Scalable Mixed-Integer Optimization with Neural Constraints via Dual Decomposition

Shuli Zeng[1], **Sijia Zhang**[1*], **Feng Wu**[1*], **Shaojie Tang**[2], **Xiangyang Li**[1*]
[1]University of Science and Technology of China
[2]State University of New York at Buffalo
zengshuli0130@mail.ustc.edu.cn, sxzsj@mail.ustc.edu.cn,
wufeng02@ustc.edu.cn, shaojiet@buffalo.edu, xiangyangli@ustc.edu.cn

Abstract

Embedding deep neural networks (NNs) into mixed-integer programs (MIPs) is attractive for decision making with learned constraints, yet state-of-the-art "monolithic" linearisations blow up in size and quickly become intractable. In this paper, we introduce a novel dual-decomposition framework that relaxes the single coupling equality $u = x$ with an augmented lagrange multiplier and splits the problem into a vanilla MIP and a constrained NN block. Each part is tackled by the solver that suits it best—branch & cut for the MIP subproblem, first-order optimisation for the NN subproblem—so the model remains modular, the number of integer variables never grows with network depth, and the per-iteration cost scales only linearly with the NN size. On the public SURROGATELIB benchmark, our method proves scalable, modular, and adaptable: it runs $120\times$ faster than an exact Big–M formulation on the largest test case; the NN sub-solver can be swapped from a log-barrier interior step to a projected-gradient routine with no code changes; and swapping the MLP for an LSTM backbone still completes the full optimisation in 47s without any bespoke adaptation.

Extended version — https://arxiv.org/abs/2511.09186

1 Introduction

Intelligent decision systems increasingly integrate neural networks into decision-making and optimization pipelines. Recent advances utilize large language models for problem modeling (Pan et al. 2025), reinforcement learning for multi-objective tasks (Xie et al. 2025) and learning-based branching policies (Zhang et al. 2025a) to enhance traditional solvers (Cappart et al. 2021; Joshi 2020; Bengio, Lodi, and Prouvost 2021). Neural networks provide data-driven surrogates that capture complex, nonlinear dependencies, and their outputs can be enforced as constraints inside mixed-integer programming (MIP) models while preserving the latter's combinatorial guarantees (Fig. 1). This fusion marries the predictive accuracy of data-driven models with the exactness of discrete optimisation: the NN supplies rich, sample-efficient knowledge of latent physics or economics (Bertsimas et al. 2016; Misaghian et al. 2022; Jalving et al. 2023),

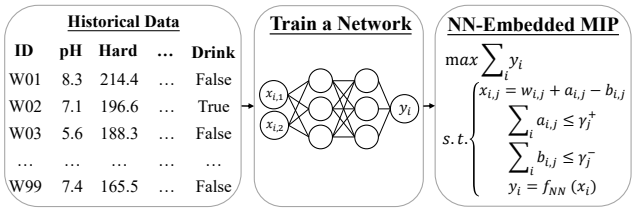

Figure 1: Workflow of the NN-Embedded MIP: historical samples are adjusted under budget constraints, evaluated by a neural classifier, and optimised via a mixed-integer solver.

while the MIP layer delivers globally optimal and certifiably feasible decisions. As a result, practitioners can exploit learned structure without sacrificing safety guarantees or the ability to produce worst-case certificates—an advantage unattainable by black-box heuristics (Droste, Jansen, and Wegener 2006) alone.

Prior research (Ceccon et al. 2022; Bergman et al. 2022; Lombardi, Milano, and Bartolini 2017; Maragno et al. 2025) has taken important steps toward incorporating neural networks as mathematical constraints. One line of work focuses on *exact, monolithic integration*: encoding the network's operations (e.g., ReLU activations) directly into a MIP or satisfiability formulation. Tjeng, Xiao, and Tedrake (2017) formulate the verification of piecewise-linear neural networks as a mixed-integer linear program, enabling exact adversarial robustness checks with orders-of-magnitude speedups over earlier methods. Building on such encodings, Anderson et al. (2020) developed stronger MILP formulations for ReLU-based networks, which further tighten the linear relaxations and improve solving efficiency. In parallel, practical frameworks have emerged to automate these integrations: for example, PySCIPOpt-ML (Turner et al. 2023) provides an interface to embed trained machine learning models (including neural nets) as constraints within the open-source solver SCIP (Bestuzheva et al. 2023). These advances demonstrate the viability of treating a neural network as a combinatorial constraint.

However, existing approaches to neural network integration face fundamental limitations. A prevailing tactic is to *fully linearise* every activation, yielding a single, rigid MILP. Any change to the network then forces a complete reformu-

*Corresponding author.

lation; the encoding relies on large Big–M bounds (Grimstad and Andersson 2019; Huchette et al. 2023; Badilla et al. 2023) that weaken relaxations (Griva, Nash, and Sofer 2008; Camm, Raturi, and Tsubakitani 1990); and runtime soars with depth and width. In our largest test, translating a 64×512 ReLU model for 500 samples with PySCIPOpt-ML inflates the problem from 500 variables to 5,900,000 variables and 7,800,000 constraints, yet SCIP still times out after two hours. Even the tighter formulations of Anderson et al. (2020) remain highly size-sensitive. These limitations motivate a framework that *decouples* the neural and combinatorial parts, achieving *scalable*, *modular*, and *adaptable* integration.

We propose a *dual-decomposition* scheme that duplicates the integer vector x with a continuous copy u, relaxes the coupling equality $u = x$ via an augmented-Lagrange term, and then alternates between two native solves: (i) a branch-and-cut MIP on x and (ii) a constrained first-order update on u. After each pair of solves the dual multipliers are updated, shrinking the mismatch until the two blocks agree; because neither block is ever reformulated into the language of the other, the method keeps the full power of specialised MILP engines and modern NN optimisers while sidestepping the Big-M explosion. This method yields three validated advantages. *Scalability*: On the X-Large case (6×128 ReLU, 400 samples) our dual-decomposition solver finishes in 24.5s, whereas PySCIPOpt-ML needs 600s or more—a $25\times$ speed-up obtained. *Modularity*: replacing the NN subsolver (projected-gradient $\leftrightarrow$ log-barrier) leaves the MIP untouched and changes total time by less than a factor of two. *Adaptability*: swapping the backbone MLP for a CNN or LSTM requires no reformulation and still converges in under 47s with the same objective value. Section 6 details these results, confirming that dual decomposition can combine learning surrogates with discrete optimisation at scale.

Our main contributions are as follows:

- Algorithm. We introduce a dual-decomposition framework that couples a mixed-integer subproblem with a neural subproblem through an augmented-Lagrange split, avoiding any Big–M reformulation.
- Theory. We prove the alternating scheme converges to a stationary point and linear per-iteration scalability, guaranteeing scalability by design.
- Experiments. Extensive experiments confirm that the solver is *scalable* (up to $120\times$ faster than monolithic baselines), *modular* (sub-solvers can be swapped without code changes), and *adaptable* (CNN and LSTM can be solved within tens of seconds on our benchmark when converged).

2 Preliminaries

Mixed Integer Linear Programming (MILP). Given a set of decision variables $\boldsymbol{x} \in \mathbb{R}^n$, the MILP problem is as follow:

$$\begin{aligned} \min \quad & \boldsymbol{c}^\top \boldsymbol{x}, \\ \text{s.t.} \quad & \boldsymbol{A}\boldsymbol{x} \geq \boldsymbol{b}, \quad \boldsymbol{l} \leq \boldsymbol{x} \leq \boldsymbol{r}, \\ & \boldsymbol{x} \in \{0,1\}^p \times \mathbb{Z}^q \times \mathbb{R}^{n-p-q}, \end{aligned} \tag{1}$$

where $\boldsymbol{c} \in \mathbb{R}^n$ is the objective coefficients, $\boldsymbol{A} \in \mathbb{R}^{m\times n}$ is the constraint coefficient matrix, $\boldsymbol{b} \in \mathbb{R}^m$ is the right-hand side vector, $\boldsymbol{l}, \boldsymbol{r} \in \mathbb{R}^n$ are the variable bounds.

Augmented Lagrangian Method (ALM). We consider the equality-constrained problem as below:

$$\min_{x\in\mathbb{R}^n} \; f(x) \;\; \text{s.t.} \;\; c_i(x) = 0, \quad i \in \mathcal{E}, \tag{2}$$

where $\mathcal{E}$ indexes the set of equality constraints. A straightforward quadratic-penalty approach enforces the constraints by minimising the following equation:

$$\Phi_k^{\text{pen}}(x) \;=\; f(x) + \mu_k \sum_{i\in\mathcal{E}} c_i(x)^2, \tag{3}$$

and gradually driving $\mu_k \to \infty$; however, large penalties typically induce severe ill-conditioning (Bertsekas 2014).

To mitigate this, the augmented Lagrangian augments the objective with a linear multiplier term as:

$$\Phi_k(x) \;=\; f(x) + \tfrac{\mu_k}{2} \sum_{i\in\mathcal{E}} c_i(x)^2 + \sum_{i\in\mathcal{E}} \lambda_i \, c_i(x), \tag{4}$$

where $\lambda \in \mathbb{R}^{|\mathcal{E}|}$ estimates the true Lagrange multipliers. At outer iteration k, one solves:

$$x^{(k)} \;=\; \arg\min_x \Phi_k(x),$$

then updates the multipliers by:

$$\lambda_i^{(k+1)} \;=\; \lambda_i^{(k)} + \mu_k \, c_i\big(x^{(k)}\big), \qquad i \in \mathcal{E}, \tag{5}$$

and increases μ_k (e.g. $\mu_{k+1} = \beta\mu_k$ with $\beta > 1$) only when constraint violations stagnate. Because the linear term offsets part of the quadratic penalty, μ_k can stay moderate, leading to better numerical conditioning and stronger convergence guarantees. In practice, safeguards that bound λ are often used to prevent numerical overflow (Hestenes 1969).

3 Problem Formulation and Motivation

We study mixed-integer programmes (MIPs) that *embed* a pretrained neural network (NN) directly into their objective and constraints. Let $x \in \mathbb{Z}^p$ denote the discrete decisions and $y := f_\theta(x) \in \mathbb{R}^q$ the NN output; f_θ is differentiable with fixed parameters θ. The general problem is

$$\begin{aligned} \min_{x\in\mathbb{Z}^p} \quad & c^\top x + d^\top y \\ \text{s.t.} \quad & A\begin{bmatrix} x \\ y \end{bmatrix} \leq b, \\ & y = f_\theta(x). \end{aligned} \tag{6}$$

with $c \in \mathbb{R}^p$ and $d \in \mathbb{R}^q$ weighting the discrete cost and the NN-dependent cost, $A \in \mathbb{R}^{m\times(p+q)}$ and $b \in \mathbb{R}^m$ capturing all affine limits that jointly constrain the decisions and the NN prediction. Because the mapping $x \mapsto y = f_\theta(x)$ is nonlinear, problem (6) inherits both the combinatorial hardness of MIPs and the nonlinearity of deep networks, making it particularly challenging to solve.

Limitations of Exact Linearization. A common strategy for incorporating a pretrained NN into a MIP is to replace each nonlinear operation by piecewise-linear surrogates—e.g. Big-M formulations (Cococcioni and Fiaschi 2021) or SOS1 constraints (Fischer and Pfetsch 2018). Although exact, these encodings introduce $\mathcal{O}(Lh)$ additional binaries and constraints for an L-layer, h-unit MLP, which can blow up the search tree even for moderate network sizes. Moreover, linearization routines support only basic primitives, whereas modern architectures-convolutional filters, recurrent gates, attention blocks and layer-normalisation-have no off-the-shelf encodings. Adapting each novel layer requires bespoke linearization, increasing modelling complexity and hindering extensibility.

Rationale for Decomposition. By contrast, treating the NN as a black-box oracle avoids this overhead but sacrifices explicit feasibility guarantees. To overcome the computational challenges associated with monolithic linearization approaches, we propose employing *augmented Lagrangian decomposition*. This technique partitions the original problem (6) into simpler subproblems, each handled by specialized solvers, while iteratively maintaining solution consistency through dual updates. This decomposition approach provides significant computational advantages, improved scalability, and robust theoretical convergence properties.

4 Our Decomposition-Based Method

We introduce an auxiliary continuous variable $u \in \mathbb{R}^p$ to duplicate the integer variable x, resulting in an equivalent but more tractable formulation featuring explicit coupling constraints:

$$\begin{aligned} \min_{x,u} \quad & c^\top x + d^\top f_\theta(u) \\ \text{s.t.} \quad & A_{\text{MIP}}\, x \;\le\; b_{\text{MIP}}, \\ & A_{\text{NN}} \begin{bmatrix} u \\ f_\theta(u) \end{bmatrix} \;\le\; b_{\text{NN}}, \\ & u \;=\; x, \qquad x \in \mathbb{Z}^p. \end{aligned} \tag{7}$$

Here $A_{\text{MIP}} \in \mathbb{R}^{m_1 \times p}$, $b_{\text{MIP}} \in \mathbb{R}^{m_1}$ store the "legacy" linear limits that involve only the integer vector x. All remaining affine constraints that depend on the NN output are written as $A_{\text{NN}} \in \mathbb{R}^{m_2 \times (p+q)}$ and $b_{\text{NN}} \in \mathbb{R}^{m_2}$ acting on the stacked vector $\big[u;\, f_\theta(u)\big]$. Across iterations u stays continuous—enabling inexpensive first-order updates of the NN block—while the coupling equation $u = x$ is enforced at convergence to recover a valid integer solution.

High-Level Decomposition Approach

The key theoretical challenge we address is the nonconvex equality constraint $u = x$. We handle this complexity by introducing dual multipliers (Lagrange multipliers) $\lambda \in \mathbb{R}^p$ and a quadratic penalty term controlled by parameter$\rho > 0$. The resulting augmented Lagrangian function is given by:

$$\begin{aligned} \mathcal{L}_\rho(x, u, \lambda) \;=\; & c^\top x + d^\top f_\theta(u) \\ & + \lambda^\top (u - x) + \frac{\rho}{2} \|u - x\|_2^2. \end{aligned} \tag{8}$$

Algorithm 1: Dual Decomposition for NN-Embedded MIP

Require: Initialization $u^{(0)}, \lambda^{(0)}, \rho > 0$, tolerance ε, maximum iterations K

1: **for** $k = 1, \dots, K$ **do**
2: Solve MIP subproblem for $x^{(k)}$ to update discrete decisions.
3: Solve NN subproblem (PGD or Log-Barrier) for $u^{(k)}$ to refine continuous inputs.
4: Update dual multiplier: $\lambda^{(k)} \leftarrow \lambda^{(k-1)} + \rho(u^{(k)} - x^{(k)})$.
5: Check convergence: if $\|u^{(k)} - x^{(k)}\|_\infty < \varepsilon$, terminate.
6: Adjust penalty parameter ρ adaptively based on residuals.
7: **end for**
8: **return** final solution pair $(x^\star, u^\star)$.

The term $\lambda^\top (u - x)$ is the *classical* Lagrange coupling: in the augmented-Lagrangian saddle-point picture (for the continuous relaxation), it drives the equality $u = x$ by adjusting the dual multipliers, while supplying first-order information to each subproblem. The quadratic add-on $\frac{\rho}{2}\|u - x\|_2^2$ augments the Lagrangian, injecting strong convexity that damps oscillations and markedly accelerates the alternating updates observed in practice. Following standard augmented-Lagrangian heuristics (Larsson and Yuan 2004), we initialize ρ conservatively and increase it adaptively whenever the primal residual $\|u - x\|_2$ ceases to decrease, thereby achieving both robustness and efficiency. Figure 2 illustrates our proposed dual decomposition workflow, highlighting the iterative, coordinated solving of the decomposed subproblems. Algorithm 1 provides the pseudocode.

Detailed Modular Subproblem Decomposition

We then decompose the augmented Lagrangian into two tractable subproblems solved alternately and coordinated by iterative updates of the dual multipliers:

MIP Subproblem. Given $(u^{(k-1)}, \lambda^{(k-1)})$, the MIP subproblem is defined as:

$$\begin{aligned} x^{(k)} \;=\; & \arg\min_{x \in \mathbb{Z}^p} \Big(c^\top x - \lambda^{(k-1)\top} x + \frac{\rho}{2} \|u^{(k-1)} - x\|_2^2 \Big) \\ & \text{s.t.} \quad A_{\text{MIP}}\, x \;\le\; b_{\text{MIP}}. \end{aligned} \tag{9}$$

This formulation retains the linear constraints $A_{\text{MIP}}\, x \;\le\; b_{\text{MIP}}$ but adjusts the objective to include the penalty term $\|u^{(k-1)} - x\|_2^2$, thereby encouraging proximity between x and the current NN solution $u^{(k-1)}$. It includes the linear term $-\lambda^{(k-1)\top} x$, the Lagrange multiplier contribution originating from the coupling constraint $u = x$ in the augmented Lagrangian. The companion constant $\lambda^{(k-1)\top} u^{(k-1)}$ is omitted because it is independent of x and thus irrelevant for the minimisation.

Modern branch-and-bound or cutting-plane solvers, including advanced column generation techniques for large-scale instances (Hu et al. 2025), can efficiently handle this

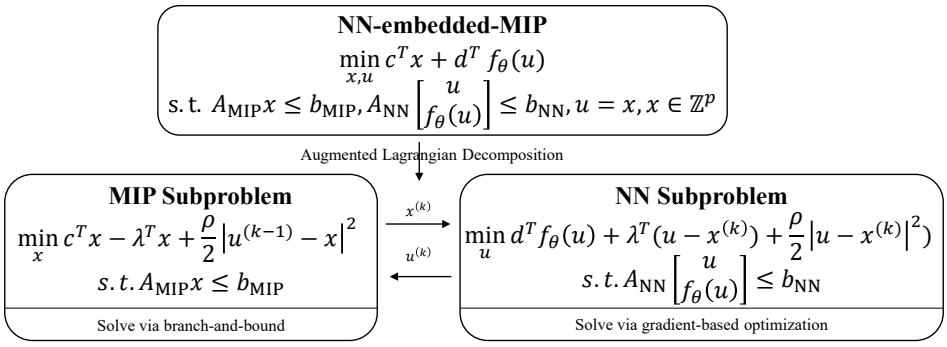

Figure 2: Illustration of the proposed NN-embedded MIP solution framework using augmented Lagrangian decomposition, emphasizing iterative primal-dual coordination.

subproblem due to its convexified structure (Mistry et al. 2021). Furthermore, since the subproblem is solved iteratively with updated parameters, dynamic reoptimization strategies (Zhang et al. 2025b) could naturally be leveraged to further accelerate convergence.

NN Subproblem. Given the discrete solution $x^{(k)}$ and multiplier $\lambda^{(k-1)}$, the continuous variables are optimized through solving:

$$\begin{aligned} u^{(k)} \;=\; \arg\min_{u\in\mathbb{R}^p} \Big\{ & \; d^\top f_\theta(u) + \lambda^{(k-1)\top}\big(u - x^{(k)}\big) \\ & + \frac{\rho}{2}\,\|u - x^{(k)}\|_2^2 \Big\} \\ \text{s.t.} \quad & A_{\text{NN}} \begin{bmatrix} u \\ f_\theta(u) \end{bmatrix} \;\le\; b_{\text{NN}}. \end{aligned} \tag{10}$$

Fixing the discrete iterate $x^{(k)}$ isolates the continuous decision u, letting us recalibrate the neural input without perturbing the combinatorial block. The first term $d^\top f_\theta(u)$ transmits the original cost coefficients to the NN, while the dual inner product $\lambda^{(k-1)\top}(u - x^{(k)})$ conveys the mismatch detected in the preceding coordination step. The proximal penalty $\frac{\rho}{2}\|u - x^{(k)}\|_2^2$ tempers aggressive updates, yielding a well-conditioned landscape. Crucially, the stacked linear constraint $A_{\text{NN}}\,[u\,;\; f_\theta(u)] \le b_{\text{NN}}$ preserves every safety or performance limit originally imposed on the network output, ensuring that each neural adjustment remains admissible before the next dual synchronization.

Solving the NN Subproblem with Constraints

The NN subproblem is non-trivial due to the nonlinear nature of $f_\theta(u)$ and the constraint $A_{\text{NN}}[\,u\,;\; f_\theta(u)] \;\le\; b_{\text{NN}}$, which induces the feasible set

$$\mathcal{C} \;:=\; \Big\{ u \in \mathbb{R}^p \;\Big|\; A_{\text{NN}} \begin{bmatrix} u \\ f_\theta(u) \end{bmatrix} \le b_{\text{NN}} \Big\}.$$

Projected Gradient Descent (PGD). When projection onto $\mathcal{C}$ is affordable—e.g., if A_{NN} encodes simple box or monotone limits—we perform

1. a gradient step on the smooth part of the objective, then
2. an orthogonal projection back onto $\mathcal{C}$.

Let $J_{f_\theta}(u)$ denote the Jacobian of f_θ at u. The gradient of the augmented Lagrangian with fixed $(x^{(k)}, \lambda^{(k-1)})$ reads

$$\nabla_u \mathcal{L}_\rho(u) = J_{f_\theta}(u)^\top d + \lambda^{(k-1)} + \rho\big(u - x^{(k)}\big).$$

With stepsize $\eta > 0$ we update

$$u \leftarrow \text{Proj}_{\mathcal{C}}\big(u - \eta \nabla_u \mathcal{L}_\rho(u)\big),$$

where $\text{Proj}_{\mathcal{C}}$ is the Euclidean projection (Liu and Ye 2009) onto $\mathcal{C}$. The proximal term $\rho\|u - x^{(k)}\|_2^2$ ensures the landscape is well-conditioned, and PGD typically converges in a handful of iterations (Haji and Abdulazeez 2021).

Log-Barrier Interior Method. If projection is itself expensive (e.g. $\mathcal{C}$ is defined by many affine cuts on f_θ), we instead absorb the constraints via a logarithmic barrier:

$$\begin{aligned} \min_{u\in\mathbb{R}^p} \Big\{ & \; d^\top f_\theta(u) + \lambda^{(k-1)\top}\big(u - x^{(k)}\big) + \frac{\rho}{2}\,\|u - x^{(k)}\|_2^2 \\ & - \mu \sum_{j=1}^{m} \log\Big(b_{\text{NN},j} - a_j^\top \begin{bmatrix} u \\ f_\theta(u) \end{bmatrix}\Big) \Big\} \end{aligned}$$

where $a_j^\top$ is the j-th row of A_{NN} and $\mu > 0$ is the barrier parameter. Starting from a strictly feasible point, we solve the above with a second-order method and gradually decrease μ ($\mu \leftarrow 0.1\mu$) until the dual feasibility tolerance is reached. This interior-point strategy dispenses with explicit projections yet retains strict feasibility along the entire Newton path (Toussaint 2017); it is especially beneficial when $\mathcal{C}$ is described by many coupled linear cuts but occupies a relatively low-volume region, where projection costs dominate.

Dual Updates and Penalty Parameter Adjustments

The dual multipliers are iteratively updated to maintain primal-dual feasibility:

$$\lambda^{(k)} \leftarrow \lambda^{(k-1)} + \rho(u^{(k)} - x^{(k)}). \tag{11}$$

The penalty parameter ρ is dynamically adjusted to effectively balance feasibility and dual optimization stability:

- **Increasing** ρ strengthens primal feasibility enforcement when constraint violations persist; after a finite number of increases we keep ρ fixed.

5 Theoretical Analysis

In this section, we analyze (i) convergence of an idealized variant of Algorithm 1 under smooth-convex assumptions, and (ii) per-iteration scalability with neural network size.

Convergence Analysis

We establish convergence guarantees for Algorithm 1 under Assumption 1.

Assumption 1. *1. $\mathcal{X} := \{x \in \mathbb{Z}^p \mid A_{MIP}x \leq b_{MIP}\}$ is non-empty and bounded;*
2. The map $g(u) := d^\top f_\theta(u)$ is convex and has L-Lipschitz gradient (L–smooth);
3. $\mathcal{C} := \{u \mid A_{NN}[u; f_\theta(u)] \leq b_{NN}\}$ is non-empty and satisfies Slater's condition;
4. A penalty $\rho > L$ is chosen once (or increased finitely many times and then fixed).

Remark 1 (Theory applicability)**.** *Theorem 5.1 assumes exact subproblem solves and a smooth convex $g(u)$. In practice we use inexact NN solves (fixed optimizer steps) and ReLU subgradients; these settings are not covered by Assumption 1.*

Theorem 5.1 (Convergence (primal feasibility))**.** *Under Assumption 1, the sequence $\{(x^{(k)}, u^{(k)}, \lambda^{(k)})\}$ generated by Algorithm 1 is bounded, and the primal residual $r^{(k)} := u^{(k)} - x^{(k)}$ satisfies*

$$\sum_{k=1}^{\infty} \|r^{(k)}\|_2^2 < \infty, \qquad \min_{t \leq k} \|r^{(t)}\|_2^2 = \mathcal{O}\big(\tfrac{1}{k}\big). \tag{12}$$

In particular, $\|u^{(k)} - x^{(k)}\|_2 \to 0$.

Proof. Define the Lyapunov (merit) sequence

$$\mathcal{M}_\rho^{(k)} := \mathcal{L}_\rho\big(x^{(k)}, u^{(k)}, \lambda^{(k-1)}\big) + \frac{1}{2\rho}\|\lambda^{(k-1)}\|_2^2.$$

The argument proceeds in three steps.

(i) Descent in the u–block. Because g has L-Lipschitz gradient and $\rho > L$, the u-subproblem objective $u \mapsto g(u) + \lambda^{(k-1)\top}(u - x^{(k)}) + \frac{\rho}{2}\|u - x^{(k)}\|_2^2$ is $(\rho - L)$-strongly convex in u. Solving the NN subproblem exactly therefore yields a strict decrease in $\mathcal{L}_\rho$ (at fixed $x^{(k)}$ and $\lambda^{(k-1)}$), with a quadratic decrease controlled by $\rho - L$.

(ii) Descent in the x–block. The discrete set $\mathcal{X}$ is finite and each MIP subproblem is solved to global optimality, hence the update in x does not increase $\mathcal{L}_\rho$ (at fixed $u^{(k-1)}$ and $\lambda^{(k-1)}$).

(iii) Dual update and telescoping. With $\lambda^{(k)} = \lambda^{(k-1)} + \rho\big(u^{(k)} - x^{(k)}\big)$, a standard calculation (see extended version for details) gives the one-step decrease

$$\mathcal{M}_\rho^{(k+1)} \;\leq\; \mathcal{M}_\rho^{(k)} - \frac{\rho - L}{2}\|u^{(k)} - x^{(k)}\|_2^2.$$

Since $\mathcal{M}_\rho^{(k)}$ is bounded below, summing the inequality over k yields $\sum_k \|u^{(k)} - x^{(k)}\|_2^2 < \infty$. The rate $\min_{t \leq k} \|u^{(t)} - x^{(t)}\|_2^2 = \mathcal{O}(1/k)$ follows immediately from this summability. □

Scalability with Neural Network Size

Next, we analyze how the complexity of our method scales with the neural network architecture.

Theorem 5.2 (Linear Scalability in Network Size)**.** *Let P denote the number of parameters in the neural network f_θ. Then each iteration of Algorithm 1 has overall complexity*

$$\mathcal{O}\big(T_{MIP} + P\big), \tag{13}$$

where T_{MIP} is the complexity of solving the MIP subproblem, independent of P. Consequently, for fixed MIP size, the runtime per iteration grows linearly *with the network size.*

Proof. (i) The MIP subproblem (9) depends only on x, λ, and $u^{(k-1)}$; its size does not increase with P. Thus, T_{MIP} is independent of the neural architecture. (ii) The NN subproblem (10) requires evaluating $f_\theta(u)$ and its Jacobian $J_{f_\theta}(u)$. For standard feedforward networks, forward and backward passes both require $\mathcal{O}(P)$ time (Goodfellow et al. 2016). (iii) The projection or barrier operations involve only affine constraints A_{NN}, whose size is fixed by the problem specification and independent of P. Therefore, the per-iteration complexity is $\mathcal{O}(T_{\text{MIP}} + P)$, implying linear scaling with the network size. □

In a Big-M linearisation, every hidden unit contributes a constant number of *additional variables and constraints*, so the model dimension itself grows roughly linearly with network depth and width. However, the resulting MILP remains NP-hard; empirical and worst-case studies show that solver time rises super-linearly, often exponentially, with that dimension increase (Bixby 1992). By leaving the network outside the MILP, our decomposition sidesteps this blow-up and retains per-iteration linear scaling.

6 Experimental Evaluations

Overview of Experiments. We perform five complementary studies to thoroughly evaluate the proposed dual-decomposition framework: **(E1)** a head-to-head comparison with Big-M linearisation; **(E2)** a stress test that scales network depth and width; **(E3)** an ablation that removes the dual-update loop; **(E4)** a modularity study that swaps heterogeneous network architectures without changing solver logic; and **(E5)** a subsolver comparison between projected-gradient descent and a log-barrier interior method. Together, these experiments probe solution quality, runtime efficiency, robustness, architecture agnosticism, and the impact of continuous-block solver choice.

Experimental Setup

Implementation Details. All experiments are implemented in `Python`: neural classifiers are trained with PyTorch (Paszke et al. 2019), while MIP subproblems are solved via PySCIPOpt 8.0.4 (Maher et al. 2016). For the linearisation baseline, we use PySCIPOpt-ML to linearise the trained classifier via its Big-M add_predictor_constr interface. For dual decomposition (DD), we adopt an ADMM-style scheme with $\rho = 10.0$, step size $\eta = 0.01$, and convergence tolerance $\varepsilon = 10^{-4}$. Each augmented-Lagrange

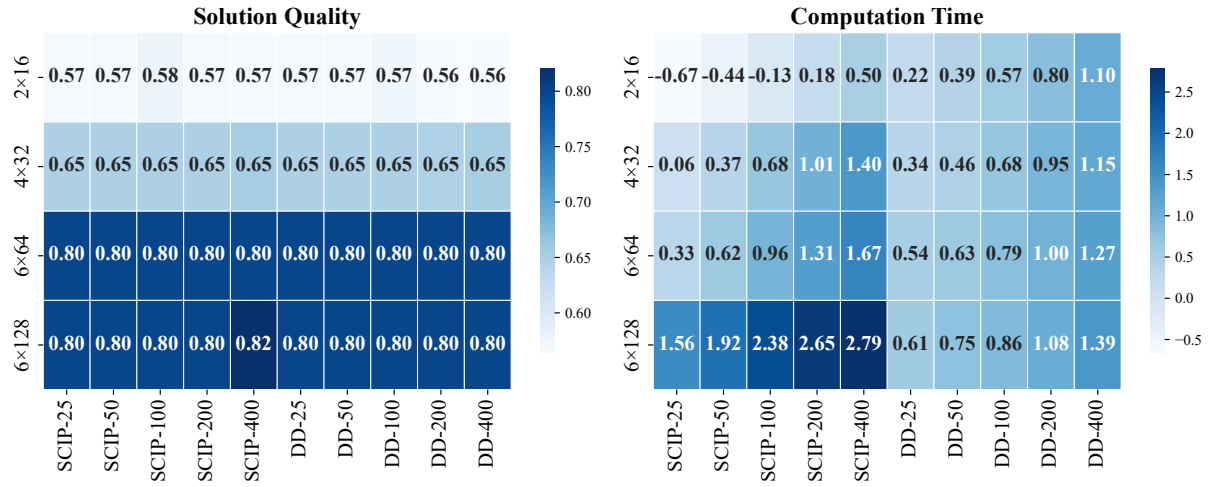

Figure 3: (E1) Comparison with Linearisation-based Methods. Solution quality (left) and computation time (right) across network sizes and problem sizes. Colours denote potability rate and $\log_{10}$ wall-clock time, respectively.

iteration first solves the MIP subproblem, then refines the neural block—performing 25 Adam (Kingma and Ba 2014) steps in the PGD variant or an L-BFGS-B (Zhu et al. 1997) update in the log-barrier variant. SCIP-ML is run with a 2h time limit for scalability stress. Unless otherwise stated, we repeat each configuration over five random seeds $\{42, 123, 456, 789, 1024\}$ and report averages.

Dataset. **(1) Water Potability**. Used in Experiments E1-E4. The dataset offers nine water-quality features with binary drinkability labels. From the undrinkable subset we draw instances of different size n. Each optimisation task selects feature adjustments—subject to per-feature budgets of ± 2.0—to maximise the number of samples classified as potable. **(2) Tree Planting** (Wood et al. 2023). Employed exclusively in Experiment E5 for the subsolver comparison. Four species placed on an $n_{\text{grid}} \times n_{\text{grid}}$ plot ($n_{\text{grid}} = 6$). Seven site features feed neural survival models.

(E1) Comparing Linearisation Methods

We benchmark DD against the Big-M linearisation in PySCIPOpt-ML on four configurations—*Small* (2×16), *Medium* (4×32), *Large* (6×32), and *X-Large* (6×128)—and five instance sizes.

Solution Quality. DD aligns with the monolithic baseline across all configurations (Fig. 3, left), evidencing virtually no performance loss from decomposition on this task. **Computation Time.** SCIP-ML runtimes exceed 10^2 s as depth or n grows, whereas DD stays near 10 s even on the hardest cases, delivering 10–120× speed-ups (Fig. 3, right).

(E2) Scalability Stress Test Across Network Size

We fix the sample size at $n = 100$ and sweep six increasingly deep MLPs[1], repeating each configuration over five random seeds. For every run we record (i) total wall-clock time, and (ii) average time per iteration.

[1] $2\times16, 4\times32, 8\times64, 16\times128, 32\times256, 64\times512$ hidden units.

Network	Time (s)	Time/iter (s)
2×16	7.7	0.041
4×32	9.7	0.058
8×64	10.3	0.108
16×128	16.5	0.296
32×256	37.2	0.835
64×512	113.6	1.846

Table 1: (E2) Scalability stress test at fixed $n = 100$. Total wall-clock time and average time per outer iteration for increasingly large NNs ($d \times w$).

Scalability. Total solve time grows with depth but remains under 120s even for the 64×512 model (Tab. 1). **Efficiency.** Per-iteration cost increases sub-linearly with network size (Tab. 1), implying that the extra overhead stems chiefly from neural evaluation, not the optimisation loop. For reference, PySCIPOpt-ML fails beyond the 16×128 model: deeper nets either exceed memory limits or time out at 300s. DD therefore avoids the prohibitive linearisation burden through its decomposition strategy.

(E3) Ablation Study: Impact of Dual Coordination

To quantify the contribution of dual coordination, we construct a degenerate variant—*single-step gradient* (SSG)—that removes the dual update and the augmented term. SSG keeps the same outer-loop structure: at each iteration it performs a single projected-gradient update on u, rounds to an integer x, and repeats until the stopping criterion or the iteration cap is met. The design is inspired by the direct gradient-descent solver of DiffILO (Geng et al. 2025), which addresses NN-embedded MIPs without dual variables, and the complete SSG pseudocode is given in extended version.

- **Solution quality.** SSG attains erratic quality—0.45, 0.50, 0.53, and 0.50 of the SCIP benchmark as n grows—whereas DD stays near unity, confirming that the

Problem Size	Quality		Iterations		Speedup	
	SSG	DD	SSG	DD	SSG	DD
$n = 20$	0.454	1.000	73	10	0.3	0.3
$n = 40$	0.498	1.000	80	12	2.7	2.2
$n = 60$	0.531	1.000	100	42	29.6	66.7
$n = 80$	0.501	0.998	100	48	20.9	111.7

Table 2: (E3) Ablation results. SSG (no dual coordination) versus full DD. Performance comparison relative to SCIP baseline (quality and speedup). Higher quality/speedup is better; fewer iterations is better.

decomposition preserves optimality.

- **Convergence.** SSG requires $73-100$ iterations (avg. ≈ 88), often stalling at the cap; DD converges in an average of 10, 12, 42, and 48 iterations for $n = 20, 40, 60, 80$, underscoring the stabilising effect of the dual updates.
- **Runtime.** SSG is slower than SCIP for $n = 20$ ($0.3\times$), modestly faster for $n = 40$ ($2.7\times$), and reaches $20-30\times$ speed-ups at larger scales. DD starts on par with SSG at $n = 40$ ($2.2\times$), then widens the gap to $66.7\times$ and $111.7\times$ for $n = 60$ and 80, respectively.

Dual coordination is indispensable: removing it degrades numerical robustness, inflates iteration counts by a large margin, and erodes the runtime advantage as n increases. Alternating dual updates are therefore critical for both rapid convergence and high-quality solutions.

(E4) Adaptability Across Neural Architectures

We run the *unchanged* DD solver on four disparate predictors: a shallow fully connected network (FC), a lightweight CONV1D, a deep RESNET, and a sequential LSTM. By contrast, linearisation frameworks such as PySCIPOpt-ML only support simple feed-forward layers and cannot encode CNN or LSTM backbones at all.

- Predictive fidelity. DD preserves the original accuracy: 0.694 (FC), 0.743 (CONV1D), 0.791 (LSTM), and 0.696 (RESNET), demonstrating zero distortion of the models.
- Convergence across backbones. Within the 50-iteration limit DD satisfies $\|u - x\|_2 < 10^{-4}$ in 0.83 of CONV1D runs, 0.75 of FC runs, 0.83 of RESNET runs, and 0.50 of LSTM runs—architectures that existing linearisation tool-chains cannot even represent.

(E5) Modular Subsolver Test: PGD *vs.* Log-Barrier

We verify *modularity* by fixing all outer-loop parameters ($\rho = 10$, 50 iterations, stopping criterion) and only swapping the NN subsolver: **DD-PGD** (25 Adam steps with projection) versus **DD-Barrier** (log-barrier objective solved by L-BFGS-B with damping and $\mu \leftarrow 0.1\mu$). The monolithic linearisation baseline (SCIP-ML) is included for reference.

Tab. 4 shows three runtime regimes: SCIP-ML grows from 8.5–860 s; **DD-Barrier** runs in 24.8/63.1/110 s ; **DD-PGD** runs in 0.4/1.4/6.5 s. This isolated subsolver swap—without any change to the MIP block, dual updates or data handling—demonstrates true modularity: users can interchange NN optimisation routines to match constraint complexity (e.g. cheap projection vs. dense polyhedral limits) while retaining all theoretical and empirical benefits of the dual-decomposition framework.

Architecture	Accuracy	Convergence Rate
CONV1D	0.743	0.83
FC	0.694	0.75
LSTM	0.791	0.50
RESNET	0.696	0.83

Table 3: (E4) Adaptability across architectures. Left: standalone test accuracy. Right: fraction of DD runs converging within the 50-iteration cap.

Network Size	SCIP (s)	DD-Barrier (s)	DD-PGD (s)
8×32	8.5	24.8	0.4
16×64	395.6	63.1	1.4
24×128	860.3	111.0	6.5

Table 4: (E5) Modular subsolver test. Wall–clock runtime (log scale in the original plot) for DD-PGD, DD-Barrier, and the linearisation baseline on the *Small*, *Medium*, and *Large* MLPs.

Summary of Findings

Across five studies we establish three main results. **(i) Scalability**: DD maintains near-optimal quality while delivering $10\times$–$120\times$ speed-ups over SCIP-ML (E1, E2). **(ii) Necessity of dual updates**: removing them degrades quality by up to 55% and inflates iteration counts by up to $10\times$ (E3). **(iii) Modularity & Adaptability**: DD functions as a plug-and-play layer—handling CNN, LSTM backbones and interchangeable PGD/Barrier sub-solvers without code changes—yet still outperforms linearisation baselines by at least an order of magnitude (E4, E5). These empirical results corroborate the theoretical claims of Section 1 and demonstrate that dual decomposition offers a practical path to embedding complex neural surrogates in large-scale mixed-integer optimisation.

7 Conclusion

We proposed a scalable and modular dual decomposition framework for integrating neural networks into mixed-integer optimization. Our method outperforms monolithic baselines in both runtime and flexibility across a range of problem sizes and neural architectures. Despite the advantages, several limitations remain and will be addressed in future work. (i) The convergence rate can be sensitive to hyperparameters such as the penalty ρ and step size, suggesting the need for adaptive or learned scheduling strategies. (ii) Our framework currently assumes access to gradient information from the neural network; extending to non-differentiable predictors remains an open challenge.

Acknowledgments

The research is partially supported by Quantum Science and Technology-National Science and Technology Major Project (QNMP) 2021ZD0302900 and China National Natural Science Foundation with No. 62132018 , 62231015, "Pioneer" and "Leading Goose" R&D Program of Zhejiang", 2023C01029, and 2023C01143, and the USTC Kunpeng-Ascend Scientific and Educational Innovation Excellence Center.

References

Anderson, R.; Huchette, J.; Ma, W.; Tjandraatmadja, C.; and Vielma, J. P. 2020. Strong mixed-integer programming formulations for trained neural networks. *Mathematical Programming*, 183(1): 3–39.

Badilla, F.; Goycoolea, M.; Muñoz, G.; and Serra, T. 2023. Computational tradeoffs of optimization-based bound tightening in ReLU networks. *arXiv preprint arXiv:2312.16699*.

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

Bergman, D.; Huang, T.; Brooks, P.; Lodi, A.; and Raghunathan, A. U. 2022. JANOS: an integrated predictive and prescriptive modeling framework. *INFORMS Journal on Computing*, 34(2): 807–816.

Bertsekas, D. P. 2014. *Constrained optimization and Lagrange multiplier methods*. Academic press.

Bertsimas, D.; O'Hair, A.; Relyea, S.; and Silberholz, J. 2016. An analytics approach to designing combination chemotherapy regimens for cancer. *Management Science*, 62(5): 1511–1531.

Bestuzheva, K.; Besançon, M.; Chen, W.-K.; Chmiela, A.; Donkiewicz, T.; Van Doornmalen, J.; Eifler, L.; Gaul, O.; Gamrath, G.; Gleixner, A.; et al. 2023. Enabling research through the SCIP optimization suite 8.0. *ACM Transactions on Mathematical Software*, 49(2): 1–21.

Bixby, R. E. 1992. Implementing the simplex method: The initial basis. *ORSA Journal on Computing*, 4(3): 267–284.

Camm, J. D.; Raturi, A. S.; and Tsubakitani, S. 1990. Cutting big M down to size. *Interfaces*, 20(5): 61–66.

Cappart, Q.; Moisan, T.; Rousseau, L.-M.; Prémont-Schwarz, I.; and Cire, A. A. 2021. Combining reinforcement learning and constraint programming for combinatorial optimization. In *Proceedings of the AAAI conference on artificial intelligence*, volume 35, 3677–3687.

Ceccon, F.; Jalving, J.; Haddad, J.; Thebelt, A.; Tsay, C.; Laird, C. D.; and Misener, R. 2022. OMLT: Optimization & machine learning toolkit. *Journal of Machine Learning Research*, 23(349): 1–8.

Cococcioni, M.; and Fiaschi, L. 2021. The Big-M method with the numerical infinite M. *Optimization Letters*, 15(7): 2455–2468.

Droste, S.; Jansen, T.; and Wegener, I. 2006. Upper and lower bounds for randomized search heuristics in black-box optimization. *Theory of computing systems*, 39(4): 525–544.

Fischer, T.; and Pfetsch, M. E. 2018. Branch-and-cut for linear programs with overlapping SOS1 constraints. *Mathematical Programming Computation*, 10(1): 33–68.

Geng, Z.; Wang, J.; Li, X.; Zhu, F.; Hao, J.; Li, B.; and Wu, F. 2025. Differentiable integer linear programming. In *The Thirteenth International Conference on Learning Representations*.

Goodfellow, I.; Bengio, Y.; Courville, A.; and Bengio, Y. 2016. *Deep learning*, volume 1. MIT press Cambridge.

Grimstad, B.; and Andersson, H. 2019. ReLU networks as surrogate models in mixed-integer linear programs. *Computers & Chemical Engineering*, 131: 106580.

Griva, I.; Nash, S. G.; and Sofer, A. 2008. *Linear and nonlinear optimization 2nd edition*. SIAM.

Haji, S. H.; and Abdulazeez, A. M. 2021. Comparison of optimization techniques based on gradient descent algorithm: A review. *PalArch's Journal of Archaeology of Egypt/Egyptology*, 18(4): 2715–2743.

Hestenes, M. R. 1969. Multiplier and gradient methods. *Journal of optimization theory and applications*, 4(5): 303–320.

Hu, Y.-X.; Wu, F.; Li, S.; Zhao, Y.; and Li, X.-Y. 2025. FFCG: Effective and Fast Family Column Generation for Solving Large-Scale Linear Program. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, 11238–11245.

Huchette, J.; Muñoz, G.; Serra, T.; and Tsay, C. 2023. When deep learning meets polyhedral theory: A survey. *arXiv preprint arXiv:2305.00241*.

Jalving, J.; Ghouse, J.; Cortes, N.; Gao, X.; Knueven, B.; Agi, D.; Martin, S.; Chen, X.; Guittet, D.; Tumbalam-Gooty, R.; et al. 2023. Beyond price taker: Conceptual design and optimization of integrated energy systems using machine learning market surrogates. *Applied Energy*, 351: 121767.

Joshi, A. V. 2020. *Machine Learning and Artificial Intelligence*. Cham: Springer International Publishing. ISBN 978-3-030-26622-6.

Kingma, D. P.; and Ba, J. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Larsson, T.; and Yuan, D. 2004. An augmented lagrangian algorithm for large scale multicommodity routing. *Computational Optimization and Applications*, 27(2): 187–215.

Liu, J.; and Ye, J. 2009. Efficient Euclidean projections in linear time. In *Proceedings of the 26th annual international conference on machine learning*, 657–664.

Lombardi, M.; Milano, M.; and Bartolini, A. 2017. Empirical decision model learning. *Artificial Intelligence*, 244: 343–367.

Maher, S.; Miltenberger, M.; Pedroso, J. P.; Rehfeldt, D.; Schwarz, R.; and Serrano, F. 2016. PySCIPOpt: Mathematical Programming in Python with the SCIP Optimization Suite. In *Mathematical Software – ICMS 2016*, 301–307. Springer International Publishing.

Maragno, D.; Wiberg, H.; Bertsimas, D.; Birbil, Ş. İ.; den Hertog, D.; and Fajemisin, A. O. 2025. Mixed-integer optimization with constraint learning. *Operations Research*, 73(2): 1011–1028.

Misaghian, M. S.; Tardioli, G.; Cabrera, A. G.; Salerno, I.; Flynn, D.; and Kerrigan, R. 2022. Assessment of carbon-aware flexibility measures from data centres using machine learning. *IEEE Transactions on Industry Applications*, 59(1): 70–80.

Mistry, M.; Letsios, D.; Krennrich, G.; Lee, R. M.; and Misener, R. 2021. Mixed-integer convex nonlinear optimization with gradient-boosted trees embedded. *INFORMS Journal on Computing*, 33(3): 1103–1119.

Pan, X.; Fang, J.; Wu, F.; Zhang, S.; Hu, Y.-X.; Li, S.; and Li, X.-Y. 2025. Guiding large language models in modeling optimization problems via question partitioning. In *Proceedings of the Thirty-Fourth International Joint Conference on Artificial Intelligence*, 2657–2665.

Paszke, A.; Gross, S.; Massa, F.; Lerer, A.; Bradbury, J.; Chanan, G.; Killeen, T.; Lin, Z.; Gimelshein, N.; Antiga, L.; et al. 2019. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32.

Tjeng, V.; Xiao, K.; and Tedrake, R. 2017. Evaluating robustness of neural networks with mixed integer programming. *arXiv preprint arXiv:1711.07356*.

Toussaint, M. 2017. A tutorial on Newton methods for constrained trajectory optimization and relations to SLAM, Gaussian Process smoothing, optimal control, and probabilistic inference. *Geometric and numerical foundations of movements*, 361–392.

Turner, M.; Chmiela, A.; Koch, T.; and Winkler, M. 2023. PySCIPOpt-ML: Embedding trained machine learning models into mixed-integer programs. *arXiv preprint arXiv:2312.08074*.

Wood, K. E.; Kobe, R. K.; Ibáñez, I.; and McCarthy-Neumann, S. 2023. Tree seedling functional traits mediate plant-soil feedback survival responses across a gradient of light availability. *Plos one*, 18(11): e0293906.

Xie, W.; Hu, Y.-X.; Xu, J.; Wu, F.; and Li, X.-Y. 2025. MURKA: Multi-Reward Reinforcement Learning with Knowledge Alignment for Optimization Tasks. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, volume 39.

Zhang, S.; Zeng, S.; Li, S.; Wu, F.; and Li, X.-Y. 2025a. Learning to Select Nodes in Branch and Bound with Sufficient Tree Representation. In *The Thirteenth International Conference on Learning Representations*.

Zhang, S.; Zeng, S.; Li, S.; Wu, F.; Tang, S.; and Li, X.-Y. 2025b. Don't Restart, Just Reuse: Reoptimizing MILPs with Dynamic Parameters. In *Forty-second International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*. PMLR.

Zhu, C.; Byrd, R. H.; Lu, P.; and Nocedal, J. 1997. Algorithm 778: L-BFGS-B: Fortran subroutines for large-scale bound-constrained optimization. *ACM Transactions on mathematical software (TOMS)*, 23(4): 550–560.