SOLVER-LABEL-FREE PRETRAINING FOR IN-CONTEXT PRIMAL SOLUTION PREDICTION IN ILPS

Anonymous authors
Paper under double-blind review

ABSTRACT

Integer linear programs (ILPs) are a foundational tool for modeling discrete decision-making problems, but solving large-scale ILPs remains computationally challenging due to their NP-hardness. This has motivated machine learning (ML) methods that predict the values of the variables in high-quality solutions, a task known as *primal solution prediction*. These predictions then guide an exact solver, which narrows its search to promising regions of the solution space. However, most of these methods are trained on solver-labeled instances, which requires hundreds of solver runs, each taking up to an hour, for every problem class. We propose **PrimalPFN**, a framework that instead uses in-context learning (ICL), predicting solutions for a new instance from a few solved instances of the same class given as input, without retraining. Following prior-data fitted networks (PFNs), we acquire the ICL capability without any solver labels by pretraining on synthetic tasks, in which instances from standard problem generators are labeled by randomly sampled rules that favor feasible, high-quality assignments. At inference time, an exact solver provides the solutions of the few input instances, so the model predicts the solution that solver would find for the new instance. For each problem class, PrimalPFN requires only two solver-labeled instances rather than hundreds, and shares them across all new instances of that class. Across three ILP problem classes, it finds solutions comparable in quality to those of class-specific supervised predictors on two classes and better ones on the third, with a gap to the best known solutions about 40% smaller. Compared with an unsupervised baseline that likewise requires no solver labels for training, PrimalPFN reduces the gap by 39–79%. Moreover, given the same labeled training instances as the supervised predictors, PrimalPFN attains a lower average gap than them on all three classes.

1 INTRODUCTION

Integer linear programs (ILPs) optimize a linear objective over integer variables subject to linear constraints. This formulation provides a unified framework for a wide range of decision-making problems across logistics, scheduling, network design, and energy systems (Klose & Drexl, 2005; Dauzère-Pérès et al., 2024; Crainic et al., 2022; Anjos & Conejo, 2017). Despite the broad practical importance of ILPs, solving them in general remains computationally challenging due to their combinatorial nature and NP-hardness. In many practical settings, ILP instances arising from the same application, which together form a *problem class*, exhibit recurring structural patterns, creating opportunities for learning-based methods to leverage information shared across instances (Gasse et al., 2019; Bengio et al., 2021). Building on this observation, a growing body of work has explored machine learning (ML) approaches that predict the values of variables in high-quality solutions, a task commonly referred to as *primal solution prediction* (Ding et al., 2020). These predictions can be used to restrict or guide the search of an exact ILP solver, for example by limiting the search to a neighborhood of the predicted assignment, allowing the solver to focus its computational effort on promising regions.

The most widely used approaches to primal solution prediction rely on supervised learning, which requires solver-labeled training data (Nair et al., 2021; Han et al., 2023; Liu et al., 2025; Pu et al., 2025). A typical protocol labels about 300 instances per problem class (e.g., 240 for training and 60 for validation) by running an exact solver such as Gurobi (Gurobi Optimization, LLC, 2026) or SCIP (Hojny et al., 2025) on each instance for up to one hour, amounting to up to 300 CPU-hours

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.

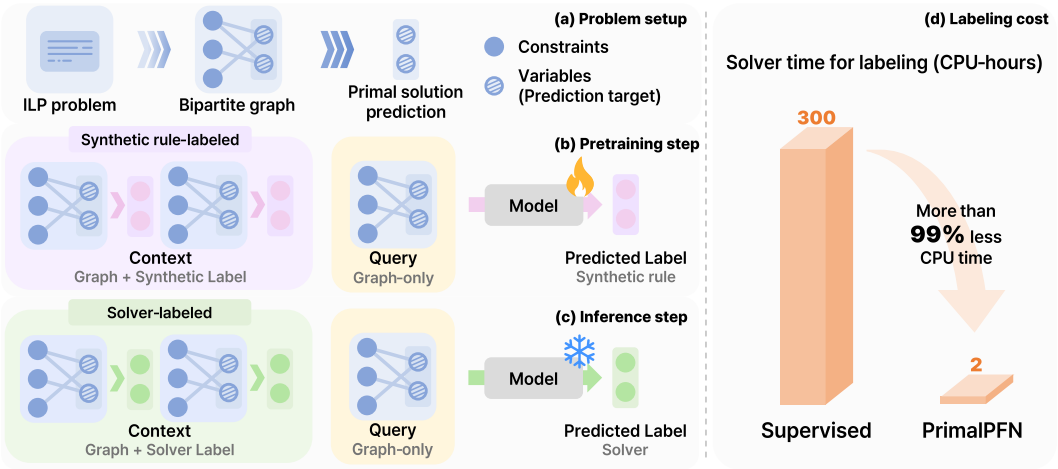

Figure 1: Overview of PrimalPFN. (a) An ILP instance is represented as a bipartite graph of constraint and variable nodes, and a binary value is predicted for each variable. (b) Pretraining uses only synthetic labels. In each episode, generated instances are labeled by a randomly sampled rule, which the model learns to infer from the labeled context and apply to the unlabeled query. (c) At inference, the frozen model receives solver-labeled context instances of the target class, so the solver's labeling plays the role of the rule. (d) Two solver-labeled instances per class, shared across all test instances of that class, reduce the solver time for labeling from up to 300 to 2 CPU-hours.

of solver computation per class (Han et al., 2023; Liu et al., 2025). Since Gurobi often exhausts this one-hour budget on challenging instances (Huang et al., 2024), faster hardware does not substantially reduce this cost. More importantly, this cost must be paid again for every new problem class, and it grows further when more training instances are used or when longer solver budgets are allocated to obtain higher-quality labels. DiffILO takes an important step toward alleviating this burden by training without solver labels, although its optimization can become trapped in suboptimal solutions, a limitation acknowledged in the original work (Geng et al., 2025). These observations motivate a research question: can we avoid extensive solver labeling while still adapting to the solver's behavior from only a few solved instances?

To answer this question, we adopt the in-context learning (ICL) paradigm, in which a pretrained model adapts to a new task from a few solved examples given as input, called the *context*, without any parameter updates. In our setting, a task is the solver's labeling of instances from a target distribution (e.g., instances of one problem class with similar size and structure), and the context consists of a few instances from this distribution solved by the solver. Prior-data fitted networks (PFNs) (Müller et al., 2022; Hollmann et al., 2023) have recently shown that ICL can be acquired by pretraining purely on synthetic data sampled from a designed prior, and that it transfers to real data. This recipe suits primal solution prediction particularly well, since ILP instances can be generated at scale (Prouvost et al., 2020) and the criterion a solver optimizes, namely the objective value subject to feasibility, can be evaluated cheaply for any assignment. Synthetic labeling rules that optimize this criterion thus act as plausible stand-ins for the solver's labeling, leaving solver solutions to inference time. This addresses both sides of the dilemma between costly supervision and label-free training, because pretraining avoids solver labeling and the few solver solutions used as context supply the supervision that unsupervised methods lack.

Building on this, we introduce **PrimalPFN** (see Figure 1), which represents each ILP instance as a bipartite graph of variables and constraints and, given the context and an unsolved *query* instance drawn from the same distribution, predicts the value of each variable in the query. To pretrain PrimalPFN without solver labels, we generate a large number of synthetic tasks, each consisting of ILP instances labeled by a randomly sampled rule. The rule first assigns values based on the graph structure and then applies an optimization-aware refinement that moves the assignment toward feasible, high-quality ones. Since the rule varies across tasks, the model learns to infer it from the context and apply it to the query, so that at inference time the solver's labeling acts as one such rule.

Experiments on three ILP problem classes show that, with only two solver-labeled instances per class, PrimalPFN performs comparably to or better than supervised predictors trained on hundreds of labeled instances and outperforms the solver-label-free baseline DiffILO. Context perturbation analyses further show that it relies on the correspondence between the context instances and their solver labels, rather than on graph structure alone.

We summarize our contributions as follows:

- We propose PrimalPFN, which extends PFN-style in-context learning to primal solution prediction by constructing the context across solved instances rather than within a single graph as in existing graph PFNs, and by pretraining on a solver-free prior whose labeling rules refine graph-based initial assignments toward feasible, high-quality ones. Ablations show that this optimization-aware refinement drives the transfer to solver-labeled contexts.
- On three ILP problem classes with Gurobi, PrimalPFN uses only two solver-labeled instances per class, yet on two classes it finds solutions comparable in quality to those of class-specific supervised predictors that use 300 labeled instances, and on the third it finds better ones, with a gap to the best known solutions about 40% smaller. It also reduces the gap of the unsupervised baseline DiffILO by 39–79%.
- We show that PrimalPFN remains effective across labeling budgets. Supervised predictors need from about 15 to more than 240 labeled training instances to match the solution quality that PrimalPFN attains with two. Conversely, with Gurobi, additionally pretraining PrimalPFN on their labeled training instances yields the best average solution quality on all three classes, with clear gains on two.

2 Preliminaries

2.1 Integer Linear Programming

An integer linear program (ILP) can be written in the following general form:

$$\min_{\mathbf{x}} \ \mathbf{c}^\top \mathbf{x}, \quad \text{s.t. } A\mathbf{x} \le \mathbf{b}, \ \ \mathbf{l} \le \mathbf{x} \le \mathbf{u}, \ \ \mathbf{x} \in \mathbb{Z}^n, \tag{1}$$

where $\mathbf{x} \in \mathbb{Z}^n$ denotes the decision variable vector constrained to be integer-valued, $\mathbf{c} \in \mathbb{R}^n$ is the objective coefficient vector, $A \in \mathbb{R}^{m \times n}$ and $\mathbf{b} \in \mathbb{R}^m$ define the m linear constraints, and $\mathbf{l} \in (\mathbb{R} \cup \{-\infty\})^n$, $\mathbf{u} \in (\mathbb{R} \cup \{+\infty\})^n$ specify lower and upper bounds on $\mathbf{x}$. Maximization problems can be converted to this form by negating $\mathbf{c}$. A solution $\mathbf{x}$ is *feasible* if it satisfies all constraints, and primal solution prediction aims to predict the values of $\mathbf{x}$ in a high-quality feasible solution. We focus on binary ILPs, i.e., $\mathbf{x} \in \{0,1\}^n$, noting that bounded integer variables can be represented using standard binary encodings.

2.2 Bipartite Graph Representation for Integer Linear Programming

An ILP instance in Eq. 1 can be represented as a weighted bipartite graph $\mathcal{G} = (\mathcal{V}_{\text{var}}, \mathcal{V}_{\text{con}}, \mathcal{E})$, where the n nodes in $\mathcal{V}_{\text{var}}$ correspond to variables and the m nodes in $\mathcal{V}_{\text{con}}$ correspond to constraints. An edge connects constraint node j and variable node i whenever $A_{ji} \neq 0$, with weight A_{ji}. Each node additionally carries features derived from the instance, such as the objective coefficient c_i for a variable node and the right-hand side b_j for a constraint node.

2.3 Posterior Predictive Distribution in Supervised Learning

Consider a supervised learning problem with inputs $x \in \mathcal{X}$ and outputs $y \in \mathcal{Y}$. A Bayesian formulation assumes a hypothesis space Φ, where each hypothesis $\phi \in \Phi$ specifies a conditional distribution $p(y \mid x, \phi)$ describing how outputs are generated from inputs, and prior knowledge about these hypotheses is represented by a prior distribution $p(\phi)$. Given a context dataset $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^N$, Bayesian inference updates the prior to the posterior

$$p(\phi \mid \mathcal{D}) \propto p(\phi) \prod_{i=1}^{N} p(y_i \mid x_i, \phi). \tag{2}$$

For a new query input x_{query}, the posterior predictive distribution (PPD) marginalizes predictions over this posterior:

$$p(y_{\text{query}} \mid x_{\text{query}}, \mathcal{D}) = \int_{\Phi} p(y_{\text{query}} \mid x_{\text{query}}, \phi)\, p(\phi \mid \mathcal{D})\, \mathrm{d}\phi. \tag{3}$$

Thus, the context provides evidence about which hypotheses are plausible, and the resulting posterior is used to make predictions for the query.

2.4 Prior-Data Fitted Networks

PFNs amortize this Bayesian inference procedure into the parameters of a neural network. Rather than explicitly computing the posterior $p(\phi \mid \mathcal{D})$ for every new dataset, a PFN with parameters θ is pretrained on synthetic tasks. In each task, a hypothesis $\phi \sim p(\phi)$ is sampled, and a context dataset $\mathcal{D}$ and a query pair $(x_{\text{query}}, y_{\text{query}})$ are generated under ϕ. Letting $p_\theta(y \mid x, \mathcal{D})$ denote the predictive distribution produced by the network, it is trained to minimize

$$\mathcal{L}(\theta) = \mathbb{E}_{\phi,\, \mathcal{D},\, (x_{\text{query}}, y_{\text{query}})} \left[-\log p_\theta(y_{\text{query}} \mid x_{\text{query}}, \mathcal{D}) \right], \tag{4}$$

whose minimizer is the PPD in Eq. 3, so that a trained network approximates it:

$$p_\theta(y_{\text{query}} \mid x_{\text{query}}, \mathcal{D}) \approx p(y_{\text{query}} \mid x_{\text{query}}, \mathcal{D}). \tag{5}$$

Consequently, a pretrained PFN can adapt its predictions to the hypothesis underlying a new context dataset in a single forward pass.

3 Method

3.1 In-Context Formulation for Primal Solution Prediction

We cast primal solution prediction as amortized Bayesian inference in the PFN framework. The input is an ILP instance represented as a bipartite graph $\mathcal{G}$, the output is a binary assignment $\mathbf{y} \in \{0,1\}^{|\mathcal{V}_{\text{var}}|}$ to the decision variables $\mathbf{x}$ in Eq. 1, and each hypothesis ϕ is a labeling rule q_ω that maps an instance to such an assignment, with parameters drawn from a prior $p(\omega)$ described in Section 3.2.

In each pretraining episode, we sample ILP instances from an instance distribution $p(\mathcal{G})$ and partition them into a context set $\mathcal{G}_{\text{ctx}} = \{\mathcal{G}^{(i)}_{\text{ctx}}\}_{i=1}^{N_{\text{ctx}}}$ and a query set $\mathcal{G}_{\text{query}} = \{\mathcal{G}^{(j)}_{\text{query}}\}_{j=1}^{N_{\text{query}}}$. We then sample rule parameters $\omega \sim p(\omega)$, shared across all instances in the episode, and apply q_ω to obtain context labels $\mathbf{y}_{\text{ctx}}$ and query labels $\mathbf{y}_{\text{query}}$. Given the labeled context $\mathcal{D} := (\mathcal{G}_{\text{ctx}}, \mathbf{y}_{\text{ctx}})$, the network predicts the query labels and is trained with the PFN objective in Eq. 4. Since modeling the joint distribution over all $2^{|\mathcal{V}_{\text{var}}|}$ assignments is intractable and the downstream search uses only variable-wise marginals, we factorize the predictive distribution over variables, so that for each query variable v,

$$p_\theta(y_v = 1 \mid \mathcal{G}_{\text{query}}, \mathcal{D}) \approx p(y_v = 1 \mid \mathcal{G}_{\text{query}}, \mathcal{D}). \tag{6}$$

Dependencies among variables, such as those induced by the constraints, are left to the solver in the downstream search. At test time, an exact solver labels the context in place of q_ω, so its labeling plays the role of the rule.

3.2 Prior Generation

The effectiveness of a PFN depends on how well its prior reflects the tasks encountered at test time. Our prior consists of two components, *instance generation* and *optimization-aware label generation*.

Instance Generation We generate ILP instances with the Ecole generators (Prouvost et al., 2020) for the three problem classes considered in our experiments, as illustrated in Figure 2(a). To expose the model to a broad range of structures and constraint tightness, we randomize the generator parameters (e.g., number of variables and constraint density) across pretraining episodes, while all instances within an episode come from the same class. Since pretraining requires no solver labels, instances can be generated in unlimited quantities at negligible cost. Because the labeling rules below are defined for any binary ILP and do not depend on the problem class, we pretrain a single model on episodes pooled from all three classes and rely on the context to convey the solver's labeling of each class.

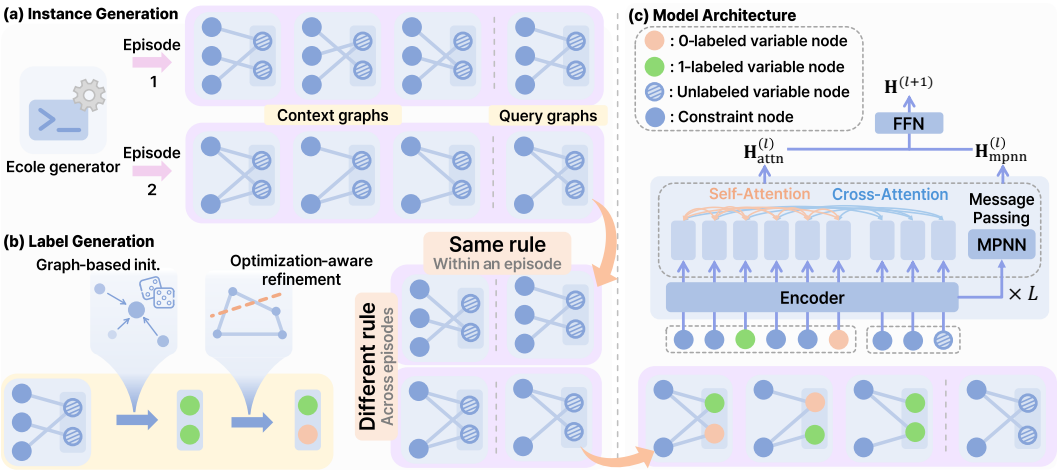

Figure 2: Pretraining pipeline and architecture of PrimalPFN. (a) In each episode, ILP instances are generated with Ecole (Prouvost et al., 2020) and divided into context and query sets. (b) A labeling rule, consisting of a graph-based initialization by a randomly initialized MPNN and an optimization-aware refinement, is sampled anew for each episode and labels all instances in that episode. (c) Each layer of the model combines attention across instances, namely self-attention among context tokens and cross-attention from query to context tokens, with message passing within each instance.

Optimization-Aware Label Generation Following the principle of synthetic priors in PFNs, we construct an episode-level labeling rule q_ω with randomly sampled parameters $\omega \sim p(\omega)$ (see Figure 2(b)). The rule consists of a *graph-based initialization*, in which a randomly initialized message-passing neural network (MPNN) scores the variables to produce an initial binary assignment, and an *optimization-aware refinement*, which improves this assignment by local search on a penalized objective that combines the objective value with constraint violations, similar to the formulation of DiffILO (Geng et al., 2025). Randomizing both steps, including the weight of the violation penalty, yields diverse rules with different objective–feasibility trade-offs. We describe our algorithm in Appendix B.2.1.

3.3 Model Architecture

Our architecture builds on the dual-branch design of NodePFN (Choi et al., 2026), which pairs an attention branch with a message-passing branch. Applying this design to ILPs requires handling three aspects that existing graph PFNs are not designed for: (1) a bipartite structure in which variable and constraint nodes play asymmetric roles and only variable nodes carry labels, (2) signed edge weights A_{ji} that encode the constraint coefficients, and (3) context drawn from other instances rather than from labeled nodes of the same graph.

We address these aspects with two branches that are computed in parallel in every layer and then fused. The *local MPNN branch* performs bipartite message passing within each instance, using separate encoders for variable and constraint features and incorporating the signed coefficients into the edge-to-node updates. The *context–query attention branch* transfers information across instances. Context labels enter as input features of the context variable nodes, context tokens attend to one another, and query tokens attend to the context tokens through cross-attention without attending to other queries. As a result, the representation of each query variable reflects both its local constraint structure and the labels of related variables in the context. Figure 2(c) illustrates the architecture, and Appendix A.2 provides the details.

3.4 Training Objective

We instantiate the PFN objective in Eq. 4 for binary assignments. Following common practice in primal solution prediction (Nair et al., 2021; Han et al., 2023), the model predicts a marginal probability $\hat{y}_{\theta,v} = p_\theta(y_v = 1 \mid \mathcal{G}_{\text{query}}, \mathcal{D})$ for each query variable v, as in Eq. 6. The negative log-likelihood

in Eq. 4 then reduces to a binary cross-entropy (BCE) loss, a proper scoring rule whose minimizer for each variable is the marginal PPD in Eq. 6, and we average it over the query variables of each episode:

$$\mathcal{L}(\theta) = \mathbb{E}\left[\frac{1}{|\mathcal{V}_{\text{query}}|}\sum_{v\in\mathcal{V}_{\text{query}}} \ell_{\text{BCE}}\left(y_v, \hat{y}_{\theta,v}\right)\right], \tag{7}$$

where the expectation is over episodes, i.e., instances $\mathcal{G} \sim p(\mathcal{G})$ and rule parameters $\omega \sim p(\omega)$, $\mathcal{V}_{\text{query}}$ denotes the variable nodes of all query instances in the episode, and y_v is the label assigned to v by q_ω.

3.5 Inference Procedure

In-Context Primal Prediction At test time, given a target distribution, we solve N of its instances with an exact solver and use them as the context $\mathcal{D} = (\mathcal{G}_{\text{ctx}}, \mathbf{y}_{\text{ctx}})$. This is the only stage of our pipeline that requires solver labels, and the same context is reused for every query instance from that distribution, so its labeling cost is shared across all of them. For each query instance $\mathcal{G}_{\text{query}}$, PrimalPFN outputs the marginal probability $\hat{y}_{\theta,v} = p_\theta(y_v = 1 \mid \mathcal{G}_{\text{query}}, \mathcal{D})$ of each variable v in a single forward pass.

Downstream Primal Heuristic To turn the predicted probabilities into solutions, we use Predict-and-Search (PS) (Han et al., 2023) as the downstream primal heuristic. PS selects the k_0 and k_1 variables most confidently predicted as 0 and 1, respectively, and invokes an ILP solver on the subproblem in which at most Δ of these variables may deviate from their predicted values (Appendix B.3). The model thus only needs to identify promising regions of the search space, while the solver searches for feasible, high-quality solutions within them. Although we evaluate with PS, the predicted marginals can in principle guide other primal heuristics that take variable-wise probabilities as input, such as Neural Diving (Nair et al., 2021) and Apollo-MILP (Liu et al., 2025).

4 Experiments

4.1 Experimental Setup

Datasets & Benchmarks We evaluate PrimalPFN on three problem classes from D-MIPLIB (Huang et al., 2024): set covering (SC) (Balas & Ho, 1980), independent set (IS) (Bergman et al., 2016), and combinatorial auctions (CA) (Leyton-Brown et al., 2000). We choose these classes following DiffILO, a representative learning-based primal heuristic for ILPs, since PrimalPFN targets binary ILPs and only a few D-MIPLIB classes are pure binary ILPs. Following prior work (Han et al., 2023; Geng et al., 2025), we use 240 training, 60 validation, and 100 test instances from the single instance distribution of each class, and obtain solver labels by running Gurobi for up to one hour per instance. To simulate a limited labeling budget, we draw $N = 2$ instances uniformly at random from the training split of each class as the context for PrimalPFN, matching the pretraining episodes ($N_{\text{ctx}} = 2$, $N_{\text{query}} = 1$), and reuse it for all 100 test instances of that class. Although few in number, these instances are large, so the context contains 8,000 to 12,000 labeled variables (Table 5), unlike tabular PFNs in which each context example carries a single label.[1]

Baselines We compare PrimalPFN with two learning-based baselines, Predict-and-Search (PS) (Han et al., 2023) and DiffILO (Geng et al., 2025), and with standalone Gurobi. PS trains a GNN solution predictor on solver-generated solutions, whereas DiffILO trains a GNN predictor without solution labels by optimizing a differentiable reformulation of the ILP. Following their standard settings, PS and DiffILO are trained separately for each problem class on the 240 training instances, with and without solver labels, respectively, and use the 60 validation instances for model selection. PrimalPFN is pretrained once on synthetically labeled instances from all three problem classes and uses the $N = 2$ context instances described above for each class, without using the validation instances. We also report a variant, denoted *w/ solver labels*, that is pretrained on a mixture of the synthetic tasks and tasks built from the same 240 solver-labeled training instances as PS, and that uses the same context as PrimalPFN.

[1] Appendix D.5 shows that the results vary little across four additional random context draws.

Table 1: Main results with Gurobi under a 1,000-second time limit. Best and second-best results are shown in bold and underlined, respectively.

	SC (BKS: 168.88)		IS (BKS: 2,115.66)		CA (BKS: 116,309.51)	
	Obj ↓	Gap(%) ↓	Obj ↑	Gap(%) ↓	Obj ↑	Gap(%) ↓
Gurobi (1000s)	169.89	0.59	1,895.97	10.39	115,093.08	1.04
Gurobi (3600s)	169.46	0.34	2,093.02	1.07	115,367.40	0.81
PS	<u>169.20</u>	<u>0.18</u>	2,104.38	0.53	115,525.21	0.67
DiffILO	169.54	0.38	2,084.05	1.49	112,709.10	3.09
PrimalPFN	169.28	0.23	<u>2,108.91</u>	<u>0.32</u>	<u>115,559.11</u>	<u>0.64</u>
w/ solver labels	**169.18**	**0.17**	**2,110.40**	**0.25**	**115,821.32**	**0.42**

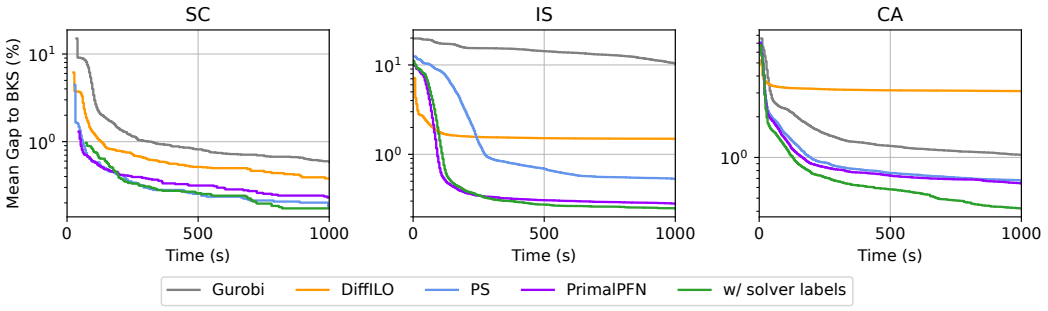

Figure 3: Gap (Eq. 8) over solving time under a 1,000-second time limit with Gurobi, averaged over the 100 test instances.

Evaluation Protocol Each method is run with a solver time limit of 1,000 seconds, and we additionally report Gurobi with a 3,600-second limit as a reference. Prediction time, below 3 seconds per instance for all learning-based methods, is not counted toward the time limit (Appendix D.6). We report the final objective value (Obj) and the relative gap to the best known solution (BKS),

$$\text{Gap}\,(\%) = \frac{|\text{Obj} - \text{BKS}|}{|\text{BKS}|} \times 100, \tag{8}$$

both computed per instance and averaged over the 100 test instances. For each test instance, the BKS is the best solution found by Gurobi within 3,600 seconds or by any evaluated method, since these methods often improve on the former. Table 1 reports the average BKS of each benchmark in its header. The three learning-based methods differ only in how the variable probabilities are predicted. The probabilities from PS's supervised GNN, DiffILO, or PrimalPFN are passed to the same search procedure of Predict-and-Search, whose trust-region radius is selected for each method on held-out instances generated separately from the benchmark splits (Appendix B.3), and the resulting restricted problem is solved with Gurobi.

4.2 Main Results

Solving Performance Table 1 summarizes the results, and Appendix D.2 reports the corresponding per-instance win/draw/loss counts. Using only two solver-labeled instances per class, PrimalPFN attains a gap of 0.32% on IS, compared with 0.53% for PS trained on 240 labeled instances, and finds better solutions on 75 of 100 instances. Across five random context sets, its gap on IS ranges from 0.30% to 0.32%, all below that of PS (Appendix D.5). On SC and CA, its gaps are comparable to those of PS (0.23% vs. 0.18% and 0.64% vs. 0.67%). On CA, it wins on 53 instances, and on SC, where most instances end in ties (69 of 100), it wins on 12 and loses on 19. PrimalPFN also outperforms DiffILO on all benchmarks, reducing its gap by 39%, 79%, and 79% on SC, IS, and CA, respectively, and finding better solutions on all 100 IS and CA instances. With the same 240 solver-labeled instances as PS, the *w/ solver labels* variant further lowers the gap on all benchmarks, achieving the lowest gaps overall and finding better solutions than PS on 76 IS and 68 CA instances.

Table 2: Effect of the context labels on the predictions and the resulting solutions. Since the gap also reflects the downstream search, which can mask changes in the predictions, we additionally report the AUROC of the predicted marginals against the BKS solution of each test instance to isolate the effect of each intervention on the predictions. Prior uses labels generated by the prior, Cross-class uses solver-labeled instances from one of the other two classes, chosen at random, and Shuffle and Random permute or randomize the solver labels. "infeas." means that no test instance has a feasible solution.

	SC		IS		CA	
	AUROC ↑	Gap(%) ↓	AUROC ↑	Gap(%) ↓	AUROC ↑	Gap(%) ↓
Solver	0.997	0.23	0.876	0.32	0.878	0.64
Prior	0.996	0.27	0.733	0.44	0.573	3.12
Cross-class	0.891	171.80	0.844	0.50	0.801	8.01
Shuffle	0.740	392.09	0.771	infeas.	0.721	10.83
Random	0.621	712.58	0.778	infeas.	0.614	27.08

Additional results with the open-source solver SCIP show that PrimalPFN attains the lowest gap on SC and IS, 44% and 24% below PS (Appendix D.1).

Gap over Time Figure 3 plots the gap in Eq. 8, averaged over the test instances, throughout the 1,000-second run, where a faster decrease indicates that high-quality solutions are found earlier. On IS, PrimalPFN and its *w/ solver labels* variant reduce the gap sharply early in the run, whereas PS begins to decline later and remains above both until the end. DiffILO decreases early on IS and CA but then plateaus at a higher gap, and standalone Gurobi remains above 10% on IS throughout. On SC, the curves of PS, PrimalPFN, and *w/ solver labels* largely overlap. On CA, PS and PrimalPFN overlap while *w/ solver labels* stays below both, consistent with Table 1.

4.3 Analysis

Effect of Context Labels To test whether PrimalPFN relies on the solver labels in the context rather than on the context graphs alone, we vary the context labels while keeping the context instances fixed, except for Cross-class, which replaces them with solver-labeled instances from another class (Table 2). Replacing the solver labels with labels generated by the prior increases the gap on all benchmarks, most notably on CA, and using solver labels from another problem class increases it drastically on SC and CA. Shuffling the labels across variables within each context instance, which preserves their frequency but breaks their correspondence with the instances, reduces AUROC to 0.72–0.77 and leaves every IS instance infeasible. Random labels increase the gap the most. The gap thus increases as the context labels move away from the solver's labeling, indicating that PrimalPFN exploits the correspondence between instances and their solver labels (Appendix D.3). AUROC and the gap need not agree, since PS fixes only the most confident predictions. For instance, prior-generated labels yield the lowest AUROC on IS and CA, yet a smaller gap than cross-class, shuffled, or random labels. Notably, even with prior-generated labels and no solver labels at all, PrimalPFN outperforms DiffILO on SC and IS and performs comparably on CA.

Prior Ablation For computational efficiency, we pretrain the full prior and two variants for 200 epochs each under otherwise identical settings (Table 3). Removing the optimization-aware refinement increases the gap from 0.31% to 0.47% on SC and from 0.79% to 4.56% on CA and leaves 9 of 100 IS instances infeasible. Replacing the graph-based initialization with random initialization yields nearly identical gaps. This indicates that the optimization-aware refinement, rather than graph-dependent labeling as in existing graph PFN priors, drives the transfer to solver-labeled contexts. We nevertheless retain the graph-based initialization, as it adds negligible cost.

Table 3: Prior ablation (Gap, %) with 200 pretraining epochs. *9 of 100 instances infeasible, averaged over the rest.

Prior	SC	IS	CA
Full	0.31	0.38	0.79
Random init.	0.31	0.36	0.79
w/o refinement	0.47	1.68*	4.56

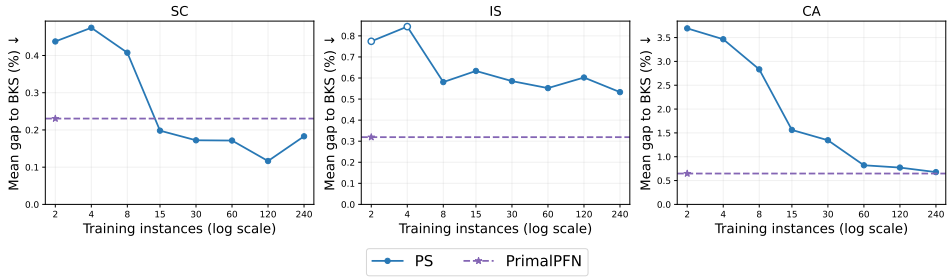

Figure 4: Gap of PS trained on varying numbers of solver-labeled instances (one run each), compared with PrimalPFN using two as context (star, extended as a dashed line). The rightmost point is PS in Table 1. Hollow markers denote points with infeasible test instances, averaged over feasible ones.

Label Efficiency To measure how many labeled instances PS needs to match PrimalPFN, we train PS on 2 to 240 labeled training instances (Figure 4), keeping its 60 labeled validation instances for model selection at every budget as in its standard protocol, which favors PS. With the same two instances, PS attains a gap of 0.44% on SC, nearly twice that of PrimalPFN, and needs about 15 instances to match it. On CA, PS reaches a comparable gap only with all 240 instances, and on IS it remains worse even then. We provide additional analysis of PrimalPFN in Appendix D.

5 Related Work

5.1 ML for Primal Solution Prediction

Among learning-based approaches to accelerating ILP solving, one line of research predicts high-quality primal solutions or partial assignments. Neural Diving (Nair et al., 2021) predicts partial solutions, and Predict-and-Search (Han et al., 2023) and Apollo-MILP (Liu et al., 2025) integrate such predictions into the solver to improve solution quality and mitigate infeasibility. These approaches rely on supervised learning and therefore require costly solver labels for each problem class. DiffILO (Geng et al., 2025) removes this requirement through unsupervised learning but may become trapped in suboptimal solutions. PrimalPFN instead replaces most of this supervision with synthetic labels and obtains solver supervision only from a few context instances at test time.

5.2 Prior-Data Fitted Networks

PFNs (Müller et al., 2022) train Transformers on datasets sampled from a prior so that they approximate the corresponding PPD in a single forward pass, and Nagler (2023) analyzes their statistical properties from a frequentist perspective. TabPFN (Hollmann et al., 2023) demonstrated the effectiveness of this framework on tabular data, and NodePFN (Choi et al., 2026) extended it to in-context node classification on previously unseen graphs. Transferring graph PFNs to ILP solution prediction is, however, nontrivial. In node classification, labeled nodes within the target graph naturally serve as context, whereas a newly encountered ILP provides no observed variable assignments before solving, so the context must be constructed across problem instances. Moreover, unlike node classification priors, the prior must favor feasible, high-quality assignments, since feasibility and solution quality are properties of an entire instance rather than of individual nodes (Section 4.3).

6 Conclusion

We present PrimalPFN, a primal solution predictor for ILPs that is pretrained without solver labels and adapts to the solver's labeling of a target class from a few solved instances given as context. With two solver-labeled instances per class, it performs comparably to or better than supervised predictors trained on hundreds of labeled instances and, with Gurobi, improves further with abundant solver labels, suggesting that synthetic labels can replace most of the solver supervision that learning-based primal heuristics require. We discuss limitations and future directions in Appendix C.

AI USE STATEMENT

We used generative AI tools in preparing this work in three ways. First, to aid and polish writing, we used them to revise the wording, grammar, and structure of paragraphs written by the authors, to check terminology and notation for consistency across sections, and to cross-check that the numbers quoted in the text match the reported tables. Second, to draft sections of the paper, we used them to produce initial text for parts of the abstract, introduction, method, and experiments from outlines, results, and instructions provided by the authors, which the authors then revised. Third, to assist implementation, we used them to write parts of the code following the authors' designs, through coding agents and code completion. We did not use generative AI tools for research ideation or method design, and all research ideas and experimental designs are the authors' own. We have reviewed all AI-assisted text and code, verified every technical claim and number against our experimental results, and take full responsibility for the final content of this paper.

ETHICS STATEMENT

We do not anticipate any significant negative societal impacts arising directly from this work.

REPRODUCIBILITY STATEMENT

We specify the input features, architecture, prior generation, training hyperparameters, and Predict-and-Search settings in Appendices A and B. Code will be released upon publication.

REFERENCES

Miguel F. Anjos and Antonio J. Conejo. Unit commitment in electric energy systems. *Found. Trends Electr. Energy Syst.*, 1(4):220–310, December 2017. ISSN 2332-6557. doi: 10.1561/3100000014. URL https://doi.org/10.1561/3100000014.

Egon Balas and Andrew Ho. *Set covering algorithms using cutting planes, heuristics, and subgradient optimization: A computational study*, pp. 37–60. Springer Berlin Heidelberg, Berlin, Heidelberg, 1980. ISBN 978-3-642-00802-3. doi: 10.1007/BFb0120886. URL https://doi.org/10.1007/BFb0120886.

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. ISSN 0377-2217. doi: https://doi.org/10.1016/j.ejor.2020.07.063. URL https://www.sciencedirect.com/science/article/pii/S0377221720306895.

David Bergman, Andre A. Cire, Willem-Jan van Hoeve, and John Hooker. *Decision Diagrams for Optimization*. Springer, Cham, 2016. doi: 10.1007/978-3-319-42849-9.

Jeongwhan Choi, Jongwoo Kim, Woosung Kang, and Noseong Park. Learning posterior predictive distributions for node classification from synthetic graph priors. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=FmxRzlu0rT.

Teodor Gabriel Crainic, Bernard Gendron, and Mohammad Rahim Akhavan Kazemzadeh. A taxonomy of multilayer network design and a survey of transportation and telecommunication applications. *European Journal of Operational Research*, 303(1):1–13, 2022. ISSN 0377-2217. doi: https://doi.org/10.1016/j.ejor.2021.12.028. URL https://www.sciencedirect.com/science/article/pii/S0377221721010821.

Stéphane Dauzère-Pérès, Junwen Ding, Liji Shen, and Karim Tamssaouet. The flexible job shop scheduling problem: A review. *Eur. J. Oper. Res.*, 314(2):409–432, April 2024. URL https://doi.org/10.1016/j.ejor.2023.05.017.

Jian-Ya Ding, Chao Zhang, Lei Shen, Shengyin Li, Bing Wang, Yinghui Xu, and Le Song. Accelerating primal solution findings for mixed integer programs based on solution prediction. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34:1452–1459, 04 2020. doi: 10.1609/aaai.v34i02.5503.

Maxime Gasse, Didier Chételat, Nicola Ferroni, Laurent Charlin, and Andrea Lodi. Exact combinatorial optimization with graph convolutional neural networks. In *Advances in Neural Information Processing Systems 32*, 2019.

Zijie Geng, Jie Wang, Xijun Li, Fangzhou Zhu, Jianye Hao, Bin Li, and Feng Wu. Differentiable integer linear programming. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=FPfCUJTsCn.

Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2026. URL https://www.gurobi.com.

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. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=pHMpgT5xWaE.

Christopher Hojny, Mathieu Besançon, Ksenia Bestuzheva, Sander Borst, Antonia Chmiela, João Dionísio, Leon Eifler, Mohammed Ghannam, Ambros Gleixner, Adrian Göß, Alexander Hoen, Rolf van der Hulst, Dominik Kamp, Thorsten Koch, Kevin Kofler, Jurgen Lentz, Stephen J. Maher, Gioni Mexi, Erik Mühmer, Marc E. Pfetsch, Sebastian Pokutta, Felipe Serrano, Yuji Shinano, Mark Turner, Stefan Vigerske, Matthias Walter, Dieter Weninger, and Liding Xu. The SCIP Optimization Suite 10.0. Technical report, Optimization Online, November 2025. URL https://optimization-online.org/2025/11/the-scip-optimization-suite-10-0/.

Noah Hollmann, Samuel Müller, Katharina Eggensperger, and Frank Hutter. TabPFN: A transformer that solves small tabular classification problems in a second. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=cp5PvcI6w8_.

Weihua Hu, Bowen Liu, Joseph Gomes, Marinka Zitnik, Percy Liang, Vijay Pande, and Jure Leskovec. Strategies for pre-training graph neural networks. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=HJlWWJSFDH.

Weimin Huang, Taoan Huang, Aaron M Ferber, and Bistra Dilkina. Distributional miplib: a multi-domain library for advancing ml-guided milp methods, 2024.

Andreas Klose and Andreas Drexl. Facility location models for distribution system design. *European Journal of Operational Research*, 162(1):4–29, 2005. ISSN 0377-2217. doi: https://doi.org/10.1016/j.ejor.2003.10.031. URL https://www.sciencedirect.com/science/article/pii/S0377221703008191. Logistics: From Theory to Application.

Kevin Leyton-Brown, Mark Pearson, and Yoav Shoham. Towards a universal test suite for combinatorial auction algorithms. In *Proceedings of the 2nd ACM Conference on Electronic Commerce*, EC '00, pp. 66–76, New York, NY, USA, 2000. Association for Computing Machinery. ISBN 1581132727. doi: 10.1145/352871.352879. URL https://doi.org/10.1145/352871.352879.

Sirui Li, Janardhan Kulkarni, Ishai Menache, Cathy Wu, and Beibin Li. Towards foundation models for mixed integer linear programming. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=6yENDA7J4G.

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 *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=mFY0tPDWK8.

Samuel Müller, Noah Hollmann, Sebastian Pineda Arango, Josif Grabocka, and Frank Hutter. Transformers can do bayesian inference. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=KSugKcbNf9.

Thomas Nagler. Statistical foundations of prior-data fitted networks. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett (eds.), *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 25660–25676. PMLR, 23–29 Jul 2023. URL https://proceedings.mlr.press/v202/nagler23a.html.

Vinod Nair, Sergey Bartunov, Felix Gimeno, Ingrid von Glehn, Pawel Lichocki, Ivan Lobov, Brendan O'Donoghue, Nicolas Sonnerat, Christian Tjandraatmadja, Pengming Wang, Ravichandra Addanki, Tharindi Hapuarachchi, Thomas Keck, James Keeling, Pushmeet Kohli, Ira Ktena, Yujia Li, Oriol Vinyals, and Yori Zwols. Solving mixed integer programs using neural networks, 2021. URL https://arxiv.org/abs/2012.13349.

Antoine Prouvost, Justin Dumouchelle, Lara Scavuzzo, Maxime Gasse, Didier Chételat, and Andrea Lodi. Ecole: A gym-like library for machine learning in combinatorial optimization solvers. In *Learning Meets Combinatorial Algorithms at NeurIPS2020*, 2020. URL https://openreview.net/forum?id=IVc9hqgibyB.

Tianle Pu, Zijie Geng, Haoyang Liu, Shixuan Liu, Jie Wang, Li Zeng, Chao Chen, and Changjun Fan. RoME: Domain-robust mixture-of-experts for MILP solution prediction across domains. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=wRQmQ6UXYF.

A Implementation Details

A.1 Data Representation

We follow the bipartite graph representation of Gasse et al. (2019) and use the subset of its features that does not require solving the LP relaxation, together with two label features for context variables (Table 4).

Table 4: Variable, constraint, and edge features used by PrimalPFN.

Index	Variable Feature Name	Description
0	Objective	Normalized objective coefficient
1	Variable coefficient	Average of the variable's coefficients over its constraints
2	Variable degree	Degree of the variable node in the bipartite graph
3	Maximum variable coefficient	Maximum of the variable's coefficients over its constraints
4	Minimum variable coefficient	Minimum of the variable's coefficients over its constraints
5	Variable type	Whether the variable is binary
6	Label information	Label of the variable, provided for context variables
7	Label mask	Whether the label information is available
Index	**Constraint Feature Name**	**Description**
0	Constraint coefficient	Average of all nonzero coefficients in the constraint
1	Constraint degree	Degree of the constraint node in the bipartite graph
2	Bias	Normalized right-hand side of the constraint
3	Sense	Sense of the constraint
Index	**Edge Feature Name**	**Description**
0	Coefficient	Constraint coefficient

A.2 Model Architecture

Following the dual-branch design of NodePFN (Choi et al., 2026), each layer of PrimalPFN combines a context–query attention branch and a local MPNN branch. Every variable and constraint node of each instance is represented as a token, and the labels of context variables enter through their input features.

Context–Query Attention Branch Context and query tokens are processed asymmetrically to enable in-context learning. Context representations incorporate the ILP features and the observed variable labels, which may be binary assignments or soft variable-wise marginals. Within each layer, context tokens attend to all context tokens, exchanging information both within and across the labeled context instances:

$$\widetilde{\mathbf{H}}_{\mathrm{ctx}}^{(\ell,\mathrm{attn})} = \mathrm{MHA}\left(\mathbf{H}_{\mathrm{ctx}}^{(\ell)}, \mathbf{H}_{\mathrm{ctx}}^{(\ell)}, \mathbf{H}_{\mathrm{ctx}}^{(\ell)}\right). \tag{9}$$

Query tokens attend to the context tokens through cross-attention:

$$\widetilde{\mathbf{H}}_{\mathrm{query}}^{(\ell,\mathrm{attn})} = \mathrm{MHA}\left(\mathbf{H}_{\mathrm{query}}^{(\ell)}, \mathbf{H}_{\mathrm{ctx}}^{(\ell)}, \mathbf{H}_{\mathrm{ctx}}^{(\ell)}\right). \tag{10}$$

The two attention operations share their projection parameters. Query tokens do not attend to other query tokens, which prevents information leakage between query instances. This asymmetric pattern lets each query instance retrieve labeling patterns from the context without parameter updates, and, because attention operates across instances, this branch is the primary mechanism for transferring information from the context to the query.

Local MPNN Branch In parallel with the attention branch, we apply message passing within each bipartite graph to capture its local constraint–variable structure. We adopt GINE (Hu et al., 2020), which incorporates edge attributes into neighborhood aggregation. For each graph $\mathcal{G}$, the local branch is written as

$$\widetilde{\mathbf{H}}_{\mathcal{G}}^{(\ell,\mathrm{mpnn})} = \mathrm{GINE}\left(\mathbf{H}_{\mathcal{G}}^{(\ell)}, \mathbf{E}_{\mathcal{G}}, \mathbf{A}_{\mathcal{G}}\right), \tag{11}$$

where $\mathbf{E}_{\mathcal{G}}$ specifies the bipartite connectivity and $\mathbf{A}_{\mathcal{G}}$ contains the corresponding signed constraint coefficients. Unlike the attention branch, message passing is restricted to nodes within the same instance and therefore captures intra-instance structural dependencies. Because the constraint coefficients enter the messages directly, the branch distinguishes not only which variables participate in a constraint but also the magnitude and sign of their contributions.

Layer Fusion We combine the two branches using branch-specific residual connections and layer normalization. For each layer, we first compute

$$\mathbf{H}_{\text{attn}}^{(\ell)} = \text{LN}_{\text{attn}}\left(\mathbf{H}^{(\ell)} + \text{Dropout}\left(\widetilde{\mathbf{H}}^{(\ell,\text{attn})}\right)\right) \tag{12}$$

and

$$\mathbf{H}_{\text{mpnn}}^{(\ell)} = \text{LN}_{\text{mpnn}}\left(\mathbf{H}^{(\ell)} + \text{Dropout}\left(\widetilde{\mathbf{H}}^{(\ell,\text{mpnn})}\right)\right), \tag{13}$$

where $\widetilde{\mathbf{H}}^{(\ell,\text{attn})}$ and $\widetilde{\mathbf{H}}^{(\ell,\text{mpnn})}$ collect the outputs of the two branches over all context and query tokens. The two branch representations are then fused by summation:

$$\mathbf{H}_{\text{fuse}}^{(\ell)} = \mathbf{H}_{\text{attn}}^{(\ell)} + \mathbf{H}_{\text{mpnn}}^{(\ell)}. \tag{14}$$

Finally, a position-wise feed-forward network with an additional residual connection produces the next-layer representation:

$$\mathbf{H}^{(\ell+1)} = \text{LN}_{\text{ff}}\left(\mathbf{H}_{\text{fuse}}^{(\ell)} + \text{Dropout}\left(\text{FFN}\left(\mathbf{H}_{\text{fuse}}^{(\ell)}\right)\right)\right). \tag{15}$$

B Experimental Details

B.1 Benchmark Details

Table 5 summarizes the average number of constraints and variables of the instances in each benchmark.

Table 5: Average instance size of each benchmark.

Benchmark	Constraints	Variables
SC	5,000	4,000
IS	23,994.82	6,000
CA	2,676.32	4,000

B.2 Training Details

B.2.1 Prior Generation

Each episode samples a labeling rule q_ω whose parameters ω are shared by all instances in the episode and resampled across episodes (Algorithm 1). The rule consists of two steps.

Graph-Based Initialization A shallow random MPNN, whose weights and input code are collected in ψ, operates on the bipartite graph of Gasse et al. (2019) and assigns a score to each variable. Variables whose scores fall in the top ρ fraction are set to one and the rest to zero, where ρ is also sampled per episode. Since ψ and ρ vary across episodes, different episodes favor different structural patterns and label densities.

Optimization-Aware Refinement Starting from the initial assignment, the rule runs simulated annealing with single-variable flips on the penalized energy

$$E_t(x) = \bar{c}_t^\top x + \mu_t V_t(x), \tag{16}$$

where $\bar{c}_t$ is the normalized objective of instance t in minimization form and $V_t(x)$ is the average constraint violation over its normalized rows. The instance-level penalty weight μ_t rescales a sampled relative weight μ by the scales of the objective and violation terms, so that the same μ induces

Algorithm 1 Episode-level labeling rule q_ω

Require: Binary ILP instances $\{\mathcal{I}_t\}_{t=1}^{T}$ of one episode
Ensure: Synthetic labels $\{x_t^\star\}_{t=1}^{T}$
Sample $\omega = (\psi, \rho, \mu, \lambda, S) \sim p(\omega)$ ▷ shared within the episode; Table 6
for $t = 1, \dots, T$ **do**
  $x \leftarrow$ indicator of the top-ρ fraction of variables ranked by $g_\psi(\mathcal{G}_t)$ ▷ initialization
  $x_t^\star \leftarrow x$
  for $q = 0, \dots, S-1$ **do** ▷ refinement
    $\tau_q \leftarrow$ temperature decayed geometrically from λs_t ▷ s_t: energy scale of $\mathcal{I}_t$
    for each variable i in the sweep order **do**
      $\widetilde{x} \leftarrow x \oplus e_i$; $\delta \leftarrow E_t(\widetilde{x}) - E_t(x)$
      With probability $\min\{1, \exp(-\delta/\tau_q)\}$, set $x \leftarrow \widetilde{x}$
      if $E_t(x) < E_t(x_t^\star)$ **then**
        $x_t^\star \leftarrow x$
      end if
    end for
  end for
end for
return $\{x_t^\star\}_{t=1}^{T}$

Table 6: Sampling distributions of the labeling-rule parameters ω, drawn once per episode. LogU denotes a log-uniform distribution.

Parameter	Role	Distribution
ψ	Random MPNN (hidden width 16, tanh)	weights $\sim \mathcal{N}(0, 1/d_{\text{in}})$, input code $\sim \mathcal{U}([-1,1]^8)$
ρ	Fraction of variables initialized to one	$\text{LogU}(0.01, 0.8)$
μ	Relative penalty weight	$\text{LogU}(0.5, 32)$
λ	Initial temperature factor	$\text{LogU}(0.05, 1)$
S	Number of annealing sweeps	$\text{Uniform}\{4, 8, 16, 32\}$

a comparable objective–feasibility trade-off across instances. The temperature starts from a sampled multiple λ of the instance's energy scale and decays geometrically to 1% of this value over S sweeps, and the rule returns the lowest-energy assignment visited, without any feasibility repair. All rule parameters are sampled per episode from the distributions in Table 6, yielding rules with different objective–feasibility trade-offs.

B.2.2 Model Training

Table 7 summarizes the model configuration and training hyperparameters. The model has about 5M parameters, and pretraining takes about 24 hours on a single NVIDIA GeForce RTX 5090 GPU. For the *w/ solver labels* variant, tasks built from the solver-labeled training instances are mixed with the synthetic tasks during pretraining, with all other settings unchanged. We select the checkpoint by the loss on held-out synthetic episodes generated from the prior, so model selection requires no solver labels.

B.3 Inference Details

We use Predict-and-Search (PS) (Han et al., 2023) as the downstream procedure in all experiments. Given the predicted marginals, PS selects the $k_0|\mathcal{V}_{\text{var}}|$ variables most confidently predicted as zero and the $k_1|\mathcal{V}_{\text{var}}|$ variables most confidently predicted as one, which form the sets $\mathcal{K}_0$ and $\mathcal{K}_1$. It then solves the original ILP with the trust-region constraint

$$\sum_{i \in \mathcal{K}_0} x_i + \sum_{i \in \mathcal{K}_1} (1 - x_i) \le \Delta,$$

which allows at most Δ of the selected variables to deviate from their predicted values. Expressing k_0 and k_1 as fractions of the number of variables accounts for differences in problem size.

Table 7: Model architecture and training hyperparameters.

Parameter	**Value**
Model architecture	
Embedding dimension	192
Number of layers	12
Number of attention heads	8
Local message passing	GINE
Attention mechanism	FlashAttention
Dropout	0.05
Training	
Optimizer	Muon + AdamW
Initial learning rate	10^{-4}
Final scheduled learning rate	10^{-5}
Weight decay	0
Batch size	1
Gradient accumulation	4
Episodes per epoch	120
Maximum training epochs	700
Scheduler	Linear

Table 8: Predict-and-Search parameters used in our experiments.

Benchmark	k_0	$k_0\|\mathcal{V}_{\text{var}}\|$	k_1	$k_1\|\mathcal{V}_{\text{var}}\|$
SC	0.6	2,400	0.01	40
IS	0.1	600	0.1	600
CA	0.4	1,600	0.01	40

Table 8 summarizes the Predict-and-Search parameters. For each prediction model and benchmark, we select the trust-region radius $\Delta \in \{0, 20, 40\}$ with a 1,000-second solver budget on an additional set of instances generated with the Ecole generator under the same settings as the benchmark, and fix it for evaluation on the test set. Although we use PS throughout, the predicted marginals can in principle be combined with other solver-based primal heuristics, such as Apollo-MILP (Liu et al., 2025).

C Discussions

C.1 Limitations

Research Scope We focus on integer linear programs (ILPs), and in particular on binary ILPs, since bounded integer variables can be encoded with binary variables. Directly handling continuous or general integer variables is beyond the scope of this work: it requires both a prior that generates meaningful labels for such variables and a prediction head that maps node representations to continuous-valued decisions, neither of which is straightforward. We believe PrimalPFN opens a promising direction toward priors that accommodate general ILPs and, ultimately, mixed-integer linear programs (MILPs).

Problem-Class Coverage Our pretraining prior draws instances from class-specific generators for the three benchmark classes. This follows the standard setting of primal solution prediction, in which the target class is known in advance, and including a new class only requires pretraining with its instances, without solver labels. Handling classes unavailable during pretraining is a different setting, which would require broadening the instance generation beyond class-specific generators.

C.2 Future Avenues

Toward General (M)ILPs and Foundation Models Our experiments consider SC, IS, and CA instances sampled from their respective generators. Broadening the prior to cover structural patterns shared across ILP classes would be a first step toward a single predictor for general ILPs, and co-designing the instance and label generators could further extend the framework to MILPs. This direction aligns with growing interest in foundation models for (M)ILP solving (Li et al., 2025): because pretraining requires no solver, scaling the model and the pretraining distribution is limited only by compute rather than by labeling cost.

D Additional Results

D.1 Results with SCIP

We repeat the experiment of Table 1 with SCIP (Hojny et al., 2025) as the solver in place of Gurobi. The predictors and their solver labels are unchanged, and only the solver used in the downstream search and as the standalone baseline is replaced. All other settings follow Section 4.1 and Appendix B.3, including the 1,000-second time limit, the PS parameters, and the selection of the trust-region radius Δ, which we repeat with SCIP. We compute gaps with respect to the BKS of Table 1, obtained from the Gurobi-based runs, so that gaps are comparable across the two solvers, and we verified that no SCIP-based run improves on this BKS. Table 9 reports the results.

Table 9: Results with SCIP under a 1,000-second time limit. Best and second-best results are shown in bold and underlined, respectively.

	SC (BKS: 168.88)		IS (BKS: 2,115.66)		CA (BKS: 116,309.51)	
	Obj ↓	Gap(%) ↓	Obj ↑	Gap(%) ↓	Obj ↑	Gap(%) ↓
SCIP	172.65	2.18	1,993.96	5.75	112,317.98	3.44
PS	169.57	0.41	2,066.67	2.31	113,891.03	2.08
DiffILO	171.26	1.39	2,052.94	2.96	112,198.04	3.54
PrimalPFN	**169.27**	**0.23**	**2,078.60**	**1.75**	<u>113,911.49</u>	<u>2.06</u>
w/ solver labels	<u>169.43</u>	<u>0.32</u>	<u>2,072.05</u>	<u>2.06</u>	**114,651.69**	**1.43**

D.2 Per-Instance Comparison for Table 1

Tables 10–12 complement the Gurobi results in Table 1 with per-instance comparisons. For each pair of methods, we count the test instances on which the row method finds a better (win), equal (draw), or worse (loss) final objective value than the column method, where two objective values are considered equal if they coincide within a relative tolerance of 10^{-6}.

Table 10: Per-instance (win, draw, loss) counts of each row method against each column method on SC.

Method	PS	DiffILO	Gurobi(1000s)	Gurobi(3600s)
PrimalPFN	(12, 69, 19)	(24, 62, 14)	(40, 48, 12)	(22, 61, 17)
w/ solver labels	(9, 79, 12)	(26, 66, 8)	(39, 56, 5)	(23, 69, 8)

Table 11: Per-instance (win, draw, loss) counts on IS, as in Table 10.

Method	PS	DiffILO	Gurobi(1000s)	Gurobi(3600s)
PrimalPFN	(75, 3, 22)	(100, 0, 0)	(100, 0, 0)	(99, 0, 1)
w/ solver labels	(76, 4, 20)	(100, 0, 0)	(100, 0, 0)	(100, 0, 0)

Table 12: Per-instance (win, draw, loss) counts on CA, as in Table 10.

Method	PS	DiffILO	Gurobi(1000s)	Gurobi(3600s)
PrimalPFN	(53, 0, 47)	(100, 0, 0)	(75, 0, 25)	(55, 0, 45)
w/ solver labels	(68, 0, 32)	(100, 0, 0)	(77, 0, 23)	(68, 0, 32)

D.3 Effect of Context-Label Interventions on Predictions

Figure 5 visualizes how changing the context labels affects individual variable predictions on all three benchmarks. We use four interventions whose expected effect is clear if the model relies on the context labels, namely all-zero and all-one labels, which should pull the predictions toward the given value, and random and shuffled labels, which break the correspondence between variables and labels and should compress the predictions toward their mean. We omit the prior and cross-class contexts of Table 2, since their expected effect on individual predictions is not known in advance. The predictions follow these expectations on all three benchmarks. The shuffled case is the most informative, because it keeps the label frequency of the original context but breaks the correspondence between each context variable and its label. Under this intervention, the predictions change nearly as much as under random labels, indicating that the model uses the instance–label correspondence in the context rather than its marginal label statistics.

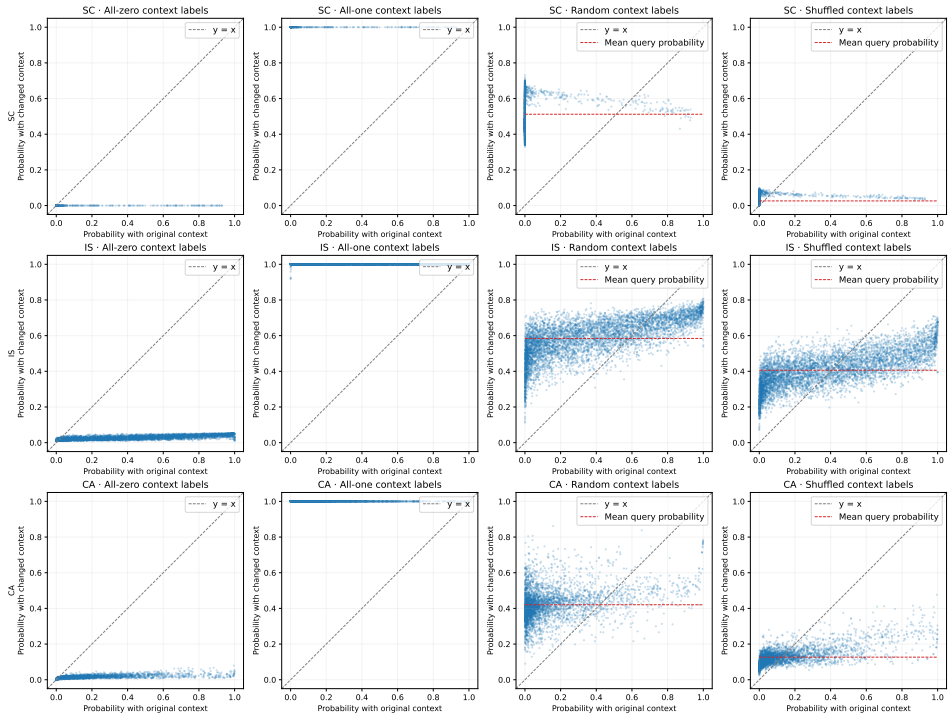

Figure 5: Context-label interventions on PrimalPFN predictions for SC, IS, and CA (rows). Each point is a query variable, plotted by its predicted probability under the original solver-labeled context (x) and under a modified context with the same graphs but changed labels (y). Columns: all labels set to zero, all set to one, labels drawn i.i.d. from Bernoulli(0.5), and labels shuffled across variables within each context instance. The red dashed line marks the mean prediction under the modified context.

D.4 Prior Ablation

Table 13 compares the full prior with two variants, all pretrained for 200 epochs: *w/o refinement*, which uses the graph-based initial assignments directly as labels, and *random init.*, which replaces the graph-based initialization with random initialization before refinement. Removing the

Table 13: Prior ablation. All variants are pretrained for 200 epochs under otherwise identical settings. *For *w/o refinement* on IS, 9 of 100 instances yield no feasible solution. Obj and Gap are averaged over the remaining 91 instances.

	SC (BKS: 168.88)		IS (BKS: 2,115.66)		CA (BKS: 116,309.51)	
	Obj ↓	Gap(%) ↓	Obj ↑	Gap(%) ↓	Obj ↑	Gap(%) ↓
Full prior	169.42	0.31	2,107.86	0.38	115,389.56	0.79
Random init.	169.42	0.31	2,108.32	0.36	115,391.84	0.79
w/o refinement	169.68	0.47	2,080.88*	1.68*	111,007.45	4.56

optimization-aware refinement substantially degrades performance on all benchmarks. The gap increases from 0.31% to 0.47% on SC and from 0.79% to 4.56% on CA, and 9 of 100 IS instances yield no feasible solution. In contrast, replacing the graph-based initialization with random initialization yields nearly identical gaps on all benchmarks. This indicates that the optimization-aware refinement, rather than graph-dependent labeling as in existing graph PFN priors, drives the transfer to solver-labeled contexts. We nevertheless retain the graph-based initialization in our main model, as it adds negligible cost to prior generation and does not hurt performance. One explanation for its limited effect is that a useful prior should concentrate its mass near plausible solver labelings rather than merely diversify them: the randomized refinement parameters vary the trade-off between objective and feasibility, a direction in which solver outputs plausibly vary, whereas random structural scores add variation that solver labelings do not exhibit.

D.5 Sensitivity to Context Selection

The results in Table 1 are obtained with a single context set per problem class. To assess their sensitivity to this choice, we draw four additional context sets, each consisting of two instances sampled uniformly at random from the training split, and evaluate PrimalPFN with each of them under the same protocol. Table 14 reports the results, with PS shown for reference. Across all five context sets, PrimalPFN attains a lower gap than PS on IS, while its gaps on SC and CA remain comparable to those of PS, ranging from 0.23% to 0.24% and from 0.54% to 0.65%, respectively. We perform this analysis for PrimalPFN only. Note that the *w/ solver labels* variant uses the same context set as in Table 1.

Table 14: PrimalPFN with five context sets, each consisting of two instances drawn uniformly at random from the training split. Context set 1 is the one used in Table 1, and PS is shown for reference. The shaded row reports the mean ± sample standard deviation over the five context sets. Gaps are computed with respect to the BKS of Table 1.

	SC (BKS: 168.88)		IS (BKS: 2,115.66)		CA (BKS: 116,309.51)	
	Obj ↓	Gap(%) ↓	Obj ↑	Gap(%) ↓	Obj ↑	Gap(%) ↓
Context set 1	169.28	0.23	2,108.91	0.32	115,559.11	0.64
Context set 2	169.29	0.24	2,108.90	0.32	115,623.59	0.59
Context set 3	169.28	0.23	2,109.27	0.30	115,565.65	0.64
Context set 4	169.28	0.23	2,108.84	0.32	115,558.01	0.65
Context set 5	169.30	0.24	2,109.28	0.30	115,680.35	0.54
Mean ± std	169.29 ± 0.01	0.23 ± 0.01	2,109.04 ± 0.22	0.31 ± 0.01	115,597.34 ± 53.83	0.61 ± 0.05
PS	169.20	0.18	2,104.38	0.53	115,525.21	0.67

D.6 Inference Time

Figure 6 reports the prediction time of each learning-based method, measured on a single NVIDIA GeForce RTX 5090 GPU and averaged over the 100 test instances. PrimalPFN is slower than PS and DiffILO because it processes the context instances together with each query. To reduce this overhead, we run PrimalPFN in BF16 precision with FlashAttention, which keeps its additional cost

to at most about 1.5 seconds per instance (on SC), while PS and DiffILO use their default FP32 implementations. On average, every method takes under 3 seconds per instance, less than 0.3% of the 1,000-second time limit. For all learning-based methods, prediction time is not counted toward the solver time limit. Given its magnitude, this choice has a negligible effect on the comparison.

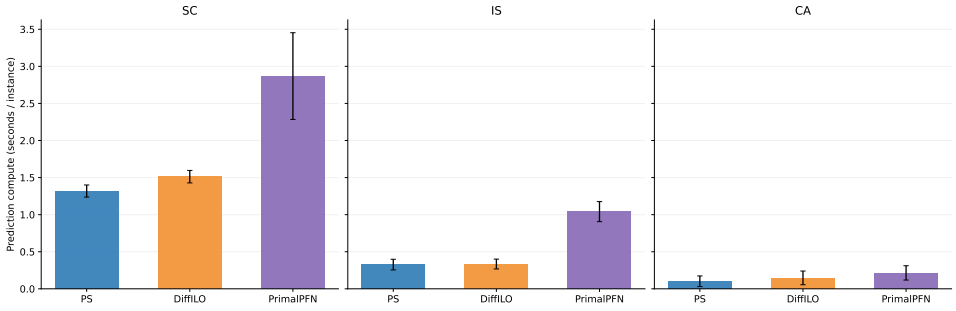

Figure 6: Prediction time per instance on a single GPU, including feature extraction, averaged over the 100 test instances. Error bars denote one standard deviation. On average, every method takes under 3 seconds per instance, less than 0.3% of the 1,000-second time limit.