SHSP: Structure-Aware Hierarchical Solution Prediction for Mixed-Integer Linear Programming

Zherong Zhang[1,2*], Guanlin Li[1,2*], Chengrui Gao[1,2], Haopu Shang[1,2], Ke Xue[1,2], Jixiang Lu[3], Weiyong Yang[3], Chao Qian[1,2†]

[1]State Key Laboratory of Novel Software Technology, Nanjing University
[2]School of Artificial Intelligence, Nanjing University
[3]State Key Laboratory of Technology and Equipment for Defense Against Power System Operational Risks, Nari Technology Co., Ltd.

Abstract

Mixed-Integer Linear Programming (MILP) is a fundamental optimization paradigm in combinatorial optimization and has been widely applied across real-world domains. Due to its NP-hard nature, obtaining optimal solutions for large-scale or highly constrained MILP instances remains computationally prohibitive. Learning-based solution prediction has therefore emerged as a promising approach to provide high-quality variable assignment for solver acceleration. However, existing methods typically adopt a one-shot prediction paradigm that predicts the marginal probabilities of all variables simultaneously. As a result, the conditional dependencies among variables are only implicitly captured through message passing, with the burden of modeling the combinatorial structure falling entirely on the representational capacity of graph neural networks. To address this limitation, we propose the **S**tructure-Aware **H**ierarchical **S**olution **P**rediction (SHSP) framework that replaces the parallel marginal decoding of one-shot methods with a novel hierarchical conditional decoding mechanism. Specifically, SHSP constructs a variable coupling graph from the constraint structure, decodes variables sequentially along a hierarchy of increasing coupling strength, and conditions each hierarchy on previously predicted assignments. To mitigate error accumulation during the decoding process, SHSP further incorporates a confidence-aware mask-and-repair mechanism to identify and correct unreliable intermediate predictions. We integrate SHSP with multiple learning-guided search methods, and evaluate it on four standard MILP benchmarks. Experimental results demonstrate that SHSP significantly outperforms existing one-shot prediction baselines, achieving a 54% average reduction in solution gap. Code is available at: https://github.com/lamda-bbo/SHSP.

1 Introduction

Mixed-Integer Linear Programming (MILP) is one of the most widely used modeling paradigms in combinatorial optimization and operations research, with broad applications in various optimization domains (Ma et al. 2019; Li et al. 2024b). Owing to the NP-hardness of MILP (Karp 1972), decades of research efforts have been devoted to developing powerful solvers such as SCIP (Achterberg 2009) and Gurobi (Gurobi Optimization, LLC 2026). Equipped with advanced techniques including cutting planes, primal heuristics, and presolving, these solvers are capable of efficiently handling a wide range of practical instances. However, solving large-scale and highly constrained instances remains computationally expensive, making the improvement of MILP solving efficiency a fundamental research challenge.

In recent years, machine learning has emerged as a promising paradigm for accelerating MILP solving (Li et al. 2024a), and existing studies generally fall into two categories. The first category learns heuristic policies for key components of branch-and-bound (B&B) solvers, such as branching strategy (Khalil et al. 2016; Gasse et al. 2019; Hou et al. 2026) and cut selection (Tang, Agrawal, and Faenza 2020; Huang et al. 2021; Paulus et al. 2022; M. et al. 2026), typically via imitation learning or reinforcement learning. The second category employs machine learning models such as graph neural networks (GNNs) to directly predict high-quality solutions for MILP instances (Ding et al. 2020; Zeng et al. 2024; Geng et al. 2025). Research in this category advances along two complementary aspects. The first aspect concerns how predictions are exploited to reduce the original problem (Nair et al. 2020; Han et al. 2023; Liu et al. 2025). The second aspect focuses on improving the prediction capacity (Pu et al. 2026; Wang, Li, and Wang 2026). A detailed discussion of related work is provided in Appendix A. Together, these efforts have established learning-based solution prediction as an effective paradigm for accelerating MILP solving.

Despite the remarkable performance of existing solution prediction methods, most of them follow a one-shot prediction paradigm (Han et al. 2023; Liu et al. 2025; Pu et al. 2026), where the marginal probabilities of all decision variables are predicted simultaneously in a single forward pass. However, variables in a MILP are strongly interdependent through shared constraints. The optimal assignment of one variable often hinges on the combinatorial structure formed by the others, to the extent that exactly predicting the complete optimal assignments would amount to solving the MILP itself. Under the one-shot paradigm, such dependencies are captured only implicitly by the GNN encoder, while the decoding stage treats all variables as independent. This mismatch can yield mutually conflicting assignments that mislead, rather than accelerate, the downstream solver.

To address these limitations, we propose **Structure-Aware**

*These authors contributed equally.
†Corresponding author. Email: qianc@lamda.nju.edu.cn.

Hierarchical Solution Prediction for MILP (SHSP), a novel learning framework that explicitly models variable dependencies through hierarchical partitioning and prediction, rather than relying on a one-shot predictor to recover them implicitly. Specifically, SHSP first constructs a weighted variable coupling graph that quantifies pairwise dependencies based on expected constraint violation and coefficient importance. Guided by the coupling scores, variables are predicted in a hierarchical manner: weakly coupled variables are resolved first, while strongly coupled ones are subsequently decoded conditioned on the resulting partial assignments. To further enhance robustness, we introduce a confidence-based mask-and-repair mechanism that mitigates error accumulation during multi-step inference. Finally, the predicted assignments are exploited through a structure-aware fixing strategy that prioritizes strongly coupled variables, whose fixed values propagate through shared constraints and tighten the feasible domains of many others, thereby effectively reducing the search space.

As a superior predictor, SHSP serves as a drop-in replacement for the vanilla GNN-based one-shot predictor and can be seamlessly integrated into diverse learning-guided acceleration frameworks (Nair et al. 2020; Han et al. 2023). To validate the effectiveness of SHSP, we conduct extensive experiments on four widely used MILP benchmarks. Experimental results demonstrate that SHSP consistently outperforms the one-shot prediction baselines across all benchmarks, reducing the average absolute primal gap by up to 100.0%, 38.1%, 94.3%, and 42.1%, respectively. Notably, a variant of SHSP even surpasses the solution quality of full-budget Gurobi on combinatorial auction instances within a substantially smaller time budget.

2 Preliminaries

2.1 Mixed-Integer Linear Programming

Mixed-integer linear programming (MILP) seeks to optimize a linear objective function over a feasible region defined by linear constraints, where a subset of decision variables take integer values. Formally, an MILP instance can be expressed as

$$\begin{aligned} \min_{x} \quad & c^\top x \\ \text{s.t.} \quad & Ax \le b, \\ & l \le x \le u, \\ & x \in \mathbb{Z}^p \times \mathbb{R}^{n-p}, \end{aligned}$$

where x denotes the n-dimensional decision vector comprising p integer variables and $n-p$ continuous variables. The vector $c \in \mathbb{R}^n$ specifies the objective coefficients, $A \in \mathbb{R}^{m\times n}$ is the constraint coefficient matrix, and $b \in \mathbb{R}^m$ is the corresponding right-hand-side vector. The vectors $l \in (\mathbb{R} \cup \{-\infty\})^n$ and $u \in (\mathbb{R} \cup \{+\infty\})^n$ impose the lower and upper bounds on the variables, respectively. Following prior works (Han et al. 2023; Liu et al. 2025; Pu et al. 2026), we focus on MILP instances whose integer variables are binary, as general integer variables can be reduced to binary ones via well-established preprocessing techniques (Nair et al. 2020).

An MILP instance can be naturally encoded as a weighted bipartite graph $G = (W \cup V, E)$ (Gasse et al. 2019), where W and V denote the sets of constraint nodes and variable nodes, respectively. An edge $(w, v) \in E$ connects a constraint node $w \in W$ and a variable node $v \in V$ if and only if the corresponding variable appears with a nonzero coefficient in the constraint. The node features typically encompass objective coefficients, variable bounds, and variable types on the variable side, as well as right-hand-side values and constraint senses on the constraint side, while the edge features carry the corresponding constraint coefficients. More details on the graph modeling are provided in Appendix B.

2.2 Predict-and-Search

Predict-and-Search (PaS) (Han et al. 2023) is a two-stage framework that integrates neural solution prediction with mathematical optimization for accelerating MILP solving. Given an MILP instance I, PaS first learns the underlying solution distribution from a collection of optimal or near-optimal solutions. Specifically, the solution distribution is approximated as $q(x|I) = \frac{\exp(-E(x,I))}{\sum_{x'\in S}\exp(-E(x',I))}$, where S denotes the collected solution set and the energy function is defined as $E(x, I) = c^\top x$ if x is feasible and $+\infty$ otherwise. Guided by this distribution, PaS trains a graph neural network p_θ to estimate the marginal probability of each binary variable. Under the variable-wise independence assumption, the joint solution distribution is factorized as $p_\theta(x \mid I) = \prod_{i=1}^{n} p_\theta(x_i \mid I)$, thereby reducing the learning task to predicting per-variable marginals. The GNN is trained by minimizing a weighted binary cross-entropy loss over the collected solutions, where each solution is weighted according to $q(x \mid I)$:

$$\mathcal{L} = -\sum_{s=1}^{S} q(x^{(s)} \mid I) \sum_{i=1}^{n} \ell(\hat{p}_i, y_{s,i})$$

where $\hat{p}_i = p_\theta(x_i = 1 \mid I)$ denotes the predicted probability of variable x_i, $y_{s,i} \in \{0, 1\}$ denotes the value of variable x_i in solution $x^{(s)}$, and $\ell(\hat{p}_i, y_{s,i}) = -y_{s,i} \log \hat{p}_i - (1 - y_{s,i}) \log(1 - \hat{p}_i)$ denotes the binary cross-entropy loss. Based on the predicted marginal probabilities, PaS selects k_1 variables with the highest marginal probabilities and fixes them to one, while selecting k_0 variables with the lowest marginal probabilities and fixes them to zero. The resulting partial solution is denoted as $\hat{x}^{[P]}$, where P represents the index set of fixed variables.

Rather than hard-fixing these variables as in neural diving (Nair et al. 2020), PaS formulates a trust-region optimization problem that constrains the feasible region to the neighborhood of the predicted partial solution. Formally, the trust region is defined as $B_P(\hat{x}^{[P]}, \Delta) = \{x^{[P]} \in \mathbb{R}^n \mid \|\hat{x}^{[P]} - x^{[P]}\|_1 \le \Delta\}$, where Δ denotes the neighborhood radius. The resulting trust-region problem is formulated as

$$\begin{aligned} \min_{x} \quad & c^\top x \\ \text{s.t.} \quad & Ax \le b, \quad l \le x \le u, \\ & x^{[P]} \in B_P(\hat{x}^{[P]}, \Delta), \quad x \in \mathbb{Z}^p \times \mathbb{R}^{n-p}. \end{aligned}$$

By coupling neural prediction with trust-region-based local optimization, PaS effectively combines the efficiency of learning-based methods and the reliability of mathematical programming solvers. Notably, when $\Delta = 0$, PaS degenerates to the hard-fixing scheme of neural diving.

3 Methodology

In this section, we present **Structure-Aware Hierarchical Solution Prediction for MILP (SHSP)**. In contrast to existing methods that predict the assignments of all decision variables simultaneously, SHSP explicitly exploits variable coupling information and decodes variable assignments sequentially following a hierarchy of increasing coupling strength, where the prediction at each decoding step is conditioned on the partial assignments of lower hierarchies obtained in preceding steps. The overall framework is illustrated in Figure 1. In the remainder of this section, we first introduce the variable coupling graph, then describe the structure-aware hierarchical predictor, and finally present the corresponding variable fixing strategy.

3.1 Variable Coupling Graph

Decision variables are often tightly coupled through shared constraints, and such structural dependencies play a critical role in obtaining high-quality solutions. To explicitly capture these dependencies, we construct a variable coupling graph, which serves as the basis for estimating coupling scores and for determining hierarchical partitioning.

Graph Modeling. We model the coupling structure among variables as an undirected weighted graph, where nodes correspond to variables and edges capture their co-occurrence in constraints. Given linear constraints, we consider all variables when estimating coupling strengths, but retain only those that co-occur with at least one binary variable in some constraints, i.e., those directly connected to binary variables. This makes the coupling graph considerably sparser without severely compromising the prediction accuracy of binary variables. An edge is created between two retained variables if they appear together in at least one constraint, with its weight quantifying the coupling strength implied by the shared constraints, as detailed below.

Edge Weights Computation. For each retained variable pair, we assign an edge weight that reflects the coupling strength implied by the constraints. The weight is determined by two complementary heuristic factors. The first factor, termed the *expected violation*, measures how strongly a variable pair is restricted by a constraint. Specifically, we treat the two variables as random variables, where binary variables follow independent uniform distributions over $\{0, 1\}$ and continuous variables follow uniform distributions over their bounds. We then compute the expected violation of the constraint induced by the pair alone. A larger expected violation indicates that the pair is more tightly restricted by the constraint and thus more strongly coupled. The second factor, termed the *coefficient importance*, characterizes the relative contribution of a variable pair within a constraint. It is computed as the sum of the magnitudes of the two corresponding coefficients, each scaled by the range of its associated variable. To eliminate scale discrepancies across constraints, both factors are normalized by their respective average values over all variable pairs within the same constraint. Note that the above two factors are both computed over all variables, including purely continuous ones.

The local coupling weight of a pair in constraint c is then given by the product of the two normalized factors, and the final edge weight W_{ij} between variables z_i and z_j is obtained by summing the local weights over all constraints containing both variables. The detailed computation of edge weights is provided in Appendix C. The resulting weighted graph captures the overall structural coupling among variables and serves as the basis for computing the variable coupling score and determining the hierarchical prediction order in structure-aware hierarchical prediction.

3.2 Structure-Aware Hierarchical Prediction

The above weighted variable coupling graph provides a quantitative measure of the structural dependency among variables. Building upon this, we design a structure-aware hierarchical prediction framework that decodes variables sequentially from weakly coupled to strongly coupled ones. This framework conditions each decoding step on the assignments fixed in preceding hierarchies, so that strongly coupled variables, whose values are the most sensitive to those of others, are predicted with explicit structural context rather than in isolation as in previous one-shot methods.

Coupling Score-based Hierarchical Partitioning. For each binary variable z_i, we define its variable coupling score (VCS) as the sum of the weights of its incident edges in the coupling graph, which quantifies the overall strength of its structural dependency on the remaining variables. All variables are sorted in ascending order of VCS and evenly partitioned into K hierarchies $\{H_1, H_2, \ldots, H_K\}$, such that H_1 contains the most weakly coupled variables and H_K the most strongly coupled ones. The decoding process then follows this weak-to-strong order. Weakly coupled variables are largely insensitive to the assignments of other variables and can thus be predicted reliably at early stages. Their fixed values in turn serve as conditioning information that anchors the prediction of strongly coupled variables in later hierarchies.

Hierarchical Decoding with Mask-and-Repair. The model decodes the K hierarchies sequentially, and each step is conditioned on the tentative assignments accumulated from all preceding hierarchies. To realize this conditioning, the input feature of every variable node is augmented with a decoding-state triplet $(m_i, \hat{x}_i, c_i)$, where $m_i \in \{0, 1\}$ indicates whether variable i should be predicted in the current step, $\hat{x}_i$ denotes its tentative binary assignment, and c_i its associated confidence. For variables that will be decoded in future steps, the triplet is set to a zero placeholder. In addition, to make the network aware of its position in the decoding process, the current step index is encoded via positional encoding (Vaswani et al. 2017), and the resulting step embedding is concatenated with the initial node embeddings and passed through a fusion layer to produce the step-aware node representations. The collection of these state features

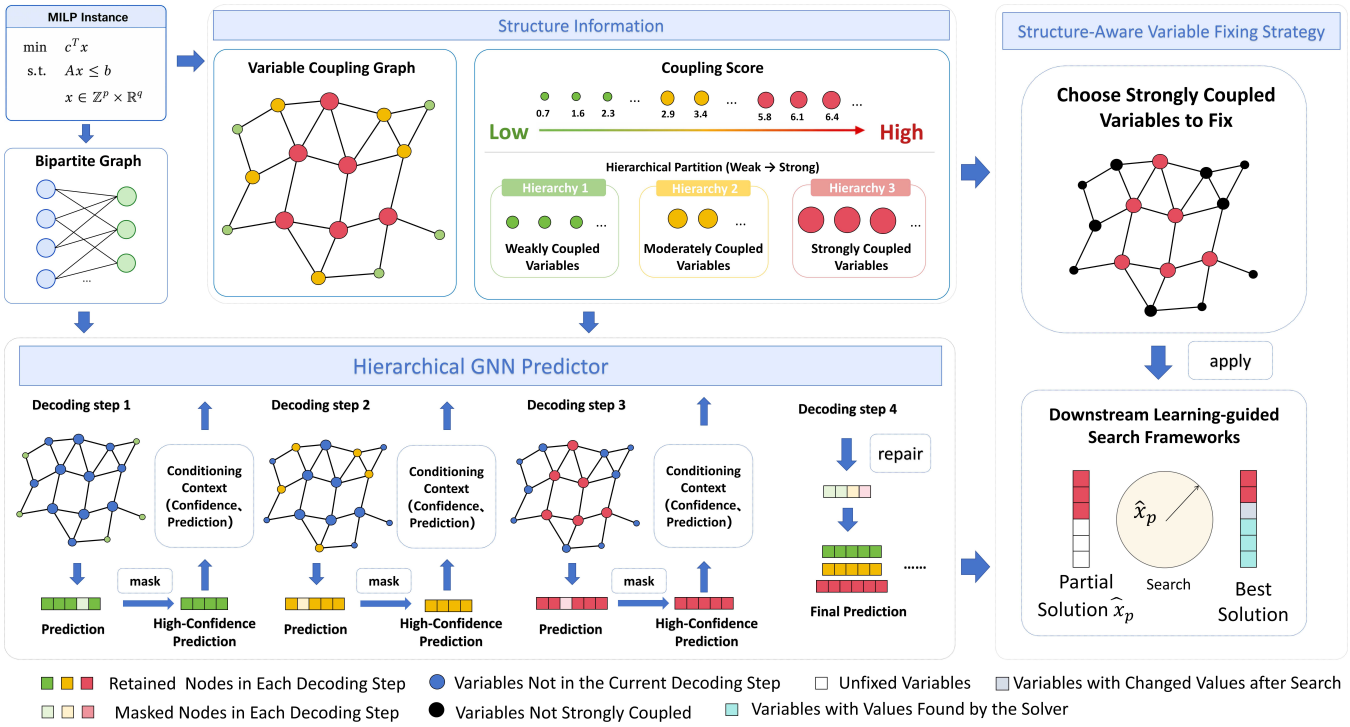

Figure 1: Overview of the proposed method. Given an original MILP instance, we first construct a variable coupling graph to capture the structural dependencies among decision variables. Based on variable coupling graph, variables are ranked according to their coupling score and divided into multiple hierarchies. Our SHSP sequentially predicts variables following the weak-to-strong hierarchy, with a mask-and-repair mechanism to mitigate error accumulation. Finally, the predicted variable assignments are combined with a structure-aware variable fixing strategy and applied to downstream learning-guided search frameworks such as Predict-and-Search (Han et al. 2023).

constitutes the conditioning context $\mathbf{h}_{<k}$, and the prediction of the k-th hierarchy is formulated as

$$\mathbf{p}_{H_k} = f_\theta(G, H_k, \mathbf{h}_{<k}),$$

where G denotes the bipartite graph and f_θ the graph neural network. For each variable $i \in H_k$, the network outputs a probability p_i, from which the binary assignment $\hat{x}_i = \mathbb{I}(p_i > 0.5)$ and the confidence $c_i = 2|p_i - 0.5|$ are derived.

Since the conditioning context is constructed from model predictions rather than ground-truth assignments, wrong early decisions may propagate and accumulate across hierarchies. To mitigate such error accumulation, we propose a mask-and-repair mechanism. Before being incorporated into the conditioning context, each newly predicted variable is stochastically masked with a probability $p_{\text{mask}}(i) \propto 1 - c_i$ that decreases with its confidence. Consequently, high-confidence predictions are likely to be retained as conditioning information, while uncertain predictions tend to be masked, reverting to the undecided placeholder state and being collected into a repair set $\mathcal{R}$. After all K hierarchies have been decoded, a final repair step decodes the variables in $\mathcal{R}$ conditioned on the full context of retained assignments, i.e., $\mathbf{p}_{\mathcal{R}} = f_\theta\big(G, \mathcal{R}, \mathbf{h}_{\mathcal{V}\setminus\mathcal{R}}\big)$, and the repaired predictions replace the masked ones to yield the final solution. This iterative masking-and-prediction paradigm shares a similar spirit with the recent work Apollo-MILP (Liu et al. 2025). However, our method differs in two key aspects: (1) it explicitly exploits the variable coupling structure to establish a hierarchy that guides the decoding process; (2) it constitutes an inherently new prediction paradigm that operates entirely within the prediction stage, rather than following Apollo-MILP's alternation between prediction and solver invocation.

Training Objective. The framework is trained by minimizing a weighted prediction loss aggregated over all hierarchies and the repair step. Given S supervised solutions with objective values $\{o_s\}_{s=1}^S$, each solution is assigned a weight $w_s = \exp(-o_s/\tau) \,/\, \sum_{j=1}^S \exp(-o_j/\tau)$, where higher-quality solutions contribute more to the supervision and τ controls the weighting smoothness. The overall training objective is defined as

$$\mathcal{L} = \frac{1}{K}\sum_{k=1}^{K}\sum_{s=1}^{S} w_s \sum_{i\in H_k} \ell\Big(p_i^{(k)}, y_{s,i}\Big) + \sum_{s=1}^{S} w_s \sum_{i\in\mathcal{R}} \ell\Big(p_i^{(\text{rep})}, y_{s,i}\Big),$$

where $p_i^{(k)}$ and $p_i^{(\text{rep})}$ denote the predicted probabilities of variable i in the k-th hierarchy and the repair step, respec-

tively, $y_{s,i}$ denotes the label of variable i in solution s, and $\ell(\cdot,\cdot)$ is the binary cross-entropy loss. Note that when computing the k-th prediction, the preceding $k-1$ states are decoded by the model itself, which aligns with the inference procedure and thus mitigates the distributional shift between training and inference. However, fully relying on model-decoded states can hinder convergence, as early predictions are highly inaccurate. We therefore incorporate a *teacher forcing* strategy that employs ground-truth assignments as conditioning information in the early stage of training, and the proportion of teacher forcing is progressively decayed as training proceeds (See Appendix E.2).

3.3 Structure-Aware Variable Fixing Strategy

Existing variable fixing strategies typically select variables solely according to their prediction confidences. However, fixing variables with similar confidence levels may have substantially different structural influences on the reduced MILP problem, so treating them uniformly overlooks the valuable structural information.

Motivated by this observation, we propose a structure-aware variable fixing strategy that jointly considers the variable coupling structure and the prediction confidences. Specifically, we first identify variables whose variable coupling scores exceed a prescribed threshold as high-coupling variables. Fixing such variables tends to tighten the feasible domains of many others through shared constraints, and thus effectively reduces the search space. The identified variables are subsequently screened according to their prediction confidences, where variables with $p_i \geq \theta_1$ are fixed to one, and those with $p_i \leq \theta_0$ are fixed to zero. This two-stage screening guarantees that only strongly coupled variables with highly reliable predictions are eligible for fixing.

To align with existing methods and ensure a fair comparison, we further cap the number of fixed variables: the variables selected above are ranked by their prediction probabilities, and at most k_1 and k_0 of them are retained for fixing to one and to zero, respectively. Note that k_1 and k_0 serve only as upper bounds. If the thresholds yield fewer variables, no additional ones are fixed to reach these bounds. In contrast to confidence-only fixing, the proposed strategy prioritizes variables that are both structurally influential and confidently predicted, thereby enabling downstream optimization frameworks to explicitly exploit structural information for more effective search-space reduction.

4 Experiments

4.1 Experimental Setup

Datasets We evaluate our framework on four widely adopted MILP benchmarks in this field: Combinatorial Auctions (CA) (Leyton-Brown, Pearson, and Shoham 2000), Workload Appointment (WA) (Gasse et al. 2022), Item Placement (IP) (Gasse et al. 2022), and Set Covering (SC) (Balas and Ho 1980). Following existing studies (Gasse et al. 2019; Han et al. 2023; Liu et al. 2025), we generate SC and CA instances using similar procedures, while the IP and WA benchmarks are obtained from two real-world problems used in NeurIPS ML4CO 2021 competition (Gasse et al. 2022). For each benchmark problem, we use 240 instances for training, 60 instances for validation, and 100 instances for testing. Additional details are provided in Appendix D.

Baselines We compare our method with both traditional MILP solvers and learning-based solution prediction approaches. For traditional optimization methods, we compare our method with Gurobi (Gurobi Optimization, LLC 2026) and SCIP (Achterberg 2009). For learning-based approaches, we consider three representative methods: Neural Diving (ND) (Nair et al. 2020), Predict-and-Search (PaS) (Han et al. 2023), and Apollo-MILP (Liu et al. 2025).

Metrics. We compare the best objective values (OBJ) achieved by different methods within a 1000-second time limit on each test instance. Following the evaluation protocol in Predict-and-Search (Han et al. 2023), we use the best solution obtained by a single-threaded Gurobi solver with a 3600-second time limit as the best-known solution (BKS), which serves as an approximation of the optimal value. The absolute primal gap is then calculated as $\text{Gap}_{abs} = |\text{OBJ}-\text{BKS}|$, measuring the distance between the obtained solution and the BKS. It is worth noting that our method achieves better solutions than the Gurobi solver with a 3600-second time limit on certain benchmark problems. In such cases, the objective values found by our method are used to update the BKS.

4.2 Main Evaluation

Solving Performance To evaluate the effectiveness of our method, we replace the original GNN predictor and the variable fixing strategy in each framework with our SHSP predictor and structure-aware variable fixing strategy, forming **ND+SHSP**, **PaS+SHSP**, and **Apollo+SHSP**, respectively. We compare the solving performance between our framework and the baselines under a time limit of 1,000 seconds. Although SHSP performs hierarchical multi-step prediction, its additional neural network inference time is less than 0.2 seconds across all benchmark datasets. The graph construction time ranges from 0.26 to 15.23 seconds and is included in the 1,000-second time budget for a fair comparison. Table 1 presents the average objective values and the corresponding average absolute primal gaps across four benchmark datasets, with Gurobi employed as the downstream solver. Similar results obtained with the SCIP solver are reported in Table 2. Detailed results of graph construction time and neural network inference time are provided in Appendix F.3.

Overall, SHSP consistently improves performance across all three learning-guided search frameworks compared with their original one-shot prediction baselines. With PaS, SHSP reduces the absolute primal gap by 99.7% on CA, nearly closing the gap entirely, along with reductions of 38.1%, 24.3%, and 42.1% on WA, IP, and SC, respectively. With Apollo, the gains are similarly substantial: the primal gap is completely eliminated on CA (100.0% reduction) and reduced by 92.9% on IP, with further reductions of 28.6% and 41.7% on WA and SC. When integrated with ND, SHSP also delivers solid improvements, reducing the absolute primal gap by 94.4% on IP and 67.0% on CA, and by 19.2% on WA. Remarkable improvements are also observed when using SCIP as the downstream solver. As shown in Table 2,

Method	CA ↑ (BKS:98489.24)		WA ↓ (BKS:706.86)		IP ↓ (BKS:11.69)		SC ↓ (BKS:123.30)	
	Obj	Gap_{abs}	Obj	Gap_{abs}	Obj	Gap_{abs}	Obj	Gap_{abs}
Gurobi (3600s)	98453.42	35.82	706.86	0.00	11.69	0.00	123.30	0.00
Gurobi (1000s)	97407.51	1081.73	707.07	0.21	12.16	0.47	123.54	0.24
ND	94340.63	4148.61	707.12	0.26	14.18	2.49	123.51	0.21
ND+SHSP	**97121.05**	**1368.19**	**707.07**	**0.21**	**11.83**	**0.14**	123.51	0.21
Improvement	-	67.02%	-	19.23%	-	94.38%	-	0.00%
PaS	97891.74	597.50	707.07	0.21	12.06	0.37	123.49	0.19
PaS+SHSP	**98487.72**	1.52	**706.99**	**0.13**	**11.97**	0.28	**123.41**	0.11
Improvement	-	99.75%	-	38.1%	-	24.32%	-	42.11%
Apollo	97997.74	491.50	707.07	0.21	11.97	0.28	123.42	0.12
Apollo+SHSP	**98489.24**	**0.00**	**707.01**	**0.15**	**11.71**	**0.02**	**123.37**	**0.07**
Improvement	-	100.00%	-	28.6%	-	92.86%	-	41.67%

Table 1: Performance comparison on four benchmark datasets (CA, WA, IP, and SC) using Gurobi under a 1000-second time limit. ↑ indicates that higher values are better, while ↓ indicates that lower values are better. Bold values denote the better results. Improvement denotes the relative reduction in Gap_{abs} relative to the corresponding downstream solution-prediction framework.

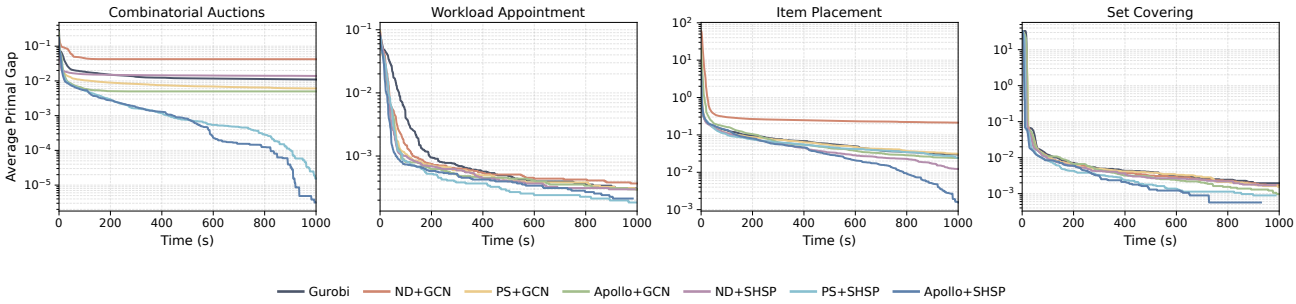

Figure 2: The convergence curves of primal gaps as the solving process proceeds. All methods are implemented using Gurobi with a total time budget of 1,000 seconds. For a fair comparison, the graph construction time of SHSP is included in this budget. Results are averaged over 100 testing instances. A lower primal gap indicates faster convergence and better solving performance.

SHSP consistently reduces the absolute primal gap across all three frameworks on all four benchmarks, demonstrating that the proposed method generalizes well across different downstream solvers. It is worth noting that Apollo+SHSP reaches the best known solution, surpassing the solution obtained by Gurobi with a 3,600-second time limit on the CA benchmark. This observation suggests that our method can provide a stronger initialization for the downstream solver, not only accelerating the search process but also guiding it toward higher-quality solutions.

Primal Gap as a Function of Runtime Figure 2 presents the curves of the average primal gap, defined as $\text{Gap}_{\text{rel}} = \frac{|\text{OBJ}-\text{BKS}|}{|\text{BKS}|}$, throughout the solving process. Overall, SHSP-based methods consistently exhibit faster convergence and achieves lower primal gaps on the benchmarks than corresponding one-shot prediction methods. This behavior is attributed to more accurate solution predictions and more effective search-space reduction of our method.

4.3 Ablation Study

Confidence-Based Mask and Repair Mechanism To evaluate the effectiveness of the proposed mask-and-repair mechanism, we compare three settings: without mask and repair (None), with mask stage only (Mask), and with both mask and repair (Mask+Repair). As shown in Table 3, applying the masking mechanism alone does not consistently improve performance and may even cause slight degradation, since masked variables are simply discarded without correction, resulting in a loss of predictive information. In contrast, coupling masking with the proposed repair strategy consistently yields the best results across all datasets and both prediction frameworks, confirming that the two components are complementary and jointly contribute to the overall performance. More results are provided in Appendix F.1.

Hierarchical Solution Prediction Predictor To evaluate the effectiveness of the proposed hierarchical solution prediction predictor, we compare it with the original GNN predictor under the same structure-aware fixing strategy. As shown in

Method	CA ↑ (BKS:98489.24)		WA ↓ (BKS:706.86)		IP ↓ (BKS:11.69)		SC ↓ (BKS:123.30)	
	Obj	Gap_{abs}	Obj	Gap_{abs}	Obj	Gap_{abs}	Obj	Gap_{abs}
SCIP (3600s)	97795.78	693.46	708.21	1.35	19.06	7.37	124.58	1.28
SCIP (1000s)	96932.37	1556.87	709.62	2.76	22.69	11.00	125.81	2.51
ND	94338.51	4150.73	707.83	0.97	16.33	4.64	125.87	2.57
ND+SHSP	**97267.46**	**1221.78**	**707.18**	**0.32**	**13.70**	**2.01**	**125.84**	**2.54**
Improvement	-	70.57%	-	67.01%	-	56.68%	-	1.17%
PaS	97223.78	1265.46	708.27	1.41	19.61	7.92	125.82	2.52
PaS+SHSP	**97230.19**	**1259.05**	**708.21**	**1.35**	**19.44**	**7.75**	**125.33**	**2.03**
Improvement	-	0.51%	-	4.26%	-	2.15%	-	19.44%
Apollo	96270.42	2218.82	708.42	1.56	20.48	8.79	125.74	2.44
Apollo+SHSP	**97192.31**	**1296.93**	**708.21**	**1.35**	**18.56**	**6.87**	**125.45**	**2.15**
Improvement	-	41.55%	-	13.46%	-	21.84%	-	11.89%

Table 2: Performance comparison on four benchmark datasets (CA, WA, IP, and SC) using SCIP under a 1000-second time limit. ↑ indicates that higher values are better, while ↓ indicates that lower values are better. Bold values denote the better results. Improvement denotes the relative reduction in Gap_{abs} relative to the corresponding downstream solution-prediction framework.

Method	CA ↑	IP ↓
PaS+SHSP (None)	98380.92	12.12
PaS+SHSP (Mask)	98416.55	12.17
PaS+SHSP (Mask+Repair)	**98487.72**	**11.97**
Apollo+SHSP (None)	98385.20	11.78
Apollo+SHSP (Mask)	98344.07	11.80
Apollo+SHSP (Mask+Repair)	**98489.24**	**11.71**

Table 3: Ablation study of the proposed confidence-based mask-and-repair strategy on the CA and IP datasets. Bold values denote the better results.

Method (Fixing Strategy)	CA ↑	IP ↓
PaS (Structure-Aware)	97184.33	12.12
PaS+SHSP (Original)	98061.51	12.34
PaS+SHSP (Structure-Aware)	**98487.72**	**11.97**
Apollo (Structure-Aware)	98015.32	11.87
Apollo+SHSP (Original)	98070.43	12.08
Apollo+SHSP (Structure-Aware)	**98489.24**	**11.71**

Table 4: Ablation study of the proposed hierarchical solution prediction and structure-aware fixing strategy on the CA and IP datasets. Bold values denote the better results.

Table 4, replacing original GNN predictor with hierarchical solution prediction predictor consistently improves the solution quality on both the PaS and Apollo frameworks. The improvement suggests that our predictor generates more accurate solution predictions, thereby enabling more effective variable fixing and subsequent problem reduction. Results on other problems are provided in Appendix F.1.

Structure-Aware Fixing Strategies To evaluate the effectiveness of the proposed structure-aware fixing strategy, we compare it with the original fixing strategy while using the same predictor. As shown in Table 4, the proposed structure-aware fixing strategy consistently outperforms the normal fixing strategy on both the PaS and Apollo frameworks. The improvement suggests that prioritizing structurally important variables leads to more effective search space reduction, allowing the solver to explore the remaining search space more efficiently and ultimately achieve better solving performance. Results on other problems are provided in Appendix F.1.

5 Conclusion

In this paper, we propose SHSP, a structure-aware hierarchical solution prediction framework for MILP solving. SHSP replaces the parallel marginal decoding with a hierarchical conditional decoding mechanism guided by a variable coupling graph, decoding variables from weakly to strongly coupled ones while conditioning each step on previously predicted assignments. It further incorporates a mask-and-repair mechanism to mitigate error accumulation, and a structure-aware fixing strategy to enable more effective search-space reduction in downstream frameworks. Extensive experiments on four standard benchmarks show that SHSP consistently outperforms one-shot baselines across three frameworks and two downstream solvers, achieving a 54% average reduction in the absolute primal gap with negligible inference overhead. In future work, we plan to learn coupling scores in an end-to-end manner and extend to broader problem classes.

References

Achterberg, T. 2009. SCIP: Solving constraint integer programs. *Mathematical Programming Computation*, 1(1): 1–41.

Balas, E.; and Ho, A. 1980. *Set Covering Algorithms Using Cutting Planes, Heuristics, and Subgradient Optimization: A Computational Study*. Springer.

Chmiela, A.; Khalil, E.; Gleixner, A.; Lodi, A.; and Pokutta, S. 2021. Learning to Schedule Heuristics in Branch and Bound. In *Advances in Neural Information Processing Systems 34*, 24235–24246. Virtual.

Ding, J.-Y.; Zhang, C.; Shen, L.; Li, S.; Wang, B.; Xu, Y.; and Song, L. 2020. Accelerating Primal Solution Findings for Mixed Integer Programs Based on Solution Prediction. In *Proceedings of the 34th AAAI Conference on Artificial Intelligence*, 1452–1459. New York, NY.

Gasse, M.; Bowly, S.; Cappart, Q.; Charfreitag, J.; Charlin, L.; Chetelat, D.; Chmiela, A.; Dumouchelle, J.; Gleixner, A.; Kazachkov, A. M.; Khalil, E.; Lichocki, P.; Lodi, A.; Lubin, M.; Maddison, C. J.; Christopher, M.; Papageorgiou, D. J.; Parjadis, A.; Pokutta, S.; Prouvost, A.; Scavuzzo, L.; Zarpellon, G.; Yang, L.; Lai, S.; Wang, A.; Luo, X.; Zhou, X.; Huang, H.; Shao, S.; Zhu, Y.; Zhang, D.; Quan, T.; Cao, Z.; Xu, Y.; Huang, Z.; Zhou, S.; Chen, B.; He, M.; Hao, H.; Zhang, Z.; An, Z.; and Mao, K. 2022. The Machine Learning for Combinatorial Optimization Competition (ML4CO): Results and Insights. In *Proceedings of the NeurIPS 2021 Competitions and Demonstrations Track*, 220–231. Virtual.

Gasse, M.; Chetelat, D.; Ferroni, N.; Charlin, L.; and Lodi, A. 2019. Exact combinatorial optimization with graph convolutional neural networks. In *Advances in Neural Information Processing Systems 33*, 15554–15566. Vancouver, Canada.

Geng, Z.; Wang, J.; Li, X.; Zhu, F.; Hao, J.; Li, B.; and Wu, F. 2025. Differentiable Integer Linear Programming. In *Proceedings of the 13th International Conference on Learning Representations*. Singapore.

Gleixner, A.; Hendel, G.; Gamrath, G.; Achterberg, T.; Bastubbe, M.; Berthold, T.; Christophel, P.; Jarck, K.; Koch, T.; Linderoth, J.; et al. 2021. MIPLIB 2017: Data-Driven Compilation of the 6th Mixed-Integer Programming Library. *Mathematical Programming Computation*, 13(3): 443–490.

Gupta, P.; Gasse, M.; Khalil, E.; Mudigonda, P.; Lodi, A.; and Bengio, Y. 2020. Hybrid Models for Learning to Branch. In *Advances in Neural Information Processing Systems 34*, 18087–18097. Virtual.

Gupta, P.; Khalil, E. B.; Chetelat, D.; Gasse, M.; Bengio, Y.; Lodi, A.; and Kumar, M. P. 2022. Lookback for Learning to Branch. arXiv:2206.14987.

Gurobi Optimization, LLC. 2026. Gurobi Optimizer Reference Manual.

Han, Q.; Yang, L.; Chen, Q.; Zhou, X.; Zhang, D.; Wang, A.; Sun, R.; and Luo, X. 2023. A GNN-guided predict-and-search framework for mixed-integer linear programming. In *Proceedings of the 11th International Conference on Learning Representations*. Kigali, Rwanda.

Hou, Z.; Li, X.; Zhang, Y.; Li, T.; and You, K. 2026. LLM4Branch: Large Language Model for Discovering Efficient Branching Policies of Integer Programs. arXiv:2605.10401.

Huang, T.; Ferber, A.; Zharmagambetov, A.; Tian, Y.; and Dilkina, B. 2024. Contrastive Predict-and-Search for Mixed Integer Linear Programs. In *Proceedings of the 41st International Conference on Machine Learning*, 19406–19424. Vienna, Austria.

Huang, Z.; Wang, K.; Liu, F.; Zhen, H.-L.; Zhang, W.; Yuan, M.; Hao, J.; Yu, Y.; and Wang, J. 2021. Learning to Select Cuts for Efficient Mixed-Integer Programming. arXiv:2105.13645.

Karp, R. M. 1972. *Complexity of Computer Computations*. Plenum Press.

Khalil, E. B.; Dilkina, B.; Nemhauser, G. L.; Ahmed, S.; and Shao, Y. 2017. Learning to Run Heuristics in Tree Search. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence*, 659–666. Melbourne, Australia.

Khalil, E. B.; Le Bodic, P.; Song, L.; Nemhauser, G.; and Dilkina, B. 2016. Learning to branch in mixed integer programming. In *Proceedings of the 30th AAAI Conference on Artificial Intelligence*, 724–731. Phoenix, AZ.

Kuang, Y.; Wang, J.; Liu, H.; Zhu, F.; Li, X.; Zeng, J.; Hao, J.; Li, B.; and Wu, F. 2024. Rethinking Branching on Exact Combinatorial Optimization Solver: The First Deep Symbolic Discovery Framework. In *Proceedings of the 12th International Conference on Learning Representations*. Vienna, Austria.

Leyton-Brown, K.; Pearson, M.; and Shoham, Y. 2000. Towards a Universal Test Suite for Combinatorial Auction Algorithms. In *Proceedings of the 2nd ACM Conference on Electronic Commerce*, 66–76. Minneapolis, MN.

Li, X.; Zhu, F.; Zhen, H.-L.; Luo, W.; Lu, M.; Huang, Y.; Fan, Z.; Zhou, Z.; Kuang, Y.; Wang, Z.; Geng, Z.; Li, Y.; Liu, H.; An, Z.; Yang, M.; Li, J.; Wang, J.; Yan, J.; Sun, D.; Zhong, T.; Zhang, Y.; Zeng, J.; Yuan, M.; Hao, J.; Yao, J.; and Mao, K. 2024a. Machine Learning Insides OptVerse AI Solver: Design Principles and Applications. arXiv:2401.05960.

Li, Y.; Wang, W.; Xu, W.; Deng, Y.; and Wu, W. 2024b. Factor graph neural network meets max-sum: A real-time route planning algorithm for massive-scale trips. In *Proceedings of the 23rd International Conference on Autonomous Agents and Multiagent Systems*, 1165–1173. Auckland, New Zealand.

Lin, J.; Xu, M.; Xiong, Z.; and Wang, H. 2024. CAMBranch: Contrastive Learning with Augmented MILPs for Branching. In *Proceedings of the 12th International Conference on Learning Representations*. Vienna, Austria.

Ling, H.; Wang, Z.; and Wang, J. 2024. Learning to Stop Cut Generation for Efficient Mixed-Integer Linear Programming. In *Proceedings of the 38th AAAI Conference on Artificial Intelligence*, 20759–20767. Vancouver, Canada.

Liu, H.; Wang, J.; Geng, Z.; Li, X.; Zong, Y.; Zhu, F.; Hao, J.; and Wu, F. 2025. Apollo-MILP: An alternating prediction-correction neural solving framework for mixed-integer linear

programming. In *Proceedings of the 13th International Conference on Learning Representations*. Singapore.

M., A.; Tandon, R.; Gupta, A.; KODAMANA, H.; and Ramteke, M. 2026. MIRACLE: Model-free Imitation and Reinforcement Learning for Adaptive Cut-Selection. In *Proceedings of the 14th International Conference on Learning Representations*. Rio de Janeiro, Brazil.

Ma, K.; Xiao, L.; Zhang, J.; and Li, T. 2019. Accelerating an FPGA-based SAT solver by software and hardware co-design. *Chinese Journal of Electronics*, 28(5): 953–961.

Nair, V.; Bartunov, S.; Gimeno, F.; von Glehn, I.; Lichocki, P.; Lobov, I.; O'Donoghue, B.; Sonnerat, N.; Tjandraatmadja, C.; Wang, P.; et al. 2020. Solving Mixed Integer Programs Using Neural Networks. arXiv:2012.13349.

Paulus, M. B.; and Krause, A. 2023. Learning to Dive in Branch and Bound. In *Advances in Neural Information Processing Systems 36*, 34260–34277. New Orleans, LA.

Paulus, M. B.; Zarpellon, G.; Krause, A.; Charlin, L.; and Maddison, C. J. 2022. Learning to Cut by Looking Ahead: Cutting Plane Selection via Imitation Learning. In *Proceedings of the 39th International Conference on Machine Learning*, 17584–17600. Baltimore, MD.

Pu, T.; Li, J.; Gao, Y.; Liu, S.; Geng, Z.; Liu, H.; Chen, C.; and Fan, C. 2026. CoCo-MILP: Inter-Variable Contrastive and Intra-Constraint Competitive MILP Solution Prediction. In *Proceedings of the 40th AAAI Conference on Artificial Intelligence*, 24882–24890. Singapore.

Puigdemont, P.; Skoulakis, S.; Chrysos, G.; and Cevher, V. 2024. Learning to Remove Cuts in Integer Linear Programming. In *Proceedings of the 41st International Conference on Machine Learning*, 41235–41255. Vienna, Austria.

Scavuzzo, L.; Chen, F.; Chetelat, D.; Gasse, M.; Lodi, A.; Yorke-Smith, N.; and Aardal, K. 2022. Learning to Branch with Tree MDPs. In *Advances in Neural Information Processing Systems 35*, 18514–18526. New Orleans, LA.

Tang, Y.; Agrawal, S.; and Faenza, Y. 2020. Reinforcement Learning for Integer Programming: Learning to Cut. In *Proceedings of the 37th International Conference on Machine Learning*, 9367–9376. Virtual.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, Ł.; and Polosukhin, I. 2017. Attention Is All You Need. In *Advances in Neural Information Processing Systems 30*, 5998–6008. Long Beach, CA.

Wang, J.; Wang, Z.; Li, X.; Kuang, Y.; Shi, Z.; Zhu, F.; Yuan, M.; Zeng, J.; Zhang, Y.; and Wu, F. 2024. Learning to Cut via Hierarchical Sequence/Set Model for Efficient Mixed-Integer Programming. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 46(12): 9697–9713.

Wang, R.; Li, X.; and Wang, M. 2026. MILPnet: A Multi-Scale Architecture with Geometric Feature Sequence Representations for Advancing MILP Problems. In *Proceedings of the 14th International Conference on Learning Representations*. Rio de Janeiro, Brazil.

Wang, Z.; Li, X.; Wang, J.; Kuang, Y.; Yuan, M.; Zeng, J.; Zhang, Y.; and Wu, F. 2023. Learning Cut Selection for Mixed-Integer Linear Programming via Hierarchical Sequence Model. In *Proceedings of the 11th International Conference on Learning Representations*. Kigali, Rwanda.

Yoon, T. 2022. Confidence Threshold Neural Diving. arXiv:2202.07506.

Zarpellon, G.; Jo, J.; Lodi, A.; and Bengio, Y. 2021. Parameterizing Branch-and-Bound Search Trees to Learn Branching Policies. In *Proceedings of the 35th AAAI Conference on Artificial Intelligence*, 3931–3939. Virtual.

Zeng, H.; Wang, J.; Das, A.; He, J.; Han, K.; Hu, H.; and Sun, M. 2024. Effective Generation of Feasible Solutions for Integer Programming via Guided Diffusion. arXiv:2406.12349.

Zhang, C.; Ouyang, W.; Yuan, H.; Gong, L.; Sun, Y.; Guo, Z.; Dong, Z.; and Yan, J. 2024. Towards Imitation Learning to Branch for MIP: A Hybrid Reinforcement Learning Based Sample Augmentation Approach. In *Proceedings of the 12th International Conference on Learning Representations*. Vienna, Austria.

A Related Work

A.1 Learning alongside MILP Solvers

In recent years, machine learning has emerged as a promising paradigm for accelerating MILP solving (Li et al. 2024a), and existing studies generally fall into two categories. The first category learns heuristic policies for key components of branch-and-bound (B&B) solvers, including branching variable selection (Khalil et al. 2016; Gasse et al. 2019; Gupta et al. 2020; Zarpellon et al. 2021; Gupta et al. 2022; Scavuzzo et al. 2022; Lin et al. 2024; Zhang et al. 2024; Kuang et al. 2024), cutting plane selection (Tang, Agrawal, and Faenza 2020; Wang et al. 2023, 2024; Huang et al. 2021; Paulus et al. 2022; Ling, Wang, and Wang 2024; Puigdemont et al. 2024), scheduling of primal heuristics (Khalil et al. 2017; Chmiela et al. 2021). These methods typically leverage imitation learning or reinforcement learning to learn effective heuristic policies from historical solving trajectories or expert policies.

Among all components, branching and cut selection have received the most attention due to their significant impact on solver performance. For branching, Khalil et al. (2016) pioneered the imitation of strong branching decisions via a learned ranking function, and more recently, Hou et al. (2026) leveraged large language models to automatically discover branching policies. For cut selection, Tang, Agrawal, and Faenza (2020) proposed to learn adaptive selection policies through reinforcement learning, while M. et al. (2026) further combined imitation learning and reinforcement learning to obtain memory-efficient policies. Collectively, these studies demonstrate that learning effective policies for individual solver components can substantially enhance the performance of modern MILP solvers.

A.2 End-to-End Prediction for MILP

The second category employs machine learning models such as graph neural networks (GNNs) to directly predict high-quality solutions for MILP instances (Ding et al. 2020; Zeng et al. 2024; Geng et al. 2025). Research in this category advances along two complementary aspects. The first aspect concerns how predictions are exploited to reduce the original problem (Nair et al. 2020; Han et al. 2023; Liu et al. 2025). Nair et al. (2020) proposed to directly fix high-confidence variables to their predicted values, yielding a reduced subproblem, which was subsequently refined in follow-up studies (Yoon 2022; Paulus and Krause 2023). Han et al. (2023) and Huang et al. (2024) relaxed such hard fixing by searching within a trust region centered at the predicted partial assignment, and Liu et al. (2025) further extended this paradigm by alternating between prediction and solver-based correction.

The second aspect focuses on improving the prediction capacity (Pu et al. 2026; Wang, Li, and Wang 2026). Pu et al. (2026) enhanced prediction quality by explicitly modeling inter-variable correlations, while Wang, Li, and Wang (2026) replaced the conventional bipartite graph representation with geometric feature sequences and performed prediction via a multi-scale attention architecture. Together, these efforts have established learning-based solution prediction as an effective paradigm for accelerating MILP solving.

B Graph Representation Details

The bipartite graph representation used in this paper is based on the representation adopted by Predict-and-Search (Han et al. 2023). To support hierarchical decoding, we further introduce three additional variable features: target indicator, previous prediction, and previous confidence. We list the graph features in Table 5.

Index	Variable Feature Name	Description
0	Objective	Normalized objective coefficient
1	Variable coefficient	Average variable coefficient in all constraints
2	Variable degree	Degree of the variable node in the bipartite graph representation
3	Maximum variable coefficient	Maximum variable coefficient in all constraints
4	Minimum variable coefficient	Minimum variable coefficient in all constraints
5	Variable type	Whether the variable is an integer variable or not
6	**Target indicator**	**Whether the variable belongs to the current prediction hierarchy**
7	**Previous prediction**	**Previously predicted value of the variable**
8	**Previous confidence**	**Confidence score of the predicted value**
9–24	Position embedding	Binary encoding of the order of appearance for each variable among all variables
Index	Constraint Feature Name	Description
0	Constraint coefficient	Average of all coefficients in the constraint
1	Constraint degree	Degree of constraint nodes
2	Bias	Normalized right-hand side of the constraint
3	Sense	The sense of the constraint
Index	Edge Feature Name	Description
0	Coefficient	Constraint coefficient

Table 5: The variable features, constraint features, and edge features used for the graph representation. Newly introduced features are shown in bold.

C Variable Coupling Graph Details

Expected Violation The *expected violation* measures how strongly a variable pair is restricted by a constraint. Violation of the variable pair (z_i, z_j) in constraint c is defined as

$$\Delta\left(\phi_{ij}^{(c)}\right) = \begin{cases} \max(b_c - \phi_{ij}^{(c)}, 0), & \circ_c \text{ is } \geq, \\ \max(\phi_{ij}^{(c)} - b_c, 0), & \circ_c \text{ is } \leq, \\ \left|\phi_{ij}^{(c)} - b_c\right|, & \circ_c \text{ is } = . \end{cases}$$

where $\phi_{ij}^{(c)} = a_{ci} z_i + a_{cj} z_j$, with a_{ci} and a_{cj} denoting the coefficients of variables z_i and z_j in constraint c, respectively.

The expected violation $P_{ij}^{(c)} = \mathbb{E}\left[\Delta\left(\phi_{ij}^{(c)}\right)\right]$ is computed as follows. For a binary–binary pair (z_i, z_j), we assume $z_i, z_j \sim \mathrm{Uniform}(\{0,1\})$, and the expected violation is computed as

$$P_{ij}^{(c)} = \frac{1}{4} \sum_{z_i, z_j \in \{0,1\}} \Delta\left(\phi_{ij}^{(c)}\right).$$

For a binary–continuous pair (z_i, z_j), we assume $z_i \sim \mathrm{Uniform}(\{0,1\})$ and $z_j \sim \mathrm{Uniform}(l_j, u_j)$, where l_j and u_j denote the lower and upper bounds of z_j. The expected violation is computed as

$$P_{ij}^{(c)} = \frac{1}{2} \sum_{z_i \in \{0,1\}} \frac{1}{u_j - l_j} \int_{l_j}^{u_j} \Delta\left(\phi_{ij}^{(c)}\right) dz_j.$$

To eliminate scale differences across constraints and measure the relative strength of a variable pair within the constraint, we normalize the expected violation as

$$\widehat{P}_{ij}^{(c)} = \frac{P_{ij}^{(c)}}{\frac{1}{|E_c|}\sum_{(u,v)\in E_c} P_{uv}^{(c)}},$$

where E_c denotes the set of retained variable pairs in constraint c.

Coefficient Importance The *coefficient importance* characterizes the relative contribution of a variable pair within a constraint. For each variable z_i, we define a scale factor s_i according to its variable type, where s_i is set to 1 for binary variables and to $u_i - l_i$ for continuous variables. Then we compute the coefficient importance $r_{ij}^{(c)}$ of the variable pair and normalize it within the constraint to obtain $\widehat{r}_{ij}^{(c)}$, which represents the relative importance of a variable pair within the constraint:

$$r_{ij}^{(c)} = \frac{|a_{ci}|s_i + |a_{cj}|s_j}{2},$$

$$\widehat{r}_{ij}^{(c)} = \frac{r_{ij}^{(c)}}{\frac{1}{|E_c|}\sum_{(u,v)\in E_c} r_{uv}^{(c)}}.$$

Edge Weight Aggregation. Finally, the local coupling weight of a pair in constraint c is computed as $w_{ij}^{(c)} = \widehat{P}_{ij}^{(c)}\widehat{r}_{ij}^{(c)}$, and the final edge weight W_{ij} between variables z_i and z_j is obtained by summing the local weights over all constraints containing both variables.

D Benchmark Details

D.1 Benchmarks in Main Evaluation

The CA and SC benchmark instances are generated following the process described by Gasse et al. (2019). Specifically, the CA dataset follows the generation method of Leyton-Brown, Pearson, and Shoham (2000), while the SC dataset is constructed using the algorithm presented by Balas and Ho (1980). The IP and WA instances are obtained from the NeurIPS ML4CO 2021 competition (Gasse et al. 2022). The statistical information is provided in Table 6.

	CA	WA	IP	SC
Constraint Number	2592	64318	195	3000
Variable Number	1500	61000	1083	5000
Binary Variables	1500	1000	1050	5000
Continuous Variables	0	60000	33	0
Integer Variables	0	0	0	0

Table 6: Summary statistics of the benchmark datasets used in the main evaluation.

D.2 Subset of MIPLIB

We use the IIS dataset, which is a subset of MIPLIB (Gleixner et al. 2021) constructed by Liu et al. (2025), to evaluate the solvers' ability to handle challenging real-world instances. According to Liu et al. (2025), the instances in the dataset are selected based on their similarity, which is measured using the 100 human-designed features (Gleixner et al. 2021). Instances with presolving times exceeding 300 seconds or those that exceed GPU memory limits during the inference process are discarded. The IIS dataset contains eleven instances, including eight training instances and three testing instances (ramos3, scpj4scip, and scpl4). Detailed statistics of the IIS dataset are reported in Table 7.

Instance	Constraint Number	Variable Number	Binary Variables
ex1010-pi	1468	25200	25200
fast0507	507	63009	63009
glass-sc	6119	214	214
iis-glass-cov	5375	214	214
iis-hc-cov	9727	297	297
ramos3	2187	2187	2187
scpj4scip	1000	99947	99947
scpk4	2000	100000	100000
scpl4	2000	200000	200000
seymour	4944	1372	1372
v150d30-2hopcds	7822	150	150

Table 7: Statistical information of the instances in the IIS dataset. All instances contain only binary variables.

E Implementation Details

E.1 SHSP Predictor Details

The predictor used in SHSP is based on the graph neural network architecture adopted in previous learning-based MILP approaches (Han et al. 2023; Liu et al. 2025). To support the hierarchical decoding process in SHSP, we introduce a learnable step embedding, which enables the network to adapt its prediction behavior across different decoding steps. Specifically, the initial representation of each variable node is computed as

$$h_{v_i}^{(0,k)} = f_s\left(\left[f_v\left(x_i\right); \mathrm{Emb}(k)\right]\right).$$

Here, v_i denotes the i-th variable node, k is the current decoding step, and x_i is the feature vector of v_i. The function $f_v(\cdot)$ denotes the variable embedding network, $\mathrm{Emb}(k)$ is the learnable embedding corresponding to decoding step k, and $f_s(\cdot)$ fuses the variable and step embeddings. The resulting representation $h_{v_i}^{(0,k)}$ is then used as the input to the same bipartite message-passing architecture adopted in previous approaches (Han et al. 2023; Liu et al. 2025).

E.2 Training Details

During predictor training, we set the initial learning rate to be 0.001 and the number of training epochs to 500. To collect the training data, we run a single-threaded Gurobi (Gurobi Optimization, LLC 2026) on each training and validation instance for 3,600 seconds and record the best 50 solutions.

For SHSP training, we incorporate a *teacher forcing* strategy that employs ground-truth assignments as conditioning information in the early stage of training, and the teacher forcing ratio ρ_t is progressively decayed as training proceeds. Specifically, with probability ρ_t, we use the ground-truth value as the history input for the next step; otherwise, we use the model prediction. The teacher forcing ratio ρ_t is linearly decayed from ρ_{start} to ρ_{end} during the first T_{decay} epochs:

$$\rho_t = \rho_{\text{start}} + \min\left(1, \frac{t}{T_{\text{decay}}}\right)(\rho_{\text{end}} - \rho_{\text{start}}),$$

where t is the current epoch. During validation and inference, we disable teacher forcing and always use the model predictions to update the conditioning context. In experiments, we set $\rho_{\text{start}} = 1.0$, $\rho_{\text{end}} = 0$, and $T_{\text{decay}} = 50$.

E.3 Integration with Downstream Learning-based Search Methods

We replace the original GNN predictor and the variable fixing strategy in each framework with our SHSP predictor and structure-aware variable fixing strategy, forming **ND+SHSP**, **PaS+SHSP**, and **Apollo+SHSP**, respectively.

For ND+SHSP and PaS+SHSP, the variable coupling graph is constructed once before neural network inference for each instance. The graph construction time is included in the solver time budget.

For Apollo+SHSP, the variable coupling graph is constructed before the first prediction and is subsequently updated for each reduced subproblem before prediction. Similarly, the time required for graph construction and updates is deducted from the corresponding solver time budget.

E.4 Inference Details

We conducted all the experiments on a single machine with an AMD EPYC 7513 32-Core Processor and NVIDIA GeForce RTX 4090 GPUs. The inference settings are described as follows.

For the baselines, the hyperparameters (k_0, k_1) for ND and (k_0, k_1, Δ) for PaS are summarized in Table 8. For Apollo-MILP, the number of iterations is set to 4. The hyperparameter settings $(k_0^{(i)}, k_1^{(i)}, \Delta^{(i)})$ for each iteration are summarized in Table 9.

Dataset	PaS (k_0, k_1, Δ)	ND (k_0, k_1)
CA	$(600, 0, 20)$	$(600, 0)$
WA	$(0, 500, 10)$	$(0, 500)$
IP	$(400, 5, 10)$	$(400, 5)$
SC	$(2000, 0, 100)$	$(2000, 0)$

Table 8: Hyperparameter settings for PaS and ND on different benchmark datasets.

For our method, we use the same hyperparameter settings as the corresponding baseline framework. The number of decoding steps is set to 2 for all benchmark datasets except WA, for which it is set to 4. The additional hyperparameter settings $(\theta_0, \theta_1, \eta)$ are reported in Table 10, where η controls the VCS threshold. Specifically, let $z_{(1)}, \ldots, z_{(n)}$ denote the variables sorted in descending order of VCS. We use the VCS of $z_{(\lceil \eta n \rceil)}$ as the threshold.

	CA	WA	IP	SC
Iteration 1	$(400, 0, 60)$	$(20, 200, 100)$	$(100, 20, 50)$	$(1000, 0, 200)$
Iteration 2	$(200, 0, 30)$	$(10, 100, 50)$	$(40, 15, 20)$	$(500, 0, 100)$
Iteration 3	$(100, 0, 15)$	$(10, 5, 5)$	$(20, 15, 10)$	$(250, 0, 50)$
Iteration 4	$(50, 0, 10)$	$(1, 10, 5)$	$(5, 50, 30)$	$(10, 0, 5)$

Table 9: Hyperparameter configuration of $(k_0^{(i)}, k_1^{(i)}, \Delta^{(i)})$ in Apollo-MILP.

Dataset	ND+SHSP	PaS+SHSP	Apollo+SHSP
CA	$(0.10, 0.90, 0.5)$	$(0.10, 0.90, 0.5)$	$(0.04, 0.90, 0.7)$
WA	$(0.10, 0.90, 0.7)$	$(0.10, 0.90, 0.7)$	$(0.10, 0.95, 0.8)$
IP	$(0.05, 0.95, 0.7)$	$(0.10, 0.90, 0.7)$	$(0.10, 0.50, 0.5)$
SC	$(0.10, 0.90, 0.5)$	$(0.10, 0.90, 0.5)$	$(0.10, 0.90, 0.7)$

Table 10: The additional hyperparameter settings $(\theta_0, \theta_1, \eta)$ for ND+SHSP, PaS+SHSP and Apollo+SHSP on different benchmark datasets.

F Additional Experimental Results

F.1 More Ablation Study Results

Confidence-Based Mask and Repair Mechanism We perform an ablation study to evaluate the proposed mask-and-repair mechanism. Specifically, we compare three variants: None (without mask or repair), Mask (mask only), and Mask+Repair (mask and repair) on the four benchmark datasets (CA, WA, IP, and SC). The results are shown in Table 11. The observations are consistent with those discussed in the main paper: the two components are complementary and jointly contribute to the overall performance.

Hierarchical Solution Prediction Predictor To evaluate the effectiveness of the proposed hierarchical solution prediction predictor, we compare it with the original GNN predictor under the same structure-aware fixing strategy on the four benchmark datasets (CA, WA, IP, and SC). The results are shown in Table 12, which suggests that our predictor generates more accurate solution predictions, thereby enabling more effective variable fixing and problem reduction.

Fixing Strategies To evaluate the effectiveness of the proposed structure-aware fixing strategy, we compare it with the original fixing strategy while using the same predictor. The results on the four benchmark datasets (CA, WA, IP, and SC) are shown in Table 12. The observations are consistent with those discussed in the main paper: prioritizing structurally important variables leads to more effective search space reduction, allowing the solver to explore the remaining search space more efficiently and achieve better performance.

F.2 Results on MIPLIB

To further demonstrate the applicability of our method, we conduct experiments on instances from challenging real-

Method	CA ↑(BKS:98489.24)		WA ↓(BKS:706.86)		IP ↓(BKS:11.69)		SC ↓(BKS:123.30)	
	Obj	Gap_{abs}	Obj	Gap_{abs}	Obj	Gap_{abs}	Obj	Gap_{abs}
PaS+SHSP (None)	98380.92	108.32	707.06	0.20	12.12	0.43	123.47	0.17
PaS+SHSP (Mask)	98416.55	72.69	707.05	0.19	12.17	0.48	123.53	0.23
PaS+SHSP (Mask+Repair)	**98487.72**	**1.52**	**706.99**	**0.13**	**11.97**	**0.28**	**123.41**	**0.11**
Apollo+SHSP (None)	98385.20	104.04	707.10	0.24	11.78	0.09	123.42	0.12
Apollo+SHSP (Mask)	98344.07	145.17	**707.01**	**0.15**	11.80	0.11	123.39	0.09
Apollo+SHSP (Mask+Repair)	**98489.24**	**0.00**	**707.01**	**0.15**	**11.71**	**0.02**	**123.37**	**0.07**

Table 11: Detailed ablation study of the proposed confidence-based mask-and-repair strategy. ↑ indicates that higher values are better, while ↓ indicates that lower values are better. Bold values denote the best results, including ties.

Method	CA ↑(BKS:98489.24)		WA ↓(BKS:706.86)		IP ↓(BKS:11.69)		SC ↓(BKS:123.30)	
	Obj	Gap_{abs}	Obj	Gap_{abs}	Obj	Gap_{abs}	Obj	Gap_{abs}
PaS (structure-aware fixing)	97184.33	1304.91	707.08	0.22	12.12	0.43	123.49	0.19
PaS+SHSP (original fixing)	98061.51	427.73	707.01	0.15	12.34	0.65	123.53	0.23
PaS+SHSP (structure-aware fixing)	**98487.72**	**1.52**	**706.99**	**0.13**	**11.97**	**0.28**	**123.41**	**0.11**
Apollo (structure-aware fixing)	98015.32	473.92	707.09	0.23	11.87	0.18	123.40	0.10
Apollo+SHSP (original fixing)	98070.43	418.81	**707.01**	**0.15**	12.08	0.39	**123.37**	**0.07**
Apollo+SHSP (structure-aware fixing)	**98489.24**	**0.00**	**707.01**	**0.15**	**11.71**	**0.02**	**123.37**	**0.07**

Table 12: Detailed ablation study of the proposed hierarchical solution prediction predictor and structure-aware fixing strategy. ↑ indicates that higher values are better, while ↓ indicates that lower values are better. Bold values denote the best results, including ties.

world dataset MIPLIB (Gleixner et al. 2021). However, applying ML-based solvers directly to the entire dataset can be difficult because of the heterogeneous nature of the instances in MIPLIB. Therefore, following Liu et al. (2025), we focus on a subset of MIPLIB that contains similar instances. More information on the selected MILP subset, referred to as IIS, is provided in Appendix D.2. We report the solving performance of the solvers in Table 13 and Table 14, where our methods outperform their corresponding baselines, showcasing the potential for real-world applications.

Method	Obj↓	Gap_{abs} ↓
Gurobi	211.00	20.00
PaS	209.00	18.00
PaS+SHSP	**206.67**	**15.67**
Apollo	214.33	23.33
Apollo+SHSP	**208.67**	**17.67**

Table 13: Results on the IIS dataset, a subset of MIPLIB used by Liu et al. (2025). All learning-based methods use Gurobi as the downstream solver, with a solving time limit of 3,600 seconds.

F.3 Runtime Results

We report network inference time and graph construction time in Table 15. The inference time of the SHSP predictor is only slightly higher than that of the original GNN predictor,

Instance	BKS	Gurobi	PaS	Apollo	PaS+SHSP	Apollo+SHSP
ramos3	186.00	228.00	227.00	242.00	**221.00**	226.00
scpj4scip	128.00	133.00	131.00	132.00	**130.00**	131.00
scpl4	259.00	272.00	269.00	269.00	269.00	269.00

Table 14: The best objectives found on each test instance in IIS. BKS denotes the best known objective values reported by MIPLIB (https://miplib.zib.de/index.html).

which remains negligible compared with the overall solver runtime. In addition, the graph construction time ranges from 0.26s to 15.23s per instance, which is also much shorter than the solver runtime.

Dataset	GNN Predictor	SHSP Predictor	Graph construction time
CA	0.0050s	0.0146s	0.82s
WA	0.0439s	0.2197s	5.48s
IP	0.0043s	0.0131s	0.26s
SC	0.0658s	0.1953s	15.23s

Table 15: Comparison of the average per-instance inference time and graph construction time (in seconds) on the four benchmark datasets.