deterministic energy-aware lot-sizing problem into a two-stage stochastic programming case. We incorporate uncertainty regarding renewable energy generation and product demand, which are typically stochastic in real-world manufacturing environments. In the first stage, the plant manager makes "here-and-now" decisions regarding the production startup and setup states ($X_{j,r}$ and $Y_{j,r}$), which must be determined before the realization of uncertain parameters. In the second stage, "wait-and-see" recourse actions include production quantities ($Q_{j,r}$), inventory/lost sales levels ($I_{j,t}, L_{j,t}$), energy use (U_r), energy procurement (P_r), and battery operations (B_r, C_r, D_r). And they are optimized for each scenario $s \in S$ to minimize the expected total cost. The objective function is thus reformulated to minimize the sum of the first-stage startup costs and the expected value of second-stage costs across all weighted scenarios:

$$\min \sum_{j,r} f_j X_{j,r} + \mathbb{E}_{s \in S} \left[\sum_{j,t} (a_j^H I_{j,t,s} + a_j^L L_{j,t,s}) + \sum_{r} \pi_{r,s} P_{r,s} \right]$$

These first-stage variables $Y_{j,r}$ remain fixed across all scenarios, while the production-oriented constraints equation 14-equation 19 and energy-oriented constraints equation 20-equation 26 are replicated for each scenario s to ensure that local energy balances and demand satisfaction are feasible under varying conditions.

E NUMERICAL SETTINGS FOR CASE STUDIES

E.1 DETERMINISTIC CASE

This problem has a specific constraint equation 12, which says that the sum of the setup decisions, over all products (including the dummy product) in a given period r, equals 1. This is a set of hard constraints that the setup variables must satisfy. Therefore, in the final layer of GNN, instead of using a general sigmoid function, we use softmax function for each vector of output in r: $\hat{p}_{j,r} = \text{softmax}(h_r)_j$. With a slight abuse of notation, let h_r denote $[h_{0,r}, h_{1,r}, \ldots, h_{J,r}]$. Since $\sum_{j=0}^{J} Y_{j,r} = 1$ (Constraint equation 12) and, by construction of the softmax, $\sum_{j=0}^{J} \hat{p}_{j,r} = 1$, the group $\{Y_{j,r}\}_{j=0}^{J}$ is a *multiple-choice* (or *one-hot*) set for each period r. Writing $\hat{p}_r := [\hat{p}_{0,r}, \ldots, \hat{p}_{J,r}]$ and $Y_r := [Y_{0,r}, \ldots, Y_{J,r}]$, the mismatch $D_{\hat{p}}(y)$ (Section 3) restricted to this group simplifies to $\sum_{j=0}^{J}[(1-\hat{p}_{j,r})Y_{j,r} + \hat{p}_{j,r}(1-Y_{j,r})] = 2(1-\hat{p}_r^T Y_r)$: only the relative differences between the category probabilities determine which assignment is preferred, so a uniform $\hat{p}_r$ carries no preference at all, even if every probability is below $1/2$. We notice furthermore that the startup decisions can be easily calculated through inequalities equation 13 if all the setup decisions are already fixed. Moreover, once the setup decisions are fixed and satisfy the hard constraints equation 12, the remaining problem is always feasible. Thus, in what follows, we focus on predicting only the Y values using GNN model. In the generic form P, we have $y = \{Y\}$, $x = \{X, Q, I, L, U, P, B, C, D\}$.

We employ 6 GCN (graph convolutional network) layers in this GNN model. To evaluate the performance of the GNN model, we conducted several tests on two different datasets.

Let (T, J, N) denote an instance set with N randomly generated instances of size (T, J), i.e. producing J different items over a planning horizon of T macro-periods. We generate a dataset $\mathcal{D}$ containing 180 varied instances, which is used for training and validating. We create furthermore two different datasets $\mathcal{D}_1$ and $\mathcal{D}_2$ for testing.

- $\mathcal{D} = \{(2,2,30), (2,3,30), (2,4,30), (2,5,30), (4,2,30), (4,3,30)\}$.
- $\mathcal{D}_1 = \{(2,3,50), (2,4,50), (2,5,50), (4,2,50), (4,3,50), (4,4,50)\}$ is a varied instance set containing in total 300 instances of different sizes, varied from 2 to 4 macro-periods and comprising 2 to 5 products.
- $\mathcal{D}_2 = \{(8,2,10), (8,3,10), (8,4,10), (8,5,10)\}$ is an instance set containing in total 40 instances with 4-days time horizon and comprising 2 to 5 products.

All the instances in dataset $\mathcal{D}$ are solved to optimality using CPLEX.

To evaluate the predictive performance and generalization capability of the proposed GNN, dataset $\mathcal{D}$ is partitioned into two disjoint subsets: 80% of the instances are allocated to the training set and the rest 20% instances constitute the validation set. We use a random seed $seed = 42$ for reproducibility.

E.2 TWO-STAGE STOCHASTIC CASE

The dataset generation procedure is similar to the one described above for the deterministic case, except that we randomly generate the demands and renewable energy quantities for each scenario $s \in \mathcal{S}$.

We have demonstrated in the deterministic case that the benchmark hard fixing strategy performs badly compared to both benchmark penalization strategy and soft penalization strategy. Therefore, in what follows, we mainly focus on the computational results obtained with benchmark penalization and soft penalization strategies.

Let (T, J, S, N) denote an instance configuration containing N randomly generated instances characterized by J different items produced over a planning horizon of T macro-periods under S scenarios. We construct the following datasets for training ($\mathcal{DS}$) and testing ($\mathcal{DS}_1, \mathcal{DS}_2$):

- $\mathcal{DS} = \{(2,2,5,20), (2,3,5,20), (2,4,5,20), (4,2,5,20), (4,3,5,20), (2,2,10,20), (2,3,10,20), (2,4,10,20), (4,2,10,20), (4,3,10,20)\}$ contains 200 instances in total

across diverse problem dimensions, varying from 2 to 4 macro-periods, 2 to 4 products, and 5 to 10 scenarios.

- $\mathcal{DS}_1 = \{(2,2,5,20),(2,3,5,20),(2,4,5,20),(4,2,5,20),(4,3,5,20),(2,2,10,20),(2,3,10,20),(2,4,10,20),(4,2,10,20),(4,3,10,20)\}$ contains 200 instances of same sizes as $\mathcal{DS}$.
- $\mathcal{DS}_2 = \{(6,2,5,5),(6,3,5,5),(6,4,5,5),(6,5,5,5),(6,2,10,5),(6,3,10,5),(6,4,10,5),(6,5,10,5)\}$ is a test set comprising 40 larger instances in total, characterized by a 6-period time horizon ($T = 6$), 2 to 5 products, and 5 to 10 scenarios.

F Soft Penalization Term

A natural candidate is to derive a penalization term from the cross-entropy loss used during training. Given a prediction $\hat{p}_i$, one could consider a penalty of the form:

$$-\big(y_i \log(\hat{p}_i) + (1 - y_i)\log(1 - \hat{p}_i)\big). \tag{31}$$

At evaluation time, since $\hat{p}_i$ is fixed, this reduces to a linear function in y_i with coefficient proportional to the log-odds $\log\left(\frac{1-\hat{p}_i}{\hat{p}_i}\right)$[1]. However, this formulation has an important drawback: when $\hat{p}_i$ is close to 0 or 1, the log-terms become very large in magnitude, effectively leading to near-hard decisions. As a result, even slightly miscalibrated predictions may induce overly strong biases, potentially degrading solution quality.

Some preliminary results are obtained for the deterministic case study using soft penalization strategy with either log-based or proposed linear penalization term. The relative gap and relative run time distribution are depicted in Figure 4.

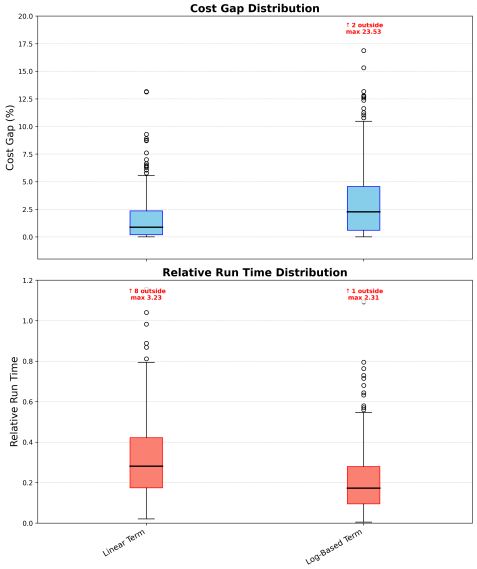


Figure 4: Performance comparison using soft penalizing strategy for dataset $\mathcal{D}_1$

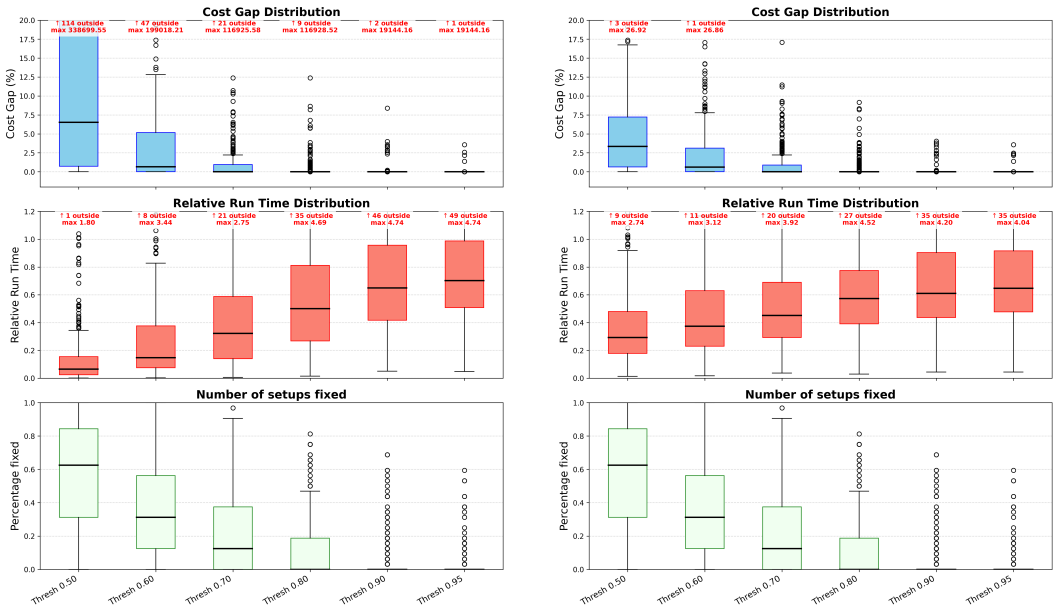
We observe from Figure 4 that the soft penalizing strategy leads to reasonable relative gap below 25% with both penalization term. The linear penalization results to smaller cost gap compared to the log term, while gives competitive run time that is in average 0.1x-0.4x times faster than the reference solving time.

[1]Indeed, the cross-entropy term can be rewritten as $-\big(y_i \log(\hat{p}_i) + (1 - y_i)\log(1 - \hat{p}_i)\big) = -\log(1 - \hat{p}_i) + y_i\big(\log(1 - \hat{p}_i) - \log(\hat{p}_i)\big) = -\log(1 - \hat{p}_i) + y_i \log\left(\frac{1-\hat{p}_i}{\hat{p}_i}\right)$, which is affine in y_i. Since $y_i \in \{0, 1\}$, this yields a linear objective contribution with coefficient $\log\left(\frac{1-\hat{p}_i}{\hat{p}_i}\right)$.

G PERFORMANCE UNDER "PREDICT-THEN-OPTIMIZE" PARADIGM

Under the deterministic setup, we evaluate benchmark fixing and penalization strategies under the Predict-then-Optimize paradigm. We train the model on $\mathcal{D}$ using the classical binary cross entropy (BCE) loss and then test the performance of the model on the small instance dataset $\mathcal{D}_1$.

Benchmark fixing We first evaluate with the fixing strategy. To satisfy the hard constraint, we only fix in each period r the setup variables with the highest probability, namely, if $\hat{p}_{j^*,r} > \tau$, $j^* = \arg\max_{j\in\{0\}\cup\mathcal{J}} \hat{p}_{j,r}$, then we fix $y_{j^*,r}$ to 1. The threshold is set to $\tau \in \{0.5, 0.6, 0.7, 0.8, 0.9, 0.95\}$. The resulting restricted MILP problems are also solved directly by CPLEX. The gap, relative run time, and The percentage of the number of setups fixed are shown in Figure 5a. Each boxplot illustrates the statistical distribution across this entire test set $\mathcal{D}_1$.



(a) Benchmark fixing strategy

(b) Benchmark penalization strategy

Figure 5: Performance comparison using benchmark fixing and penalization strategy for dataset $\mathcal{D}_1$

From the figure, we observe a clear trade-off between variable fixing and solution accuracy. The proportion of fixed setup variables declines from a mean of nearly 60% to almost 0% as the threshold τ is increased from 0.5 to 0.95. As a direct consequence of the expanded search space, the relative solving time increases accordingly. While the cost gap can be extremely large in the worst case (more than 33 times larger than the true optimal objective), it decreases as the threshold increases. The substantial relative cost gap suggests that even if we are more than 95% sure about the decisions we make, a few bad decisions in the setup may drastically alter the solution, leading to a cost gap of 19,144% in a worst case scenario. This is due to the fact that we set a high penalty for each unit lost sales, a small change in the production schedule can result in an inability to satisfy primary demand, causing the objective function to escalate by several orders of magnitude compared to the true optimum.

Benchmark penalization We then evaluate the benchmark penalization strategy (see Figure 5b), setting $M = 100$ as the default across all penalization-based approaches. A more detailed investigation into the impact of M on the trade-off between decision quality and relative run time is provided in Section 4.2. Comparing Figure 5b with Figure 5a reveals that benchmark penalization significantly reduces the cost gap relative to benchmark fixing, without sacrificing solver execution speed.