BIPNN: Learning to Solve Binary Integer Programming via Hypergraph Neural Networks

Sen Bai[*]
baisen@cust.edu.cn

Chunqi Yang[*]
yangchunqi@mails.cust.edu.cn

Xin Bai[†‡]
baixinbs@163.com

Xin Zhang[*]
zhangxin@cust.edu.cn

Zhengang Jiang[*]
jiangzhengang@cust.edu.cn

Abstract

Binary (0-1) integer programming (BIP) is pivotal in scientific domains requiring discrete decision-making. As the advance of AI computing, recent works explore neural network-based solvers for integer linear programming (ILP) problems. Yet, they lack scalability for tackling nonlinear challenges. To handle nonlinearities, state-of-the-art Branch-and-Cut solvers employ linear relaxations, leading to exponential growth in auxiliary variables and severe computation limitations. To overcome these limitations, we propose **BIPNN** (**B**inary **I**nteger **P**rogramming **N**eural **N**etwork), an **unsupervised** learning framework to solve nonlinear BIP problems via hypergraph neural networks (HyperGNN). Specifically, **(I)** BIPNN reformulates BIPs-constrained, discrete, and nonlinear ($\sin$, $\log$, $\exp$) optimization problems-into unconstrained, differentiable, and polynomial loss functions. The reformulation stems from the observation of a precise one-to-one mapping between polynomial BIP objectives and hypergraph structures, enabling the unsupervised training of HyperGNN to optimize BIP problems in an end-to-end manner. On this basis, **(II)** we propose a GPU-accelerated and continuous-annealing-enhanced training pipeline for BIPNN. The pipeline enables BIPNN to optimize large-scale nonlinear terms in BIPs fully in parallel via straightforward gradient descent, thus significantly reducing the training cost while ensuring the generation of discrete, high-quality solutions. Extensive experiments on synthetic and real-world datasets highlight the superiority of our approach.

1 Introduction

For decades, binary integer programming (BIP)—a powerful mathematical tool characterized by discrete binary decision variables (0 or 1)—is of critical importance in numerous domains, such as operational optimization [1, 2, 3], quantum computing [4, 5, 6], computational biology [7, 8], materials science and computational chemistry [9, 10]. However, BIP is generally known to be NP-complete [11], making large-scale BIP instances computationally intractable.

Along with AI computing shines in scientific discovery, the potential of neural network-based IP solvers has emerged in recent years. To address integer linear programming (ILP) problems, MIP-GNN [12] leverages graph neural networks (GNN) to improve the performance. Another GNN&GBDT-guided framework [13] for large-scale ILP problems can save up 99% of running time

[*]School of Computer Science and Technology, Changchun University of Science and Technology.
[†]Huawei Technologies Co. Ltd.
[‡]Corresponding author

39th Conference on Neural Information Processing Systems (NeurIPS 2025).

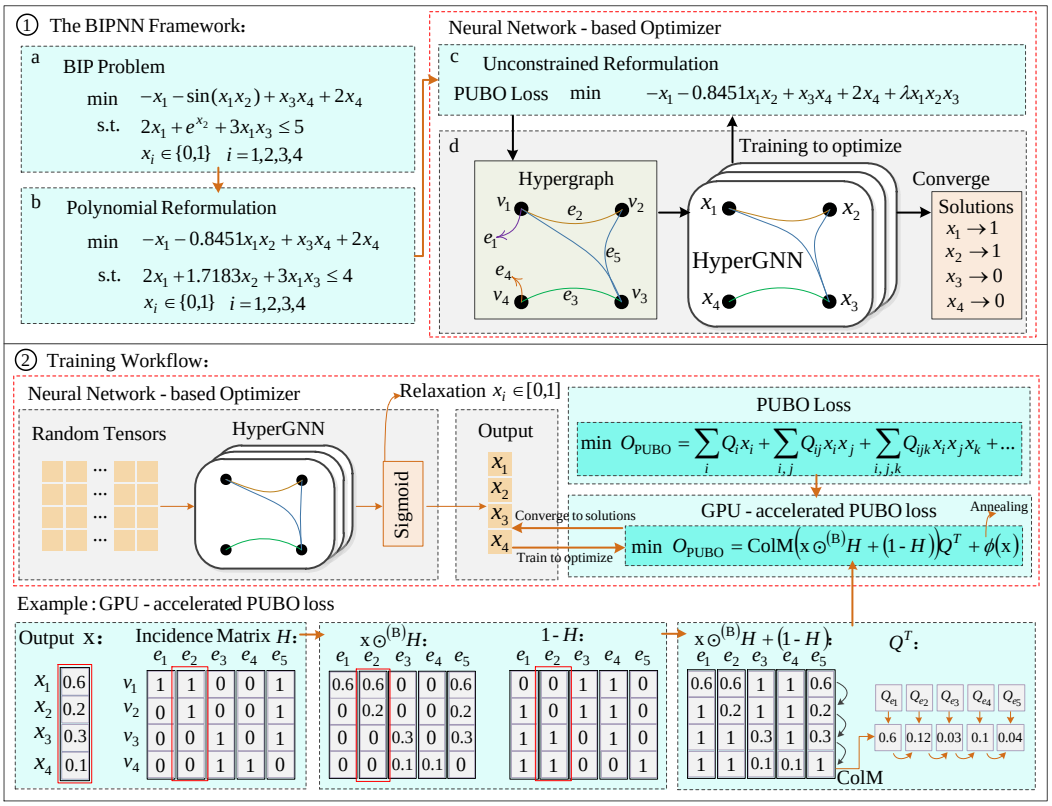


Figure 1: The BIPNN framework. As shown in c and d, BIPNN's hypergraph modeling (Sec. 3.1) stems from the key observation that when variables are treated as nodes, any monomial with coefficient can be represented as a hyperedge, while polynomials can be expressed through hypergraphs – and vice versa.

in achieving the same solution quality as SCIP [14], a leading IP solver. However, these neural network-based ILP solvers lack scalability for nonlinear BIPs (please refer to Appendix A).

To handle nonlinearities, state-of-the-art Branch-and-Cut solvers (e.g., SCIP [15]) rely on linear relaxation, which introduces a number of auxiliary variables. Once linearized, these problems are solved using linear programming (LP) solvers (e.g., the Simplex method[4]). Consequently, large-scale nonlinear BIPs often suffer from prohibitive computational costs. As BIP solvers continue to evolve, linearization remains indispensable for making nonlinearities more tractable for BIP solvers.

These limitations motivate us to develop a streamlined and general-purpose BIP solver to advance the state of the art. To profoundly adapt to real-world applications, our work grapples with challenges arising from neural networks' unique characteristics beyond linearization-based methods, as summarized below:

Challenge 1. Meticulously modeling nonlinear terms in BIP objectives and constraints;

Challenge 2. Utilizing GPU's parallel computing capability.

To this end, in this work we propose **BIPNN** (**B**inary **I**nteger **P**rogramming **N**eural **N**etwork), an unsupervised BIP solver that bridges the gap between nonlinear BIP and deep neural networks. Our overarching idea stems from the observation of one-to-one mapping correspondence between polynomial BIP objectives and hypergraph structures (upper right of Fig. 1). As depicted in Fig. 1, our framework consists of three phases:

[4]To be precise, the Simplex method is designed to solve linear programming (LP) problems in polynomial time, meaning they belong to the class P [16].

1) In the first phase, we employ broadly applicable penalty term method to convert constrained BIP problems into polynomial unconstrained binary optimization (PUBO[5]) formalism. To handle exponential and trigonometric terms, we propose a novel transformation to represent them in the form of polynomials. These refined polynomial objectives are adaptable to neural network-based solvers when applied as loss functions.

2) In the second phase, we leverage hypergraph neural networks (HyperGNN) to address **Challenge 1**, capturing high-order correlations between binary decision variables, or in other words the polynomial terms in the refined PUBO objective. By applying a relaxation strategy to the PUBO objective to generate a differentiable loss function with which we train the HyperGNN in an unsupervised manner.

3) Nevertheless, when we train these HyperGNNs to minimize the PUBO objectives, we encounter severe obstacles of low computational efficiency in these polynomial losses with numerous variables. In the third phase, leveraging GPUs, we further propose an algorithm to address **Challenge 2** via matrix operations on the incidence matrices of hypergraphs.

In summary, we contribute:

1) BIPNN, an unsupervised HyperGNN-based solver that allows learning approximate BIP solutions in an end-to-end differentiable way with strong empirical performance.

2) An empirical study of the performance of BIPNN on synthetic and real-world data, demonstrating that unsupervised neural network solvers outperform classic BIP solvers such as Gurobi and SCIP in tackling large-scale nonlinear BIP problems.

3) Large-scale nonlinear optimization has long been challenging due to its inherent complexity and scalability issues. We advance this field by employing several nonlinearity modeling methods for BIP, including the polynomial reformulation and unconstrained reformulation. These methods provide instructive guidance for unsupervised neural network-based solvers.

2 Notations and Definitions

In the following, we will formulate the BIP problem and articulate the definition of hypergraphs.

Definition 1 (Formulation of BIP). *Nonlinear BIP is an optimization problem where the decision variables* $\mathbf{x} = (x_1, x_2, ..., x_m)$ *are restricted to binary values (*0 *or* 1*), and the objective function* O_{BIP} *or constraints (or both) are nonlinear. Below is the general formulation.*

$$\begin{aligned} \min \quad & O_{\text{BIP}} = f(\mathbf{x}) \\ \text{s. t.} \quad & g_k(\mathbf{x}) \leq 0 \quad \text{for all} \quad k = 1, 2, \ldots, K \\ & q_l(\mathbf{x}) = 0 \quad \text{for all} \quad l = 1, 2, \ldots, L \\ & x_i \in \{0, 1\} \quad \text{for all} \quad i = 1, 2, \ldots, n \end{aligned} \tag{1}$$

where $f(\mathbf{x})$*,* $g_k(\mathbf{x})$ *and* $q_l(\mathbf{x})$ *are nonlinear functions of the decision variables* $\mathbf{x}$*.* □

Definition 2 (Hypergraph). *A hypergraph is defined by* $G = (V, E)$*, where* $V = \{v_1, v_2, ..., v_{|V|}\}$ *stands for a set of vertices and* $E = \{e_1, e_2, ..., e_{|E|}\}$ *denotes a set of hyperedges. Each hyperedge* $e_j \in E$ *is a subset of* V*. A hypergraph* G *can be represented by the incidence matrix (Fig. 1 at the bottom)* $H \in \{0, 1\}^{|V| \times |E|}$*, where* $H_{ij} = 1$ *if* $v_i \in e_j$*, or otherwise* $H_{ij} = 0$*.* □

3 BIPNN: HyperGNN-based Optimizer for PUBO-formulated BIP

For easier comprehension of our approach, in this section we first elaborate how to solve an unconstrained, PUBO-formulated BIP problem as depicted in Eq. 2. Then, in Sec. 4, we will show how to transform a general BIP problem with constraints and nonlinear terms into PUBO formalism.

3.1 Modeling PUBO-formulated BIPs via Hypergraphs

BIPNN employs a HyperGNN-based optimizer (upper right of Fig. 1) to solve PUBO-formulated BIP problems. Inspired by the binary characteristic of variables, we can reformulate general BIPs as

[5]The mathematical formulation PUBO is well-known in quantum computing, for modeling complex optimization problems in a way quantum computers may solve efficiently.

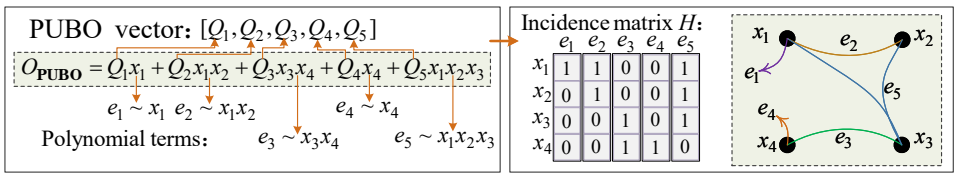


Figure 2: Modeling PUBO-formulated BIPs via hypergraphs.

PUBO problems through the polynomial reformulation in Sec.4.1 and unconstrained reformulation in Sec.4.2. A PUBO problem is to optimize the cost function:

$$O_{\text{PUBO}} = \sum_i Q_i x_i + \sum_{i,j} Q_{ij} x_i x_j + \sum_{i,j,k} Q_{ijk} x_i x_j x_k + \cdots \tag{2}$$

where $x_i \in \{0, 1\}$ are binary descision variables and the set of all decision variables is denoted by $\mathbf{x} = (x_1, x_2, \cdots, x_m)$. As shown in Fig. 2, for ease of representation, a PUBO objective O_{PUBO} with n terms can be decomposed into two components: the PUBO vector $Q = [Q_1, Q_2, ..., Q_n]$, and n linear or polynomial terms such as x_i, x_ix_j, or $x_ix_jx_k$. In this way, we discover multi-variable interactions in O_{PUBO} can be modeled as a hypergraph $G = (V, E)$, where $|E| = n$, and each hyperedge $e \in E$ encodes a single descision variable x_i or a polynomial term such as x_ix_j or $x_ix_jx_k$.

3.2 Neural Network-based Optimizer

The training workflow of the neural network-based optimizer is illustrated at the bottom of Fig. 1.

HyperGNN Architecture. Initially, for a PUBO-transformed hypergraph $G = (V, E)$, HyperGNNs take the incidence matrix H of G and a randomly initialized $X^{(0)} \in \mathbb{R}^{m \times d}$ as inputs. Subsequently, BIPNN applies the sigmoid function to produce the output vector $\mathbf{x} = (x_1, x_2, \cdots, x_m)$, where $x_i \in [0, 1]$ are the relaxation of decision variables $x_i \in \{0, 1\}$. The HyperGNN model operates as follows:

$$\mathbf{x} = \text{sigmoid}(\text{HyperGNN}(H, X^{(0)})) \tag{3}$$

where HyperGNN is a multi-layer hypergraph convolutional network. HyperGNNs are graph-based deep learning architectures that generalize GNNs to operate on hypergraphs. These HyperGNN models includes HGNN+ [17], HyperGCN [18], UniGCN [19] (or UniGAT, UniSAGE, UniGIN). We have tested these models and observed that HGNN+ always outperformed other methods (refer to Appendix F.4). Thus in this work we use HGNN+ to build BIPNN. The l-th convolutional layer of HGNN is built as below:

$$X^{(l)} = \text{ReLU}(D_v^{-\frac{1}{2}} H W D_e^{-1} H^T D_v^{-\frac{1}{2}} X^{(l-1)} \Theta^{(l)}) \tag{4}$$

where $X^{(l)}$ is the embedding matrix generated by the l-th convolutional layer. For a vertex v_i in the hypergraph, the degree is defined as $\deg(v_i) = \sum_{e_j \in E} H_{ij}$. Similarly, the degree of an edge $e_j \in E$ is defined as $\deg(e_j) = \sum_{v_i \in V} H_{ij}$. D_v and D_e are used to denote the diagonal matrices of the vertex degrees and edge degrees, respectively. $W = \text{diag}(|e_1|, \cdots, |e_{|E|}|)$ can be regarded as the weight of hyperedges. The parameter $\Theta^{(l)} \in \mathbb{R}^{d^{(l-1)} \times d^{(l)}}$ is learned during training.

Training to Optimize. As an unsupervised learning model, BIPNN relaxes the PUBO objective O_{PUBO} into a differentiable loss function and trains to optimize it. Specifically, O_{PUBO} can be expressed by the output $\mathbf{x}$ and the incidence matrix H as depicted in Fig. 1. We aim to find the optimal solution $\mathbf{x}^{\text{s}} = \text{argmin} O_{\text{PUBO}}(\mathbf{x}, H)$. As training progresses, $x_i \in \mathbf{x}$ will gradually converge to binary solutions.

GPU-accelerated Training. For a large-scale BIP problem, numerous polynomial terms in O_{PUBO} lead to a high computational cost. To address this, an intuitive idea is to leverage GPU-supported matrix operations to accelerate training. However, PUBO problems lack a straightforward matrix formulation. To this end, we propose GPU-accelerated PUBO objective as follows.

$$O_{\text{PUBO}} = \text{ColM}(\mathbf{x} \odot^{(\text{B})} H + (1 - H))Q^T \tag{5}$$

where $\mathbf{x}$ is the output of HyperGNN, H is the incidence matrix, and $Q = [Q_1, Q_2, ..., Q_n]$ is the PUBO vector. More concretely, $\mathbf{x} \odot^{(\mathrm{B})} H$ denotes the element-wise Hadamard product with broadcasting between m-dimensional vector $\mathbf{x}$ and matrix $H \in \mathbb{R}^{m \times n}$. We add $1 - H$ on $\mathbf{x} \odot^{(\mathrm{B})} H$ to fill zero-valued elements with 1. Based on this operation, we use the column-wise multiplication denoted by ColM on the first dimension of the matrix obtained by $\mathbf{x} \odot^{(\mathrm{B})} H + (1 - H)$. Through the ColM operation we obtain an n-dimensional vector, of which each element represents a polynomial term in O_{PUBO}. The final loss function is computed by scaling each polynomial term with its respective coefficient Q_i. The detailed explanation is illustrated in Fig. 1.

Eq. 5 transforms the computation of large-scale polynomials into efficient matrix operations. It leverages PyTorch-supported broadcasting mechanisms to distribute the vector of decision variables across each monomial (hyperedge), enabling parallel computation.

Time Complexity Analysis. For $\mathbf{x} \in \mathbb{R}^m$, $Q \in \mathbb{R}^{1 \times n}$, and $H \in \mathbb{R}^{m \times n}$, the time complexity of Eq. 5 is $O(m \times n)$. For GPU-accelerated training, element-wise operations such as Hadamard product are fully parallelizable. Column-wise product over m leads to time complexity $O(\log m)$. Thus, the theoretical best GPU time complexity is $O(\log m)$. Utilizing T cores, the realistic GPU time complexity is $O(\frac{m \times n}{T})$.

Annealing Strategy. To achieve unsupervised learning, BIPNN relaxes PUBO problems into continuous space. The differentiable relaxation of discrete decision variables sometimes leads to continuous solutions $x_i \in [0, 1]$. To address this, we employ the continuous relaxation annealing (CRA) [20] method. Specifically, BIPNN uses the following loss function: $O_{\mathrm{PUBO}} = \mathrm{ColM}(\mathbf{x} \odot^{(\mathrm{B})} H + (1 - H))Q^T + \phi(\mathbf{x})$, where $\phi(\mathbf{x}) = \gamma \sum_{i=1}^{n} (1 - (2x_i - 1)^{\alpha})$ is the penalty term, γ controls the penalty strength and α is an even integer. We initialize $\gamma < 0$ and gradually increase it to a positive value as training progresses. The annealing strategy enhances the performance of BIPNN in three aspects, **(i)** In the high-temperature phase ($\gamma < 0$), it smooths the HyperGNN, preventing it from getting trapped in local optima; **(ii)** In the low-temperature phase ($\gamma > 0$), it enforces the discreteness of solutions; **(iii)** It effectively accelerates the training process.

4 BIPNN: Polynomial & Unconstrained Reformulation of BIP

In this section, we explain how to reformulate nonlinear BIPs as unconstrained and polynomial optimization problems, which are compatible with our neural network-based optimizer.

4.1 Polynomial Reformulation of BIP

Our approach is inspired by the observation that for any binary variable, a nonlinear term such as e^x can be exactly fitted by a polynomial equivalent $h(x) = ax + b$, such that $h(x) = e^x$ for $x \in \{0, 1\}$. That is, $h(x) = (e - 1)x + 1$, where $h(0) = 1$ and $h(1) = e$. To handle univariate nonlinearities, including trigonometric, logarithmic, and exponential terms (e.g., $\sin x$, $\log x$, and e^x), we have the following transformation: $h(x) = (h(1) - h(0))x + h(0)$. For multivariate terms such as $e^{x_i x_j}$ and $\sin(x_i x_j)$, where $x_i x_j \in \{0, 1\}$, we can perform the transformation as follows: $h(\prod_{i \in S} x_i) = (h(1) - h(0)) \prod_{i \in S} x_i + h(0)$.

BIPNN employs a more general method to handle more intricate multivariate nonlinear terms (such as $\sin(x_i + x_j)$). For a set of binary decision variables $x_1, x_2, ..., x_n$, a non-linear function $h(x_1, x_2, ..., x_n)$ can be transformed into the polynomial forms as follows.

$$h(x_1, x_2, ..., x_m) = \sum_{S \subseteq \{1,2,...,m\}} c_S \prod_{i \in S} x_i \tag{6}$$

By setting up a system of equations based on all possible combinations of $x_1, x_2, ..., x_m$, we can determine the coefficients c_S to precisely fit $h(x_1, x_2, ..., x_m)$ by leveraging simple inclusion-exclusion principle (refer to Appendix D.1) as below.

$$c_S = \sum_{T \subseteq S} (-1)^{|S| - |T|} f(T) \tag{7}$$

where $f(T)$ represents the function value when the variables in the subset T are 1 and the others are 0. For each subset S, it needs to calculate $2^{|S|}$ values of $f(T)$. □

As an example, we have $\sin(x_1 + x_2) = 0.8415x_1 + 0.8415x_2 - 0.7737x_1x_2$. A toy example of $\sin(x_1 + x_2 + x_3)$ is illustrated in Appendix D.1. To be noticed, polynomial reformulation of all nonlinear terms in a BIP objective is not necessary. If the transformation becomes overly complex, we may opt to retain the original nonlinear term and directly incorporate it as part of the loss function of HyperGNN.

4.2 Unconstrained Reformulation of BIP

We propose a novel penalty method to transform the constrained BIP problem into an unconstrained form. In penalty methods [21, 22], unconstrained reformulation is achieved by adding "penalty terms" to the objective function that penalize violations of constraints. A well-constructed penalty term must be designed such that it equals 0 if and only if the constraint is satisfied, and takes a positive value otherwise. Specifically, given a BIP problem in Eq. 1, for inequality constraints $g_k(\mathbf{x}) \leq 0$, we have penalty terms $P_k(\mathbf{x}) = \lambda_k \cdot (\max(0, g_k(\mathbf{x})))^2$, for equality constraints $q_l(\mathbf{x}) = 0$, we have penalty terms $Q_l(\mathbf{x}) = \mu_l \cdot (q_l(\mathbf{x}))^2$, where λ_k, μ_l are sufficiently large penalty coefficients (For the selection of proper values of λ_k, μ_l, please refer to Appendix C). By combining all terms into a single objective function, we have an unconstrained BIP objective:

$$\min \quad O_{\text{BIP}} = f(\mathbf{x}) + \sum_{k=1}^{K} \lambda_k \cdot (\max(0, g_k(\mathbf{x})))^2 + \sum_{l=1}^{L} \mu_l \cdot (q_l(\mathbf{x}))^2 \tag{8}$$

As part of the loss function of BIPNN, O_{BIP} must be differentiable to enable gradient-based optimization. However, $\max(0, g_k(\mathbf{x}))$ is not a continuously differentiable function, thus finding an appropriate penalty term is crucial. We propose two methods to address this issue:

1) **ReLU-based Penalty**. We can use $\text{ReLU}(g_k(\mathbf{x}))^2 = (\max(0, g_k(\mathbf{x})))^2$ to handle constraints. This is a general method for a large number of variables x_i in a constraint $g_k(\mathbf{x})$.

2) **Polynomial Penalty**. In the following, we present an algorithm to construct polynomial penalty terms with 2^Δ time complexity for $g_k(\mathbf{x})$, where Δ is the number of variables in constraint $g_k(\mathbf{x})$.

For binary variables, do there exist polynomial penalty terms that correspond to BIP constraints? To answer this question, we have the following discussion. For $x_1 + 2x_2 - 2 \leq 0$, we observe that the violating subset $\{x_1 = 1, x_2 = 1\}$ corresponds to polynomial penalty term $\lambda(x_1x_2)$. For another constraint $x_1 + 3x_2 - 2 \leq 0$, the violating subsets $\{x_1 = 0, x_2 = 1\}$ and $\{x_1 = 1, x_2 = 1\}$ correspond to the polynomial penalty term $\lambda(x_2 + x_1x_2)$ or λx_2. Through an in-depth analysis, we propose a novel method to transform nonlinear BIP constraints into polynomial penalty terms. To handle an inequality constraint $g(\mathbf{x}) \leq 0$ for the BIP problem in Eq. 1, our method consists of three steps (to see a toy example, refer to Appendix D.2):

(i) Initially, we express the constraint $g(\mathbf{x}) \leq 0$ as a boolean indicator function: $\psi(\mathbf{x}) = \begin{cases} 1 & \text{if } g(\mathbf{x}) > 0 \text{ (violation)} \\ 0 & \text{otherwise (feasible)} \end{cases}$, then define minimal violation subsets $\mathcal{V}$ as the smallest variable combinations causing constraint violations:

$$\mathcal{V} = \left\{ S \subseteq \{1, ..., n\} \,\middle|\, \psi(\mathbf{x}) = 1 \text{ when } x_i = 1 \; \forall i \in S \text{ and } x_j = 0 \; \forall j \notin S \right\} \tag{9}$$

each $S \in \mathcal{V}$ cannot be reduced further without eliminating the violation.

(ii) Generate a penalty term for each minimal violation subset $S \in \mathcal{V}$:

$$P(\mathbf{x}) = \lambda \sum_{S \in \mathcal{V}} \prod_{i \in S} x_i \tag{10}$$

where λ is the penalty coefficient.

(iii) Combine each term into the BIP objective function:

$$\min \quad O_{\text{BIP}} = f(\mathbf{x}) + P(\mathbf{x}) \tag{11}$$

In the worst case, when an enumeration method is used in step **(i)**, it requires calculating 2^Δ subsets, where Δ is the number of variables in constraint $g(\mathbf{x})$. Nevertheless, in most real-world problems

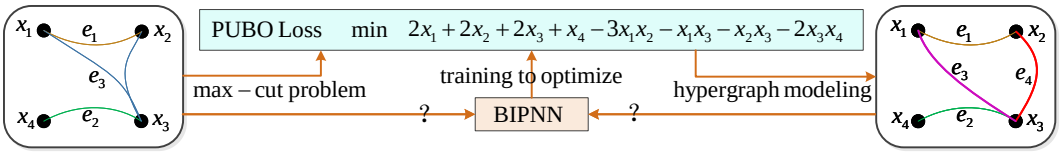


Figure 3: To solve the hypergraph max-cut problem, BIPNN generates a new hypergraph structure. However, both of these hypergraphs can be utilized for training the HyperGNN model.

(e.g. max-cut, and maximal independent set or MIS) involving graphs, the variables associated with each constraint often exhibit locality. □

The polynomial penalty method facilitates to incorporate penalty terms to PUBO objectives and use GPU-accelerated training pipeline to solve BIPs. As far as we know, only a few number of constraint/penalty pairs [22] associated have been identified in existing literature. Our work significantly expands the potential application domains of the penalty method.

5 Discussion

Feasible Solutions. Firstly, a PUBO problem always has feasible solutions. The feasible set is the entire space of binary variable combinations, since there are no constraints to exclude any combination. Every possible binary assignment $x_i \in \{0, 1\}$ is inherently feasible. Secondly, the feasibility of a nonlinear BIP problem depends on the constraint compatibility—whether there exists at least one binary variable assignment $\mathbf{x} \in \{0, 1\}^m$ that satisfies all nonlinear constraints simultaneously. In BIPNN, we determine the existence of feasible solutions through **(i)** Training-phase feasibility check: if all penalty terms (e.g., constraint violations) converge to zero during training, feasible solutions exist; otherwise, the problem is infeasible. **(ii)** Post-training verification: we sample candidate solutions from the trained model and explicitly verify whether they satisfy all constraints.

The Effectiveness of BIPNN's Hypergraph Generation Mechanism. As depicted in Fig. 3, when BIPNN is applied to solve combinatorial optimization (CO) problems on hypergraphs, it generates an alternative hypergraph structure. However, both of the hypergraphs can be used as the input of BIPNN. A critical question arises: which type of hypergraph structure achieves better performance when applied to HyperGNN? The main difference between these two hypergraphs is that the hypergraph generated by BIPNN breaks down the original hypergraph's high-order hyperedges into numerous low-order ones. We argue that BIPNN training with the original hypergraph structure is more computationally efficiency, while BIPNN-generated hypergraph structure leads to more optimal solutions. In Sec. 6.3, we will empirically compare the solution quality of both methods.

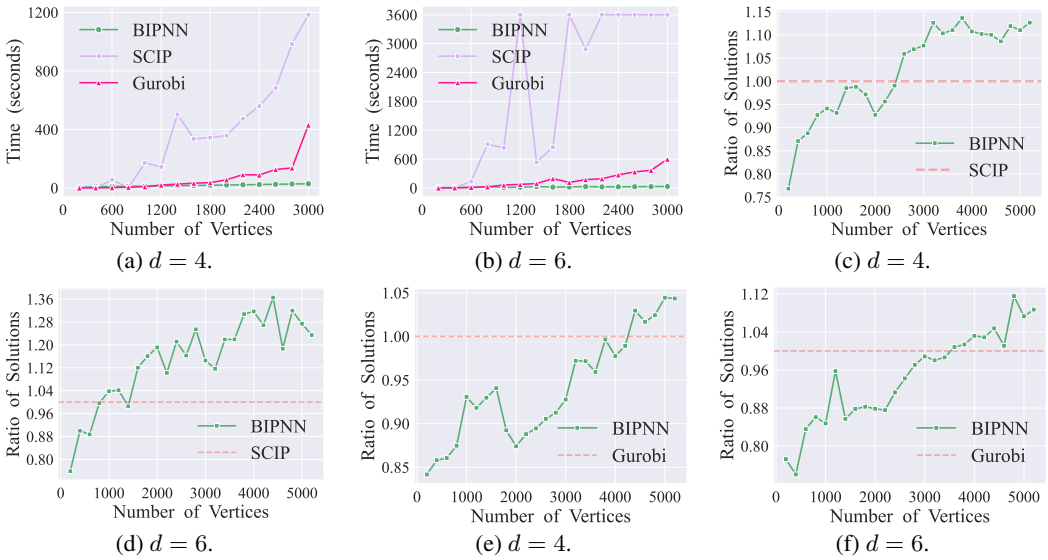
6 Experimental Results

In this section, we describe our empirical experiments on BIPNN and baseline optimization tools. Our source codes can be obtained on GitHub[6].

Benchmarks. To evaluate BIPNN on BIP problems with diverse scales, the datasets are generated using DHG library[7]. To evaluate the quality of solutions and computational efficiency of BIPNN, datasets of varying scales are generated in three steps: Initially, DHG library is applied to generate hypergraph structures (where $|E| = 2|V|$). Subsequently, a random coefficient is assigned to each hyperedge (representing a polynomial term) to generate PUBO objective functions. Thereafter, several constraints (penalty terms) were randomly incorporated into the PUBO objectives. To demonstrate the effectiveness of BIPNN on real-world settings, we also conduct experiments on the hypergraph max-cut problem (refer to Appendix E), a well-known BIP problem benchmark. Moreover, we conduct experiments on publicly-available hypergraph datasets (refer to Appendix F.1).

[6]https://github.com/Classpi/BIPNN-Learning-to-Solve-Binary-Integer-Programming-via-Hypergraph-Neural-Networks

[7]https://deephypergraph.readthedocs.io/en/latest/index.html



(a) $d = 4$. (b) $d = 6$. (c) $d = 4$.

(d) $d = 6$. (e) $d = 4$. (f) $d = 6$.

Figure 4: Comparison of BIPNN and BIP solvers such as Gurobi and SCIP. d is the degree of polynomial terms in BIP objective functions. (a)(b) show the solving time required for BIPNN, SCIP and Gurobi to obtain the same solution. (c)(d) show the ratio of the solutions of BIPNN to SCIP; (e)(f) illustrate the ratio of the solutions of BIPNN to Gurobi; Runtime is restricted to half an hour.

Baseline Methods. In our experiments, the baseline methods include optimization techniques and tools such as Gurobi[8], SCIP [14] (refer to Appendix A), Tabu search [23].

Implementation Details. Experiments are conducted on an Intel Core i9-12900K CPU with 24 cores, and an NVIDIA GeForce RTX 3090 GPU with 24 G of memory. We adopt two-layer HGNN+ [17] as the HyperGNN model for the experiments.

6.1 Comparison with Linearization-based BIP Solvers

Gurobi and SCIP are exact solvers based on the branch-and-cut algorithm. Theoretically, given sufficient time and computational resources, they guarantee exact solutions. However, for large-scale problems, due to time constraints, Gurobi and SCIP may terminate prematurely and return approximate solutions. To conduct the experiment, we generate a specific BIP instance for each size of variables. Specifically, for a BIPNN-generated hypergraph, the number of vertices (variables) $|V|$ ranges from 200 to 3000. The degrees of vertices are set to 4 (Fig. 4a) and 6 (Fig. 4b) respectively.

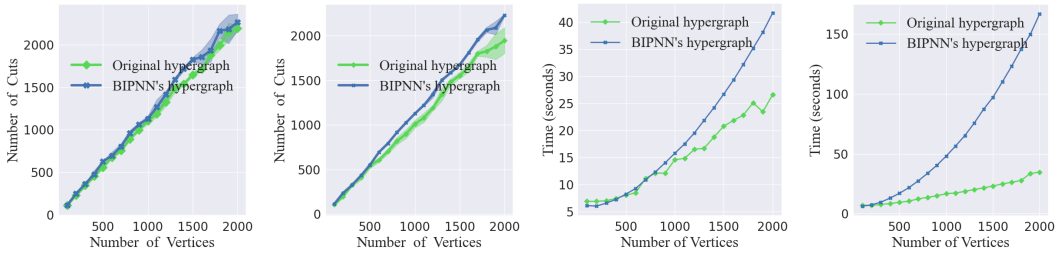
Fig. 4a and Fig. 4b show the comparison of the solving time for BIPNN, SCIP and Gurobi. We evaluate the solving time taken by BIPNN to obtain the best approximate solution and the time required by SCIP and Gurobi to find the same solution. Experimental results demonstrate that the solving time of BIPNN grows linearly and slowly with increasing problem size, while SCIP and Gurobi's solving time exhibits exponential growth. This trend becomes more pronounced when the degree of polynomial terms is 6.

Moreover, we impose half an hour time limit and evaluate the solution quality of BIPNN, SCIP and Gurobi across varying scales of BIP instances. Fig. 4c, 4d, 4e, 4f show the comparative ratio of solutions obtained by BIPNN, SCIP and Gurobi. The comparative ratio is defined as $\frac{O^s_{\text{BIPNN}}}{O^s_{\text{Linear}}}$, where O^s_{BIPNN} is the solution obtained by BIPNN, and O^s_{Linear} is the solution obtained by SCIP and Gurobi. Experimental results demonstrate that BIPNN starts outperforming SCIP when the number of variables exceeds $2,500$ when $d = 4$. As the problem size increases, BIPNN's solutions increasingly outperform SCIP's solutions. For $d = 6$, BIPNN outperforms SCIP when the number of vertices exceeds $1,000$. The results on Gurobi show a pattern similar to SCIP. The number of variables required for BIPNN to outperform Gurobi is $4,300$ when $d = 4$. For $d = 6$, BIPNN outperforms Gurobi when the number of vertices exceeds $3,500$.

[8]https://www.gurobi.com/

Table 1: The solutions of graph/hypergraph max-cut problems.

Method	BAT	EAT	UAT	DBLP	CiteSeer	AmzPhoto	Primary	High	Cora	PubMed
Gurobi	655	3,997	8,431	2,869	3,960	72,461	8,448	5,195	1,400	7,226
SCIP	655	3,849	7,899	2,869	3,960	59,258	7,603	4,599	1,215	7,185
Tabu	652	3,970	8,402	2,710	3,717	71,970	8,500	5,160	1,360	6,868
BIPNN	653	**3,997**	**8,463**	2,847	3,944	**83,446**	**8,516**	**5,252**	1,397	7,188



(a) $d = 4$. (b) $d = 6$. (c) $d = 4$. (d) $d = 6$.

Figure 5: Comparison of the quality of solutions and time efficiency of BIPNN when it applys its generated hypergraph structure or the original hypergraph structure to solve hypergraph max-cut problems. d is the degree of polynomial terms in BIP objective functions. (a)(b) show the numbers of cuts; (c)(d) show the solving time.

Additional experimental results on large-scale BIP problems can be found in Appendix F.2. The performance of BIPNN compared with Tabu search is illustrated in Appendix F.3.

6.2 Comparison on Real-world Datasets

We compare BIPNN against baseline methods on real-world graph and hypergraph datasets, including BAT, EAT, UAT, DBLP, CiteSeer, AmzPhoto, Primary, High, Cora and PubMed (refer to Appendix F.1). Graph datasets include BAT, EAT, UAT, DBLP, CiteSeer, and AmzPhoto. Hypergraph datasets include Primary, High, Cora, and PubMed. Graph and hypergraph max-cut problems are selected as the BIP problem benchmarks. We impose 1 hour time limit and evaluate the number of cuts obtained by Gurobi, SCIP, Tabu, and BIPNN.

As depicted in Tab. 1, Gurobi outperforms SCIP and Tabu on all graph and hypergraph datasets. Moreover, experimental results show that the performance of Gurobi and BIPNN varies across different datasets. Gurobi achieved the best performance on four graph datasets and two hypergraph datasets, while BIPNN achieved the best performance on three graph datasets and two hypergraph datasets.

6.3 Comparative Analysis on Hypergraph Generation Mechanism

In Sec. 5 and Fig. 3, we propose to evaluate the effectiveness of BIPNN's hypergraph generation mechanism by comparing the effects of its generated hypergraph structures against the original hypergraph structures in a hypergraph CO problem. In this section, we select hypergraph max-cut as benchmark and conduct experiments to evaluate the performance of BIPNN under both of the hypergraph structures. Experimental results are depicted in Fig. 5. The number of variables ranges from 100 to 2000. The degrees of polynomial terms d are set to $d = 4$ and $d = 6$ respectively. We perform 10 tests each time and record the average value of the cut numbers.

As illustrated in Fig. 5a and Fig. 5b, the hypergraph structure generated by BIPNN can identify more cuts in comparison. However, as depicted in Fig. 5c and Fig. 5d, when the parameter d is larger, the number of hyperedges (polynomial terms in PUBO objectives) in the hypergraph structure generated by BIPNN increases sharply, leading to significantly higher computational costs. The results align with the theoretical analysis we presented in Sec. 5. Unlike conventional machine learning tasks (e.g., node classification), we hypothesize that the hypergraph structure for BIPNN should be related to the loss function. Thus HyperGNN's energy flow directs each polynomial term along proper gradient paths.

6.4 Ablation Study

GPU Acceleration. The superior time efficiency of BIPNN is primarily attributed to the GPU-accelerated algorithm employed in computing large-scale PUBO loss functions. Fig. 6 shows a comparison of the training times for BIPNN with or without the GPU-accelerated algorithm. We evaluate the training time of BIPNN on the hypergraph max-cut problem. The number of variables ranges from 200 to 1000. The degree of polynomial terms is set to 4.

We train BIPNN for a fixed number of 1000 epochs. As Fig. 6 illustrates, when GPU acceleration is applied to compute the PUBO loss function, the training time does not exhibit significant growth with an increasing number of variables. In contrast, without GPU acceleration, the training time increases rapidly as the number of variables rises.

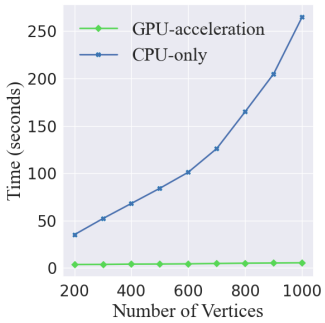


Figure 6: Comparison of the training time for BIPNN with or without GPU accelerated algorithm for PUBO losses.

Annealing Strategy. We validate the effectiveness of the annealing strategy of BIPNN on the hypergraph max-cut problem. The experiments are conducted on Cora with $1,330$ vertices. The metrics include the number of cuts and discreteness of variables. The penalty strength γ is set to -2.5 initially and its value is gradually increased during training. The value of γ reaches 0 after 500 epochs and continued to increase thereafter.

As illustrated in Fig. 7, the annealing strategy ensures BIPNN to get better solutions while guaranteeing all variables to converge to discrete values. It demonstrates that negative γ values enable BIPNN to escape local optima, thereby discovering better solutions. Moreover, when γ is set to positive values, it facilitates the convergence of variables toward discrete values.

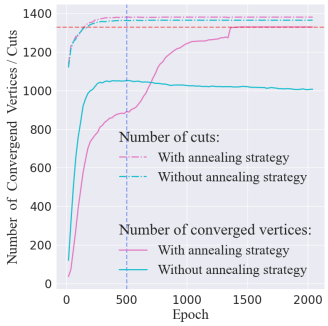


Figure 7: Quality and discreteness of solutions with or without the annealing strategy.

7 Conclusion

This work proposes BIPNN, a novel neural network solver for nonlinear BIP problems. It reformulates nonlinear BIPs into PUBO cost functions, which correspond to hypergraph structures. On this basis, these PUBO cost functions are used as loss functions for HyperGNNs, enabling the model to solve BIPs in an unsupervised training manner. Compared with existing BIP solvers (e.g., Gurobi, SCIP) that rely on linearization, BIPNN reduces the training cost by optimizing nonlinear BIPs via straightforward gradient descent. Empirical results demonstrate that BIPNN achieves state-of-the-art performance in learning approximate solutions for large-scale BIP problems.

Acknowledgments and Disclosure of Funding

This work is sponsored by the Natural Science Foundation of Jilin Province under grants 20240101349JC, National Natural Science Foundation of China under grants 82472118.

References

[1] Yan Qiao, Yanjun Lu, Jie Li, Siwei Zhang, Naiqi Wu, and Bin Liu. An efficient binary integer programming model for residency time-constrained cluster tools with chamber cleaning requirements. *IEEE Transactions on Automation Science and Engineering*, 19(3):1757–1771, 2021.

[2] Theodore P Papalexopoulos, Christian Tjandraatmadja, Ross Anderson, Juan Pablo Vielma, and David Belanger. Constrained discrete black-box optimization using mixed-integer programming. In *International Conference on Machine Learning*, pages 17295–17322. PMLR, 2022.

[3] Libin Wang, Han Hu, Qisen Shang, Haowei Zeng, and Qing Zhu. Structuredmesh: 3-d structured optimization of façade components on photogrammetric mesh models using binary integer programming. *IEEE Transactions on Geoscience and Remote Sensing*, 62:1–12, 2024.

[4] Giacomo Nannicini, Lev S Bishop, Oktay Günlük, and Petar Jurcevic. Optimal qubit assignment and routing via integer programming. *ACM Transactions on Quantum Computing*, 4(1):1–31, 2022.

[5] Akshay Ajagekar, Kumail Al Hamoud, and Fengqi You. Hybrid classical-quantum optimization techniques for solving mixed-integer programming problems in production scheduling. *IEEE Transactions on Quantum Engineering*, 3:1–16, 2022.

[6] Lei Fan and Zhu Han. Hybrid quantum-classical computing for future network optimization. *IEEE Network*, 36(5):72–76, 2022.

[7] Mercè Llabrés, Gabriel Riera, Francesc Rosselló, and Gabriel Valiente. Alignment of biological networks by integer linear programming: virus-host protein-protein interaction networks. *BMC bioinformatics*, 21(Suppl 6):434, 2020.

[8] Jianshen Zhu, Naveed Ahmed Azam, Fan Zhang, Aleksandar Shurbevski, Kazuya Haraguchi, Liang Zhao, Hiroshi Nagamochi, and Tatsuya Akutsu. A novel method for inferring chemical compounds with prescribed topological substructures based on integer programming. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 19(6):3233–3245, 2021.

[9] Vladimir V Gusev, Duncan Adamson, Argyrios Deligkas, Dmytro Antypov, Christopher M Collins, Piotr Krysta, Igor Potapov, George R Darling, Matthew S Dyer, Paul Spirakis, et al. Optimality guarantees for crystal structure prediction. *Nature*, 619(7968):68–72, 2023.

[10] Georgia Stinchfield, Joshua C Morgan, Sakshi Naik, Lorenz T Biegler, John C Eslick, Clas Jacobson, David C Miller, John D Siirola, Miguel Zamarripa, Chen Zhang, et al. A mixed integer linear programming approach for the design of chemical process families. *Computers & Chemical Engineering*, 183:108620, 2024.

[11] Richard M Karp. *Reducibility among combinatorial problems*. Springer, 2010.

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

[13] Huigen Ye, Hua Xu, Hongyan Wang, Chengming Wang, and Yu Jiang. Gnn&gbdt-guided fast optimizing framework for large-scale integer programming. In *International conference on machine learning*, pages 39864–39878. PMLR, 2023.

[14] Stephen Maher, Matthias Miltenberger, João Pedro Pedroso, Daniel Rehfeldt, Robert Schwarz, and Felipe Serrano. PySCIPOpt: Mathematical programming in python with the SCIP optimization suite. In *Mathematical Software – ICMS 2016*, pages 301–307. Springer International Publishing, 2016.

[15] Tobias Achterberg. Scip: solving constraint integer programs. *Mathematical Programming Computation*, 1:1–41, 2009.

[16] Narendra Karmarkar. A new polynomial-time algorithm for linear programming. In *Proceedings of the sixteenth annual ACM symposium on Theory of computing*, pages 302–311, 1984.

[17] Yue Gao, Yifan Feng, Shuyi Ji, and Rongrong Ji. Hgnn+: General hypergraph neural networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(3):3181–3199, 2022.

[18] Naganand Yadati, Madhav Nimishakavi, Prateek Yadav, Vikram Nitin, Anand Louis, and Partha Talukdar. Hypergcn: A new method for training graph convolutional networks on hypergraphs. *Advances in neural information processing systems*, 32, 2019.

[19] Jing Huang and Jie Yang. Unignn: a unified framework for graph and hypergraph neural networks. In *the Thirtieth International Joint Conference on Artificial Intelligence (IJCAI)*, 2021.

[20] Yuma Ichikawa. Controlling continuous relaxation for combinatorial optimization. *Advances in Neural Information Processing Systems (NeurIPS)*, 37:47189–47216, 2024.

[21] Jorge Nocedal and Stephen J Wright. *Numerical optimization*. Springer, 1999.

[22] Fred Glover, Gary Kochenberger, Rick Hennig, and Yu Du. Quantum bridge analytics i: a tutorial on formulating and using qubo models. *Annals of Operations Research*, 314(1):141–183, 2022.

[23] Fred Glover and Manuel Laguna. *Tabu search*. Springer, 1998.

[24] 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.

[25] Nasimeh Heydaribeni, Xinrui Zhan, Ruisi Zhang, Tina Eliassi-Rad, and Farinaz Koushanfar. Distributed constrained combinatorial optimization leveraging hypergraph neural networks. *Nature Machine Intelligence*, pages 1–9, 2024.

[26] Zhixiao Xiong, Huigen Ye, Fangyu Zong, Yutong Zhang, Hua Xu, and Hongyan Wang. Neuralqp: A general hypergraph-based optimization framework for large-scale quadratically constrained quadratic programs.

A Related works (Sec. 1)

Gurobi & SCIP. Gurobi is a commercial high-performance mathematical optimization solver. It is widely used in industry and academia for solving large-scale linear programming (LP), mixed-integer linear programming (MILP), quadratic programming (QP), and mixed-integer quadratic programming (MIQP) problems. Gurobi is known for its speed. Moreover, starting with version 12.0, Gurobi supports nonlinear constraints. SCIP is a powerful open-source optimization solver. It is primarily designed for solving MILP and nonlinear programs (MINLP) and is highly customizable. SCIP is widely used in academic research due to its flexibility.

Table 2: Categorizing several related neural network BIP solvers.

Method	Supervised	Unsupervised	Linear	Quadratic	Polynomial	Hypergraph Modeling
BIPNN		✓			✓	objective & constraints
DiffLO [24]		✓	✓			
HypOp [25]		✓			✓	constraints only
NeuralQP [26]	✓			✓		constraints only
MIP-GNN [12]	✓		✓			

In Tab. 2, we present recent neural network-based solvers for BIPs.

DiffILO [24]. DiffILO is an unsupervised method with performance that can exceed Gurobi and SCIP. Its methodology is the most similar to that of BIPNN. However, DiffILO is designed for linear BIPs, while BIPNN primarily addresses general nonlinear BIPs. DiffILO employs bipartite graphs and GNNs to model constraints, whereas BIPNN utilizes HyperGNNs to represent nonlinear BIPs.

HypOp [25]. HypOp is also an unsupervised method. Compared with HypOp, BIPNN first discovers the mapping correspondence betweeen polynomials and hypergraph structures. Compared to linear and quadratic problems, the core challenge in solving nonlinear BIPs lies in leveraging GPU to accelerate training. BIPNN proposes a computational framework specifically for this purpose. In contrast, HypOp utilizes distributed HyperGNN training, yet its hypergraph modeling solely targets constraints rather than objective functions, thus rendering HypOp inapplicable to general BIPs.

MIP-GNN [12] & NeuralQP [26]. They are supervised and designed for LP and QP problems. Supervised learning methods require precomputing vast amounts of labels [24]. Obtaining labels for nonlinear BIPs is challenging, since Gurobi and SCIP have higher computational overhead when solving nonlinear BIPs. Although MIP-GNN and NeuralQP demonstrate strong performance, they are specifically designed for LP and QP problems. In contrast, BIPNN is designed for solving more complex nonlinear BIP problems, making it inherently difficult to match the performance of specialized LP and QP solvers.

Overall, BIPNN is proposed for general nonlinear BIPs, and it is fast. We believe BIPNN and DiffILO collectively address numerous limitations inherent in penalty-based deep learning methods for solving nonlinear BIP problems. In the future, they could potentially integrate commercial solvers (e.g., Gurobi, SCIP) to enhance large-scale BIP optimization. BIPNN's polynomial reformulation, unconstrained reformulation, and GPU-accelerated PUBO solver, may also potentially enhance other neural network-based BIP solvers.

B Discussion: supervised methods are not applicable to nonlinear BIPs (Sec. 1)

Table 3: SCIP's runtime when solving BIPs with polynomial objectives of degree 4.

Number of monomials	2400	3600	4800	6000
Linear	0.0037 s	0.0047 s	0.0060 s	0.0073 s
Nonlinear	173 s	350 s	553 s	1215 s

Most existing supervised methods are designed for ILP and QP problems. They require precomputing labels via solvers like Gurobi and SCIP. However, Gurobi and SCIP exhibit exponentially slower

performance when solving general nonlinear BIPs compared to ILPs, making label acquisition practically infeasible. To verify this, we measured SCIP's runtime for BIPs with polynomial objectives of degree 4 (Tab. 3).

Moreover, recent work DiffILO [24] for linear BIPs reveals that supervised methods frequently produce infeasible solutions, due to the misalignment between training objectives (minimizing prediction error) and inference objectives (generating high-quality feasible solutions).

C The selection of proper values of penalty factors (Sec. 4.2)

In common Maximal Independent Set (MIS) problems, the penalty factor is typically fixed at 2. However, for general BIP problems, the choice of penalty factors critically impacts both constraint satisfaction and solution quality. This problem has been well-solved by DiffILO [24], an unsupervised linear BIP solver. Specifically, we can use

$$\mu_l = \mu_0 + \eta \cdot q_l \tag{12}$$

where μ_0 is the initial value of the penalty factor μ_l, η is the learning rate, and q_l is the value of the penalty term. We can initialize μ_0 with a small value (e.g., $2 \sim 5$) to broaden the model's search space. The learning rate η should be experimentally tuned across different values (e.g., $0.001 \sim 1$).

D Examples of polynomial reformulation and unconstrained reformulation

D.1 A toy example of the polynomial reformulation of BIP (Sec. 4)

For $\sin(x_1 + x_2 + x_3)$, where $x_1, x_2, x_3 \in \{0, 1\}$, we can construct a polynomial to precisely fit the function, such that it matches $\sin(x_1 + x_2 + x_3)$ for all combinations of $x_1, x_2, x_3 \in \{0, 1\}$. For multiple binary variables, the polynomial can be generalized as:

$$P(x_1, x_2, x_3) = a_1x_1 + a_2x_2 + a_3x_3 + b_{12}x_1x_2 + b_{13}x_1x_3 + b_{23}x_2x_3 + cx_1x_2x_3 + d \tag{13}$$

Based on all possible combinations of x_1, x_2, x_3, we can set up the following equations:

1) When $x_1 = 0, x_2 = 0, x_3 = 0$: $P(0, 0, 0) = d = \sin(0) = 0$. Thus, $d = 0$.

2) When $x_1 = 0, x_2 = 0, x_3 = 1$: $P(0, 0, 1) = a_3 = \sin(1) \approx 0.8415$. Thus, $a_3 = 0.8415$.

3) When $x_1 = 0, x_2 = 1, x_3 = 0$: $P(0, 1, 0) = a_2 = \sin(1) \approx 0.8415$. Thus, $a_2 = 0.8415$.

4) When $x_1 = 1, x_2 = 0, x_3 = 0$: $P(1, 0, 0) = a_1 = \sin(1) \approx 0.8415$. Thus, $a_1 = 0.8415$.

5) When $x_1 = 0, x_2 = 1, x_3 = 1$: $P(0, 1, 1) = a_2 + a_3 + b_{23} = \sin(2) \approx 0.9093$.

Substituting $a_2 = 0.8415$ and $a_3 = 0.8415$: $b_{23} = -0.7737$.

6) When $x_1 = 1, x_2 = 0, x_3 = 1$: $P(1, 0, 1) = a_1 + a_3 + b_{13} = \sin(2) \approx 0.9093$

Substituting $a_1 = 0.8415$ and $a_3 = 0.8415$: $b_{13} = -0.7737$.

7) When $x_1 = 1, x_2 = 1, x_3 = 0$: $P(1, 1, 0) = a_1 + a_2 + b_{12} = \sin(2) \approx 0.9093$

Substituting $a_1 = 0.8415$ and $a_2 = 0.8415$: $b_{12} = -0.7737$

8) When $x_1 = 1, x_2 = 1, x_3 = 1$: $P(1, 1, 1) = a_1 + a_2 + a_3 + b_{12} + b_{13} + b_{23} + c = \sin(3) \approx 0.1411$.

Substituting known values: $c = -0.0623$.

Based on the above calculations, the polynomial is:

$$P(x_1, x_2, x_3) = 0.8415(x_1 + x_2 + x_3) - 0.7737(x_1x_2 + x_1x_3 + x_2x_3) - 0.0623x_1x_2x_3 \tag{14}$$

D.2 A toy example of the unconstrained reformulation of BIP (Sec. 4)

For a nonlinear constraint with exponential term $g(\mathbf{x})$: $2x_1 + e^{x_2} + 3x_1x_3 \leq 5$, where $x_1, x_2, x_3 \in \{0, 1\}$, we can find the minimal violation subsets $\mathcal{V}$ based on all possible combinations of x_1, x_2, x_3.

1) When $x_1 = 0, x_2 = 0, x_3 = 0$: $g(\mathbf{x}) = 1 \leq 5$, feasible.

2) When $x_1 = 0, x_2 = 0, x_3 = 1$: $g(\mathbf{x}) = 1 \leq 5$, feasible.

3) When $x_1 = 0, x_2 = 1, x_3 = 0$: $g(\mathbf{x}) = e \leq 5$, feasible.

4) When $x_1 = 1, x_2 = 0, x_3 = 0$: $g(\mathbf{x}) = 3 \leq 5$, feasible.

5) When $x_1 = 0, x_2 = 1, x_3 = 1$: $g(\mathbf{x}) = e \leq 5$, feasible.

6) When $x_1 = 1, x_2 = 0, x_3 = 1$: $g(\mathbf{x}) = 6 \geq 5$, violation.

7) When $x_1 = 1, x_2 = 1, x_3 = 0$: $g(\mathbf{x}) = e + 2 \leq 5$, feasible.

8) When $x_1 = 1, x_2 = 1, x_3 = 1$: $g(\mathbf{x}) = 5 + e \geq 5$, violation (not minimal).

Identified minimal violation subsets: $\{x_1, x_3\}$. Thus,

$$P(\mathbf{x}) = \lambda(x_1 x_3) \tag{15}$$

Final BIP objective:

$$O_{\text{BIP}} = f(\mathbf{x}) + \lambda(x_1 x_3) \tag{16}$$

E The hypergraph max-cut problem (Sec. 6)

The max-cut problem of a hypergraph $G = (V, E)$ involves partitioning the vertex set into two disjoint subsets such that the number of hyperedges crossing the partitioned blocks is maximized.

PUBO Form. The hypergraph max-cut problem on G can be formulated by optimizing a PUBO objective as follows:

$$\min \quad O_{\max-\text{cut}} = \sum_{e \in E} (1 - \prod_{i \in e} x_i - \prod_{i \in e} (1 - x_i)) \tag{17}$$

where $x_i \in \{0, 1\}$ are binary decision variables.

For a simple example illustrated in Fig. 3, the original hypergraph consists of three hyperedges: $\{x_1, x_2\}$, $\{x_3, x_4\}$, and $\{x_1, x_2, x_3\}$. Thus, the max-cut objective of G is to minimize $2x_1 + 2x_2 + 2x_3 + x_4 - 3x_1x_2 - x_1x_3 - x_2x_3 - 2x_3x_4$. BIPNN typically generates a new hypergraph structure with five hyperedges, $\{x_1, x_2\}$, $\{x_3, x_4\}$, $\{x_1, x_3\}$, and $\{x_2, x_3\}$, to solve this PUBO objective. we found that both hypergraphs can be utilized for HyperGNN training in BIPNN framework.

F Datasets and additional results

F.1 Datasets

Table 4: Summary statistics of six real-world graphs: the number of vertices $|V|$, the number of edges $|E|$. Four hypergraphs: the number of vertices $|V|$, the number of hyperedges $|E|$, the size of the hypergraph $\sum_{e \in E} |e|$.

Graphs	$\lvert V\rvert$	$\lvert E\rvert$	**Hypergraphs**	$\lvert V\rvert$	$\lvert E\rvert$	$\sum_{e \in E} \lvert e\rvert$
BAT	131	1,003	Primary	242	12,704	30,729
EAT	399	5,993	High	327	7,818	18,192
UAT	1,190	13,599	Cora	1,330	1,503	4,599
DBLP	2,591	3,528	PubMed	3,824	7,523	33,687
CiteSeer	3,279	4,552				
AmzPhoto	7,535	119,081				

F.2 Comparison with Gurobi and SCIP

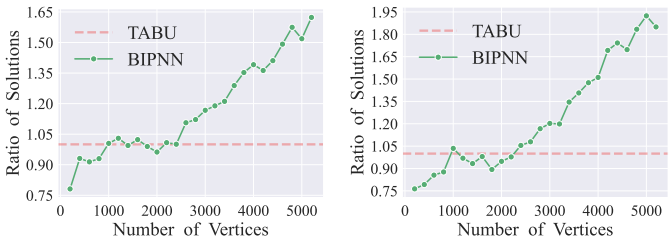
On larger-scale BIPs, the advantage of BIPNN becomes more pronounced. For SCIP and Gurobi, it is often difficult to find an initial solution within reasonable time on large-scale BIPs. To illustrate this issue, we have supplemented the following experimental results.

Table 5: Comparison of BIPNN and BIP solvers such as Gurobi and SCIP on large-scale datasets.

Number of variables (monomials)	SCIP	Gurobi	BIPNN
10,000 (20,000)	-2,066	-1,870	-2,566
10,000 (30,000)	-105	-1,988	-3,317
10,000 (50,000)	-117	None	-4,323
20,000 (40,000)	None	None	-5,039

Tab. 5 shows optimization results of different methods on the objective function, with computation time constrained to half an hour. The degree of each monomial is 6. BIPNN outperforms SCIP and Gurobi.

F.3 Comparison with Tabu Search



(a) $d = 4$. (b) $d = 6$.

Figure 8: Comparison of BIPNN and Tabu search. d is the degree of polynomial terms in BIP objective functions. (a)(b) illustrate the ratio of the solutions of BIPNN to Tabu; Runtime is restricted to half an hour.

Tabu search is a heuristic method that typically provides approximate solutions. We impose half an hour time limit and evaluate the difference in solution quality for Tabu when the degrees of polynomial terms are set to 4 and 6. The number of vertices (variables) $|V|$ in the hypergraph generated by BIPNN ranges from 200 to $5,000$. Experimental results are depicted in Fig. 8a ($d = 4$) and Fig. 8b ($d = 6$). As shown in the figures, BIPNN achieves the performance comparable to Tabu when the number of variables exceeds $1,000$. When the number of variables exceeds $2,500$, BIPNN significantly outperforms Tabu as the variable count increases further.

F.4 Comparison of various HyperGNNs

Table 6: The performance of BIPNN with various HyperGNNs.

Datasets	HyperGNNs	Runtime (s)	VRAM (MB)	Number of cuts
Cora	HGNN+	**64**	**421**	**1,394**
	UniGCN	66	**421**	**1,394**
	UniGAT	84	479	1,358
	HyperGCN	4,517	533	1,317
Primary	HGNN+	**57**	**229**	8,495
	UniGCN	59	**229**	**8,496**
	UniGAT	70	231	8,442
	HyperGCN	4,818	235	8,363
High	HGNN+	**46**	**177**	**5,227**
	UniGCN	47	179	5,213
	UniGAT	55	189	5,209
	HyperGCN	3,367	233	5,118

We have evaluated the performance of BIPNN with various HyperGNNs, including HGNN+, UniGCN, UniGAT, and HyperGCN. HGNN+ always outperformed others in both training time

and VRAM usage. The evaluation results on the hypergraph max-cut problem have been depicted in Tab. 6. The evaluation metrics include runtime, VRAM consumption, and the number of cuts. We conduct the experiments on three hypergraph datasets, including Cora, Primary, and High, with all models trained for 5000 epochs.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: In this paper, we introduce a novel unsupervised neural network frameworks for solving nonlinear BIP problems. We provide theoretical justification and conduct extensive experiments on synthetic and real-world datasets.

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: We provided information on method limitations in section 6.

   Guidelines:

   - The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory Assumptions and Proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [Yes]

Justification: In section 3 and 4, where we discuss the proposed method's theoretical proposition, we provide a full set of assumptions and a complete proof.

Guidelines:

- The answer NA means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental Result Reproducibility**

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: We provide a fully detailed experimental results in section 6. We also provided the code for our experiments.

Guidelines:

- The answer NA means that the paper does not include experiments.
- If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [Yes]

Justification: We provide the code of our experiments.

Guidelines:

- The answer NA means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental Setting/Details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

   Answer: [Yes]

   Justification: All implementation details for experiments have been provided in section 6.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment Statistical Significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: We reported the standard deviation error of the mean and all the results in section 6.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).

- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
- If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments Compute Resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: Compute resources used are presented in section 6.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code Of Ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: We have reviewed the NeurIPS Code of Ethics and ensured our compliance with its guidelines.

   Guidelines:

   - The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader Impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [NA]

    Justification: We believe this work does not have a direct social impact.

    Guidelines:

    - The answer NA means that there is no societal impact of the work performed.
    - If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.

- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

    Answer: [NA]

    Justification: All the datasets used in this work are known and widely used. We are not aware of any risks involved in these datasets.

    Guidelines:

    - The answer NA means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: We share all the packages we used for our code.

    Guidelines:

    - The answer NA means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.

- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New Assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: We document our experiments thoroughly, anonymize the data, and provide the code.

    Guidelines:

    - The answer NA means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and Research with Human Subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [NA]

    Justification: This paper does not involve crowdsourcing or research with human subjects.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional Review Board (IRB) Approvals or Equivalent for Research with Human Subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [NA]

    Justification: The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.

- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.