DCEMBED: SCALABLE OPTIMIZATION OVER NEURAL SURROGATES

Anonymous authors
Paper under double-blind review

ABSTRACT

Neural surrogates can accelerate large-scale optimization by replacing expensive or intractable model components with more efficient learned approximations, but solving the resulting embedded problems can remain prohibitively costly. For instance, standard exact encodings of neural networks with ReLU activations allow for the embedded problem to be solved by mixed-integer solvers, but add large numbers of binary variables to accommodate the nonlinearity of the activations, which can render the problem computationally prohibitive. To address this challenge, we propose `DCEmbed`,[1] a local heuristic for optimization problems with embedded neural surrogates that leverages the difference-of-convex (DC) representation of the neural network and avoids introducing auxiliary activation binaries. Exploiting shared structure within the DC representation of a ReLU neural network, we derive a reduced-size, exact formulation for its convex components that can be embedded in optimization problems using just two linear inequalities and one continuous auxiliary variable per hidden neuron. Using this formulation, the resulting embedded problem can be solved via an iterative penalty convex-concave procedure, where only the concave portions of the neural terms are approximated at each stage. The original objective, constraints, and any discrete decisions are retained exactly, allowing the use of standard convex or mixed-integer optimization solvers to optimize the host and surrogate jointly at each iteration of the procedure. In experiments on synthetic quadratic programs, mixed-integer resource allocation, and neural two-stage stochastic programming, our method demonstrates substantially faster progress toward high-quality feasible solutions than approaches using exact mixed-integer embeddings. In particular, `DCEmbed` achieves $4\times$ lower normalized primal integral than the best exact baseline on the resource allocation problem, while in two-stage stochastic programming it reaches the global surrogate optimum $\sim 5\times$ faster than Gurobi ML.

1 INTRODUCTION

Neural surrogates are increasingly used to approximate optimization model components that are expensive to evaluate or difficult to express analytically. Applications include approximating recourse costs in two-stage stochastic programming (Dumouchelle et al., 2022), modeling nonlinear physics in power system unit commitment (Kody et al., 2022), and incorporating constraints learned from data (Fajemisin et al., 2024; Maragno et al., 2025). We analyze settings in which trained neural networks replace selected components of the objective and/or constraints of an optimization problem that has discrete and/or continuous decision variables (the *host model*). We refer to the remaining, explicitly modeled part as the *residual model*. When the inputs to a neural surrogate depend on decision variables, the surrogate must be optimized jointly with this residual model. This can make the embedded problem difficult to solve even when mature optimization solvers can efficiently exploit the structure of the residual model.

Standard exact mixed-integer encodings of ReLU networks retain the host objective and constraints, but introduce binary variables to represent neural activation states. The number of binaries scales with network size and repeated embeddings, leading to high-dimensional combinatorial problems whose solution can become computationally prohibitive. Alternative gradient-based solvers like

[1] An implementation is in the supplementary material and will be released publicly upon acceptance.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

projected gradient descent (PGD) avoid activation binaries by directly optimizing neural surrogates or penalties, but cannot directly accommodate discrete decisions in the host problem. Moreover, PGD can require costly projections onto the residual constraints, while poor numerical conditioning can slow progress. These limitations motivate a significant need for new methods that can use existing solvers to optimize the neural terms and residual model jointly while retaining any original discrete decisions and avoiding the combinatorial burden of neural activation binaries.

To address this gap, we propose `DCEmbed`, a scalable heuristic optimization method that accommodates neural surrogates in the objective as well as in inequality and equality constraints. We begin by expressing a trained ReLU network as a difference-of-convex (DC) functions (Awasthi et al., 2024). At the core of `DCEmbed` is an exact, reduced-size formulation for optimizing over these convex components within the host model. We use this formulation within the penalty convex-concave procedure (CCP) (Lipp & Boyd, 2016), which solves a sequence of subproblems by retaining one convex component of each neural term and linearizing the other. Only the neural terms are approximated, with the residual model expressions and original binary decisions retained exactly. Off-the-shelf optimization solvers can therefore optimize the residual and convexified neural terms jointly, without the additional combinatorial search introduced by neural activation binaries.

Key Challenges. A central challenge is to turn the DC decomposition of a neural network into a representation that is both scalable and composable within a larger, structured optimization model. The formulation must recover the desired convex components exactly, keep the subproblems small as networks get larger or are embedded repeatedly, and reuse hidden variables when several predictions enter different parts of the model. These requirements are complicated by the recursive dependence between the two convex components across layers in the DC decomposition. The representation must also support neural objectives, inequalities, and equalities while preserving the residual expressions and original binary decisions. Additionally, CCP's conservative treatment of neural constraints can potentially exclude high quality feasible solutions and limit progress.

Contributions. We make the following contributions.

- **Exact neural formulations.** We show that one continuous auxiliary variable and two linear inequalities per hidden neuron are sufficient to represent the epigraphs of a trained ReLU network's DC components exactly, without activation binaries. Compared with a direct representation of both DC components, this halves the variable count and reduces the number of layerwise constraints required. The resulting formulation yields smaller CCP subproblems and supports the reuse of hidden variables and constraints.
- **Activation-binary-free optimization of embedded surrogates.** We combine this formulation with penalty CCP to optimize neural objectives, inequalities, and equalities jointly with the residual model and its original binary decisions. Penalty CCP enables exploration of points excluded by vanilla CCP's conservative approximation of neural constraints. Because we add only continuous variables and linear inequalities, and no activation binaries, a convex residual model remains convex while a mixed-integer convex model remains mixed-integer convex. This allows existing mature optimization solvers to optimize the embedded surrogate and residual model together.
- **Superior computational scaling.** Across synthetic quadratic programs and two mixed-integer applications, `DCEmbed` finds strong feasible incumbents even when exact mixed-integer formulations fail, and outperforms PGD on most tested continuous architectures. On the largest resource allocation instance, it achieves over $4\times$ lower normalized primal integral than the best exact baseline. In two-stage stochastic programming, it reaches the global surrogate optimum approximately $5\times$ faster than Gurobi ML at the largest scenario count, while exact mixed-integer formulations remain 4.81% suboptimal at the time limit.

2 Related Work

Surrogate models in optimization. Surrogate embeddings can vary in both the quantities learned and their role within the optimization model. Constraint learning augments an optimization model with feasibility conditions inferred from data (Fajemisin et al., 2024; Maragno et al., 2025). Other embeddings approximate physical relationships (Kody et al., 2022), or learn recourse functions and

solution mappings of optimization problems (Dumouchelle et al., 2022; Chen et al., 2024). Across these settings, the solution method depends on whether the learned terms enter the objective or constraints, and whether the host contains discrete decisions. Next, we discuss mixed-integer formulations, continuous optimization methods, and other alternatives for optimizing embedded models.

Formulations for ReLU networks. The big-M formulation is a standard exact encoding of a ReLU network using binary variables and linear constraints (Fischetti & Jo, 2018; Grimstad & Andersson, 2019). Using this encoding preserves the host objective and constraints but adds binary variables for the activations, turning continuous hosts into mixed-integer programs and increasing the number of discrete variables for mixed-integer hosts. Performance depends on the bounds on the preactivations, or inputs to each ReLU, which determine the strength of the continuous relaxation. Bound tightening can strengthen these relaxations at the cost of additional preprocessing (Grimstad & Andersson, 2019; Sosnin & Tsay, 2024), while stronger polyhedral formulations trade increased model size for tighter relaxations (Anderson et al., 2020; Tsay et al., 2021). Exact ReLU network embeddings are implemented in software packages such as OMLT (Ceccon et al., 2022), Gurobi Machine Learning (Gurobi Optimization, LLC), and PySCIPOpt-ML (Turner et al., 2024). We instead optimize the embedded problem locally through a DC representation, preserving the host structure without introducing activation binaries or requiring additional computation for bound tightening.

Continuous optimization over trained neural networks. For continuous hosts, projected gradient descent (PGD) (Beck, 2017) differentiates through the fixed network and projects onto the residual constraints, with neural constraint violations handled through penalty terms. However, costly projections and poor numerical conditioning can limit the efficacy of PGD, and it does not directly accommodate discrete host decisions. Awasthi et al. (2024) construct a DC decomposition for ReLU networks and apply the difference-of-convex algorithm (DCA) to scalar neural objectives over convex domains, such as scalar function and adversarial margin minimization. In both applications, the network enters the host through a scalar objective optimized over a simple-to-project-onto convex input domain. Liu & Dvorkin (2025) propose a different DC formulation based on complementarity conditions for individual ReLUs, whose violations are penalized before applying DCA. Here, we derive an exact, reduced-size epigraph formulation for the convex components in the network-level decomposition of Awasthi et al. (2024). A single set of continuous hidden variables and linear constraints serves to represent both convex components. This shared formulation enables penalty CCP to optimize neural objectives, inequalities, and equalities jointly with the residual objective, constraints, and discrete decisions.

Neural network verification and optimization. Neural network verifiers construct bounds on a network or computation graph over a bounded input region, refining the domain when the bounds are inconclusive. The same machinery can be used for continuous global optimization. In α, β-CROWN, gradient-based search provides feasible incumbents, while a combination of certified bounds and branch-and-bound excludes input subdomains that cannot contain a better solution (Wang et al., 2021; Zhang et al., 2022; Li et al., 2026). In this case, both the host and neural terms enter the verification-based bounding framework. We instead retain the host objective and constraints explicitly within each CCP subproblem, allowing the underlying solver to handle their convex structure and any original discrete decisions directly.

3 Problem Setting and Background

We consider a *host optimization model* with continuous decision variables $y \in \mathbb{R}^{n_y}$ and binary decision variables $\delta \in \{0,1\}^{n_\delta}$. The components to be replaced by neural surrogates are denoted q_{obj} in the objective, $q_{\mathrm{ineq},i}$ in inequality constraint i, and $q_{\mathrm{eq},j}$ in equality constraint j. For simplicity of notation, let x denote the set of decision variables used as input to these functions. Thus, x is a subvector of (y, δ). Separating these components from the remaining model, we can write

$$\begin{aligned} \underset{y,\delta}{\text{minimize}} \quad & \phi_0(y,\delta) + q_{\mathrm{obj}}(x) \\ \text{s.t.} \quad & \phi_i(y,\delta) + q_{\mathrm{ineq},i}(x) \le 0, \quad i = 1,\dots,m_{\mathrm{ineq}}, \\ & \psi_j(y,\delta) + q_{\mathrm{eq},j}(x) = 0, \quad j = 1,\dots,m_{\mathrm{eq}}, \\ & y \in \mathcal{Y}, \quad \delta \in \{0,1\}^{n_\delta}, \end{aligned} \tag{1}$$

where $\mathcal{Y} \subseteq \mathbb{R}^{n_y}$ is a closed set. The scalar-valued q terms denote possibly nonconvex, expensive to evaluate, or otherwise difficult components meant to be replaced by neural surrogates. The remaining expressions ϕ_0, ϕ_i, and ψ_j, together with the variables, constitute the *residual model*.

To approximate the q terms in Problem 1, we use a standard neural surrogate architecture (Goodfellow et al., 2016) and take an affine combination of its outputs. In our notation, $f_\theta : \mathbb{R}^{n_0} \to \mathbb{R}^p$ is a network with fixed parameters θ. For a network with L ReLU layers, let $z^0(x) = x$ and define

$$\begin{aligned} a^\ell(x) &= W^\ell z^{\ell-1}(x) + b^\ell, \quad \ell = 1, \dots, L, \\ z^\ell(x) &= \mathrm{ReLU}\left(a^\ell(x)\right), \qquad \ell = 1, \dots, L, \\ f_\theta(x) &= z^L(x), \end{aligned} \tag{2}$$

where $\mathrm{ReLU}(v)_k = \max\{v_k, 0\}$ is applied componentwise, and $W^\ell \in \mathbb{R}^{n_\ell \times n_{\ell-1}}$ and $b^\ell \in \mathbb{R}^{n_\ell}$ for $\ell = 1, \dots, L$. The approximation of each q term in Problem 1 is specified by coefficient vectors $\nu \in \mathbb{R}^p$ and offsets $\beta \in \mathbb{R}$ (indexed by their role). In this way, the *embedded problem* is

$$\begin{aligned} \underset{y,\delta}{\text{minimize}} \quad & \phi_0(y,\delta) + \left(\nu_{\text{obj}}^\top f_\theta(x) + \beta_{\text{obj}}\right) \\ \text{s.t.} \quad & \phi_i(y,\delta) + \left(\nu_{\text{ineq},i}^\top f_\theta(x) + \beta_{\text{ineq},i}\right) \le 0, \quad i = 1, \dots, m_{\text{ineq}}, \\ & \psi_j(y,\delta) + \left(\nu_{\text{eq},j}^\top f_\theta(x) + \beta_{\text{eq},j}\right) = 0, \qquad j = 1, \dots, m_{\text{eq}}, \\ & y \in \mathcal{Y}, \quad \delta \in \{0,1\}^{n_\delta}. \end{aligned} \tag{3}$$

For simplicity, Problem 3 shows a single network evaluation shared across the objective and constraints. If the objective or a constraint has no surrogate term, we set its corresponding coefficient vector ν and offset β to be zero. Surrogate predictions can either enter directly in the constraints and objective as above, or an equality constraint can link a prediction to a host variable which can then appear elsewhere in the model.

3.1 Difference-of-Convex (DC) Representation

A function f is *difference-of-convex (DC)* if it can be written as $f = g - h$, with g and h convex. The following result constructs a DC decomposition of each hidden activation in a ReLU network. To state the construction, define the positive and negative parts of a vector or matrix M entrywise as $M^+ = \max\{M, 0\}$ and $M^- = \max\{-M, 0\}$, so $M = M^+ - M^-$ and $|M| = M^+ + M^-$.

Theorem 3.1 (Awasthi et al., 2024)**.** *For each hidden layer $\ell = 1, \dots, L$ of a* ReLU *network, there exist componentwise convex functions G^ℓ and H^ℓ such that $z^\ell(x) = G^\ell(x) - H^\ell(x)$. One such decomposition is given by the following recursion. Suppressing the argument x for simplicity, initialize $G^1 = z^1, H^1 = 0$ for $\ell = 1$, and for $\ell \ge 2$, set*

$$H^\ell = W^{\ell,-} G^{\ell-1} + W^{\ell,+} H^{\ell-1}, \qquad G^\ell = \max\left\{W^{\ell,+} G^{\ell-1} + W^{\ell,-} H^{\ell-1} + b^\ell, H^\ell\right\}, \tag{4}$$

where the maximum is taken componentwise.

We include a proof of this result in Appendix A.1 for completeness. This construction provides recursive expressions for the convex components within the DC decomposition, but does not give an explicit formulation of their epigraphs using continuous variables and linear constraints. As we will show in Section 4, obtaining such a formulation will allow the components to be optimized jointly with the residual model while retaining its structure and avoiding neural activation binaries.

3.2 Convex–Concave Procedure (CCP)

The convex-concave procedure (CCP) constructs convex upper approximations of DC functions by retaining one convex component and linearizing the other (Lipp & Boyd, 2016). For a scalar DC function $f = g - h$, we define the affine lower bound of h at iterate x^k as

$$\widehat{h}^k(x) = h(x^k) + (d_h^k)^\top (x - x^k), \quad d_h^k \in \partial h(x^k), \tag{5}$$

and define $\widehat{g}^k(x)$ analogously. Since h and g are convex, we have $\widehat{h}^k(x) \le h(x)$ and $\widehat{g}^k(x) \le g(x)$. Thus,

$$f(x) \le g(x) - \widehat{h}^k(x), \quad -f(x) \le h(x) - \widehat{g}^k(x). \tag{6}$$

The first bound provides an upper approximation when f appears in the (minimization) objective, and a convex restriction when f appears in an inequality constraint. Equalities are treated by applying the restriction in both directions. CCP solves the resulting subproblem and updates the affine approximations at the new iterate, repeating until a stopping criterion is met. Noting that these restrictions can exclude points that satisfy the original constraints, penalty CCP relaxes the restrictions with nonnegative slack variables and penalizes their sum in the objective, allowing exploration of otherwise infeasible points in the restricted set. The next section will discuss how this procedure can be efficiently employed for Problem 3 under our exact formulation of the neural DC components.

4 METHOD

In this section, we develop an exact formulation of the neural DC components that reduces the size of each CCP subproblem. Because these subproblems are solved repeatedly, reducing their size can significantly lower the overall computational cost of CCP, particularly for deeper networks and repeated surrogate embeddings. First, we show in Theorem 4.1 that both components can be expressed directly in terms of the network's hidden activations. Using these expressions, we establish an exact epigraph formulation with one continuous auxiliary variable and two linear inequalities per hidden neuron in Theorem 4.2. We use this representation within penalty CCP to optimize the neural objectives and constraints jointly with the residual model.

Theorem 4.1 (Shared representation of DC components)**.** *For each affine measurement in Problem 3, let $f_r(x) = \nu_r^\top z^L(x) + \beta_r$, where r denotes* obj, (ineq, i), *or* (eq, j). *We define:*

$$\begin{aligned} \rho_r^L &= |\nu_r|, \\ \rho_r^{\ell-1} &= |W^\ell|^\top \rho_r^\ell, \qquad \ell = L, L-1, \dots, 2, \\ c_{r,\ell-1} &= \left(W^{\ell,-}\right)^\top \rho_r^\ell, \quad \ell = L, L-1, \dots, 2. \end{aligned} \tag{7}$$

Then, $f_r = G_r - H_r$ has the convex components

$$G_r(x) = \beta_r + (\nu_r^+)^\top z^L(x) + \sum_{\ell=1}^{L-1} c_{r,\ell}^\top z^\ell(x), \quad H_r(x) = (\nu_r^-)^\top z^L(x) + \sum_{\ell=1}^{L-1} c_{r,\ell}^\top z^\ell(x). \tag{8}$$

Proof. Applying Theorem 3.1 to the last hidden layer gives $z^L = G^L - H^L$. Writing $\nu_r = \nu_r^+ - \nu_r^-$, we have the affine measurement representation $f_r = G_r - H_r$ with

$$G_r = \beta_r + (\nu_r^+)^\top G^L + (\nu_r^-)^\top H^L, \quad H_r = (\nu_r^-)^\top G^L + (\nu_r^+)^\top H^L. \tag{9}$$

We define the parameterization $\eta^\ell = H^\ell$, and thus $G^\ell = z^\ell + \eta^\ell$. Substituting into Eq. 4 gives

$$\eta^1 = 0, \qquad \eta^\ell = W^{\ell,-} z^{\ell-1} + |W^\ell| \eta^{\ell-1}, \quad \ell = 2, \dots, L. \tag{10}$$

Substituting the parameterization into Eq. 9 gives

$$G_r(x) = \beta_r + \left(\nu_r^+\right)^\top z^L(x) + |\nu_r|^\top \eta^L(x), \quad H_r(x) = \left(\nu_r^-\right)^\top z^L(x) + |\nu_r|^\top \eta^L(x). \tag{11}$$

Expanding (10) and combining with $\eta^1 = 0$ yields the equivalence $|\nu_r|^\top \eta^L(x) = \sum_{\ell=1}^{L-1} c_{r,\ell}^\top z^\ell(x)$. Finally, substituting this equivalence into Eq. 11 yields the desired representation. □

The following theorem gives exact epigraph formulations for the components in Eq. 8 by replacing the hidden activations with continuous auxiliary variables and imposing a set of linear inequalities.

Theorem 4.2 (Extended Representation of DC Components)**.** *For any input x, set $u^0 = x$ and define the set*

$$\mathcal{U}(x) = \left\{ u = (u^1, \dots, u^L) \;\middle|\; \begin{array}{ll} u^\ell \geq W^\ell u^{\ell-1} + b^\ell, & \ell = 1, \dots, L \\ u^\ell \geq 0, & \end{array} \right\}, \tag{12}$$

where all inequalities are componentwise. Let $\overline{G}_r(u)$ and $\overline{H}_r(u)$ be the affine expressions obtained from Eq. 8 by replacing each $z^\ell(x)$ with u^ℓ. Then, for every input x,

$$G_r(x) = \min_{u \in \mathcal{U}(x)} \overline{G}_r(u), \qquad H_r(x) = \min_{u \in \mathcal{U}(x)} \overline{H}_r(u). \tag{13}$$

Proof Sketch. We fix x and take any $u \in \mathcal{U}(x)$. We define the common offsets

$$\widetilde{\eta}^1 = 0, \qquad \widetilde{\eta}^\ell = W^{\ell,-}u^{\ell-1} + |W^\ell|\widetilde{\eta}^{\ell-1}, \quad \ell = 2, \dots, L.$$

and

$$\widetilde{G}^\ell(x) = u^\ell + \widetilde{\eta}^\ell(x), \quad \widetilde{H}^\ell(x) = \widetilde{\eta}^\ell(x).$$

We can show by induction that $\widetilde{G}^\ell(x) \geq G^\ell(x)$ and $\widetilde{H}^\ell(x) \geq H^\ell(x)$, which combined with the affine measurement representation ultimately gives $\overline{G}_r(u) \geq G_r(x)$ and $\overline{H}_r(u) \geq H_r(x)$. Conversely, we can show that there exists a feasible u that attains this bound by considering the choice $u^\ell = z^\ell(x) = \mathrm{ReLU}\left(W^\ell z^{\ell-1}(x) + b^\ell\right)$. See Appendix A.2 for the full proof. □

Sharing and formulation size. The set $\mathcal{U}(x)$ (12) depends only on the trained network and its input, while the measurement coefficients enter through the expressions $G_r(u)$ and $H_r(u)$ in Eq. 8. The assignment $u^\ell = z^\ell(x)$ for $\ell = 1, \dots, L$ simultaneously minimizes both expressions for all r. Therefore, the epigraph constructions of these convex components can share u and its layerwise constraints across the objective and constraints of each CCP subproblem. Let $N_{\mathrm{hidden}} = \sum_{\ell=1}^{L} n_\ell$ denote the total number of hidden units in an L layer ReLU network. Our formulation uses N_{hidden} continuous hidden variables and $2N_{\mathrm{hidden}}$ layerwise inequalities, compared with $2N_{\mathrm{hidden}} - n_1$ variables and $3N_{\mathrm{hidden}} - n_1$ constraints for a direct representation. This reduction can substantially improve computational performance as demonstrated in Section 5.1.

4.1 Penalty CCP for the Embedded Problem

We let x^k denote the input selected from (y^k, δ^k) at iteration k. For each affine measurement r, let $\widehat{H}_r^k$ and $\widehat{G}_r^k$ denote the affine lower bounds of H_r and G_r, respectively, at x^k. Applying Theorem 4.2 to the embedded problem in (3), the CCP subproblem at iteration k becomes

$$\begin{aligned}
\underset{y,\delta,u,s,t}{\text{minimize}} \quad & \phi_0(y,\delta) + \overline{G}_{\mathrm{obj}}(u) - \widehat{H}^k_{\mathrm{obj}}(x) + \tau_k \left(\sum_i s_i + \sum_j t_j \right) \\
\text{s.t.} \quad & \phi_i(y,\delta) + \overline{G}_{\mathrm{ineq},i}(u) - \widehat{H}^k_{\mathrm{ineq},i}(x) \leq s_i, && i = 1, \dots, m_{\mathrm{ineq}}, \\
& \psi_j(y,\delta) + \overline{G}_{\mathrm{eq},j}(u) - \widehat{H}^k_{\mathrm{eq},j}(x) \leq t_j, && j = 1, \dots, m_{\mathrm{eq}}, \\
& -\psi_j(y,\delta) + \overline{H}_{\mathrm{eq},j}(u) - \widehat{G}^k_{\mathrm{eq},j}(x) \leq t_j, && j = 1, \dots, m_{\mathrm{eq}}, \\
& u \in \mathcal{U}(x), \quad s \geq 0, \quad t \geq 0, \quad y \in \mathcal{Y}, \quad \delta \in \{0,1\}^{n_\delta},
\end{aligned} \tag{14}$$

where $\tau_k > 0$ is the slack penalty parameter. For constraints without a neural term, the corresponding slack is fixed to zero. Each equality constraint has been expanded into two restrictions sharing the same slack t_j. Theorem 4.2 ensures that sharing u across the objective and constraints introduces no additional approximation of the penalized CCP subproblem beyond the neural linearizations.

Problem 14 retains the residual expressions and original binary variables, adding only continuous auxiliary variables and linear inequalities for the convexified neural terms. As a result, the subproblems above are convex or mixed-integer convex when $\mathcal{Y}$ is convex, ϕ_0 and ϕ_i are convex on $\mathcal{Y} \times [0,1]^{n_\delta}$, and each ψ_j is affine, even when the components q being approximated are nonconvex.

Neural feasibility and stopping criteria. Let $w = (y, \delta)$ denote the original host variables. We evaluate the embedded objective and violation as

$$\begin{aligned}
& F(w) = \phi_0(y,\delta) + f_{\mathrm{obj}}(x), \\
& V(w) = \max\left\{0,\ \max_i [\phi_i(y,\delta) + f_{\mathrm{ineq},i}(x)]_+,\ \max_j |\psi_j(y,\delta) + f_{\mathrm{eq},j}(x)| \right\},
\end{aligned} \tag{15}$$

where $[a]_+ = \max\{a, 0\}$. A candidate point is considered feasible when we have $V(w) \leq \epsilon_{\mathrm{feas}}$, where ϵ_{feas} is a desired neural feasibility tolerance. We terminate the algorithm when two consecutive feasible candidates satisfy $\left|F(w^{k+1}) - F(w^k)\right| \leq \epsilon_{\mathrm{obj}} \max\{1, \left|F(w^k)\right|\}$.

Penalty update. Let $\mu > 1$ denote the penalty scaling factor and $\tau_{\max}$ the maximum penalty. Then, we set $\tau_{k+1} = \min\{\mu\tau_k, \tau_{\max}\}$ when $V(w^{k+1}) > \epsilon_{\mathrm{feas}}$ and keep $\tau_{k+1} = \tau_k$ otherwise.

5 EXPERIMENTS

To assess `DCEmbed`'s performance, we evaluate how quickly it finds high quality feasible solutions within a fixed-time budget. A standard metric to capture the tradeoff between speed and solution quality is the *primal integral* (Berthold, 2013; Gasse et al., 2022). We use a *normalized primal integral* (NPI) that scales the objective gap by the improvement available from a common initialization, making scores comparable across instances with different objective scales and initial gaps:

$$\mathrm{NPI} = \frac{100}{T} \int_0^T \frac{J_{\mathrm{best}}(t) - J_{\mathrm{ref}}}{J_0 - J_{\mathrm{ref}}}\, dt, \tag{16}$$

where $J_{\mathrm{best}}(t)$ is the best feasible objective found by time t, J_0 is the initial objective, and J_{ref} is the best known feasible objective for the instance. Lower values indicate stronger primal progress, and 100% indicates no improvement over the initial incumbent.

All experiments use neural surrogates trained in PyTorch (Paszke et al., 2019), with all optimization models formulated in Gurobi (Gurobi Optimization, LLC, 2026). All baseline methods that use Gurobi are configured to prioritize finding feasible incumbents rather than solving to optimality. For detailed descriptions of the baselines and experiment settings, see Appendices B and C.

5.1 NETWORK SCALING IN SYNTHETIC CONVEX PROGRAMS

We first study how `DCEmbed` scales with the width and depth of the embedded neural surrogate using synthetic quadratic programs (QPs) with repeated neural inequality constraints. We randomly generate convex QPs, where the decision vector $x \in \mathbb{R}^{150}$ is partitioned into 30 blocks $x^{(r)} \in \mathbb{R}^5$:

$$\begin{aligned} \underset{x\in[-1,1]}{\text{minimize}} \quad & \frac{1}{2}\|Gx\|_2^2 + c^\top x \\ \text{s.t.} \quad & Ex = 0, \quad |Cx| \le \mu \mathbf{1}, \\ & f_\theta(x^{(r)}) \le 0, \quad r = 1, \dots, 30. \end{aligned} \tag{17}$$

We train networks to approximate a fixed nonconvex constraint function and vary the width $w \in \{8, 16, 32, 64, 128, 256\}$ at depth $d = 3$, and vary the depth $d \in \{1, \dots, 6\}$ at width $w = 64$. We compare `DCEmbed`, Gurobi ML, big-M with optimization-based bound tightening (OBBT) (Grimstad & Andersson, 2019), and projected gradient descent (PGD). We repeat both sweeps over three independent trials using separately sampled residual models and newly trained networks. Within each trial, the residual model is fixed across architectures, and all methods receive the same initialization for each architecture.

Results. Figure 1 shows that `DCEmbed` achieves substantially lower normalized primal integral than both Gurobi ML and big-M with OBBT. Neither of these latter exact formulations improve upon the initial feasible incumbent for wider and deeper networks. At $(w, d) = (64, 3)$, `DCEmbed` achieves a mean NPI of 0.72% compared to 7.97% for PGD and 100% for both exact embeddings. `DCEmbed` achieves lower NPI than PGD across most architectures, although its advantage narrows as the neural network size grows, with PGD performing best at the largest width and slightly better at the largest depth. We also compare `DCEmbed`'s shared hidden representation with separate layerwise representations of the two convex components (*two-path*). The shared formulation reduces mean NPI by approximately 2.5–3.2$\times$ at widths 128–256 and 3.5–4.5$\times$ at depths 4–6.

5.2 RESOURCE ALLOCATION FOR WATER TREATMENT

We adapt the shared resource allocation benchmark from SurrogateLIB (Turner et al., 2024) to evaluate repeated neural equalities within a mixed-integer residual model. The benchmark considers the allocation of shared resources to treat a set of water samples. Each sample $i \in \mathcal{I} = \{1, \dots, N\}$ has nine measured attributes. Let $b_i \in \mathbb{R}^9$ denote a sample's untreated feature vector, and let $r_i, a_i \in \mathbb{R}^9$ denote the amounts removed from and added to each attribute, producing the treated feature vector x_i. The objective maximizes the number of samples predicted potable, subject to shared

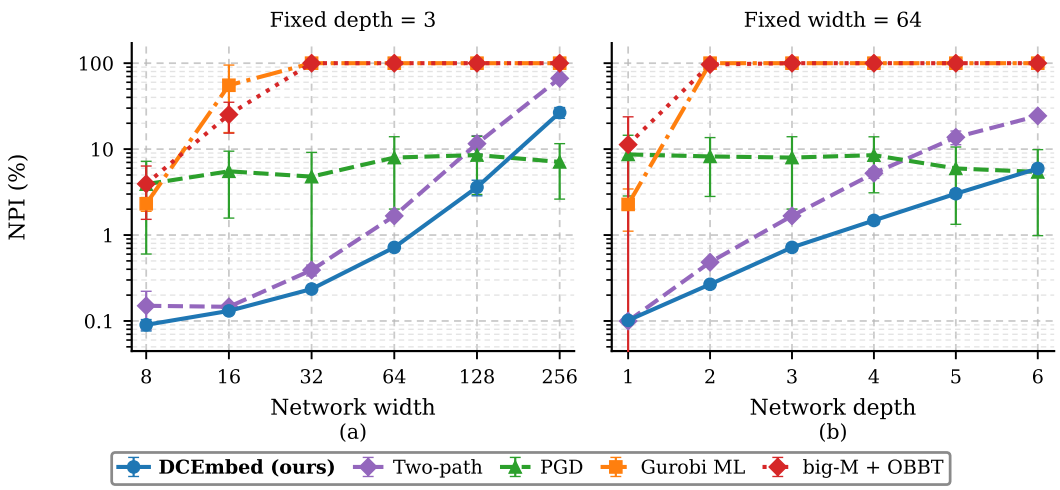

Figure 1: NPI (lower is better) on QPs as network (a) width and (b) depth increases. `DCEmbed` substantially outperforms the exact embeddings and improves on the two-path formulation.

removal and addition budgets $R, A \in \mathbb{R}^9$. The resulting embedded optimization problem is

$$\begin{aligned} \underset{x,r,a,s,y}{\text{maximize}} \quad & \sum_{i\in\mathcal{I}} y_i \\ \text{s.t.} \quad & x_i = b_i - r_i + a_i, \quad s_i = f_\theta(x_i), && i \in \mathcal{I}, \\ & y_i \in \{0,1\}, \quad y_i = 1 \Longrightarrow s_i \geq \epsilon, && i \in \mathcal{I}, \\ & \sum_{i\in\mathcal{I}} r_i \leq R, \quad \sum_{i\in\mathcal{I}} a_i \leq A, \quad r, a \geq 0. \end{aligned} \tag{18}$$

The binary variable y_i allows a sample to be counted only when its classifier score s_i meets the potability threshold ϵ. Each sample contributes one neural output equality and one original binary decision, with treatment decisions coupled through the shared budgets. The explicit score s_i connects the surrogate to Gurobi's native indicator constraint, which remains part of the residual model.

We use a fixed classifier trained on the Water Potability dataset (Kadiwal, 2021) with four hidden ReLU layers of width 32 to predict sample potability. We vary $N \in \{25, 50, 100, 200, 400, 800\}$ over three candidate cohorts and compare `DCEmbed`, Gurobi ML, and big-M with OBBT using the same network and initialization.

Results. In Figure 2(a), we see `DCEmbed` achieves lower mean NPI against exact methods at every candidate count. At $N = 800$, it achieves an NPI of 22.6% compared with 97.4% for big-M with OBBT and 100% for Gurobi ML. At $N = 800$, the OBBT big-M model retains roughly $89,160$ neural activation binaries in addition to the 800 original binary host variables. `DCEmbed` removes the neural activation binaries and solves a sequence of mixed-integer convex subproblems containing the original decisions. Compared to two-path, `DCEmbed` achieves lower mean NPI at all but the smallest size, with an average mean NPI improvement of $\sim 2\times$ across all sizes. The smaller gains relative to the QP benchmark may reflect the additional difficulty of discrete subproblems.

5.3 Scenario Scaling in Two-Stage Stochastic Programming

We revisit the pooling benchmark from Neur2SP (Dumouchelle et al., 2022), a two-stage stochastic program in which 16 binary first-stage decisions x are chosen before uncertainty ξ is realized. For each realization, a nonconvex second-stage problem determines the optimal recourse cost $Q(x, \xi)$. Neur2SP's per-scenario neural surrogate (*NN-P*) learns $\Phi^P_\theta(x, \xi) \approx Q(x, \xi)$, replacing each second-stage optimization with a neural prediction. For S scenarios with probabilities p_s, the embedded

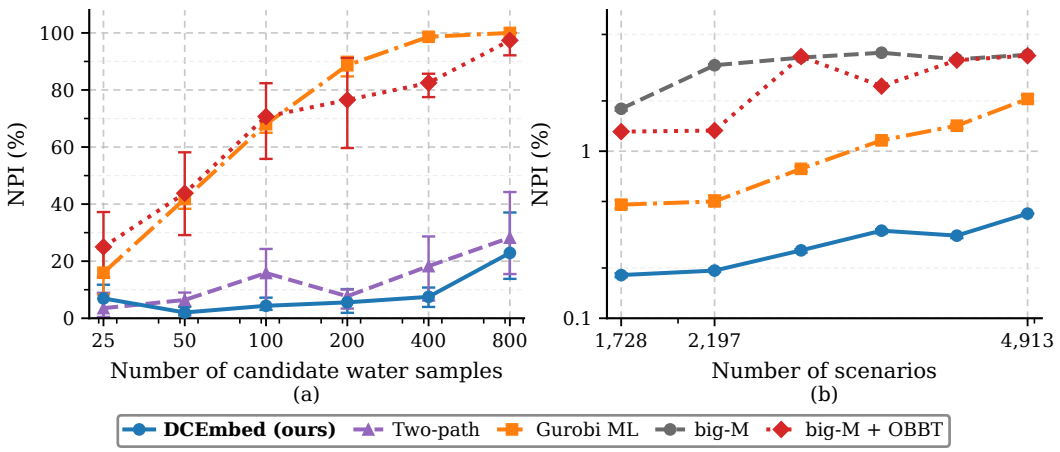

Figure 2: Scaling in (a) resource allocation for water treatment and (b) neural two-stage stochastic programming. `DCEmbed` achieves lower NPI than exact embeddings across all tested sizes.

problem minimizes the first-stage cost plus the predicted expected recourse cost as

$$\underset{x\in\mathcal{X}}{\text{minimize}} \quad c^\top x + \sum_{s=1}^{S} p_s \Phi_\theta^P(x, \xi_s), \tag{19}$$

where $\mathcal{X}$ contains the first-stage constraints and binary decisions. The objective embeds S evaluations of the same trained neural network, all coupled through the first-stage decisions x.

We use the released *NN-P* network with one hidden ReLU layer of width 64 and vary S from $1,728$ to $4,913$ over 6 sizes. We compare `DCEmbed`, Gurobi ML, and Neur2SP's original and OBBT big-M formulations on the same first-stage model, network, and scenarios across three trials.

Results. Figure 2(b) shows that `DCEmbed` achieves the lowest average NPI across all scenario counts under the $180s$ budget. At $S = 4,913$, its NPI is 0.423%, approximately $4.8\times$ and $8.8\times$ lower than Gurobi ML and big-M with OBBT, respectively. Both `DCEmbed` and Gurobi ML recover the global optimum (computed offline) of the neural surrogate problem, but `DCEmbed` reaches it approximately $5\times$ faster than Gurobi ML at $S = 4,913$ scenarios, with median times of 18.82 and 95.54 seconds, respectively. At this size, both big-M variants remain 4.81% suboptimal at the time limit. Even after bound tightening, the OBBT formulation contains $170,714$ neural activation binaries. `DCEmbed` avoids these additional binaries, finding optimal designs earlier through the sequence of subproblems containing only the 16 original first-stage binaries. The *NN-P* architecture is a single layer, and thus `DCEmbed`'s formulation is equivalent to two-path.

6 Conclusion

We presented `DCEmbed`, a scalable local heuristic for optimizing trained neural surrogates embedded in the objectives, inequalities, and equalities of optimization problems. The method combines an exact, reduced-size epigraph formulation of the DC components of the surrogate with penalty CCP, retaining the residual expressions and original discrete decisions without adding activation binaries. Across various benchmarks, `DCEmbed` shows stronger progress towards high-quality feasible solutions than exact mixed-integer embeddings. It achieves over $4\times$ lower NPI in resource allocation and finds the surrogate optimum $\sim 5\times$ faster than Gurobi ML in two-stage stochastic programming.

Avoiding activation binaries shifts the computational bottleneck to the sequence of CCP subproblems, whose cost remains an important limitation at larger scales. While our formulation reduces their size for a fixed DC decomposition, a natural next step is to tailor the decomposition itself to the iterative procedure, balancing progress per iteration with subproblem cost to improve scalability and solution quality. Furthermore, `DCEmbed` is a local heuristic that provides no convergence or optimality guarantees, and thus establishing such guarantees remains a valuable theoretical direction.

7 AI USE STATEMENT

In this work, we used generative AI coding tools to assist with the implementation of our method and baselines. Other tasks requiring disclosure were either performed without generative AI tools or were not applicable to this work. Additionally, we used generative AI tools to help identify relevant literature and edit the research paper to improve readability. We have reviewed all AI-assisted work. LLM-generated code was verified and tested for correctness. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REFERENCES

Ross Anderson, Joey Huchette, Will Ma, Christian Tjandraatmadja, and Juan Pablo Vielma. Strong mixed-integer programming formulations for trained neural networks. *Mathematical Programming*, 183(1):3–39, September 2020. ISSN 1436-4646. doi: 10.1007/s10107-020-01474-5. URL https://doi.org/10.1007/s10107-020-01474-5.

Pranjal Awasthi, Anqi Mao, Mehryar Mohri, and Yutao Zhong. DC-programming for neural network optimizations. *Journal of Global Optimization*, 95(1):5–21, January 2024. ISSN 0925-5001, 1573-2916. doi: 10.1007/s10898-023-01344-2. URL https://link.springer.com/10.1007/s10898-023-01344-2.

Amir Beck. *First-Order Methods in Optimization*. Society for Industrial and Applied Mathematics, Philadelphia, PA, October 2017. ISBN 9781611974980 9781611974997. doi: 10.1137/1.9781611974997. URL http://epubs.siam.org/doi/book/10.1137/1.9781611974997.

Timo Berthold. Measuring the impact of primal heuristics. *Operations Research Letters*, 41(6): 611–614, 2013. ISSN 0167-6377. doi: https://doi.org/10.1016/j.orl.2013.08.007. URL https://www.sciencedirect.com/science/article/pii/S0167637713001181.

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

Wenbo Chen, Haoruo Zhao, Mathieu Tanneau, and Pascal Van Hentenryck. Compact optimality verification for optimization proxies. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org, 2024. event-place: Vienna, Austria.

Justin Dumouchelle, Rahul Patel, Elias B. Khalil, and Merve Bodur. Neur2SP: neural two-stage stochastic programming. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, NIPS '22, Red Hook, NY, USA, 2022. Curran Associates Inc. ISBN 9781713871088. event-place: New Orleans, LA, USA.

Adejuyigbe O. Fajemisin, Donato Maragno, and Dick Den Hertog. Optimization with constraint learning: A framework and survey. *European Journal of Operational Research*, 314(1):1–14, April 2024. ISSN 03772217. doi: 10.1016/j.ejor.2023.04.041. URL https://linkinghub.elsevier.com/retrieve/pii/S0377221723003405.

Matteo Fischetti and Jason Jo. Deep neural networks and mixed integer linear optimization. *Constraints*, 23(3):296–309, July 2018. ISSN 1383-7133, 1572-9354. doi: 10.1007/s10601-018-9285-6. URL http://link.springer.com/10.1007/s10601-018-9285-6.

Maxime Gasse, Simon Bowly, Quentin Cappart, Jonas Charfreitag, Laurent Charlin, Didier Chételat, Antonia Chmiela, Justin Dumouchelle, Ambros Gleixner, Aleksandr M. Kazachkov, Elias Khalil, Pawel Lichocki, Andrea Lodi, Miles Lubin, Chris J. Maddison, Morris Christopher, Dimitri J. Papageorgiou, Augustin Parjadis, Sebastian Pokutta, Antoine Prouvost, Lara Scavuzzo, Giulia Zarpellon, Linxin Yang, Sha Lai, Akang Wang, Xiaodong Luo, Xiang Zhou, Haohan Huang, Shengcheng Shao, Yuanming Zhu, Dong Zhang, Tao Quan, Zixuan Cao, Yang Xu, Zhewei Huang, Shuchang Zhou, Chen Binbin, He Minggui, Hao Hao, Zhang Zhiyu, An Zhiwu, and

Mao Kun. The Machine Learning for Combinatorial Optimization Competition (ML4CO): Results and Insights. In Douwe Kiela, Marco Ciccone, and Barbara Caputo (eds.), *Proceedings of the NeurIPS 2021 Competitions and Demonstrations Track*, volume 176 of *Proceedings of Machine Learning Research*, pp. 220–231. PMLR, December 2022. URL https://proceedings.mlr.press/v176/gasse22a.html.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

Bjarne Grimstad and Henrik Andersson. ReLU networks as surrogate models in mixed-integer linear programs. *Computers & Chemical Engineering*, 131:106580, December 2019. ISSN 00981354. doi: 10.1016/j.compchemeng.2019.106580. URL https://linkinghub.elsevier.com/retrieve/pii/S0098135419307203.

Gurobi Optimization, LLC. Gurobi Machine Learning. URL https://gurobi-machinelearning.readthedocs.io/. Published: Software documentation.

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

Aditya Kadiwal. Water Quality: Drinking Water Potability, 2021. URL https://www.kaggle.com/datasets/adityakadiwal/water-potability.

Alyssa Kody, Samuel Chevalier, Spyros Chatzivasileiadis, and Daniel Molzahn. Modeling the AC power flow equations with optimally compact neural networks: Application to unit commitment. *Electric Power Systems Research*, 213:108282, December 2022. ISSN 03787796. doi: 10.1016/j.epsr.2022.108282. URL https://linkinghub.elsevier.com/retrieve/pii/S0378779622004771.

Haoyu Li, Xiangru Zhong, Hao Cheng, Bin Hu, and Huan Zhang. Bridging Control with Neural Network Verifier alpha-beta-CROWN: A Tutorial, May 2026. URL http://arxiv.org/abs/2605.26577. arXiv:2605.26577 [eess.SY].

Thomas Lipp and Stephen Boyd. Variations and extension of the convex–concave procedure. *Optimization and Engineering*, 17(2):263–287, June 2016. ISSN 1389-4420, 1573-2924. doi: 10.1007/s11081-015-9294-x. URL http://link.springer.com/10.1007/s11081-015-9294-x.

Xinwei Liu and Vladimir Dvorkin. Optimization Over Trained Neural Networks: Difference-of-Convex Algorithm and Application to Data Center Scheduling. *IEEE Control Systems Letters*, 9:835–840, 2025. ISSN 2475-1456. doi: 10.1109/LCSYS.2025.3577571. URL https://ieeexplore.ieee.org/document/11028923.

Donato Maragno, Holly Wiberg, Dimitris Bertsimas, Ş. İlker Birbil, Dick Den Hertog, and Adejuyigbe O. Fajemisin. Mixed-Integer Optimization with Constraint Learning. *Operations Research*, 73(2):1011–1028, March 2025. ISSN 0030-364X, 1526-5463. doi: 10.1287/opre.2021.0707. URL https://pubsonline.informs.org/doi/10.1287/opre.2021.0707.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: an imperative style, high-performance deep learning library. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems*. Curran Associates Inc., Red Hook, NY, USA, 2019.

Philip Sosnin and Calvin Tsay. Scaling Mixed-Integer Programming for Certification of Neural Network Controllers Using Bounds Tightening. In *2024 IEEE 63rd Conference on Decision and Control (CDC)*, pp. 1645–1650, December 2024. doi: 10.1109/CDC56724.2024.10886078. URL https://ieeexplore.ieee.org/document/10886078. ISSN: 2576-2370.

Calvin Tsay, Jan Kronqvist, Alexander Thebelt, and Ruth Misener. Partition-based formulations for mixed-integer optimization of trained ReLU neural networks. In *Proceedings of the 35th International Conference on Neural Information Processing Systems*, NIPS '21, Red Hook, NY, USA, 2021. Curran Associates Inc. ISBN 9781713845393.

Mark Turner, Antonia Chmiela, Thorsten Koch, and Michael Winkler. PySCIPOpt-ML: Embedding Trained Machine Learning Models into Mixed-Integer Programs, May 2024. URL http://arxiv.org/abs/2312.08074. arXiv:2312.08074 [math.OC].

Shiqi Wang, Huan Zhang, Kaidi Xu, Xue Lin, Suman Jana, Cho-Jui Hsieh, and J. Zico Kolter. Beta-CROWN: Efficient Bound Propagation with Per-neuron Split Constraints for Neural Network Robustness Verification. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P. S. Liang, and J. Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems*, volume 34, pp. 29909–29921. Curran Associates, Inc., 2021. URL https://proceedings.neurips.cc/paper_files/paper/2021/file/fac7fead96dafceaf80c1daffeae82a4-Paper.pdf.

Huan Zhang, Shiqi Wang, Kaidi Xu, Yihan Wang, Suman Jana, Cho-Jui Hsieh, and Zico Kolter. A Branch and Bound Framework for Stronger Adversarial Attacks of ReLU Networks. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato (eds.), *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 26591–26604. PMLR, July 2022. URL https://proceedings.mlr.press/v162/zhang22ae.html.

A Proofs

A.1 Proof of Theorem 3.1

Theorem 3.1 (Awasthi et al., 2024)**.** *For each hidden layer $\ell = 1, \ldots, L$ of a* ReLU *network, there exist componentwise convex functions G^ℓ and H^ℓ such that $z^\ell(x) = G^\ell(x) - H^\ell(x)$. One such decomposition is given by the following recursion. Suppressing the argument x for simplicity, initialize $G^1 = z^1, H^1 = 0$ for $\ell = 1$, and for $\ell \geq 2$, set*

$$H^\ell = W^{\ell,-} G^{\ell-1} + W^{\ell,+} H^{\ell-1}, \qquad G^\ell = \max\left\{W^{\ell,+} G^{\ell-1} + W^{\ell,-} H^{\ell-1} + b^\ell, H^\ell\right\}, \tag{4}$$

where the maximum is taken componentwise.

Proof. We proceed by induction on the layer index ℓ. We show that each activation z^ℓ can be written as

$$z^\ell(x) = G^\ell(x) - H^\ell(x),$$

where every coordinate of G^ℓ and H^ℓ is convex.

For the first layer, define

$$G^1(x) = \mathrm{ReLU}(W^1 x + b^1), \qquad H^1(x) = 0.$$

We can clearly see that $z^1(x)$ admits a DC decomposition as

$$z^1(x) = \mathrm{ReLU}(W^1 x + b^1) = G^1(x) - H^1(x).$$

We now assume that

$$z^{\ell-1}(x) = G^{\ell-1}(x) - H^{\ell-1}(x),$$

where $G^{\ell-1}$ and $H^{\ell-1}$ are componentwise convex. We decompose W^ℓ entrywise as

$$W^\ell = W^\ell_+ - W^\ell_-, \qquad W^\ell_+, W^\ell_- \geq 0,$$

and define

$$P^\ell(x) = W^\ell_+ G^{\ell-1}(x) + W^\ell_- H^{\ell-1}(x) + b^\ell,$$
$$Q^\ell(x) = W^\ell_- G^{\ell-1}(x) + W^\ell_+ H^{\ell-1}(x).$$

The preactivation a^ℓ can be written as

$$
\begin{aligned}
a^\ell(x) &= W^\ell z^{\ell-1}(x) + b^\ell \\
&= P^\ell(x) - Q^\ell(x).
\end{aligned}
$$

We can use the identity $\mathrm{ReLU}(p-q) = \max\{p,q\} - q$ to obtain

$$
\begin{aligned}
z^\ell(x) &= \mathrm{ReLU}\big(P^\ell(x) - Q^\ell(x)\big) \\
&= \max\big\{P^\ell(x), Q^\ell(x)\big\} - Q^\ell(x).
\end{aligned}
$$

Therefore, setting

$$
G^\ell(x) = \max\big\{P^\ell(x), Q^\ell(x)\big\}, \qquad H^\ell(x) = Q^\ell(x),
$$

yields the desired DC representation. □

A.2 Proof of Theorem 4.2

Theorem 4.2 (Extended Representation of DC Components)**.** *For any input x, set $u^0 = x$ and define the set*

$$
\mathcal{U}(x) = \left\{ u = (u^1, \dots, u^L) \;\middle|\; \begin{array}{ll} u^\ell \geq W^\ell u^{\ell-1} + b^\ell, & \ell = 1, \dots, L \\ u^\ell \geq 0, & \end{array} \right\}, \tag{12}
$$

where all inequalities are componentwise. Let $\overline{G}_r(u)$ and $\overline{H}_r(u)$ be the affine expressions obtained from Eq. 8 by replacing each $z^\ell(x)$ with u^ℓ. Then, for every input x,

$$
G_r(x) = \min_{u \in \mathcal{U}(x)} \overline{G}_r(u), \qquad H_r(x) = \min_{u \in \mathcal{U}(x)} \overline{H}_r(u). \tag{13}
$$

Proof. We first prove that every feasible $u \in \mathcal{U}(x)$ satisfies

$$
\overline{G}_r(u) \geq G_r(x), \quad \overline{H}_r(u) \geq H_r(x) \tag{20}
$$

We define

$$
\begin{aligned}
&\widetilde{\eta}^1 = 0 \\
&\widetilde{\eta}^\ell = W^{\ell,-} u^{\ell-1} + |W^\ell| \widetilde{\eta}^{\ell-1}, \quad \ell = 2, \dots, L
\end{aligned}
$$

and

$$
\widetilde{G}^\ell(x) = u^\ell + \widetilde{\eta}^\ell(x), \quad \widetilde{H}^\ell(x) = \widetilde{\eta}^\ell(x).
$$

To begin, we will show by induction that

$$
\begin{aligned}
\widetilde{G}^\ell(x) &\geq G^\ell(x) \\
\widetilde{H}^\ell(x) &\geq H^\ell(x)
\end{aligned} \tag{21}
$$

For the first layer, we take

$$
\begin{aligned}
&\widetilde{G}^1(x) = u^1 \geq \mathrm{ReLU}(W^1 x + b^1) = G^1(x) \\
&\widetilde{H}^1(x) = 0 = H^1(x).
\end{aligned}
$$

Now, we assume that $\widetilde{G}^{\ell-1}(x) \geq G^{\ell-1}(x)$ and $\widetilde{H}^{\ell-1}(x) \geq H^{\ell-1}(x)$.

We can define

$$
\begin{aligned}
\widetilde{P}^\ell(x) &= W^{\ell,+} \widetilde{G}^{\ell-1}(x) + W^{\ell,-} \widetilde{H}^{\ell-1}(x) + b^\ell \\
\widetilde{Q}^\ell(x) &= W^{\ell,-} \widetilde{G}^{\ell-1}(x) + W^{\ell,+} \widetilde{H}^{\ell-1}(x).
\end{aligned}
$$

Note that $\widetilde{P}^\ell(x) \geq P^\ell(x), \widetilde{Q}^\ell(x) \geq Q^\ell(x)$, and that $\widetilde{Q}^\ell(x) = \widetilde{\eta}^\ell(x) = \widetilde{H}^\ell(x)$. Thus, it follows that $\widetilde{H}^\ell(x) = \widetilde{Q}^\ell(x) \geq Q^\ell(x) = H^\ell(x)$.

Because $u^\ell \geq 0$, we have

$$\widetilde{G}^\ell(x) = u^\ell + \widetilde{\eta}^\ell(x) \geq \widetilde{Q}^\ell(x).$$

Because $u^\ell \geq W^\ell u^{\ell-1} + b^\ell$, we have

$$\begin{aligned}\widetilde{G}^\ell(x) &\geq W^\ell u^{\ell-1} + \widetilde{\eta}^\ell + b^\ell \\ &= \widetilde{P}^\ell(x)\end{aligned}$$

Thus, we have

$$\begin{aligned}&\widetilde{G}^\ell(x) \geq \max\left(\widetilde{P}^\ell(x), \widetilde{Q}^\ell(x)\right) \\ \implies &\widetilde{G}^\ell(x) \geq G^\ell(x).\end{aligned}$$

This completes the induction, proving (21). Now, recall that

$$\overline{G}_r(u) = \beta_r + (\nu_r^+)^\top u^L + \sum_{\ell=1}^{L-1} c_{r,\ell}^\top u^\ell$$

We can rewrite $\overline{G}_r(u)$ and $G_r(u)$ as

$$\overline{G}_r(u) = \beta_r + (\nu_r^+)^\top \widetilde{G}^L + (\nu_r^-)^\top \widetilde{H}^L$$

and

$$G_r(x) = \beta_r + (\nu_r^+)^\top G^L + (\nu_r^-)^\top H^L.$$

Combined with the result of the induction, we can clearly see that $\overline{G}_r(u) \geq G_r(x)$. A similar argument shows that $\overline{H}_r(u) \geq H(x)$.

The above argument has proven (20). We now show that there exists a feasible u that attains this bound. Consider the choice

$$u^\ell = z^\ell(x) = \mathrm{ReLU}\left(W^\ell z^{\ell-1}(x) + b^\ell\right).$$

This is clearly feasible. Thus, with this assignment, we can see that $\overline{G}_r(u)$ attains the same value as $G_r(x)$, and $\overline{H}_r(u)$ attains the same value as $H_r(x)$. □

B Baseline Methods

B.1 Big-M Formulation

For a ReLU neural network, consider the preactivations $a^\ell(x) \in \mathbb{R}^{n_\ell}$ of layer ℓ. Let $a_j^\ell(x)$ denote the j-th component (neuron) of the preactivation $a^\ell(x)$, with activation $z_j^\ell(x) = \max\{a_j^\ell(x), 0\}$.

Assuming we have valid bounds on the preactivations given by

$$L_j^\ell \leq a_j^\ell(x) \leq U_j^\ell,$$

we can represent the graph of the ReLU exactly by

$$\begin{aligned}&z_j^\ell(x) \geq a_j^\ell(x), \quad z_j^\ell(x) \leq a_j^\ell(x) - L_j^\ell\left(1 - \delta_j^\ell\right) \\ &z_j^\ell(x) \geq 0, \quad z_j^\ell(x) \leq U_j^\ell \delta_j^\ell \\ &\delta_j^\ell \in \{0, 1\}.\end{aligned}$$

B.2 Big-M + Optimization Based Bound Tightening (OBBT)

OBBT strengthens the formulation in Appendix B.1 by tightening the preactivation bounds before optimization (Grimstad & Andersson, 2019). Starting from valid bounds, we process the hidden layers in order. For layer ℓ, let $\mathcal{F}_{\ell-1}$ denote the feasible set defined by the input constraints and

the constraints representing earlier layers, using their current bounds. With $z^0 = x$, we solve the following auxiliary problems:

$$\begin{aligned} \widehat{L}_j^\ell &\le \min_{(x,z)\in\mathcal{F}_{\ell-1}} \left(W_{j:}^\ell z^{\ell-1} + b_j^\ell\right), \\ \widehat{U}_j^\ell &\ge \max_{(x,z)\in\mathcal{F}_{\ell-1}} \left(W_{j:}^\ell z^{\ell-1} + b_j^\ell\right), \end{aligned} \tag{22}$$

and we update the preactivation bounds as:

$$L_j^\ell \leftarrow \max\{L_j^\ell, \widehat{L}_j^\ell\}, \qquad U_j^\ell \leftarrow \min\{U_j^\ell, \widehat{U}_j^\ell\}. \tag{23}$$

The resulting activation bounds are carried into the auxiliary problems for subsequent layers. A neuron requires no activation binary when $U_j^\ell \le 0$ or $L_j^\ell \ge 0$, since its output is then fixed to zero or its preactivation, respectively. Our implementation uses a single complete forward pass, with a time-limited solve for each minimization and maximization problem. Even with a 0.1 second limit per solve, the preprocessing time for OBBT can be substantial. For the water treatment instance with 800 samples, a complete forward pass requires $153,600$ auxiliary solves, taking up to 256 minutes (more than $85\times$ the evaluation budget).

B.3 Projected Gradient Descent

For the continuous QP benchmark, PGD projects onto the convex residual feasible set $\mathcal{X}$ and penalizes the neural inequalities as

$$\begin{aligned} P_\rho(x) &= \frac{1}{2}\|Gx\|_2^2 + c^\top x + \frac{\rho}{2}\sum_{r=1}^{30}[f_\theta(x^{(r)})]_+^2, \\ x^{k+1} &= \Pi_{\mathcal{X}}(x^k - \alpha\nabla P_\rho(x^k)). \end{aligned} \tag{24}$$

We fix $\alpha = 0.1$ and $\rho = 100$ across architectures and trials. We formulate and solve the projection step in Gurobi as a QP. Between consecutive neural feasible and neural infeasible iterates, we search along the line segment between them for a better feasible solution.

C Experiment Details

Within each instance, all methods use the same fixed neural surrogate, residual model, and initial feasible incumbent. Solvers run single-threaded, and performance is evaluated over a $180s$ horizon. The final exact baseline solves use `MIPFocus=1` in Gurobi to prioritize finding feasible incumbents.

We use the same OBBT protocol across all benchmarks, sizes, and trials. Bound-tightening of the big-M formulation precedes the $180s$ optimization horizon, and its time cost is not counted towards the solution time. This grants the tightened baseline additional preprocessing time followed by its full optimization budget.

For `DCEmbed`, we use the penalty update and objective stopping rule in Section 4 with $\mu = 10$, $\tau_{\max} = 10^6\tau_0$, and $\epsilon_{\text{obj}} = 10^{-6}$. The neural feasibility tolerance is 10^{-5} for the QP and water experiments. Neur2SP embeds only a neural objective. Initial penalties are $\tau_0 = 100$ for the QPs, and $\tau_0 = 10$ for the water experiments. These settings are fixed across problem sizes and trials within each benchmark.

C.1 Neural Surrogate Scaling in Random QPs

The residual model has 150 variables $x \in [-1, 1]^{150}$, 18 linear equalities, and 150 linear inequalities. We use $G \in \mathbb{R}^{132\times150}$, $E \in \mathbb{R}^{18\times150}$, and $C \in \mathbb{R}^{75\times150}$, sampling their entries and the linear objective coefficients independently as

$$G_{ij} \sim \mathcal{N}\left(0, \frac{1}{132}\right), \qquad E_{ij}, C_{ij} \sim \mathcal{N}\left(0, \frac{1}{150}\right), \qquad c_i \sim \mathcal{N}\left(0, \frac{1}{150^2}\right). \tag{25}$$

Each trial fixes the residual model across architectures, and all methods start from the feasible point $x^0 = 0$.

Each surrogate approximates the same nonconvex function on $[-1, 1]^5$:

$$q(u) = 0.30u_1^2 + 0.20u_2^2 + 0.15u_3^2 + 0.15u_4^2 + 0.10u_5^2 + 0.10u_1u_4 + 0.08u_2u_5 + 0.08\big(1 - \cos(\pi u_3)\big) - 0.15. \tag{26}$$

We train the neural surrogate with varying architectures of different depths and widths. The surrogate is trained by minimizing mean-squared error using $20,000$ iterations of Adam and a minibatch size of $4,000$. The training data is uniformly sampled from $[-1, 1]^5$. The learning rate decays from 2×10^{-3} to 10^{-5}, and the checkpoint with the lowest validation error is retained. The three trials vary residual model and training seeds, and within each trial, architectures share the same training data.

C.2 Scenario Scaling in Two-Stage Stochastic Programming

We use the pooling problem and released *NN-P* architecture from Dumouchelle et al. (2022), to which we refer the reader for the original problem and training details. No additional training is performed on the *NN-P* model. We follow the benchmark's scenario discretization with $m \in \{12, \dots, 17\}$ points for each of its three uncertain quantities, giving $S = m^3$ scenarios. The original profit objective is negated to obtain the cost minimization convention used here.

To provide the same initialization to every method, we construct a pool of eight feasible first-stage decisions. To construct the pool, we optimize the first-stage linear objective and a randomly sampled Gaussian linear objective over the original constraints. We evaluate the surrogate objective at these designs and retain the best five. `DCEmbed` uses them as sequential starts, while the exact formulations receive the same designs as MIP starts. Thus all methods receive identical candidate designs and screening information for initialization. We enumerate all the feasible first-stage decisions offline to establish the global optimum of the surrogate problem for evaluation.

C.3 Resource Allocation for Water Treatment

The Water Potability dataset pairs nine water quality measurements (pH, hardness, dissolved solids, chloramines, sulfate, conductivity, organic carbon, trihalomethanes, and turbidity) with a binary potability label. We train a classifier with four hidden ReLU layers of width 32 on $2,011$ training points. Each feature is normalized by subtracting its mean and dividing by its standard deviation. Training uses binary cross-entropy with logits and Adam for 80 epochs, with a batch size of 128, learning rate of 0.005, and weight decay of 0.001. The network and normalization are then fixed for all experiments.

The network outputs a logit, for which zero corresponds to a predicted potability probability of 0.5. We count a treated sample only when its score reaches the small positive margin $\epsilon = 10^{-4}$. Candidate water samples are records labeled non-potable that do not already satisfy this score requirement before treatment. For each of the sampling seeds, we shuffle the eligible records once and use the first N as the candidate set. Addition and removal budgets follow the SurrogateLIB generator and remain fixed as N increases. All methods start from the untreated configuration with no samples counted as potable.

D Additional Results

D.1 Solution Trajectories

NPI combines solution quality and time into a single metric. To examine these effects separately, we show feasible incumbent trajectories in Figure 3 at the largest tested sizes for the experiments in Section 5. For the QP experiment, PGD improves rapidly before stalling, while `DCEmbed` continues to improve and reaches better final objectives. `DCEmbed` also makes substantially faster progress than the two-path formulation, while both exact embeddings remain at the initial incumbent. In water treatment, `DCEmbed` finds strong incumbents early, whereas the exact embeddings make little or no progress. For Neur2SP, `DCEmbed` reaches the same optimum as Gurobi ML substantially earlier, while both big-M variants stall at suboptimal designs.

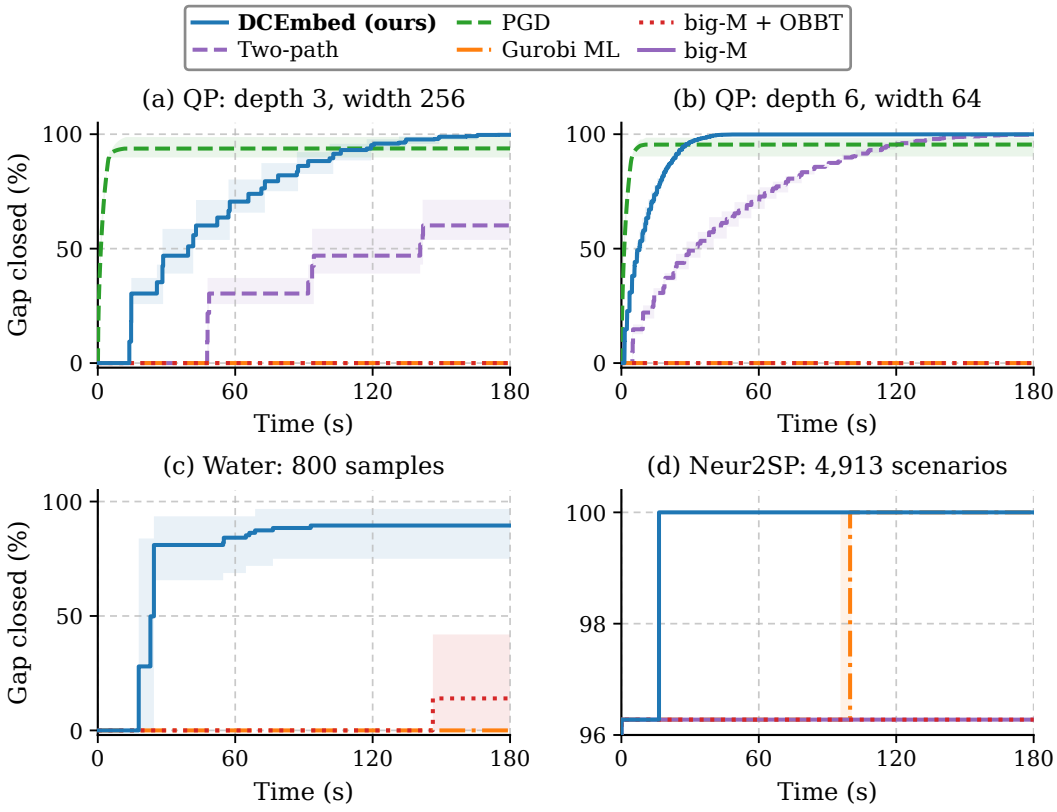

Figure 3: Primal solution trajectories showing gap closed over time (higher is better).

D.2 Final Solution Quality

Tables 1–3 report the best feasible objectives reached within the evaluation budget. The best objectives, including ties, are **bolded** in the tables below.

Table 1: Synthetic QP objectives (lower is better). Values are averaged over three trials, with zero indicating the initial incumbent objective.

Depth	Width	DCEmbed (ours)	Two-path	PGD	Gurobi ML	big-M + OBBT
1	64	**−0.06527**	**−0.06527**	−0.05990	**−0.06527**	**−0.06527**
2	64	**−0.06522**	**−0.06522**	−0.06014	0.00000	−0.00216
3	8	**−0.06531**	−0.06529	−0.06315	**−0.06531**	**−0.06531**
3	16	**−0.06532**	**−0.06532**	−0.06203	−0.03866	**−0.06532**
3	32	**−0.06549**	−0.06547	−0.06267	0.00000	0.00000
3	64	−0.06526	**−0.06527**	−0.06036	0.00000	0.00000
3	128	−0.06520	**−0.06522**	−0.05996	0.00000	0.00000
3	256	**−0.06487**	−0.03864	−0.06059	0.00000	0.00000
4	64	**−0.06535**	−0.06504	−0.06007	0.00000	0.00000
5	64	**−0.06522**	−0.06455	−0.06162	0.00000	0.00000
6	64	**−0.06494**	−0.06481	−0.06174	0.00000	0.00000

Table 2: Number of potable water samples (higher is better), averaged over three trials.

Samples	DCEmbed (ours)	Gurobi ML	big-M + OBBT
25	**10.00**	**10.00**	9.33
50	**20.33**	15.33	15.00
100	**26.00**	17.67	13.33
200	**26.33**	8.33	12.67
400	**30.33**	1.33	8.00
800	**31.33**	0.00	4.33

Table 3: Neur2SP objective (higher is better), averaged over three trials.

Scenarios	DCEmbed (ours)	Gurobi ML	big-M	big-M + OBBT
1,728	**171.5118**	**171.5118**	**171.5118**	**171.5118**
2,197	**169.7280**	**169.7280**	**169.7280**	**169.7280**
2,744	**169.9166**	**169.9166**	**169.9166**	161.8368
3,375	**169.9523**	**169.9523**	161.5369	**169.9523**
4,096	**169.1168**	**169.1168**	161.4644	161.4644
4,913	**169.4088**	**169.4088**	161.2606	161.2606