GLinSAT: The General Linear Satisfiability Neural Network Layer By Accelerated Gradient Descent

Hongtai Zeng[1] **Chao Yang**[2] **Yanzhen Zhou**[1] **Cheng Yang**[2] **Qinglai Guo**[1*]
[1] State Key Laboratory of Power Systems, Department of Electrical Engineering, Tsinghua University
[2] Decision Intelligence Lab, Alibaba DAMO Academy
zenght20@mails.tsinghua.edu.cn, xiuxin.yc@alibaba-inc.com, zhouyzh@126.com, charis.yangc@alibaba-inc.com, guoqinglai@tsinghua.edu.cn

Abstract

Ensuring that the outputs of neural networks satisfy specific constraints is crucial for applying neural networks to real-life decision-making problems. In this paper, we consider making a batch of neural network outputs satisfy bounded and general linear constraints. We first reformulate the neural network output projection problem as an entropy-regularized linear programming problem. We show that such a problem can be equivalently transformed into an unconstrained convex optimization problem with Lipschitz continuous gradient according to the duality theorem. Then, based on an accelerated gradient descent algorithm with numerical performance enhancement, we present our architecture, GLinSAT, to solve the problem. To the best of our knowledge, this is the first general linear satisfiability layer in which all the operations are differentiable and matrix-factorization-free. Despite the fact that we can explicitly perform backpropagation based on automatic differentiation mechanism, we also provide an alternative approach in GLinSAT to calculate the derivatives based on implicit differentiation of the optimality condition. Experimental results on constrained traveling salesman problems, partial graph matching with outliers, predictive portfolio allocation and power system unit commitment demonstrate the advantages of GLinSAT over existing satisfiability layers. Our implementation is available at `https://github.com/HunterTracer/GLinSAT`.

1 Introduction

Constrained decision-making problems are pervasive across various disciplines. For example, logistics companies need to arrange delivery routes to minimize transportation costs while ensuring that all orders are delivered on time. Power system operators need to decide how to allocate electricity production between different power plants to meet ever-changing electricity demand while maintaining system stability. Unfortunately, directly solving these complex constrained decision-making problems via commercial optimization solvers requires a large amount of time. As a result, in scenarios that require rapid response, traditional solvers may not be suitable due to their long computation time. With the development of deep learning, it is hopeful that neural networks can capture the domain characteristics and complex relationships involved in constrained decision-making problems through their powerful expressive capability and the solution time can be thus reduced. In recent years, research on how to use neural networks to solve constrained decision-making problems has become a topic of general interest. Despite the great success of neural networks on classification and regression tasks, making the outputs of neural networks satisfy specific constraints is not straightforward, which still needs to be further investigated.

*Corresponding author.

38th Conference on Neural Information Processing Systems (NeurIPS 2024).

 https://doi.org/10.52202/079017-3896

A natural idea to impose constraints on the neural network outputs is to penalize the constraint violation in the training stage of supervised learning or reinforcement learning [1, 2, 3]. However, such an approach requires a careful selection of each penalty coefficient to achieve a balance between decision objectives and constraint violations. As the complexity of constraints increases, choosing appropriate penalty factors may require a large number of attempts, which is time-consuming. Moreover, it is often difficult to theoretically guarantee the boundedness of constraint violations [4], which makes penalty-based methods less attractive. Ref. [5] managed to determine the width of neural networks required for ensuring feasibility by modeling these networks using binary variables and solving a complex mixed-integer bilevel programming. However, this approach necessitates shrinking the original feasible region and can only handle inequality constraints, which limits its broader application. There are also methods in the literature that are better suited for inequality constraint satisfaction. Ref. [6] uses gauge function to map the neural network outputs from a ℓ^∞-norm unit ball into a given polyhedral. Despite its success in the field of control, this method may encounter difficulties in handling equality constraints since the polyhedral need to contain the origin as an interior point. Ref. [7] first calculates a reference point within the interior of a convex region using convex programming in the offline stage, and then computes a feasible point based on this reference point through simple arithmetic operations in the online stage. Despite its efficiency potentially being affected when constraints are not fixed, the method may encounter difficulties in satisfying equality constraints, as it requires computing the null space of the equality constraints. Since the basis for the null space of a matrix is typically dense, calculating the null space for large matrices may present both efficiency and memory challenges. Another way to encode constraints in neural networks is to reformulate the original problem as a Markov decision problem [8, 9]. During the solution process, the decision variables are generated one by one and the value range of the next variable is determined by the value of the current variable so that constraints can be satisfied compulsorily. However, not all decision-making problems can be equivalently converted to Markov decision problems which limits the application of such an approach.

Due to the limitations of the above approaches, many researchers want to use a more reliable way to ensure that the outputs of neural networks satisfy specific constraints. A promising way is to integrate optimization solvers as neural network layers. When we embed a solver for end-to-end learning, we need to pay special attention to the following two issues: the first one is the ***supported constraint types***, and the second one is the ***efficiency***.

As for the issue of supported constraint types, some frameworks can directly impose combinatorial constraints on neural network outputs through integrating black-box commercial mixed-integer programming solvers at the cost of inexact gradient estimates and poor utilization of GPUs (since modern commercial solvers are CPU-based) [10, 11, 12]. These approaches need to solve combinatorial optimization problems in both training and inference stages, which is time-consuming. Instead of directly handling the combinatorial constraints, some researchers manage to make neural network outputs satisfy constraints obtained from the continuous relaxation of the original problem, e.g. the widely used double stochastic matrix constraint in Ref. [13, 14, 15] solved by Sinkhorn algorithm [16, 17, 18]. Another example is the positive semi-definite (PSD) matrix constraint with unit diagonals as a continuous relaxation of the original MAXSAT problem [19]. However, both of the above methods can only express specific constraints, which limits their application. Recently, LinSAT, which is based on a generalized Sinkhorn algorithm, is proposed to impose positive linear constraints on neural network outputs [20]. However, the requirement for all elements in constraints to be non-negative limits the application of LinSAT. For example, even for a simple constraint $x \leq y$, namely $x - y \leq 0$, LinSAT cannot be used due to the negative coefficient in front of y, which shows the limited expressiveness of LinSAT. Decision variables with negative constraint coefficients occur a lot in real-life decision-making problems, such as the bin packing problem [21], chemical process scheduling [22], power system unit commitment [23], etc.

To deal with general linear constraints in a differentiable way, currently, there are two main approaches, CvxpyLayers [24] and OptNet [25]. However, when solving a batch of problems, both of them may encounter efficiency issues. Although CvxpyLayers can achieve parallelism through multiprocessing on the CPU, there are only a dozen of cores in one CPU, leading to limited parallelism performance. On the other hand, OptNet presents a GPU-based batch quadratic programming interior point solver where batch matrix factorization are performed to accelerate the solution process. Unfortunately, batch matrix factorization may be still a computational bottleneck even when GPU is used. Although some scholars have also studied how to parallelize parts of operations in matrix factorization on the

GPU [26, 27], the degree of parallelism still highly depend on the structure of the matrix and its elimination tree. Two nodes in the elimination tree can be computed in parallel only when there is no direct branch connecting them. As a result, matrix factorization cannot fully utilize the parallel computing ability of the GPU due to the sequential characteristics in the elimination tree [28].

In this paper, we investigate how to apply bounded and general linear constraints to neural network outputs in a differentiable way while ensuring batch processing can be performed efficiently on the GPU.

The contributions of this paper include:

1) To impose general linear and bounded constraints on neural network outputs in a differentiable way, we formulate the corresponding projection problem as an entropy-regularized linear programming where negative logistic entropic regularization terms are added into the objective function. We show that such an entropy-regularized linear programming problem can be transformed into an unconstrained convex optimization problem with Lipschitz continuous gradient, and thus can be solved by gradient descent based algorithms where no matrix factorization operation is required.

2) We design GLinSAT, a general linear satisfiability layer to impose linear constraints on neural network outputs based on a state-of-the-art accelerated gradient descent algorithm with numerical performance enhancement. Since the main operations in GLinSAT is matrix-vector product and no matrix factorization is involved, it is convenient to execute these operations in parallel on the GPU. Although all the operations involved in GLinSAT are differentiable which means that we can directly use the automatic differentiation mechanism to perform back propagation, we also provide an alternative way for derivative calculation based on the optimality condition to reduce the memory consumption.

3) We then provide experimental results to demonstrate the capabilities of our proposed method. Experiments on constrained traveling salesman problems, partial graph matching with outliers, predictive portfolio allocation and power system unit commitment show the efficacy of our proposed method and advantages over existing methods. A comparison of methodologies for imposing constraints on neural networks outputs is presented in Table 1. A pipeline that shows how our approach works is provided in Fig. 1.

Table 1: Comparison with existing optimizer layers for imposing constraints on the outputs of neural networks

Ref.	Abbr. of approach	Constraint type	GPU parallel computing	Matrix factorization free	Exact gradient	Explicit back-propagation	Implicit back-propagation
[10, 11, 12]	Perturbed optimizer	combinatorial	–	–	✗	✗	✓
[13, 14, 15]	Sinkhorn	double stochastic matrix	✓	✓	✓	✓	✗
[19]	SATNet	PSD matrix with unit diagonals	✓	✓	✓	✗	✓
[20]	LinSAT	positive linear	✓	✓	✓	✓	✗
[24]	CvxpyLayers	linear and cone	✗	✗	✓	✗	✓
[25]	OptNet	linear	✓	✗	✓	✗	✓
This article	GLinSAT	linear	✓	✓	✓	✓	✓

Note: Explicit/Implicit backpropagation means that this algorithm performs backward propagation based on automatic differentiation mechanism/implicit differentiation. – means that this algorithm feature is dependent on the implementation of the backend solver.

https://doi.org/10.52202/079017-3896

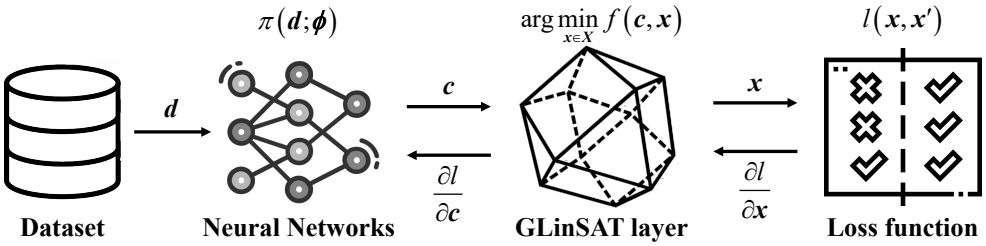

Figure 1: A pipeline that shows how GLinSAT layer works.

2 Methodology

Sec. 2.1 formulates the neural network output projection problem as an entropy-regularized linear programming by introducing logistic entropy regularization terms in the objective function. Based on duality theorem, the original problem can be transformed into an unconstrained convex optimization problem with Lipschitz continuous gradient. Then, in Sec. 2.2, based on a variant of accelerated gradient descent method, we design GLinSAT, which solves the projection problem using a GPU-friendly algorithm with several numerical enhancements. The corresponding time complexity is also provided in Sec. 2.2. Moreover, although all the operations in the forward pass of GLinSAT are differentiable, in Sec. 2.3, we provide an alternative way based on the optimality condition to calculate the derivatives for memory saving.

2.1 Reformulation of the neural network output projection problem

Here, we want to use a differentiable way to project the output of the neural network $\boldsymbol{c}' \in \mathbb{R}^{n'}$ into variables $\boldsymbol{x}' \in \mathbb{R}^{n'}$ that are as similar as possible but satisfy the following constraints (1).

$$\boldsymbol{A}_1'\boldsymbol{x}' \leq \boldsymbol{b}_1' \tag{1a}$$
$$\boldsymbol{A}_2'\boldsymbol{x}' \geq \boldsymbol{b}_2' \tag{1b}$$
$$\boldsymbol{A}_3'\boldsymbol{x}' = \boldsymbol{b}_3' \tag{1c}$$
$$\boldsymbol{l}' \leq \boldsymbol{x}' \leq \boldsymbol{u}' \tag{1d}$$

where $\boldsymbol{A}_1' \in \mathbb{R}^{m_1' \times n'}$, $\boldsymbol{A}_2' \in \mathbb{R}^{m_2' \times n'}$, $\boldsymbol{A}_3' \in \mathbb{R}^{m_3' \times n'}$, $\boldsymbol{b}_1' \in \mathbb{R}^{m_1'}$, $\boldsymbol{b}_2' \in \mathbb{R}^{m_2'}$, $\boldsymbol{b}_3' \in \mathbb{R}^{m_3'}$, $\boldsymbol{l}', \boldsymbol{u}' \in \mathbb{R}^{n'}$. Moreover, we also suppose that the feasible region in (1) is non-empty.

Apparently, any general linear constraints with bounded variables like (1) can be converted to the standard form like (2) by shifting bounds and introducing slack variables (see Appendix A.4):

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b} \tag{2a}$$
$$\boldsymbol{0} \leq \boldsymbol{x} \leq \boldsymbol{u} \tag{2b}$$

where $m = m_1' + m_2' + m_3'$, $n = n' + m_1' + m_2'$, $\boldsymbol{A} \in \mathbb{R}^{m \times n}$, $\boldsymbol{b} \in \mathbb{R}^m$, $\boldsymbol{u} \in \mathbb{R}^n_+$. Here, we denote the vector obtained from padding $(m_1' + m_2')$ zeros after the original vector $\boldsymbol{c}'$ as $\boldsymbol{c}$. Now, the original problem is transformed into a problem of projecting $\boldsymbol{c} \in \mathbb{R}^n$ onto $\boldsymbol{x} \in \mathbb{R}^n$ that satisfy constraints (2). In the following sections, we mainly focus on such a transformed problem in standard form.

In this paper, we aim for the vector $\boldsymbol{x}$ after projection to be as close as possible to the vector $\boldsymbol{c}$ prior to projection, while adhering to specified constraints. Here, we use the dot product as a measure of vector similarity. Consequently, our objective function becomes that of maximizing the dot product of vectors $\boldsymbol{c}$ and $\boldsymbol{x}$, which is equivalently described as minimizing the dot product of $-\boldsymbol{c}$ and $\boldsymbol{x}$. Besides, as pointed by [29], the optimal solution to an linear programming may not be differentiable (or even continuous) with respect to its parameters. Therefore, additional regularization terms need to be included in the objective to make the optimization problem differentiable. Inspired by entropy-regularized optimal transport, here we formulate the projection problem as an entropy-regularized linear programming to make the entire problem differentiable. Logistic entropy regularization terms

are added into the objective as follows:

$$\min_{\mathbf{0}\leq\boldsymbol{x}\leq\boldsymbol{u}} f(\boldsymbol{x}) = \min_{\mathbf{0}\leq\boldsymbol{x}\leq\boldsymbol{u}} -\boldsymbol{c}^T\boldsymbol{x} + \frac{1}{\theta}\sum_{j=1}^{n}\left(\frac{x_j}{u_j}\log\frac{x_j}{u_j} + \left(1-\frac{x_j}{u_j}\right)\log\left(1-\frac{x_j}{u_j}\right)\right) \tag{3a}$$

$$\text{s.t. } \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b} \tag{3b}$$

where $\theta > 0$ is the inverse temperature parameter that controls the approximation degree between the entropy-regularized problem and the original linear programming. The regularization coefficient $1/\theta$ controls the smoothness of the outputs. The smaller $1/\theta$ is, the more the outputs tend to be at the extreme point of the feasible region. As $\theta \to +\infty$, the optimal solution of the entropy-regularized problem should approach that of the original linear programming.

Remark 1. *It is noteworthy that unlike entropy-regularized optimal transport problems where only regularization terms in the form of $x\log x$ are involved in the objective, here logistic entropy regularization terms with respect to both x/u and its complement $1-x/u$ are added into the objective.* ***Actually, additionally incorporating the complementary entropy regularization terms is the most important part for the derivation of the Lagrange dual problem.*** *Otherwise, we cannot obtain a simple closed-form expression of the dual objective in the following derivation. Similarly, if we use the common ℓ^2-norm as the regularization term, we cannot obtain an analytical expression either.*

If we denote the dual variables with respect to the equality constraints (3b) as $\boldsymbol{y}$, the Lagrange dual function for (3) can be expressed as follows:

$$g(\boldsymbol{y}) = \inf_{\mathbf{0}\leq\boldsymbol{x}\leq\boldsymbol{u}}\left(-\boldsymbol{c}^T\boldsymbol{x} + \frac{1}{\theta}\mathbf{1}^T\left(\frac{\boldsymbol{x}}{\boldsymbol{u}}\circ\log\frac{\boldsymbol{x}}{\boldsymbol{u}} + \left(\mathbf{1}-\frac{\boldsymbol{x}}{\boldsymbol{u}}\right)\circ\log\left(\mathbf{1}-\frac{\boldsymbol{x}}{\boldsymbol{u}}\right)\right) - \boldsymbol{y}^T\boldsymbol{A}\boldsymbol{x}\right) + \boldsymbol{b}^T\boldsymbol{y} \tag{4}$$

where $\boldsymbol{a}\circ\boldsymbol{b}$, $\frac{\boldsymbol{b}}{\boldsymbol{a}}$ represents the element-wise multiplication and division of vector $\boldsymbol{a}$ and $\boldsymbol{b}$ respectively.

Since the derivative magnitude of $x\log x + (1-x)\log(1-x)$ tends to infinity when $x \to 0^+$ or $x \to 1^-$, the infimum in (4) can be attained only on a stationary point instead of a boundary point. When the infimum in (4) is attained, by making the derivative of the inner function equal to zero, we have:

$$-\boldsymbol{c} - \boldsymbol{A}^T\boldsymbol{y} + \frac{\mathbf{1}}{\theta\boldsymbol{u}}\circ\log\frac{\boldsymbol{x}}{\boldsymbol{u}-\boldsymbol{x}} = \mathbf{0} \tag{5}$$

After simplifying the above formula, we can get that when the infimum in (4) is attained, the optimal value of $\boldsymbol{x}(\boldsymbol{y})$ can be expressed as:

$$\boldsymbol{x}(\boldsymbol{y}) = \boldsymbol{u}\circ\boldsymbol{\sigma}\left(-\theta\boldsymbol{u}\circ\left(-\boldsymbol{c}-\boldsymbol{A}^T\boldsymbol{y}\right)\right) \tag{6}$$

where $\boldsymbol{\sigma}(\cdot)$ is the sigmoid function.

Substituting equation (6) into equation (4), we have:

$$g(\boldsymbol{y}) = \frac{1}{\theta}\mathbf{1}^T\log\boldsymbol{\sigma}\left(\theta\boldsymbol{u}\circ\left(-\boldsymbol{c}-\boldsymbol{A}^T\boldsymbol{y}\right)\right) + \boldsymbol{b}^T\boldsymbol{y} \tag{7}$$

Since $\log\boldsymbol{\sigma}(\cdot)$ is a strictly concave function, by minimizing the opposite of $g(\boldsymbol{y})$, we can obtain the following Lagrange dual problem (8), which is exactly an unconstrained convex optimization problem.

$$\min_{\boldsymbol{y}\in\mathbb{R}^m} -g(\boldsymbol{y}) = \min_{\boldsymbol{y}\in\mathbb{R}^m} -\frac{1}{\theta}\mathbf{1}^T\log\boldsymbol{\sigma}\left(\theta\boldsymbol{u}\circ\left(-\boldsymbol{c}-\boldsymbol{A}^T\boldsymbol{y}\right)\right) - \boldsymbol{b}^T\boldsymbol{y} \tag{8}$$

We can easily show that $f(\boldsymbol{x})$ is a strongly convex function and $-g(\boldsymbol{y})$ has Lipschitz continuous gradient (see explanations in Appendix A.5). Therefore, gradient descent based algorithms can be directly applied to solve such a problem.

2.2 Forward pass in GLinSAT

In the previous section, we have shown that the original entropy-regularized linear programming problem (3) can be equivalently converted into an unconstrained convex optimization problem (8) with Lipschitz continuous gradient. Theoretically, it can be solved readily through gradient descent based method. However, in the actual calculation process, it will be hard to choose a suitable step

 https://doi.org/10.52202/079017-3896

size if we just use vanilla gradient descent method. If the step size is much greater than the local Lipschitz constant, the algorithm may diverge. Otherwise, the convergence may be too slow.

Considering the strong convexity property of the entropy regularization terms, here we use a variant of accelerated gradient descent method, adaptive primal-dual accelerated gradient descent (APDAGD), which can adaptively approximate the local Lipschitz constant [30]. The detailed procedure of solving the entropy-regularized linear programming problem (3) in GLinSAT is provided in Algorithm 1.

Compared with the original version of APDAGD, here we improve the numerical performance of Algorithm 1 from the following two aspects. First, we use a smoother way to update the approximation of the local Lipschitz constant M in GLinSAT. In Algorithm 1, M is decreased only when the decrease of the dual objective satisfies the corresponding condition for at least two consecutive times. As a result, when M is already a good estimate of the local Lipschitz constant, the frequency of needless updates can be reduced, which will lead to less computation time. Second, to handle the round-off error, we also use a small number δ to relax the criterion for the decrease of the objective function. Otherwise, due to the existence of numerical error, the criterion of sufficient decrease in objective may be never satisfied. If we do not relax the criterion, M may become a large number and the algorithm will get stuck.

In addition, it is noteworthy that most of the operations involved in Algorithm 1 are calculation of matrix-vector products, vector-vector element-wise products and unary functions. Therefore, it is convenient to execute these operations in parallel on the GPU for solving a batch of entropy-regularized linear programming problems.

As for the time complexity of Algorithm 1, based on Theorem 1 and Theorem 2 in the supplementary material of [30], it can be easily proved that the number of iterations required by Algorithm 1 is roughly proportional to $\sqrt{\theta}$ and inversely proportional to $\sqrt{\varepsilon}$. The corresponding result is given in Corollary 1 and the detailed discussions can be found in Appendix A.6.

Algorithm 1: Solving the entropy-regularized linear programming problem in GLinSAT

Input: $\boldsymbol{A} \in \mathbb{R}^{m\times n}$, $\boldsymbol{b} \in \mathbb{R}^m$, $\boldsymbol{c} \in \mathbb{R}^n$, $\boldsymbol{u} \in \mathbb{R}^n_+$, inverse temerature $\theta > 0$, tolerance $\varepsilon > 0$, initial estimate of Lipschitz constant $L^{(0)}$, initial estimate of dual variables $\boldsymbol{y}^{(0)}$, numerical precision $\delta > 0$

Set $k = 0$, $M^{(0)} = L^{(0)}$, $\boldsymbol{\eta}^{(0)} = \boldsymbol{\zeta}^{(0)} = \boldsymbol{y}^{(0)}$, $\boldsymbol{x}^{(0)} = \boldsymbol{u} \circ \boldsymbol{\sigma}\left(-\theta\boldsymbol{u} \circ \left(-\boldsymbol{c} - \boldsymbol{A}^T\boldsymbol{y}^{(0)}\right)\right)$, $\beta^{(0)} = \alpha^{(0)} = 0$, f = `False`;

while $\left\|\boldsymbol{A}\boldsymbol{x}^{(k)} - \boldsymbol{b}\right\|_2 > \varepsilon$ **do**

- Set $\alpha^{(k+1)} = \left(1 + \sqrt{1 + 4M^{(k)}\beta^{(k)}}\right) / \left(2M^{(k)}\right)$;
- Set $\beta^{(k+1)} = \beta^{(k)} + \alpha^{(k+1)}$;
- Set $\tau^{(k+1)} = \alpha^{(k+1)}/\beta^{(k+1)}$;
- Set $\boldsymbol{\lambda}^{(k+1)} = \boldsymbol{\eta}^{(k)} + \tau^{(k+1)}\left(\boldsymbol{\zeta}^{(k)} - \boldsymbol{\eta}^{(k)}\right)$;
- Set $\boldsymbol{x}\left(\boldsymbol{\lambda}^{(k+1)}\right) = \boldsymbol{u} \circ \boldsymbol{\sigma}\left(-\theta\boldsymbol{u} \circ \left(-\boldsymbol{c} - \boldsymbol{A}^T\boldsymbol{\lambda}^{(k+1)}\right)\right)$;
- Set $\boldsymbol{\zeta}^{(k+1)} = \boldsymbol{\zeta}^{(k)} - \alpha^{(k+1)}\left(\boldsymbol{A}\boldsymbol{x}\left(\boldsymbol{\lambda}^{(k+1)}\right) - \boldsymbol{b}\right)$;
- Set $\boldsymbol{\eta}^{(k+1)} = \boldsymbol{\eta}^{(k)} + \tau^{(k+1)}\left(\boldsymbol{\zeta}^{(k+1)} - \boldsymbol{\eta}^{(k)}\right)$;
- **if** $\left(-g\left(\boldsymbol{\eta}^{(k+1)}\right)\right) - \left(-g\left(\boldsymbol{\lambda}^{(k+1)}\right)\right) - \delta \leq -\left\|\boldsymbol{A}\boldsymbol{x}\left(\boldsymbol{\lambda}^{(k+1)}\right) - b\right\|_2^2 / \left(2M^{(k)}\right)$ **then**
  - **if** f = `True` **then**
    - Set $M^{(k+1)} = M^{(k)}/2$;
  - **else**
    - Set $M^{(k+1)} = M^{(k)}$;
  - **end**
  - Set $\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} + \tau^{(k+1)}\left(\boldsymbol{x}\left(\boldsymbol{\lambda}^{(k+1)}\right) - \boldsymbol{x}^{(k)}\right)$, f = `True`;
  - Set $k = k + 1$;
- **else**
  - Set $M^{(k)} = 2M^{(k)}$, f = `False`;
- **end**

end

Output: Optimal primal variables $\boldsymbol{x}^{(k)}$, Optimal dual variables $\boldsymbol{\eta}^{(k)}$

Corollary 1. *Assume that the optimal dual solution $\boldsymbol{y}^*$ of problem (3) satisfies $\|\boldsymbol{y}^*\|_2 \leq R$. Then, for given tolerance $\varepsilon > 0$, the number of required iterations is $O\left(\|\boldsymbol{A}\|_2 \max(\boldsymbol{u}) \sqrt{\theta R/\varepsilon}\right)$.*

2.3 Backward pass in GLinSAT

Since all the operations involved in Algorithm 1 are differentiable with respect to $\boldsymbol{c}$, a natural idea is to directly use the auto differential mechanism to calculate the derivatives in the backward pass. However, directly backward propagation may require ever growing memory to store computational graphs and may cost much time when the forward pass requires a lot of iteration steps. To save the memory usage and accelerate the derivative calculation, we also provide an alternative way based on the optimality condition to calculate the derivatives in GLinSAT.

First, by calculating the derivative of $-g(\boldsymbol{y})$, we can obtain the optimality condition as follows:

$$\boldsymbol{h}(\boldsymbol{y}) = \boldsymbol{A}\left(\boldsymbol{u} \circ \boldsymbol{\sigma}\left(-\theta \boldsymbol{u} \circ \left(-\boldsymbol{c} - \boldsymbol{A}^T \boldsymbol{y}\right)\right)\right) - \boldsymbol{b} = \boldsymbol{0} \tag{9}$$

According to implicit differentiation and chain rule, differentiating equation (9), we can get:

$$\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{c}} = -\left(\frac{\partial \boldsymbol{h}}{\partial \boldsymbol{y}}\right)^{-1} \frac{\partial \boldsymbol{h}}{\partial \boldsymbol{c}} \tag{10}$$

Furthermore, according to equation (6), the derivative of loss function l with respect to $\boldsymbol{c}$ can be calculated as:

$$\frac{\partial l}{\partial \boldsymbol{c}} = \frac{\partial l}{\partial \boldsymbol{x}} \frac{\partial \boldsymbol{x}}{\partial \boldsymbol{c}} + \frac{\partial l}{\partial \boldsymbol{x}} \frac{\partial \boldsymbol{x}}{\partial \boldsymbol{y}} \frac{\partial \boldsymbol{y}}{\partial \boldsymbol{c}} + \frac{\partial l}{\partial \boldsymbol{y}} \frac{\partial \boldsymbol{y}}{\partial \boldsymbol{c}} = \frac{\partial l}{\partial \boldsymbol{x}} \frac{\partial \boldsymbol{x}}{\partial \boldsymbol{c}} - \left(\frac{\partial l}{\partial \boldsymbol{x}} \frac{\partial \boldsymbol{x}}{\partial \boldsymbol{y}} + \frac{\partial l}{\partial \boldsymbol{y}}\right) \left(\frac{\partial \boldsymbol{h}}{\partial \boldsymbol{y}}\right)^{-1} \frac{\partial \boldsymbol{h}}{\partial \boldsymbol{c}} \tag{11}$$

In the actual implementation of GLinSAT, we do not explicitly form these Jacobian matrices $\frac{\partial \boldsymbol{x}}{\partial \boldsymbol{c}}$, $\frac{\partial \boldsymbol{x}}{\partial \boldsymbol{y}}$, $\left(\frac{\partial \boldsymbol{h}}{\partial \boldsymbol{c}}\right)^{-1} \frac{\partial \boldsymbol{h}}{\partial \boldsymbol{c}}$. Instead, we directly form the matrix-vector products $\frac{\partial l}{\partial \boldsymbol{x}} \frac{\partial \boldsymbol{x}}{\partial \boldsymbol{c}}$, $\frac{\partial l}{\partial \boldsymbol{x}} \frac{\partial \boldsymbol{x}}{\partial \boldsymbol{y}}$. In addition, since the jacobian matrix $\frac{\partial \boldsymbol{h}}{\partial \boldsymbol{y}}$ is positive semi-definite (see derivations in Appendix A.7), we can use conjugate gradient method to calculate the inverse-matrix-vector products in GLinSAT. Therefore, only matrix-vector product operations are involved in the calculation of derivatives. Moreover, for the sake of completeness, in GLinSAT, we also implement derivatives with respect to $\boldsymbol{A}$, $\boldsymbol{b}$, $\boldsymbol{u}$ for future potential usage. The detailed derivation process of all derivatives is provided in Appendix A.7.

3 Experimental Results

In this section, experiments on constrained traveling salesman problems, partial graph matching with outliers, predictive portfolio allocation and power system unit commitment are used to demonstrate the advantages of GLinSAT through comparison with the state-of-the-art linear satisfiability layers LinSAT [20], CvxpyLayers [24] and OptNet [25]. For OptNet, LinSAT and GLinSAT, the regularization coefficients of nonlinear terms are all set to $1/\theta$. For CvxpyLayers, the projection problem to be solved is set to the same as (3). The first three experiments originate from Ref. [20]. The last experiment is the unit commitment problem in actual power systems. In the following sections, GLinSAT-(Dense/Sparse)-(Explicit/Implicit) means that GLinSAT is used with dense/sparse matrix and backpropagation is performed using automatic differential/implicit differential. LinSAT-(Dense/Sparse)-(100/500) means that LinSAT is used with dense/sparse matrix and maximum iteration number is set to 100/500. The reason we cannot set the maximum iteration number in LinSAT to $+\infty$, as we do in GLinSAT, is that LinSAT may iterate endlessly and get stuck in such a case. All the experiments are conducted on a computer with a 24-core Intel(R) Xeon(R) Platinum 8360H CPU and a NVIDIA Tesla A100 GPU through Pytorch 2.2. Our code is provided in `https://github.com/HunterTracer/GLinSAT`.

3.1 Constrained traveling salesman problem

Using the traveling salesman problem (TSP) dataset in [20], here we test the performance of each satisfiability layer through experiments on TSP with starting and ending cities constraint and priority constraint respectively. The mathematical formulation of TSP with starting and ending cities constraint (TSP-StartEnd) and TSP with priority constraint (TSP-Priority) is provided in Appendix

 https://doi.org/10.52202/079017-3896

Table 2: Average allocated GPU memory and solution time of different satisfiability layers during batch processing of projection and backpropagation when $\frac{1}{\theta}$ is set to 0.1 in TSP training phase

	TSP-StartEnd				TSP-Priority			
	GPU Mem./MB		Time/s		GPU Mem./MB		Time/s	
Layer	Proj.	Backprop.	Proj.	Backprop.	Proj.	Backprop.	Proj.	Backprop.
CvxpyLayers	—	—	112.1	18.39	—	—	116.5	19.94
OptNet	14305	5005	18.92	0.929	14333	5005	20.26	1.136
LinSAT-Dense-100	14977	181.2	**0.278**	0.417	15009	180.9	**0.276**	0.418
LinSAT-Dense-500	74108	181.2	1.323	1.927	74272	180.9	1.317	1.929
GLinSAT-Dense-Explicit	4380	**4.868**	0.382	0.240	4898	**4.880**	0.440	0.270
GLinSAT-Dense-Implicit	**13.35**	53.23	0.306	**0.143**	**13.36**	53.22	0.349	**0.146**
LinSAT-Sparse-100	7971	132.0	0.281	0.358	8026	132.6	0.286	0.368
LinSAT-Sparse-500	39020	133.5	1.356	1.614	39309	133	1.397	1.645
GLinSAT-Sparse-Explicit	2787	5.426	0.603	0.326	3127	5.983	0.652	0.355
GLinSAT-Sparse-Implicit	62.95	24.71	0.454	0.158	63.27	24.51	0.495	0.165

Note: The GPU memory used by CvxpyLayers is not counted since CvxpyLayers use the CPU parallel mechanism. Statistics of CvxpyLayers and OptNet are based on the first epoch since we cannot obtain a well-trained model in reasonable time.

A.8. The detailed experimental settings are provided in Appendix A.8. We report the average batch processing performance in Table 2 where $\frac{1}{\theta}$ is set to 0.1. The results when $\frac{1}{\theta}$ is set to 10^{-2} are similar therefore we display the results in Table A.1 and Table A.2 in Appendix A.8.

From Table 2, it can be seen that GLinSAT-Dense-Implicit outperforms all the other methods with minimum total storage and shortest total computation time. We can also find that our proposed GLinSAT is memory-efficient. Even though we choose the GLinSAT-Dense-Explicit method which will cost the most memory among all versions of GLinSAT, the total memory usage is still less than that of LinSAT-Sparse-100. We also attempted to set the maximum number of iterations for LinSAT to $+\infty$, but at this point LinSAT will get stuck and we cannot obtain a reasonable result. Contrarily, for our proposed GLinSAT, the convergence is guaranteed, so setting the maximum iteration number to $+\infty$ will not affect the result.

To obtain feasible tours, we exploit two kinds of post-processing methods in the validation stage [20]. The first one is rounding and the second one is beam search where the width of the beam is set to 2048. Table 3 shows the average tour length and feasibility ratio of each method. Since CvxpyLayers and OptNet are hundreds of times slower than LinSAT and GLinSAT, we are unable to obtain trained models within reasonable time and thus the results are not included.

Table 3: Mean tour length and feasibility ratio obtained from using different $\frac{1}{\theta}$ and post-processing methods in TSP validation stage

	TSP-StartEnd				TSP-Priority			
	Rounding with $\frac{1}{\theta} = 10^{-2}$		Beamsearch with $\frac{1}{\theta} = 10^{-1}$		Rounding with $\frac{1}{\theta} = 10^{-2}$		Beamsearch with $\frac{1}{\theta} = 10^{-1}$	
Layer	Mean Length	Feas. Ratio	Mean Length	Feas. Ratio	Mean Length	Feas. Ratio	Mean Length	Feas. Ratio
LinSAT-Dense-100	4.007	15.3%	3.843	**100%**	4.114	41.6%	3.952	**100%**
LinSAT-Dense-500	3.926	93.6%	3.823	**100%**	4.098	91.5%	3.947	**100%**
GLinSAT-Dense-Explicit	3.939	94.2%	3.817	**100%**	4.079	**93.5%**	3.934	**100%**
GLinSAT-Dense-Implicit	**3.922**	94.2%	**3.811**	**100%**	**4.068**	93.4%	**3.927**	**100%**
LinSAT-Sparse-100	—	—	3.843	**100%**	—	—	4.567	**100%**
LinSAT-Sparse-500	—	—	3.818	**100%**	—	—	4.400	**100%**
GLinSAT-Sparse-Explicit	3.939	94.2%	3.817	**100%**	4.078	92.6%	3.935	**100%**
GLinSAT-Sparse-Implicit	3.929	**94.6%**	3.818	**100%**	4.073	93.4%	3.933	**100%**

Note: The output of LinSAT-Sparse when $1/\theta = 10^{-2}$ is not a real number so that the results are not shown.

From Table 3, we can see that GLinSAT-Dense-Implicit results in the shortest mean tour length when beamsearch is used. It is also noteworthy that LinSAT will produce poor solution when we apply rounding to the results and the max iteration number is set to 100. Although setting the maximum number of iterations to 500 can improve LinSAT's performance, LinSAT's performance is still not as good as GLinSAT. Considering that the total computation time of LinSAT is more than five times that of GLinSAT at this time, it makes LinSAT less competitive.

3.2 Partial graph matching with outliers

The detailed mathematical formulation of partial graph matching with outliers is provided in A.9. We carry out experiments on Pascal VOC Keypoint dataset [31] with Berkeley annotations [32] under the unfiltered setting [20, 33].

Considering there are graphs with different sizes in one batch, we stack constraints as block diagonal matrices and forward them to LinSAT and GLinSAT. However, CvxpyLayers and OptNet currently cannot handle large block diagonal matrices. Disciplined parameterized programming compilation in CvxpyLayers and matrix factorization of large matrices in OptNet will cost a significant amount of time. Therefore, we can only use a for-loop to handle a batch with different sizes separately. The average GPU memory usage and solution time across different satisfiability layers is provided in Table A.3 of Appendix A.9. In the validation stage, we use Hungarian algorithm and greedy strategy for obtaining feasible integer solutions [20]. We regard the cost of matching a pair of nodes as the outputs of satisfiability layers, then use Hungarian algorithm to obtain a maximum matching. Finally, we use greedy strategy to preserve pairs with top-p matching scores for constraint satisfaction. The matching F1 scores between graph pairs across various satisfiability layers are shown in Table 4. The result of LinSAT-Dense-500 is not given due to out-of-memory (OOM) issues. According to Table 4, we can find GLinSAT yields the highest F1 scores across all satisfiability layers.

Table 4: Mean F1 scores across different satisfiability layers in partial graph matching problem

	$\frac{1}{\theta} = 10^{-1}$	$\frac{1}{\theta} = 10^{-2}$		$\frac{1}{\theta} = 10^{-1}$	$\frac{1}{\theta} = 10^{-2}$
Layer	Mean F1	Mean F1	Layer	Mean F1	Mean F1
CvxpyLayers	0.616	0.605	OptNet	0.619	0.613
LinSAT-Dense-100	0.619	0.614	LinSAT-Dense-500	×	×
GLinSAT-Dense-Explicit	**0.620**	**0.620**	GLinSAT-Dense-Implicit	0.619	**0.620**
LinSAT-Sparse-100	**0.620**	0.618	LinSAT-Sparse-500	0.619	0.611
GLinSAT-Sparse-Explicit	0.619	**0.620**	GLinSAT-Sparse-Implicit	**0.620**	**0.620**

3.3 Predictive portfolio allocation

In this section, we use the predictive portfolio allocation dataset in [20]. Denote $x_i \in [0, 1]$ as the predicted portfolio decision variable of asset i, $\mathcal{S}$ as the preferred portfolio asset. Our portfolio allocation needs to maximize the Sharpe ratio [34] while ensuring decision variables satisfy constraints $\sum_{i=1}^{n} x_i = 1, \sum_{i \in \mathcal{S}} x_i \geq q$ where q is a pre-defined positive constant. The details of experiments are provided in Appendix A.10. The average memory usage and solution time is shown in Table A.4 of Appendix A.10. According to Table A.4, we can find that our proposed GLinSAT is the fastest layer among all layers. In Table 5, we show the mean Sharpe ratio obtained from different satisfiability layers. Our proposed method always yields a high Sharpe ratio whether θ takes 10^{-1} or 10^{-2}.

Table 5: Mean Sharpe ratio obtained from different satisfiability layers in portfolio allocation problem

	$\frac{1}{\theta} = 10^{-1}$	$\frac{1}{\theta} = 10^{-2}$		$\frac{1}{\theta} = 10^{-1}$	$\frac{1}{\theta} = 10^{-2}$
Layer	S. Ratio	S. Ratio	Layer	S. Ratio	S. Ratio
CvxpyLayers	2.535	**2.600**	OptNet	**2.553**	2.381
LinSAT-Dense-100	2.245	2.409	LinSAT-Dense-500	2.245	2.409
GLinSAT-Dense-Explicit	2.535	**2.608**	GLinSAT-Dense-Implicit	2.535	**2.608**
LinSAT-Sparse-100	2.245	2.409	LinSAT-Sparse-500	2.245	2.409
GLinSAT-Sparse-Explicit	2.535	**2.608**	GLinSAT-Sparse-Implicit	2.535	**2.608**

https://doi.org/10.52202/079017-3896

3.4 Power system unit commitment

In this section, we carry out experiments about the unit commitment problem on a real provincial power system. In the unit commitment problem, there are hard constraints and soft constraints. Generally, constraints directly related to generators are regarded as hard constraints, e.g. the generator logical constraints, generator minimum up-time and down-time constraints. Constraints related to the section power and load balance are usually regarded as soft constraints where violation with large penalty coefficient is introduced in the objective [35]. The unit commitment problem can be formulated into a mixed-integer linear programming (MILP) problem, which is detailed in Appendix A.11. Based on one-year power system load data, we first use Gurobi [36] to solve the MILP within a 0.1% optimality gap.

After obtaining the integer solution of the unit commitment problem, we then use supervised learning to train neural networks with satisfiability layers so that they can predict the optimal state of a unit while satisfying logical constraints, minimum up-time and down-time constraints. As pointed by [37], when we consider logical constraints and minimum up-time and down-time constraints, these constraints formulate a convex hull so that the extreme points of the corresponding feasible region are binary. As a result, we could expect that the outputs of satisfiability layers tend to be binary when $1/\theta \to 0$, thereby making all constraints, including integer constraints, more likely to be satisfied after rounding operations. Once we obtain the predicted integer commitment status of the generators, we can fix the integer variables in the unit commitment problem and solve the corresponding linear programming problem, thereby providing a good initial point for the original mixed-integer programming problem.

Since negative coefficients occur in constraints, LinSAT cannot be used. In Table A.5 of Appendix A.11, we compare the performance of batch processing with different layers. When we stack constraints into block diagonal form to exploit parallelism, there are about 1000000 rows and 2000000 columns in the matrix. GLinSAT-Sparse-Implicit is the only way that will not report out-of-memory issues when we use GLinSAT. Both CvxpyLayers and OptNet cannot directly handle such a giant matrix within reasonable time thus we can only use a sequential way instead.

We train neural networks with $\frac{1}{\theta} = 0.1$. Table 6 shows the feasibility ratio and average gap on feasible solutions obtained from fixing unit state variables to rounded outputs of neural networks and then solving the continuous unit commitment problem in validation stage. Since CvxpyLayers and OptNet are significantly slower than GLinSAT, we are unable to obtain trained models within reasonable time and the results are not included in Table 6. Table 6 shows that if we use sigmoid function to replace the satisfiability layer in training and validation, we cannot obtain any feasible solution. As $\frac{1}{\theta} \to 0$, the feasibility ratio increases. When using GLinSAT with $\frac{1}{\theta} \leq 0.0005$, the feasibility ratio reaches 100%. In addition, when we set $\frac{1}{\theta}$ to exactly zero and solve the resulted projection problem in the form of linear programming (LP) via Gurobi, 100% feasible solutions are also found.

Table 6: Feasibility ratio and average gap obtained from using different $1/\theta$ in validation

Training final layer	Validation final layer	$1/\theta$	Feasibility Ratio	Average Gap
Sigmoid	Sigmoid	—	0%	—
GLinSAT-Sparse-Implicit	GLinSAT-Sparse-Implicit	0.01	86.23%	0.1119%
GLinSAT-Sparse-Implicit	GLinSAT-Sparse-Implicit	0.005	95.41%	0.1381%
GLinSAT-Sparse-Implicit	GLinSAT-Sparse-Implicit	0.001	98.17%	**0.1109%**
GLinSAT-Sparse-Implicit	GLinSAT-Sparse-Implicit	0.0005	**100%**	0.1114%
GLinSAT-Sparse-Implicit	GLinSAT-Sparse-Implicit	0.0001	**100%**	0.1114%
GLinSAT-Sparse-Implicit	Gurobi-LP	0	**100%**	0.1114%

4 Conclusion

In this paper, we reformulate the neural network output projection problem into a convex optimization problem with Lipschitz continuous gradient. We then propose GLinSAT, a general linear satisfiability layer to impose linear constraints on neural network outputs where all the operations are differentiable and matrix-factorization-free. GLinSAT can fully leverage the parallel computing capabilities of the GPU. We showcase four applications of GLinSAT and the advantages of our proposed framework over existing satisfiability layers are illustrated.

Acknowledgments and Disclosure of Funding

This work was supported in part by the National Natural Science Foundation of China under Grant U22B2097, 52321004 and in part by Alibaba Innovative Research Program. We would like to express our sincerest gratitude to the anonymous reviewers for their insightful feedback on our work. We are also immensely thankful to Wotao Yin for his invaluable support throughout the research process.

References

[1] Nikolaos Karalias and Andreas Loukas. Erdos goes neural: an unsupervised learning framework for combinatorial optimization on graphs. *Advances in Neural Information Processing Systems*, 33:6659–6672, 2020.

[2] Steven Bohez, Abbas Abdolmaleki, Michael Neunert, Jonas Buchli, Nicolas Heess, and Raia Hadsell. Value constrained model-free continuous control. *arXiv preprint arXiv:1902.04623*, 2019.

[3] Abhinav Bhatia, Pradeep Varakantham, and Akshat Kumar. Resource constrained deep reinforcement learning. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 29, pages 610–620, 2019.

[4] Runzhong Wang, Li Shen, Yiting Chen, Xiaokang Yang, Dacheng Tao, and Junchi Yan. Towards one-shot neural combinatorial solvers: Theoretical and empirical notes on the cardinality-constrained case. In *The Eleventh International Conference on Learning Representations*, 2022.

[5] Tianyu Zhao, Xiang Pan, Minghua Chen, and Steven H Low. Ensuring dnn solution feasibility for optimization problems with convex constraints and its application to dc optimal power flow problems. *The Eleventh International Conference on Learning Representations*, 2023.

[6] Daniel Tabas and Baosen Zhang. Computationally efficient safe reinforcement learning for power systems. In *2022 American Control Conference (ACC)*, pages 3303–3310. IEEE, 2022.

[7] Jesus Tordesillas, Jonathan P How, and Marco Hutter. Rayen: Imposition of hard convex constraints on neural networks. *arXiv preprint arXiv:2307.08336*, 2023.

[8] Irwan Bello, Hieu Pham, Quoc V Le, Mohammad Norouzi, and Samy Bengio. Neural combinatorial optimization with reinforcement learning. *arXiv preprint arXiv:1611.09940*, 2016.

[9] Elias Khalil, Hanjun Dai, Yuyu Zhang, Bistra Dilkina, and Le Song. Learning combinatorial optimization algorithms over graphs. *Advances in neural information processing systems*, 30, 2017.

[10] Marin Vlastelica Pogančić, Anselm Paulus, Vit Musil, Georg Martius, and Michal Rolinek. Differentiation of blackbox combinatorial solvers. In *International Conference on Learning Representations*, 2019.

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

[12] Anselm Paulus, Michal Rolínek, Vít Musil, Brandon Amos, and Georg Martius. Comboptnet: Fit the right np-hard problem by learning integer programming constraints. In *International Conference on Machine Learning*, pages 8443–8453. PMLR, 2021.

[13] Rodrigo Santa Cruz, Basura Fernando, Anoop Cherian, and Stephen Gould. Deeppermnet: Visual permutation learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3949–3957, 2017.

[14] Runzhong Wang, Junchi Yan, and Xiaokang Yang. Learning combinatorial embedding networks for deep graph matching. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 3056–3065, 2019.

 https://doi.org/10.52202/079017-3896

[15] Marco Cuturi, Olivier Teboul, and Jean-Philippe Vert. Differentiable ranking and sorting using optimal transport. *Advances in neural information processing systems*, 32, 2019.

[16] Richard Sinkhorn and Paul Knopp. Concerning nonnegative matrices and doubly stochastic matrices. *Pacific Journal of Mathematics*, 21(2):343–348, 1967.

[17] Marco Cuturi. Sinkhorn distances: Lightspeed computation of optimal transport. *Advances in neural information processing systems*, 26, 2013.

[18] Deeparnab Chakrabarty and Sanjeev Khanna. Better and simpler error analysis of the sinkhorn–knopp algorithm for matrix scaling. *Mathematical Programming*, 188(1):395–407, 2021.

[19] Po-Wei Wang, Priya Donti, Bryan Wilder, and Zico Kolter. Satnet: Bridging deep learning and logical reasoning using a differentiable satisfiability solver. In *International Conference on Machine Learning*, pages 6545–6554. PMLR, 2019.

[20] Runzhong Wang, Yunhao Zhang, Ziao Guo, Tianyi Chen, Xiaokang Yang, and Junchi Yan. Linsatnet: the positive linear satisfiability neural networks. In *International Conference on Machine Learning*, pages 36605–36625. PMLR, 2023.

[21] Salma Mezghani, Boukthir Haddar, and Habib Chabchoub. The evolution of the rectangular bin packing problem-a review of research topics, applications, and cited papers. 2023.

[22] Christodoulos A Floudas and Xiaoxia Lin. Mixed integer linear programming in process scheduling: Modeling, algorithms, and applications. *Annals of Operations Research*, 139: 131–162, 2005.

[23] Bernard Knueven, James Ostrowski, and Jean-Paul Watson. On mixed-integer programming formulations for the unit commitment problem. *INFORMS Journal on Computing*, 32(4): 857–876, 2020.

[24] Akshay Agrawal, Brandon Amos, Shane Barratt, Stephen Boyd, Steven Diamond, and J Zico Kolter. Differentiable convex optimization layers. *Advances in neural information processing systems*, 32, 2019.

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

[26] Steven C Rennich, Darko Stosic, and Timothy A Davis. Accelerating sparse cholesky factorization on gpus. *Parallel Computing*, 59:140–150, 2016.

[27] Meng Tang, Mohamed Gadou, and Sanjay Ranka. A multithreaded algorithm for sparse cholesky factorization on hybrid multicore architectures. *Procedia Computer Science*, 108:616–625, 2017.

[28] Timothy A Davis, Sivasankaran Rajamanickam, and Wissam M Sid-Lakhdar. A survey of direct methods for sparse linear systems. *Acta Numerica*, 25:383–566, 2016.

[29] Bryan Wilder, Bistra Dilkina, and Milind Tambe. Melding the data-decisions pipeline: Decision-focused learning for combinatorial optimization. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 1658–1665, 2019.

[30] Pavel Dvurechensky, Alexander Gasnikov, and Alexey Kroshnin. Computational optimal transport: Complexity by accelerated gradient descent is better than by sinkhorn's algorithm. In *International conference on machine learning*, pages 1367–1376. PMLR, 2018.

[31] Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The pascal visual object classes (voc) challenge. *International journal of computer vision*, 88: 303–338, 2010.

[32] Lubomir Bourdev and Jitendra Malik. Poselets: Body part detectors trained using 3d human pose annotations. In *2009 IEEE 12th international conference on computer vision*, pages 1365–1372. IEEE, 2009.

[33] Michal Rolínek, Paul Swoboda, Dominik Zietlow, Anselm Paulus, Vít Musil, and Georg Martius. Deep graph matching via blackbox differentiation of combinatorial solvers. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XXVIII 16*, pages 407–424. Springer, 2020.

[34] William F Sharpe. The sharpe ratio. *Streetwise–the Best of the Journal of Portfolio Management*, 3:169–185, 1998.

[35] Jianghua Wu, Peter B Luh, Yonghong Chen, Mikhail A Bragin, and Bing Yan. A novel optimization approach for sub-hourly unit commitment with large numbers of units and virtual transactions. *IEEE Transactions on Power Systems*, 37(5):3716–3725, 2021.

[36] Gurobi optimization. [Online]. Available: `https://www.gurobi.com`.

[37] Deepak Rajan, Samer Takriti, et al. Minimum up/down polytopes of the unit commitment problem with start-up costs. *IBM Res. Rep*, 23628:1–14, 2005.

[38] Martin WP Savelsbergh. Preprocessing and probing techniques for mixed integer programming problems. *ORSA Journal on Computing*, 6(4):445–454, 1994.

[39] Armin Fügenschuh and Alexander Martin. Computational integer programming and cutting planes. *Handbooks in Operations Research and Management Science*, 12:69–121, 2005.

[40] Tobias Achterberg. Constraint integer programming. 2007.

[41] Tobias Achterberg, Robert E Bixby, Zonghao Gu, Edward Rothberg, and Dieter Weninger. Presolve reductions in mixed integer programming. *INFORMS Journal on Computing*, 32(2): 473–506, 2020.

[42] Ambros Gleixner, Leona Gottwald, and Alexander Hoen. Papilo: A parallel presolving library for integer and linear optimization with multiprecision support. *INFORMS Journal on Computing*, 35(6):1329–1341, 2023.

[43] Runzhong Wang, Junchi Yan, and Xiaokang Yang. Neural graph matching network: Learning lawler's quadratic assignment problem with extension to hypergraph and multiple-graph matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(9):5261–5279, 2021.

[44] Defu Cao, Yujing Wang, Juanyong Duan, Ce Zhang, Xia Zhu, Congrui Huang, Yunhai Tong, Bixiong Xu, Jing Bai, Jie Tong, et al. Spectral temporal graph neural network for multivariate time-series forecasting. *Advances in neural information processing systems*, 33:17766–17778, 2020.

 https://doi.org/10.52202/079017-3896

A Appendix

A.1 Related Work

Constraints handling paradigm in neural networks for decision making. Some simple constraints can be directly encoded by neural network activation functions, e.g., ReLU function for non-negative constraints, sigmoid function for bounded constraints, softmax function for sum-to-one constraints. However, it is difficult for neural network outputs to satisfy complicated constraints by only using these activation functions. For a few problems with special structures, the constraints can be directly handled by the well-designed action space of sequential decisions in reinforcement learning [8, 9]. However, as the complexity of constraints increases, it will be hard to design a suitable action space. Considering the difficulty for agents to find feasible solutions through random exploration, the resulting sparse rewards may also lead to slow convergence. Another common way to deal with constraint violation is to penalize such violation in the training stage. Ref. [1] incorporates the constraint violation in the loss function of supervised learning. Ref. [2, 3] augment the reward function in reinforcement learning with the sum of the constraint violation penalty weighted by the Lagrange multipliers. However, it cannot be guaranteed that the constraints can be always satisfied by directly penalizing the constraint violation, see Ref. [4]. Ref. [5, 6, 7] are better suited for inequality constraint satisfaction. These methods may encounter difficulties in satisfying equality constraints, either in terms of efficiency or expressiveness. Another way to handle constraints is to incorporate optimization solvers into neural network layers, which is detailed in the next paragraph.

Optimizers as neural network layers for constraint satisfaction. To integrate optimizers into neural networks, it is necessary to calculate derivatives with respect to the parameters and perform batch processing efficiently. Ref. [10, 11, 12] exploit black-box solvers to impose combinatorial constraints on decision variables. However, only approximated gradients can be obtained through perturbations on the problem. Ref [19] relax the original combinatorial constraint into the positive semi-definite matrix constraint with unit diagonals and integrates GPU-based MAXSAT solver into neural network layers. In Ref. [13, 14, 15], Sinkhorn algorithm [16, 17, 18] is used to make neural network outputs satisfy the double stochastic matrix constraint, which is a linear relaxation of permutation, matching and sorting constraints. Since Sinkhorn algorithm only involves iterative normalization of rows and columns in matrices, it is straightforward to parallel the operations on GPUs and calculate the derivatives based on automatic differential mechanism. Ref. [25] presents OptNet, a neural network layer that integrates a GPU-based batched quadratic programming solver, qpth. The forward pass exploits a primal-dual interior point method to find the solution and the derivatives are calculated based on the matrix factorization obtained from the forward propagation. Ref. [24] further presents CvxpyLayers to incorporate convex programming into neural network layers. Although CvxpyLayers can represent more optimization problems, it relies on CPUs for parallelism which may lead to efficiency issues if a large batch of optimization problems needs solving. Ref. [20] designs LinSAT, a differentiable layer to encode the positive linear constraints based on a multi-set Sinkhorn algorithm. Although such an algorithm is easy to be parallel on GPUs, it can only imposing positive linear constraints on neural network outputs.

A.2 Broader Impacts

This paper is aimed at making the outputs of neural networks satisfy bounded and general linear constraints. The proposed framework GLinSAT can be used for end-to-end neural network training while ensuring the feasibility of neural network outputs, providing a promising approach for applying neural networks to decision-making problems. There is no foreseeable negative societal consequence that is a direct result of the proposed method.

A.3 Limitations

In this paper, we manage to impose bounded and linear constraints on neural network outputs. Considering that real-life decision variables often have finite upper and lower bounds simultaneously, our proposed method can actually be applied to a lot of decision-making problems. However, for variables with one-sided boundary or no explicit boundary, our method cannot be directly used. A possible workaround is to manually calculate the implicit bounds of these variables through domain propagation, see [38, 39, 40, 41, 42]. In addition, it should be noted that currently our algorithm can

only deal with linear constraints. In the future, we need to conduct further research on neural network layers that can efficiently handle cone constraints on GPUs.

A.4 Reformulation of general linear constraints with bounded variables into standard form

Denote $\boldsymbol{c}' \in \mathbb{R}^{n'}$ as the output of an neural network. In this section, we consider projecting the output of the neural network $\boldsymbol{x}' \in \mathbb{R}^{n'}$ into variables $\boldsymbol{x}' \in \mathbb{R}^{n'}$ that satisfy the following linear constraints and bounded constraints:

$$\boldsymbol{A}_1'\boldsymbol{x}' \leq \boldsymbol{b}_1' \tag{A.1a}$$
$$\boldsymbol{A}_2'\boldsymbol{x}' \geq \boldsymbol{b}_2' \tag{A.1b}$$
$$\boldsymbol{A}_3'\boldsymbol{x}' = \boldsymbol{b}_3' \tag{A.1c}$$
$$\boldsymbol{l}' \leq \boldsymbol{x}' \leq \boldsymbol{u}' \tag{A.1d}$$

where $\boldsymbol{A}_1' \in \mathbb{R}^{m_1' \times n'}$, $\boldsymbol{b}_1' \in \mathbb{R}^{m_1'}$, $\boldsymbol{A}_2' \in \mathbb{R}^{m_2' \times n'}$, $\boldsymbol{b}_2' \in \mathbb{R}^{m_2'}$, $\boldsymbol{A}_3' \in \mathbb{R}^{m_3' \times n'}$, $\boldsymbol{b}_3' \in \mathbb{R}^{m_3'}$, $\boldsymbol{l} \in \mathbb{R}^{n'}$, $\boldsymbol{u}' \in \mathbb{R}^{n'}$.

Obviously, we can convert all inequality constraints into equality constraints by introducing bounded slack variables as follows:

$$\boldsymbol{A}_1'\boldsymbol{x}' + \boldsymbol{s}_1' = \boldsymbol{b}_1 \tag{A.2a}$$
$$\boldsymbol{A}_2'\boldsymbol{x}' - \boldsymbol{s}_2' = \boldsymbol{b}_2 \tag{A.2b}$$
$$\boldsymbol{A}_3'\boldsymbol{x}' = \boldsymbol{b}_3' \tag{A.2c}$$
$$\boldsymbol{l}' \leq \boldsymbol{x}' \leq \boldsymbol{u}' \tag{A.2d}$$
$$\boldsymbol{0} \leq \boldsymbol{s}_1' \leq \overline{\boldsymbol{s}_1'} \tag{A.2e}$$
$$\boldsymbol{0} \leq \boldsymbol{s}_2' \leq \overline{\boldsymbol{s}_2'} \tag{A.2f}$$

where $\overline{\boldsymbol{s}_1'} = \boldsymbol{b}_1 - \boldsymbol{A}_1^{+'}\boldsymbol{l}' - \boldsymbol{A}_1^{-'}\boldsymbol{u}'$, $\overline{\boldsymbol{s}_2'} = \boldsymbol{A}_2^{+'}\boldsymbol{u}' + \boldsymbol{A}_2^{-'}\boldsymbol{l}' - \boldsymbol{b}_2$, $\boldsymbol{A}_{1+}', \boldsymbol{A}_{2+}'$ and $\boldsymbol{A}_{1-}', \boldsymbol{A}_{2-}'$ are the positive and negative parts of matrix $\boldsymbol{A}_1', \boldsymbol{A}_2'$ respectively.

We use the following notation:

$$\boldsymbol{x} = \begin{bmatrix} \boldsymbol{x}' - \boldsymbol{l}' \\ \boldsymbol{s}_1' \\ \boldsymbol{s}_2' \end{bmatrix}, \boldsymbol{A} = \begin{bmatrix} \boldsymbol{A}_1' & \boldsymbol{I} & \\ \boldsymbol{A}_2' & & -\boldsymbol{I} \\ \boldsymbol{A}_3' & & \end{bmatrix}, \boldsymbol{b} = \begin{bmatrix} \boldsymbol{b}_1' - \boldsymbol{A}_1'\boldsymbol{l}' \\ \boldsymbol{b}_2' - \boldsymbol{A}_2'\boldsymbol{l}' \\ \boldsymbol{b}_3' \end{bmatrix}, \boldsymbol{c} = \begin{bmatrix} \boldsymbol{c}' \\ \boldsymbol{0} \\ \boldsymbol{0} \end{bmatrix}, \boldsymbol{u} = \begin{bmatrix} \boldsymbol{u}' - \boldsymbol{l}' \\ \overline{\boldsymbol{s}_1'} \\ \overline{\boldsymbol{s}_2'} \end{bmatrix}$$

Then, the original problem is transformed into a problem of projecting $\boldsymbol{c} \in \mathbb{R}^{n'+m_1'+m_2'}$ onto variables $\boldsymbol{x} \in \mathbb{R}^{n'+m_1'+m_2'}$ that satisfy the following linear constraints and bounded constraints:

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b} \tag{A.3a}$$
$$\boldsymbol{0} \leq \boldsymbol{x} \leq \boldsymbol{u} \tag{A.3b}$$

A.5 Property of the primal and dual objective function

We first show that the primal objective function $f(\boldsymbol{x})$ is strongly convex, where $f(\boldsymbol{x})$ is:

$$f(\boldsymbol{x}) = -\boldsymbol{c}^T\boldsymbol{x} + \frac{1}{\theta}\sum_{j=1}^{n}\left(\frac{x_j}{u_j}\log\frac{x_j}{u_j} + \left(1 - \frac{x_j}{u_j}\right)\log\left(1 - \frac{x_j}{u_j}\right)\right) \tag{A.4}$$

The second order derivative of $f(\boldsymbol{x})$ can be expressed as follows:

$$\nabla^2 f(\boldsymbol{x}) = \frac{1}{\theta}\mathbf{diag}\left(\frac{\boldsymbol{1}}{\boldsymbol{x} \circ (\boldsymbol{u} - \boldsymbol{x})}\right) \tag{A.5}$$

where $\mathbf{diag}(\cdot)$ maps a vector to its corresponding diagonal matrix, $\circ$ represents the element-wise product, $\frac{\boldsymbol{1}}{\boldsymbol{z}}$ represents the element-wise reciprocal of vector $\boldsymbol{z}$.

Since we have $\boldsymbol{0} \leq \boldsymbol{x} \leq \boldsymbol{u}$, we have $\boldsymbol{x} \circ (\boldsymbol{u} - \boldsymbol{x}) \leq \frac{\boldsymbol{u}}{2} \circ \frac{\boldsymbol{u}}{\boldsymbol{2}} = \frac{1}{4}\boldsymbol{u} \circ \boldsymbol{u}$. As a result, we have $\nabla^2 f(\boldsymbol{x}) \succeq \frac{4}{\theta \max(\boldsymbol{u})^2}\boldsymbol{I}$, which means f is a $\frac{4}{\theta \max(\boldsymbol{u})^2}$-strongly convex function.

https://doi.org/10.52202/079017-3896

We next show that the gradient of the opposite dual objective function $\nabla\left(-g\left(\boldsymbol{y}\right)\right)$ is Lipschitz continuous, where $-g\left(\boldsymbol{y}\right)$ is:

$$-g\left(\boldsymbol{y}\right)=-\frac{1}{\theta}\mathbf{1}^{T}\log\boldsymbol{\sigma}\left(\theta\boldsymbol{u}\circ\left(-\boldsymbol{c}-\boldsymbol{A}^{T}\boldsymbol{y}\right)\right)-\boldsymbol{b}^{T}\boldsymbol{y} \tag{A.6}$$

The first order derivative of $-g\left(\boldsymbol{y}\right)$ can be expressed as follows:

$$\nabla\left(-g\left(\boldsymbol{y}\right)\right)=\boldsymbol{A}\boldsymbol{x}\left(\boldsymbol{y}\right)-\boldsymbol{b} \tag{A.7}$$

where $\boldsymbol{x}\left(\boldsymbol{y}\right)=\boldsymbol{u}\circ\boldsymbol{\sigma}\left(-\theta\boldsymbol{u}\circ\left(-\boldsymbol{c}-\boldsymbol{A}^{T}\boldsymbol{y}\right)\right)$.

Using the first order derivative of $-g\left(\boldsymbol{y}\right)$, we have:

$$\left\|\nabla\left(-g\left(\boldsymbol{y}_{1}\right)\right)-\nabla\left(-g\left(\boldsymbol{y}_{2}\right)\right)\right\|_{2}=\left\|\boldsymbol{A}\left(\boldsymbol{x}\left(\boldsymbol{y}_{1}\right)-\boldsymbol{x}\left(\boldsymbol{y}_{2}\right)\right)\right\|_{2}\leq\left\|\boldsymbol{A}\right\|_{2}\left\|\boldsymbol{x}\left(\boldsymbol{y}_{1}\right)-\boldsymbol{x}\left(\boldsymbol{y}_{2}\right)\right\|_{2} \tag{A.8}$$

According to equation the strong convexity of $f\left(\boldsymbol{x}\right)$, we have:

$$\begin{aligned}&\frac{4}{\theta\max\left(\boldsymbol{u}\right)^{2}}\left\|\boldsymbol{x}\left(\boldsymbol{y}_{1}\right)-\boldsymbol{x}\left(\boldsymbol{y}_{2}\right)\right\|_{2}^{2}\leq\left(\nabla f\left(\boldsymbol{x}\left(\boldsymbol{y}_{1}\right)\right)-\nabla f\left(\boldsymbol{x}\left(\boldsymbol{y}_{2}\right)\right)\right)^{T}\left(\boldsymbol{x}\left(\boldsymbol{y}_{1}\right)-\boldsymbol{x}\left(\boldsymbol{y}_{2}\right)\right)\\&=\left(\left(-\boldsymbol{c}+\frac{\mathbf{1}}{\theta\boldsymbol{u}}\circ\log\frac{\boldsymbol{x}\left(\boldsymbol{y}_{1}\right)}{\boldsymbol{u}-\boldsymbol{x}\left(\boldsymbol{y}_{1}\right)}\right)-\left(-\boldsymbol{c}+\frac{\mathbf{1}}{\theta\boldsymbol{u}}\circ\log\frac{\boldsymbol{x}\left(\boldsymbol{y}_{2}\right)}{\boldsymbol{u}-\boldsymbol{x}\left(\boldsymbol{y}_{2}\right)}\right)\right)^{T}\left(\boldsymbol{x}\left(\boldsymbol{y}_{1}\right)-\boldsymbol{x}\left(\boldsymbol{y}_{2}\right)\right)\\&=\left(\boldsymbol{A}\left(\boldsymbol{y}_{1}-\boldsymbol{y}_{2}\right)\right)^{T}\left(\boldsymbol{x}\left(\boldsymbol{y}_{1}\right)-\boldsymbol{x}\left(\boldsymbol{y}_{2}\right)\right)\leq\left\|\boldsymbol{A}\right\|_{2}\left\|\boldsymbol{y}_{1}-\boldsymbol{y}_{2}\right\|_{2}\left\|\boldsymbol{x}\left(\boldsymbol{y}_{1}\right)-\boldsymbol{x}\left(\boldsymbol{y}_{2}\right)\right\|_{2}\end{aligned} \tag{A.9}$$

which implies:

$$\left\|\boldsymbol{x}\left(\boldsymbol{y}_{1}\right)-\boldsymbol{x}\left(\boldsymbol{y}_{2}\right)\right\|_{2}\leq\frac{\theta\max\left(\boldsymbol{u}\right)^{2}\left\|\boldsymbol{A}\right\|_{2}}{4}\left\|\boldsymbol{y}_{1}-\boldsymbol{y}_{2}\right\|_{2} \tag{A.10}$$

Combining equation (A.8) and (A.10), we have:

$$\left\|\nabla\left(-g\left(\boldsymbol{y}_{1}\right)\right)-\nabla\left(-g\left(\boldsymbol{y}_{2}\right)\right)\right\|_{2}\leq\frac{\theta\max\left(\boldsymbol{u}\right)^{2}\left\|\boldsymbol{A}\right\|_{2}^{2}}{4}\left\|\boldsymbol{y}_{1}-\boldsymbol{y}_{2}\right\|_{2} \tag{A.11}$$

which means that $\nabla\left(-g\left(\boldsymbol{y}\right)\right)$ is $\frac{\theta\max(\boldsymbol{u})^{2}\left\|\boldsymbol{A}\right\|_{2}^{2}}{4}$-Lipschitz continuous.

A.6 Time Complexity of Algorithm 1

According to Theorem 2 in the supplementary material of [30], the l_2-norm of $\boldsymbol{A}\boldsymbol{x}^{(k)}-\boldsymbol{b}$ at the k-th iteration is bounded by $16LR/k^{2}$, where L is the Lipschitz constant of the gradient of the dual objective function, R is the upper bound of l_2-norm of the optimal dual variables. Since we have shown that $\nabla\left(-g\left(\boldsymbol{y}\right)\right)$ is $\frac{\theta\max(\boldsymbol{u})^{2}\left\|\boldsymbol{A}\right\|_{2}^{2}}{4}$-Lipschitz in Appendix A.5, we can see that:

$$\left\|\boldsymbol{A}\boldsymbol{x}^{(k)}-\boldsymbol{b}\right\|_{2}\leq\frac{4\theta\max\left(\boldsymbol{u}\right)^{2}\left\|\boldsymbol{A}\right\|_{2}^{2}R}{k^{2}} \tag{A.12}$$

Let $\left\|\boldsymbol{A}\boldsymbol{x}^{(k)}-\boldsymbol{b}\right\|_{2}=\varepsilon$, We can obtain the upper bound of outer cycle iteration number as follows:

$$k\leq2\max\left(\boldsymbol{u}\right)\left\|\boldsymbol{A}\right\|_{2}\sqrt{\frac{\theta R}{\varepsilon}} \tag{A.13}$$

According to Theorem 1 in the supplementary material of [30], the number of all the iterations after an iteration k is $O\left(k\right)$. Therefore, the time complexity of Algorithm 1 is $O\left(\max\left(\boldsymbol{u}\right)\left\|\boldsymbol{A}\right\|_{2}\sqrt{\theta R/\varepsilon}\right)$. As a result, for optimization problems that with similar upper bounds of dual variables, the required iteration number is approximately proportional to $\sqrt{\theta/\varepsilon}$. Such time complexity is better than the complexity of sinkhorn-based algorithms, in which the iteration number is approximately proportional to θ/ε^{2}, see [18, 20].

A.7 Derivative Calculation in the backward pass of GLinSAT

The optimality condition can be expressed as follows:

$$\boldsymbol{h}\left(\boldsymbol{y}\right)=\boldsymbol{A}\left(\boldsymbol{u}\circ\boldsymbol{\sigma}\left(-\theta\boldsymbol{u}\circ\left(-\boldsymbol{c}-\boldsymbol{A}^{T}\boldsymbol{y}\right)\right)\right)-\boldsymbol{b}=\boldsymbol{0} \tag{A.14}$$

Let $\boldsymbol{v}=\boldsymbol{A}$ or $\boldsymbol{b}$ or $\boldsymbol{c}$ or $\boldsymbol{u}$, according to chain rule, we have:

$$\frac{\partial l}{\partial \boldsymbol{v}}=\frac{\partial l}{\partial \boldsymbol{x}}\frac{\partial \boldsymbol{x}}{\partial \boldsymbol{v}}+\frac{\partial l}{\partial \boldsymbol{x}}\frac{\partial \boldsymbol{x}}{\partial \boldsymbol{y}}\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{v}}+\frac{\partial l}{\partial \boldsymbol{y}}\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{v}}=\frac{\partial l}{\partial \boldsymbol{x}}\frac{\partial \boldsymbol{x}}{\partial \boldsymbol{v}}-\left(\frac{\partial l}{\partial \boldsymbol{x}}\frac{\partial \boldsymbol{x}}{\partial \boldsymbol{y}}+\frac{\partial l}{\partial \boldsymbol{y}}\right)\left(\frac{\partial \boldsymbol{h}}{\partial \boldsymbol{y}}\right)^{-1}\frac{\partial \boldsymbol{h}}{\partial \boldsymbol{v}} \tag{A.15}$$

where $\boldsymbol{x}\left(\boldsymbol{y}\right)=\boldsymbol{u}\circ\boldsymbol{\sigma}\left(-\theta\boldsymbol{u}\circ\left(-\boldsymbol{c}-\boldsymbol{A}^{T}\boldsymbol{y}\right)\right)$.

We can first calculate $\frac{\partial x_q}{\partial y_p}$ as follows:

$$\frac{\partial x_q}{\partial y_p}=\theta x_q(u_q-x_q)A_{pq} \tag{A.16}$$

By writing the above equation into matrix form, we have:

$$\frac{\partial \boldsymbol{x}}{\partial \boldsymbol{y}}=\mathbf{diag}(\theta\boldsymbol{x}\circ(\boldsymbol{u}-\boldsymbol{x}))\boldsymbol{A}^{T} \tag{A.17}$$

In the actual computation process, the matrix $\frac{\partial \boldsymbol{x}}{\partial \boldsymbol{y}}$ is not explicitly formulated for saving GPU memory. Instead, we directly formulate the derivatives $\frac{\partial l}{\partial \boldsymbol{x}}\frac{\partial \boldsymbol{x}}{\partial A_{pq}}$, $\frac{\partial l}{\partial \boldsymbol{x}}\frac{\partial \boldsymbol{x}}{\partial b_p}$, $\frac{\partial l}{\partial \boldsymbol{x}}\frac{\partial \boldsymbol{x}}{\partial c_q}$, $\frac{\partial l}{\partial \boldsymbol{x}}\frac{\partial \boldsymbol{x}}{\partial u_q}$, $\frac{\partial l}{\partial \boldsymbol{x}}\frac{\partial \boldsymbol{x}}{\partial y_p}$ in (A.15) as follows:

$$\frac{\partial l}{\partial \boldsymbol{x}}\frac{\partial \boldsymbol{x}}{\partial A_{pq}}=\frac{\partial l}{\partial x_q}\frac{\partial x_q}{\partial A_{pq}}=\frac{\partial l}{\partial x_q}\theta x_q\left(u_q-x_q\right)y_p \tag{A.18a}$$

$$\frac{\partial l}{\partial \boldsymbol{x}}\frac{\partial \boldsymbol{x}}{\partial b_p}=0 \tag{A.18b}$$

$$\frac{\partial l}{\partial \boldsymbol{x}}\frac{\partial \boldsymbol{x}}{\partial c_q}=\frac{\partial l}{\partial x_q}\frac{\partial x_q}{\partial c_q}=\frac{\partial l}{\partial x_q}\theta x_q\left(u_q-x_q\right) \tag{A.18c}$$

$$\frac{\partial l}{\partial \boldsymbol{x}}\frac{\partial \boldsymbol{x}}{\partial u_q}=\frac{\partial l}{\partial x_q}\frac{\partial x_q}{\partial u_q}=\frac{\partial l}{\partial x_q}\frac{x_q-\theta x_q\left(u_q-x_q\right)\left(-c_q-\sum\limits_{i=1}^{m}y_iA_{iq}\right)}{u_q} \tag{A.18d}$$

$$\frac{\partial l}{\partial \boldsymbol{x}}\frac{\partial \boldsymbol{x}}{\partial y_p}=\sum_{q=1}^{n}\frac{\partial l}{\partial x_q}\frac{\partial x_q}{\partial y_p}=\sum_{q=1}^{n}\frac{\partial l}{\partial x_q}\theta x_q\left(u_q-x_q\right)A_{pq} \tag{A.18e}$$

By writing the above equations into matrix form, we have:

$$\frac{\partial l}{\partial \boldsymbol{x}}\frac{\partial \boldsymbol{x}}{\partial \boldsymbol{A}}=\boldsymbol{y}\left(\frac{\partial l}{\partial \boldsymbol{x}}\circ\theta\boldsymbol{x}\circ(\boldsymbol{u}-\boldsymbol{x})\right)^{T} \tag{A.19a}$$

$$\frac{\partial l}{\partial \boldsymbol{x}}\frac{\partial \boldsymbol{x}}{\partial \boldsymbol{b}}=\boldsymbol{0} \tag{A.19b}$$

$$\frac{\partial l}{\partial \boldsymbol{x}}\frac{\partial \boldsymbol{x}}{\partial \boldsymbol{c}}=\frac{\partial l}{\partial \boldsymbol{x}}\circ\theta\boldsymbol{x}\circ(\boldsymbol{u}-\boldsymbol{x}) \tag{A.19c}$$

$$\frac{\partial l}{\partial \boldsymbol{x}}\frac{\partial \boldsymbol{x}}{\partial \boldsymbol{u}}=\frac{\partial l}{\partial \boldsymbol{x}}\circ\left(\frac{\boldsymbol{x}-\theta\boldsymbol{x}\circ(\boldsymbol{u}-\boldsymbol{x})\circ\left(-\boldsymbol{c}-\boldsymbol{A}^{T}\boldsymbol{y}\right)}{\boldsymbol{u}}\right) \tag{A.19d}$$

$$\frac{\partial l}{\partial \boldsymbol{x}}\frac{\partial \boldsymbol{x}}{\partial \boldsymbol{y}}=\boldsymbol{A}\left(\frac{\partial l}{\partial \boldsymbol{x}}\circ\theta\boldsymbol{x}\circ(\boldsymbol{u}-\boldsymbol{x})\right) \tag{A.19e}$$

We then calculate $\frac{\partial h_p}{\partial y_q}$ as follows:

$$\frac{\partial h_p}{\partial y_q}=\sum_{j=1}^{n}A_{pj}\theta x_j\left(u_j-x_j\right)A_{qj} \tag{A.20}$$

https://doi.org/10.52202/079017-3896

By writing the above equations into matrix form, we have:

$$\frac{\partial \boldsymbol{h}}{\partial \boldsymbol{y}} = \boldsymbol{A}\mathbf{diag}\left(\theta \boldsymbol{x} \circ (\boldsymbol{u}-\boldsymbol{x})\right)\boldsymbol{A}^T \tag{A.21}$$

where $\mathbf{diag}(\cdot)$ maps a vector to its corresponding diagonal matrix. When $\boldsymbol{x}$ is the optimal solution, we have $\mathbf{0} < \boldsymbol{x} < \boldsymbol{u}$. Therefore, $\boldsymbol{A}\mathbf{diag}\left(\theta \boldsymbol{x} \circ (\boldsymbol{u}-\boldsymbol{x})\right)\boldsymbol{A}^T$ is a positive semi-definite matrix.

Here, we denote $\left(\frac{\partial l}{\partial \boldsymbol{x}}\frac{\partial \boldsymbol{x}}{\partial \boldsymbol{y}} + \frac{\partial l}{\partial \boldsymbol{y}}\right)\left(\frac{\partial \boldsymbol{h}}{\partial \boldsymbol{y}}\right)^{-1}$ as $\frac{\partial l}{\partial \boldsymbol{h}}$. After we finish the calculation of $\frac{\partial l}{\partial \boldsymbol{h}}$ by conjugate gradient method, we can calculate $\frac{\partial l}{\partial \boldsymbol{h}}\frac{\partial \boldsymbol{h}}{\partial A_{pq}}$, $\frac{\partial l}{\partial \boldsymbol{h}}\frac{\partial \boldsymbol{h}}{\partial b_p}$, $\frac{\partial l}{\partial \boldsymbol{h}}\frac{\partial \boldsymbol{h}}{\partial c_q}$, $\frac{\partial l}{\partial \boldsymbol{h}}\frac{\partial \boldsymbol{h}}{\partial u_q}$ as follows:

$$\frac{\partial l}{\partial \boldsymbol{h}}\frac{\partial \boldsymbol{h}}{\partial A_{pq}} = \sum_{i=1}^{m}\frac{\partial l}{\partial h_i}\frac{\partial h_i}{\partial A_{pq}} = \frac{\partial l}{\partial h_p}x_q + y_p\left(\sum_{i=1}^{m}\frac{\partial l}{\partial h_i}A_{iq}\right)\theta x_q\left(u_q - x_q\right) \tag{A.22a}$$

$$\frac{\partial l}{\partial \boldsymbol{h}}\frac{\partial \boldsymbol{h}}{\partial b_p} = \frac{\partial l}{\partial h_p}\frac{\partial h_p}{\partial b_p} = -\frac{\partial l}{\partial h_p} \tag{A.22b}$$

$$\frac{\partial l}{\partial \boldsymbol{h}}\frac{\partial \boldsymbol{h}}{\partial c_q} = \sum_{i=1}^{m}\frac{\partial l}{\partial h_i}\frac{\partial h_i}{\partial c_q} = \left(\sum_{i=1}^{m}\frac{\partial l}{\partial h_i}A_{iq}\right)\theta x_q\left(u_q - x_q\right) \tag{A.22c}$$

$$\frac{\partial l}{\partial \boldsymbol{h}}\frac{\partial \boldsymbol{h}}{\partial u_q} = \sum_{i=1}^{m}\frac{\partial l}{\partial h_i}\frac{\partial h_i}{\partial u_q} = \left(\sum_{i=1}^{m}\frac{\partial l}{\partial h_i}A_{iq}\right)\frac{x_q - \theta x_q\left(u_q - x_q\right)\left(-c_q - \sum\limits_{i=1}^{m} y_i A_{iq}\right)}{u_q} \tag{A.22d}$$

By writing the above equations into matrix form, we have:

$$\frac{\partial l}{\partial \boldsymbol{h}}\frac{\partial \boldsymbol{h}}{\partial \boldsymbol{A}} = \frac{\partial l}{\partial \boldsymbol{h}}\boldsymbol{x}^T + \boldsymbol{y}\left(\left(\boldsymbol{A}^T\frac{\partial l}{\partial \boldsymbol{h}}\right)\circ\left(\theta \boldsymbol{x}\circ(\boldsymbol{u}-\boldsymbol{x})\right)\right)^T \tag{A.23a}$$

$$\frac{\partial l}{\partial \boldsymbol{h}}\frac{\partial \boldsymbol{h}}{\partial \boldsymbol{b}} = -\frac{\partial l}{\partial \boldsymbol{h}} \tag{A.23b}$$

$$\frac{\partial l}{\partial \boldsymbol{h}}\frac{\partial \boldsymbol{h}}{\partial \boldsymbol{c}} = \left(\boldsymbol{A}^T\frac{\partial l}{\partial \boldsymbol{h}}\right)\circ\left(\theta \boldsymbol{x}\circ(\boldsymbol{u}-\boldsymbol{x})\right) \tag{A.23c}$$

$$\frac{\partial l}{\partial \boldsymbol{h}}\frac{\partial \boldsymbol{h}}{\partial \boldsymbol{u}} = \left(\boldsymbol{A}^T\frac{\partial l}{\partial \boldsymbol{h}}\right)\circ\left(\frac{\boldsymbol{x} - \theta \boldsymbol{x}\circ(\boldsymbol{u}-\boldsymbol{x})\circ\left(-\boldsymbol{c} - \boldsymbol{A}^T\boldsymbol{y}\right)}{\boldsymbol{u}}\right) \tag{A.23d}$$

Finally, by substituting equations (A.19) and (A.23) into (A.15), we can obtain the corresponding gradient as follows:

$$\frac{\partial l}{\partial \boldsymbol{A}} = \boldsymbol{y}\left(\left(\frac{\partial l}{\partial \boldsymbol{x}} - \boldsymbol{A}^T\frac{\partial l}{\partial \boldsymbol{h}}\right)\circ\left(\theta \boldsymbol{x}\circ(\boldsymbol{u}-\boldsymbol{x})\right)\right)^T - \frac{\partial l}{\partial \boldsymbol{h}}\boldsymbol{x}^T \tag{A.24a}$$

$$\frac{\partial l}{\partial \boldsymbol{b}} = \frac{\partial l}{\partial \boldsymbol{h}} \tag{A.24b}$$

$$\frac{\partial l}{\partial \boldsymbol{c}} = \left(\frac{\partial l}{\partial \boldsymbol{x}}\circ\theta \boldsymbol{x}\circ(\boldsymbol{u}-\boldsymbol{x})\right) - \left(\boldsymbol{A}^T\frac{\partial l}{\partial \boldsymbol{h}}\right)\circ\left(\theta \boldsymbol{x}\circ(\boldsymbol{u}-\boldsymbol{x})\right) \tag{A.24c}$$

$$\frac{\partial l}{\partial \boldsymbol{u}} = \left(\frac{\partial l}{\partial \boldsymbol{x}} - \boldsymbol{A}^T\frac{\partial l}{\partial \boldsymbol{h}}\right)\circ\left(\frac{\boldsymbol{x} - \theta \boldsymbol{x}\circ(\boldsymbol{u}-\boldsymbol{x})\circ\left(-\boldsymbol{c} - \boldsymbol{A}^T\boldsymbol{y}\right)}{\boldsymbol{u}}\right) \tag{A.24d}$$

A.8 Experimental details about constrained traveling salesman problem

Here, we first provide the mathematical formulation of TSP with starting and ending cities constraint (TSP-StartEnd) and TSP with priority constraint (TSP-Priority). We first focus on TSP-StartEnd, which can be modeled as follows:

$$\min_{\boldsymbol{X}\in\{0,1\}^{n\times n}} \quad \sum_{i=1}^{n}\sum_{j=1}^{n} D_{i,j}\sum_{k=1}^{n-1} X_{i,k}X_{j,k+1} \tag{A.25a}$$

$$\text{s.t.} \quad X_{s,1} = 1, X_{e,n} = 1, \tag{A.25b}$$

$$\boldsymbol{X}^T\mathbf{1}_n = \mathbf{1}_n, \boldsymbol{X}\mathbf{1}_n = \mathbf{1}_n \tag{A.25c}$$

where $X_{i,j} = 1$ means city i is the j-th visited city in a tour, $D_{i,j}$ refers to the distance between city i and city j.

On the basis of problem (A.25), the TSP-Priority problem can be obtained by introducing the following priority constraints:

$$\sum_{j=1}^{m+1} X_{p,j} = 1 \tag{A.26}$$

where p is the city that needs to be visited in the first m steps.

Due to the focus of this experiment on comparing the performance of each satisfiability layer, designing a better network structure is beyond the scope of this paper. Therefore, we directly used the SOTA network structure in solving TSP, which consists of a three-layers Transformer, followed by a three-layer MLP with ReLU activation [20]. The hidden sizes are all set to 256 and the number of multi-head attention is set to 8. All the training and test data is consisted of 20 nodes that are uniformly sampled from a unit square. For all satisfiability layers, the constraints are the continuous relaxation of the original TSP problem and all the common settings are the same as follows. The learning rate is set to 10^{-4}. The batch size is set to 1024. When we stack constraints into block diagonal form to exploit parallelism, there are about 40,000 rows and 400,000 columns in the whole matrix. The training epoch number is set to 50. The constraint tolerance is set to 10^{-3}. For GLinSAT, the initial estimate of Lipschitz constant is set to θ, the initial estimate of the dual variable is set to zero vector, the numerical precision is set to $10\epsilon_{\text{machine}}$, where $\epsilon_{\text{machine}}$ is the machine epsilon. In each epoch, we generate 256000 random cases as the training set. In the training stage, the objective function in (A.25a) is used as the loss function. In the validation stage, we use two kinds of post-processing methods. The first one is rounding. The second one is beam search where the width of the beam is set to 2048. All the experiments are conducted on a computer with a Intel(R) Xeon(R) Platinum 8360H CPU and a NVIDIA Tesla A100 GPU with 80GB memory through Pytorch 2.2.

Here, we additionally show the training performance when $1/\theta$ is set to 10^{-2} as follows:

Table A.1: Average allocated GPU memory and solution time of different satisfiability layers during batch processing of projection and backpropagation when $1/\theta$ is set to 10^{-2} in TSP training phase

	TSP-StartEnd				TSP-Priority			
	GPU Mem./MB		Time/s		GPU Mem./MB		Time/s	
Satisfiability layer	Proj.	Backprop.	Proj.	Backprop.	Proj.	Backprop.	Proj.	Backprop.
CvxpyLayers	—	—	107.3	31.12	—	—	107.4	30.22
OptNet	14305	5005	18.89	0.930	14333	5005	20.40	1.125
LinSAT-Dense-100	14977	181.2	**0.278**	0.417	15009	180.9	**0.277**	0.419
LinSAT-Dense-500	74108	181.2	1.339	1.930	74272	180.9	1.318	1.932
GLinSAT-Dense-Explicit	11095	**4.852**	0.976	0.552	11155	**4.857**	0.965	0.555
GLinSAT-Dense-Implicit	**13.34**	53.23	0.888	**0.072**	**13.36**	53.22	1.096	**0.078**
LinSAT-Sparse-100	—	—	—	—	—	—	—	—
LinSAT-Sparse-500	—	—	—	—	—	—	—	—
GLinSAT-Sparse-Explicit	7085	5.051	1.552	0.774	7007	5.519	1.508	0.812
GLinSAT-Sparse-Implicit	62.95	24.71	1.355	0.078	63.27	24.51	1.721	0.083

Note: LinSAT-(Dense/Sparse)-(100/500) means that LinSAT is used with dense/sparse matrix and max iteration number is set to 100/500. GLinSAT-(Dense/Sparse)-(Explicit/Implicit) means that GLinSAT is used with dense/sparse matrix and backpropagation is performed using automatic differential/implicit differential. The GPU memory used by CvxpyLayers is not counted since CvxpyLayers is CPU-based.
Note: The output of LinSAT-Sparse when $1/\theta = 0.01$ is not a real number so that the results are not shown.

From Table A.1, we can see GLinSAT-Dense-Implicit is the most memory efficient satisfiability layer among all these layers. As to the solution time, we can find that GLinSAT is slightly slower than LinSAT-Dense-100. **The reason LinSAT has a fast calculation speed is that the algorithm terminates due to reaching the maximum number of iterations rather than because of convergence.** Actually we find that LinSAT often reports warnings like "non-zero constraint violation

 https://doi.org/10.52202/079017-3896

within max iterations", which indicates the algorithm has not converged. After we increase the max iteration number to 500, the number of warnings has decreased, but there are still some warnings that indicate the algorithm has not converged. When we set maximum iteration number to $+\infty$, the algorithm progress will stuck. Compared with LinSAT, our proposed GLinSAT is more reliable and is guaranteed to converge. Table A.2 shows the corresponding validation results about the mean tour length and feasibility ratio. From Table A.2, we can see that the performance of GLinSAT is superior to that of LinSAT.

Table A.2: Mean tour length and feasibility ratio obtained from using different $1/\theta$ and post-processing methods in TSP validation stage

Layer	TSP-StartEnd				TSP-Priority			
	Rounding with $1/\theta$ = 10^{-3}		Beamsearch with $1/\theta = 0.1$		Rounding with $1/\theta$ = 10^{-3}		Beamsearch with $1/\theta = 0.1$	
	Mean Length	Feas. Ratio	Mean Length	Feas. Ratio	Mean Length	Feas. Ratio	Mean Length	Feas. Ratio
LinSAT-Dense-100	—	—	8.090	**100%**	—	—	7.031	**100%**
LinSAT-Dense-500	—	—	3.873	**100%**	—	—	4.011	**100%**
GLinSAT-Dense-Explicit	4.030	92.7%	3.869	**100%**	4.171	91.5%	3.983	**100%**
GLinSAT-Dense-Implicit	**3.930**	94.1%	**3.790**	**100%**	**4.053**	**94.6%**	**3.924**	**100%**
LinSAT-Sparse-100	—	—	—	—	—	—	—	—
LinSAT-Sparse-500	—	—	—	—	—	—	—	—
GLinSAT-Sparse-Explicit	4.022	92.6%	3.859	**100%**	4.078	92.6%	3.935	**100%**
GLinSAT-Sparse-Implicit	3.926	**94.8%**	3.803	**100%**	4.073	93.4%	3.933	**100%**

Note: The output of LinSAT-Sparse when $\frac{1}{\theta}$ = 10^{-2} is not a real number so that we cannot obtain any trained model. The output of LinSAT-Dense when $\frac{1}{\theta}$ = 10^{-3} is not a real number so that the results are not shown.

A.9 Experimental details about partial graph matching with outliers

Here, we first provide the mathematical formulation of partial graph matching with outliers. Denote m, n as the number of nodes of two graphs respectively. The partial graph matching problem with p inliers can be expressed as follows:

$$\boldsymbol{X}^T\mathbf{1}_m \leq \mathbf{1}_n \tag{A.27a}$$
$$\boldsymbol{X}\mathbf{1}_n \leq \mathbf{1}_m \tag{A.27b}$$
$$\mathbf{1}_m^T\boldsymbol{X}\mathbf{1}_n = p \tag{A.27c}$$
$$\boldsymbol{X} \in \{0,1\}^{m\times n} \tag{A.27d}$$

where $X_{i,j} = 1$ means the i-th node in the left graph matches the j-th node in the right graph.

When we use GLinSAT as the satisfiability layer, it is necessary to canonize the original inequality constraints. By introducing bounded slack variables into equations (A.27a) and (A.27b), we can reformulate constraints (A.27) into the standard form as follows:

$$\boldsymbol{X}^T\mathbf{1}_m + \boldsymbol{s}_n = \mathbf{1}_n \tag{A.28a}$$
$$\boldsymbol{X}\mathbf{1}_n + \boldsymbol{t}_m = \mathbf{1}_m \tag{A.28b}$$
$$\mathbf{1}_m^T\boldsymbol{X}\mathbf{1}_n = p \tag{A.28c}$$
$$\boldsymbol{X} \in \{0,1\}^{m\times n} \tag{A.28d}$$

where $\mathbf{0}_n \leq \boldsymbol{s}_n \leq \mathbf{1}_n, \mathbf{0}_m \leq \boldsymbol{t}_m \leq \mathbf{1}_m$.

In the training stage, we follow the experimental codes in Ref. [20] where the neural networks are trained on the basis of a pretrained SOTA graph matching NGMv2 model [43] named "pretrained_params_vgg16_ngmv2_afat-i_voc". Different satisfiability layers are used to make the outputs satisfy the continuous relaxation of constraints (A.27).

It is noteworthy that since the sizes of graphs differ a lot in one batch, we stack constraints into block diagonal forms in LinSAT and GLinSAT to exploit parallelism of the GPU. However, it is difficult

for CvxpyLayers and OptNet to directly handle large block diagonal matrices since disciplined parameterized programming compilation and matrix factorization of large matrices will cost a large amount of time. Therefore, we can only use a sequential way to handle batched graphs with different sizes for CvxpyLayers and OptNet. The batch size is set to 128 across all the experiments. When we stack constraints into block diagonal form to exploit parallelism, there are about 2,500 rows and 13,000 columns in the whole matrix. The constraint tolerance is set to 10^{-3}. In the training stage, binary cross entropy loss is used as the loss function. For experiments about OptNet and GLinSAT-Explicit, we use double-precision floating-point numbers during projection. If single-precision floating-point numbers are used, OptNet will encounter numerical issues in its forward pass while reporting warnings like "Returning an inaccurate and potentially incorrect solution". GLinSAT-Explicit will not encounter numerical issues in its forward pass. However, sometimes the gradient calculated by auto differential may be not a real number. We believe the problem is that single-precision floating-point numbers amplify the cumulative error of backpropagation. When we use double-precision floating-point numbers, everything works fine. As a result, we use single-precision floating-point numbers on all the other layers except OptNet and GLinSAT-Explicit.

In the validation stage, we use Hungarian algorithm and greedy strategy for post-processing [20]. We can regard the cost of matching a pair of nodes as the outputs of satisfiability layers, then use Hungarian algorithm to obtain a maximum matching. Finally, we can use greedy strategy to preserve pairs with p-highest matching scores and obtain the solution.

All the experiments are conducted on a computer with a Intel(R) Xeon(R) Platinum 8360H CPU and a NVIDIA Tesla A100 GPU with 80GB memory through Pytorch 2.2. For GLinSAT, the initial estimate of Lipschitz constant is set to θ, the initial estimate of the dual variable is set to zero vector, the numerical precision is set to $10\epsilon_{\text{machine}}$, where $\epsilon_{\text{machine}}$ is the machine epsilon. Here, we show the average memory usage and the solution time of different satisfiability layers in Table A.3.

Table A.3: Average allocated GPU memory and solution time of different satisfiability layers during batch processing of projection and backpropagation in the training phase of partial graph matching

	$\frac{1}{\theta} = 10^{-1}$				$\frac{1}{\theta} = 10^{-2}$			
	GPU Mem./MB		Time/s		GPU Mem./MB		Time/s	
Layer	Proj.	Backprop.	Proj.	Backprop.	Proj.	Backprop.	Proj.	Backprop.
CvxpyLayers	—	—	64.26	16.09	—	—	12.76	15.07
OptNet	167.3	826.2	4.290	**3.712**	167.2	826.3	4.437	3.726
LinSAT-Dense-100*	23944	322.1	0.611	3.887	23944	322.1	0.611	3.887
LinSAT-Dense-500	×	×	×	×	×	×	×	×
GLinSAT-Dense-Explicit	1249	**0.997**	1.891	4.418	1311	**1.052**	1.991	4.147
GLinSAT-Dense-Implicit	129.0	736.5	**1.333**	3.995	129.0	736.4	**1.302**	**3.515**
LinSAT-Sparse-100*	803.0	397.7	1.593	3.906	803.6	397.7	1.615	3.877
LinSAT-Sparse-500*	2772	304.3	2.922	5.534	2772	304.3	2.923	5.292
GLinSAT-Sparse-Explicit	648.7	172.9	1.794	4.435	687.6	139.8	1.986	4.620
GLinSAT-Sparse-Implicit	**0.001**	862.1	1.544	3.884	**0.001**	862.1	1.449	3.865

Note: The GPU memory used by CvxpyLayers is not counted since CvxpyLayers is CPU-based.
Note: The symbol "*" means the outputs of this satisfiability layer cannot meet constraints. The symbol "×" means the layer leads to out-of-memory (OOM) issues.

In Table A.3, although LinSAT-100 seems to be the fastest method, the price is that the required constraints are not satisfied at all. When we set the maximum iteration number of LinSAT to 100, almost every batch LinSAT will report warnings like "non-zero constraint violation within max iterations". When we set the maximum iteration number to 500, LinSAT-Dense will soon run out of memory while LinSAT-Sparse will still display the warning message in almost every epoch. Notably, the computational speed of LinSAT at this point has already fallen behind that of GLinSAT. If we set the maximum iteration number to $+\infty$ in LinSAT-Sparse, the algorithm will get stuck. Compared with our proposed GLinSAT, when the maximum iteration number is set to $+\infty$, the convergence of GLinSAT is guaranteed. In summary, we can conclude that our proposed GLinSAT is the fastest satisfiability layer while ensuring the outputs satisfy the linear and bounded constraints from the results shown in Table A.3.

https://doi.org/10.52202/079017-3896

A.10 Experimental details about predictive portfolio allocation

Here, we first restate the mathematical formulation of predictive portfolio allocation. We denote x_i as the predicted portfolio decision variable of asset i and $\mathcal{S}$ as the preferred portfolio asset. The portfolio allocation needs to maximize the Sharpe ratio [34] while satisfying the following constraints:

$$\sum_{i=1}^{n} x_i = 1, \sum_{i \in \mathcal{S}} x_i \geq q \tag{A.29a}$$

$$0 \leq x_i \leq 1, \forall 1 \leq i \leq n \tag{A.29b}$$

where q is a pre-defined positive constant. Following the codes in [20], here we set q as 0.5 and set C as {AAPL, MSFT, AMZN, TSLA, GOOGL}. We also use the first 120-day historical data to train a neural network that could maximize the Sharpe ratio for the future 120 days where StemGNN [44] are used as the network backbone to extract the features.

When we use GLinSAT as the satisfiability layer, it is necessary to canonize the original inequality constraints. By introducing bounded slack variables into constraints (A.29a), we can reformulate constraints (A.29) into the standard form as follows:

$$\sum_{i=1}^{n} x_i = 1, \sum_{i \in \mathcal{S}} x_i - w = q \tag{A.30a}$$

$$0 \leq w \leq |\mathcal{S}| - q, 0 \leq x_i \leq 1, \forall 1 \leq i \leq n \tag{A.30b}$$

where $|\mathcal{S}|$ refers to the number of elements in the preference set $\mathcal{S}$.

To conduct fair comparison, we train 50 epochs with a batch size of 128, a learning rate of 10^{-5} and a constraint tolerance of 10^{-3} across all satisfiability layers. When we stack constraints into block diagonal form to exploit parallelism, there are about 250 rows and 60,000 columns in the whole matrix. In the training stage, a weighted sum of prediction MSE error on future asset prices and the opposite of Sharpe ratio is used as the loss function. All the experiments are conducted on a computer with a Intel(R) Xeon(R) Platinum 8360H CPU and a NVIDIA Tesla A100 GPU with 80GB memory through Pytorch 2.2. For GLinSAT, the initial estimate of Lipschitz constant is set to θ, the initial estimate of the dual variable is set to zero vector, the numerical precision is set to $10\epsilon_{\text{machine}}$, where $\epsilon_{\text{machine}}$ is the machine epsilon. In the main text, we have reported the mean Sharpe ratio of each method in Table 5. Here, we provide the results related to the training performance in Table A.4. From Table A.4, we can see that the total memory usage is similar between each variant of LinSAT and GLinSAT since there are only two constraints in the original problem. The projection time of LinSAT and GLinSAT is significantly less than that of CvxpyLayers and OptNet while GLinSAT-Dense-Implicit use the shortest total calculation time among all satisfiability layers.

Table A.4: Average allocated GPU memory and solution time of different satisfiability layers during batch processing of projection and backpropagation in the training phase of predictive portfolio allocation

	$\frac{1}{\theta} = 10^{-1}$				$\frac{1}{\theta} = 10^{-2}$			
	GPU Mem./MB		Time/s		GPU Mem./MB		Time/s	
Layer	Proj.	Backprop.	Proj.	Backprop.	Proj.	Backprop.	Proj.	Backprop.
CvxpyLayers	—	—	12.85	1.476	—	—	12.27	1.423
OptNet	2012	765.0	5.599	1.035	2012	765.0	4.606	1.279
LinSAT-Dense-100	81.34	434.9	0.091	0.393	105.8	410.5	0.127	0.413
LinSAT-Dense-500	81.34	434.9	0.266	0.386	136.6	385.6	0.315	0.413
GLinSAT-Dense-Explicit	112.9	402.6	0.143	0.376	107.9	407.2	0.148	0.377
GLinSAT-Dense-Implicit	**1.565**	514.1	**0.090**	0.320	**1.565**	513.6	**0.090**	0.322
LinSAT-Sparse-100	223.6	**295.0**	0.113	0.400	290.7	227.6	0.161	0.421
LinSAT-Sparse-500	223.6	**295.0**	0.114	0.398	371.1	**224.1**	0.225	0.442
GLinSAT-Sparse-Explicit	78.24	437.7	0.211	0.391	74.80	440.8	0.213	0.392
GLinSAT-Sparse-Implicit	4.501	512.0	0.143	**0.316**	4.501	511.7	0.139	**0.315**

A.11 Experimental details about power system unit commitment

In this section, we carry out experiments on power system unit commitment where the data comes from a real provincial power system.

We first briefly introduce the unit commitment problem. Unit commitment problem is a core optimization problem in power system operation and planning. It mainly involves deciding how to most economically and safely arrange the startup-shutdown status and output power of generators while ensuring constraints related to equipment and grid operation can be satisfied. Generally, constraints can be divided into soft constraints and hard constraints. In general, constraints directly related to generators are often regarded as hard constraints, e.g. the generator minimum up-time and down-time constraints. Constraints related to the section power and load balance are usually regarded as soft constraints. For these constraints, we often penalize the corresponding violation in the objective [35].

Next, we will use the common three-binary formulation [23] to model the unit commitment problem. Let T denote the total number of time steps that considered in unit commitment problem. G is the total number of generators. We denote $\mathcal{T} = \{1, \cdots, T\}$ and $\mathcal{G} = \{1, \cdots, G\}$. Let $u_g(t)$ denote whether the generator g is on at time t, $v_g(t)$ denote whether the generator is turned on at time t, and $w_g(t)$ denote if the generator is turned off at time t. Then, $u_g(t), v_g(t), w_g(t)$ satisfy the following logical constraints:

$$u_g(t) - u_g(t-1) = v_g(t) - w_g(t), t \in \mathcal{T} \tag{A.31}$$

where T is the total number of time steps that considered in the unit commitment problem.

Units need to satisfy minimum up-time constraints and down-time constraints as follows:

$$\sum_{i=t-UT_g+1}^{t} v_g(i) \leq u_g(t), g \in \mathcal{G}, t = UT_g, \cdots T \tag{A.32a}$$

$$\sum_{i=t-DT_g+1}^{t} w_g(i) \leq 1 - u_g(t), g \in \mathcal{G}, t = DT_g, \cdots T \tag{A.32b}$$

where UT_g, DT_g are the minimum up-time and minimum down-time for generator g respectively.

Let $p_g(t)$ denote the power produced by generator g at time t. Let $\underline{p_g}, \overline{p_g}$ denote the lower bound and upper bound of generator g. Then, we have constraints related to the bound of generator output as follows:

$$\underline{p_g} u_g(t) \leq p_g(t) \leq \overline{p_g} u_g(t), g \in \mathcal{G}, t \in \mathcal{T} \tag{A.33}$$

We also need to consider the ramping capability of each generator. The ramping constraints can be formulated as follows:

$$p_g(t) - p_g(t-1) \leq -RU_g v_g(t) + \left(\underline{p_g} + RU_g\right) u_g(t) - \underline{p_g} u_g(t-1), g \in \mathcal{G}, t \in \mathcal{T} \tag{A.34a}$$

$$p_g(t-1) - p_g(t) \leq -RD_g w_g(t) + \left(\underline{p_g} + RD_g\right) u_g(t-1) - \underline{p_g} u_g(t), g \in \mathcal{G}, t \in \mathcal{T} \tag{A.34b}$$

where RU_g, RD_g denote the ramp-up rate and ramp-down rate of generator g.

The load generation balance constraint is modeled as the following soft constraint form:

$$\sum_{g \in \mathcal{G}} p_g(t) + s^+(t) - s^-(t) = \sum_{d \in \mathcal{D}} l_d(t), t \in \mathcal{T} \tag{A.35}$$

where $\mathcal{D}$ is the set of all loads, $l_d(t)$ represents the d-th load at time t, $s^+(t) \geq 0, s^-(t) \geq 0$ are the non-negative slack variables at time t which will be later penalized in the objective.

We denote the set of sections as $\mathcal{K}$. The soft constraints on section power are provided as follows:

$$\underline{F_k} \leq \sum_{g \in \mathcal{G}} H_{kg} p_g(t) - \sum_{d \in \mathcal{D}} H_{kd} l_d(t) + s_k^+(t) - s_k^-(t) \leq \overline{F_k}, k \in \mathcal{K}, t \in \mathcal{T} \tag{A.36}$$

where $\underline{F_k}, \overline{F_k}$ are the lower bound and upper bound of the k-th section power, H_{kg} is the generation shift factor that indicates the change of the k-th section power with respect to a change in injection

 https://doi.org/10.52202/079017-3896

at generator g, H_{kg} is the load shift factor that indicates the change of the k-th section power with respect to a change in injection at load d, $s_k^+(t) \geq 0, s_k^-(t) \geq 0$ are the non-negative slack variables related to the k-th section at time t which will be later penalized in the objective.

Finally, we want to minimize the system operation cost and we can obtain the optimization problem as follows:

$$\min \quad \sum_{t\in\mathcal{T}}\sum_{g\in\mathcal{G}}\left(c_g p_g(t) + c_g^{SU} v_g(t)\right) + \sum_{t\in\mathcal{T}} M\left(s^+(t) + s^-(t)\right) + \sum_{t\in\mathcal{T}}\sum_{k\in\mathcal{K}} M_k\left(s_k^+(t) + s_k^-(t)\right) \tag{A.37a}$$

$$\text{s.t.} \quad (\text{A.31}) - (\text{A.36}),$$

$$u_g(t) \in \{0,1\}, v_g(t) \in \{0,1\}, w_g(t) \in \{0,1\}, g \in \mathcal{G}, t \in \mathcal{T}, \tag{A.37b}$$

$$s_k^+(t) \geq 0, s_k^-(t) \geq 0, t \in \mathcal{T}, \tag{A.37c}$$

$$s^+(t) \geq 0, s^-(t) \geq 0, k \in \mathcal{K}, t \in \mathcal{T} \tag{A.37d}$$

where M, M_k are pre-defined penalty coefficients, c_g is the generator cost coefficient, c_g^{SU} is the generator start-up cost.

The power system we use in this article contains about 360 units. We set the total number of time steps T as 96 where the interval between two adjacent time steps is set to 15 minutes. More than 1,400 sections need to be considered in the unit commitment problem. The penalty coefficient for load imbalance is set to 10^{11} and the penalty coefficient for section power violation is set to 10^7. Based on the one-year load data, we solve the unit commitment problem via Gurobi within a 0.1% optimality gap. We can then obtain the optimal unit states for further supervised learning.

In supervised learning, we want to predict the optimal value of u_g as accurately as possible so that we can fix these binary variables and quickly obtain a high-quality solution from solving a linear programming problem. Therefore, we should require the predicted variables u_g to satisfy the hard constraints (A.31) and (A.32), namely the logical constraint and the minimum up-time and down-time constraints. In addition, $u_g(t), v_g(t), w_g(t)$ should be in the range between 0 and 1. We use satisfiability layers to ensure the above constraints can be satisfied.

When we use GLinSAT as the satisfiability layer, it is necessary to canonize the original inequality constraints. By introducing bounded slack variables into constraints (A.29a), we can reformulate constraints (A.29) into the standard form as follows:

$$\sum_{i=t-UT_g+1}^{t} v_g(i) + sv_g(t) = u_g(t), g \in \mathcal{G}, t = UT_g, \cdots T \tag{A.38a}$$

$$\sum_{i=t-DT_g+1}^{t} w_g(i) + sw_g(t) = 1 - u_g(t), g \in \mathcal{G}, t = DT_g, \cdots T \tag{A.38b}$$

Since we have $u_g(t) \in [0,1], v_g(t) \in [0,1], w_g(t) \in [0,1]$, the value range of variables $sv_g(t)$ and $sw_g(t)$ can be quickly inferred, which is exactly $sv_g(t) \in [0,1], sw_g(t) \in [0,1]$.

We use a MLP-based neural network to learn the optimal unit states. The loads at each time-step are forwarded to a 2-layer MLP where the hidden sizes are set to 32. Then, we concatenate the embeddings at all time steps, and forward the concatenated embedding to a 1-layer MLP where the hidden size is set to 3072. Finally, we use a 2-layer MLP to read out the optimal unit states where the hidden size is set to 512. We use ReLU as the activation function in the hidden layers. We set the learning rate to 10^{-3} and train neural networks for 100 epochs. We use 70% of one-year data as the training set and the other as the validation set. In the training stage, binary cross entropy loss is used as the loss function. For all satisfiability layers, the batch size is set to 16 and the regularization parameter $\frac{1}{\theta}$ is set to 0.1. It is worth noting that although the batch size is not that large, the scale of the optimization problem in each batch is still quite large. Each instance involves nearly 360 projection problems, and each projection problem involves about 160 constraints and 350 variables. When we stack constraints into block diagonal form to exploit parallelism, there are about 1,000,000 rows and 2,000,000 columns in the whole matrix. GLinSAT-Sparse-Implicit is the only way that will not report out-of-memory issues when we use GLinSAT. For GLinSAT, the initial estimate of Lipschitz constant is set to θ, the initial estimate of the dual variable is set to zero vector, the

numerical precision is set to $10\epsilon_{\text{machine}}$, where $\epsilon_{\text{machine}}$ is the machine epsilon. As to CvxpyLayers and OptNet, neither of them can directly handle such a giant matrix within reasonable time thus we can only use a sequential way instead. The performance of each satisfibility layer is provided in Table A.5. The results of LinSAT layers are not reported in Table A.5 since LinSAT layers only support positive linear constraints. All the experiments are conducted on a computer with a Intel(R) Xeon(R) Platinum 8360H CPU and a NVIDIA Tesla A100 GPU with 80GB memory through Pytorch 2.2.

Table A.5: Average allocated GPU memory and solution time of different satisfiability layers during batch processing of projection and backpropagation in the training stage of predicting unit states

	GPU Mem./MB		Time/s	
Layer	Proj.	Backprop.	Proj.	Backprop.
CvxpyLayers	—	—	2771	684.0
OptNet	33012	96.64	257.4	23.60
GLinSAT-Sparse-Implicit	**930.7**	**76.86**	**26.78**	**1.636**

Note: Statistics of CvxpyLayers and OptNet are based on the first epoch since we cannot obtain a well-trained model in reasonable time.

In the validation stage, we use different values of $\frac{1}{\theta}$ to test their performance. When $\frac{1}{\theta}$ is set to 10^{-3}, we use the double-precision floating-point numbers during projection to avoid potential numerical issues. When $\frac{1}{\theta}$ is set to 0, the projection problem turns into a linear programming problem and we solve it via Gurobi. After we have obtained the outputs of satisfiability layers, we round the outputs of final layers to 0 or 1 and then fix the unit state variables u_g using these rounded outputs. Once we fix all unit state variables to 0 or 1, the original mixed-integer linear programming (MILP) problem turns to a linear programming (LP) problem. We use Gurobi to solve such an LP problem and record whether the problem with fixed unit states is feasible and record the optimal solution if exists. We also compare the optimal objective obtained from fixing variables with the original optimal objective. The corresponding average gaps between them are shown in Table 6.

To illustrate the importance of the satisfiable layer, we additionally substitute the original satisfiability layer with a simple sigmoid activation function in both training and validation stages and report the corresponding result in Table 6. We can easily see that satisfiability layers are essential to produce feasible unit states.

 https://doi.org/10.52202/079017-3896

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 consider making a batch of neural network outputs satisfy bounded and general linear constraints. We present GLinSAT, which is the first general linear satisfiability layer in which all the operations are differentiable and matrix-factorization-free. Experimental results demonstrate the advantages of GLinSAT over existing methods.

   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 have discussed our limitations in Appendix A.3. For variables with one-sided boundary or no explicit boundary, our method cannot be directly used. A possible workaround is to manually calculate the implicit bounds of these variables through domain propagation but we have not implemented such an algorithm in GLinSAT. Also, currently our proposed framework cannot deal with conic constraints.

   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 the derivation process, we always assume the feasible region is non-empty as shown in Sec. 2.1. The derivation process of our results can be found in Sec. 2.1, Appendix A.5, A.6 and A.7.

   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 have detailed our experimental settings in Appendix A.8, A.9, A.10 and A.11 and the code released in github can be also be used as a reference.

   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).

https://doi.org/10.52202/079017-3896

(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 our codes and instructions on the first three experiments, but currently we are unable to open source the data and code about the unit commitment problem in a real power system due to data security issues and confidentiality agreements.

   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: We have detailed our experimental settings in Appendix A.8, A.9, A.10 and A.11 and the code released in github can be also be used as a reference.

   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: [No]

   Justification: Error bars are not reported because it would be too computationally expensive.

   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: All the experiments are conducted on a computer with a Intel(R) Xeon(R) Platinum 8360H CPU and a NVIDIA Tesla A100 GPU with 80GB memory through Pytorch 2.2. We also provide the batch processing performance in Table 2, Table A.1, Table A.3, Table A.4, Table A.5.

   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: The research conducted in the paper conform with the NeurIPS Code of Ethics.

   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?

https://doi.org/10.52202/079017-3896

Answer: [Yes]

Justification: We have discussed potential impacts in Appendix A.2.

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: This paper poses no risk for misusing our proposed method.

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 have properly credited the used Pascal VOC Keypoint dataset [31] with Berkeley annotations [32] under the unfiltered setting [20, 33]. We also cite the TSP dataset and portfolio allocation dataset provided in Ref. [20].

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: [NA]

    Justification: This paper does not release new assets.

    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 nor 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: This paper does not involve crowdsourcing nor research with human subjects.

 https://doi.org/10.52202/079017-3896

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.