BINARY OPTIMIZATION OVER TRAINED NEURAL NETWORKS: PARALLEL SAMPLING ON AN ANNEALED LANDSCAPE

Anonymous authors
Paper under double-blind review

ABSTRACT

In constraint learning, optimization problems are formulated with an objective function or constraints that are learned from data and represented as pretrained neural networks. Although promising as a modeling paradigm, this approach has scalability challenges that have been progressively addressed with tailored algorithms. However, those algorithms assume that the decision variables are continuous. In this work, we introduce CLEBO (Constraint Learning via Energy-Based Optimization), to our knowledge the first algorithm tailored for constraint learning over binary domains. CLEBO performs massively parallel first-order optimization using an annealed relaxation inducing binary variables toward binary values. We present two variants that differ in how neural gradients are computed. The first is a relaxed estimator that differentiates through the neural network. The second is a discrete sampling estimator that avoids unreliable gradients by not evaluating the network at non-binary inputs. Compared to other solving methods, CLEBO also stands out for being a GPU-native sampler that is agnostic to both the network architecture and its activation functions. Experiments on three benchmark problems demonstrate that CLEBO computes solutions that are competitive with exact mixed-integer programming solvers up to the sizes that such solvers can handle. For larger networks in which finding the optimal solution is computationally prohibitive, CLEBO often finds higher-quality solutions in substantially less time.

1 INTRODUCTION

A motivating application. Consider the design of a fragrant molecule as in Zhang et al. (2024): a chemist assembles molecular fragments into a graph so as to maximize the probability that it smells of banana, where that probability is given by a *trained* graph neural network (GNN) fit to labeled molecules, while respecting the rules of chemical validity (valences, bond counts, acyclicity). The objective here is not a closed-form function but a neural network learned from data, whereas the constraints are explicit combinatorial rules.

Constraint learning. The above example is an instance of *constraint learning* (CL): an optimization problem in which part of the objective or constraints is a pretrained neural network rather than an analytic expression (Kolb, 2016; Lombardi & Milano, 2018; Bergman et al., 2022; Fajemisin et al., 2024). Specialized methods apply when the problem has exploitable structure. With a continuous decision space, the network can be optimized directly via gradient descent (Schweidtmann & Mitsos, 2019), and tools such as MathOptAI.jl support embedding trained models into continuous JuMP formulations (Dowson et al., 2026). Alternatively, local search can be performed over the network's linear regions (Perakis & Tsiourvas, 2022; Tong et al., 2024; 2026). However, none of these are directly applicable if the optimization problem has binary decision variables.

We may resort to the general-purpose route of embedding the trained network into a Mixed-Integer Linear Program (MILP). Piecewise-linear activations such as the ReLU (Nair & Hinton, 2010; Glorot et al., 2011) admit exact formulations through auxiliary binary variables and linear constraints (Vielma et al., 2010), including strong and partition-based formulations developed for trained ReLU networks (Anderson et al., 2020; Tsay et al., 2021). This was observed independently across the planning (Say et al., 2017), verification (Cheng et al., 2017), deep learning (Serra et al., 2018),

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.

and optimization (Fischetti & Jo, 2018) communities, and has since been automated by tools such as OMLT (Ceccon et al., 2022) and OptiCL (Maragno et al., 2025). But this approach has two limitations. First, the encoding is architecture-specific, placing modern architectures out of reach since attention and general message-passing layers have no exact MILP formulation. Second, it scales poorly. Each hidden unit adds an auxiliary binary variable with a big-M constraint, which is known to provide weak relaxations (Balas, 1985), so the embedded model becomes intractable well before the network is large enough.

Rounding, annealing, and sampling for binary optimization. Most optimization courses raise one question before discussing integer programming: *why not just round?* Solve a *relaxation* that ignores integrality, and if an optimum happens to be integer, we are done and the solution is provably optimal. But when it is fractional, rounding to the nearest integer may violate constraints and, even when it does not, optimality is lost. This is no minor gap: linear programs solve in polynomial time (Khachiyan, 1980; Karmarkar, 1984) while integer programs are NP-hard (Karp, 1972; Garey & Johnson, 1979). To find optimal discrete decisions, integer programming (Schrijver, 1986; Wolsey, 1998; Conforti et al., 2014) is often used, since rounding can lead to poor-quality solutions.

An alternative approach is to avoid rounding the solution of a single relaxation and instead let a set of relaxations converge to binary values. Through *annealing*, the continuous relaxations are gradually driven toward discreteness, so that rounding at termination is negligible. A recent family of GPU-native solvers indeed casts combinatorial optimization as parallel stochastic search over relaxed or probabilistic discrete solutions (Sun et al., 2023; Sun & Yang, 2023; Shen et al., 2025; Ichikawa & Arai, 2025; Huang et al., 2026; Li et al., 2026), with the energy-based members using Boltzmann, Langevin, or free-energy formulations to bias the search toward feasible, low-cost discrete configurations. These methods typically exploit analytic problem structure, such as an Ising or quadratic unconstrained binary optimization (QUBO) Hamiltonian, a graphical structure, or linear constraints with an exploitable dual. None of them targets pretrained neural objectives or constraints, which is the setting we address.

This work. As a more scalable and reliable alternative, we propose CLEBO (Constraint Learning via Energy-Based Optimization). To our knowledge, this is the first heuristic algorithm tailored for optimization over trained neural networks with binary inputs. Our algorithm iterates in parallel over sequences of relaxations of the optimization model, denoting each of those sequences as a *chain*. Moreover, we combine (a) forward and backward passes on the neural network with (b) an *annealing* process that gradually drives decision variables toward binary values. Each of those elements depart from the baseline approach of directly using an MILP model. We make the following contributions:

1. **Generality.** CLEBO handles learned objectives and constraints without architecture-specific MILP encodings. It scales to larger networks and supports architectures beyond those amenable to exact MILP reformulation, such as graph attention networks, by accessing the network only through forward and backward passes.
2. **Novelty.** We introduce three complementary components. Firstly, our sampled gradient estimators evaluate learned components only at binary inputs, avoiding unreliable fractional-input gradients while retaining exact pathwise gradients for analytic terms. Secondly, we enforce structural constraints by construction, including the tree structure in molecular design through sequential parent selection. Finally, we handle the remaining constraints through scale-normalized penalties, with penalty weights searched in a single GPU run, so that no per-instance tuning is needed.
3. **Performance.** Across three benchmarks, CLEBO matches the MILP where the MILP is tractable and keeps working where it is not. On the largest random networks, the one-hour MILP often returns only the trivial all-zero solution, while CLEBO stays within 0.6% of the best-known solution in under two minutes. On the knapsack, where the MILP is strongest, CLEBO is feasible on every instance and within 0.9% of the best-known objective in a fraction of the MILP's time. On molecular design, it matches the proven optimum wherever the MILP establishes one and, at the largest size, exceeds the one-hour incumbent by a wide margin.

Organization. Section 1 serves as both the introduction and the discussion of related work. Section 2 presents the problem setting and the background on constraint learning and energy-based annealing. Section 3 develops our energy formulation, gradient estimators, and implementation details. Section 4 describes the experimental setup and reports results across the three benchmarks. Finally, Section 5 summarizes the findings and outlines directions for future work.

2 BACKGROUND

In this section, we provide the background on constraint learning, continuous relaxations, and energy-based optimization for combinatorial optimization (CO).

Constraint learning. We consider constrained binary optimization problems of the form

$$\min_{\mathbf{x}\in\{0,1\}^n} \; f_\theta(\mathbf{x}) \quad \text{s.t.} \quad g_j(\mathbf{x}) \le 0,\; j \in \{1,\dots,m\}, \tag{1}$$

where $f_\theta : \mathbb{R}^n \to \mathbb{R}$ is a learned objective and $g_j : \mathbb{R}^n \to \mathbb{R}$, $j \in \{1,\dots,m\}$, are explicit analytic constraints. The latter are typically linear inequalities. Learned constraints, as in our knapsack benchmark, are handled similarly (see Appendix B.1 for details). Each learned component is a pretrained neural network treated as a black box, accessible only via forward evaluation and gradients, with no assumptions on its architecture.

Continuous relaxations & energy-based optimization for CO. A standard approach for binary optimization with analytic constraints and an analytic objective, $f : \mathbb{R}^n \to \mathbb{R}$, is to first rewrite the problem as an unconstrained problem via penalties, i.e.,

$$c(\mathbf{x}) = f(\mathbf{x}) + \sum_{j=1}^{m} \lambda_j [g_j(\mathbf{x})]_+, \tag{2}$$

where $[v]_+ = \max(0, v)$ denotes the positive part and $\lambda_1, \dots, \lambda_m \in \mathbb{R}_{\ge 0}$ are hyperparameters. Such a problem is solved by relaxing the binary variables $\mathbf{x} \in \{0,1\}^n$ to $\mathbf{p} \in [0,1]^n$ and minimizing a differentiable surrogate $c(\mathbf{p})$ by gradient descent, rounding the resulting solution. However, a plain relaxation tends to stall in poor local optima with fractional values that are not convenient to round.

Continuous Relaxation Annealing (CRA) (Ichikawa, 2024) addresses these limitations by gradually transforming a smooth relaxation, so that it converges toward a discrete solution. A single annealing trajectory can nevertheless remain sensitive to its initialization and converge to one local mode. Parallel Quasi-Quantum Annealing (PQQA) (Ichikawa & Arai, 2025) addresses this limitation by evolving in parallel many trajectories, which we denote as chains; injecting Langevin noise to explore the landscape; and rewarding diversity to prevent the chains from collapsing to the same solution.

Let $\mathbf{P} = \{\mathbf{p}^s\}_{s=1}^S$ denote a *population* of S solutions with binary variables relaxed. We use $\mathbf{p}$ for a generic chain associated with one such member of the population and retain the superscript s whenever an expression depends on a particular member. PQQA minimizes the population energy

$$E(\mathbf{P}; \gamma, \rho, \alpha) = \sum_{s=1}^{S} \left[c(\mathbf{p}^s) + \gamma \phi(\mathbf{p}^s, \alpha) \right] - S\rho\, \sigma(\mathbf{P}), \tag{3}$$

where the sharpening and diversity terms are, respectively,

$$\begin{aligned} \phi(\mathbf{p}^s, \alpha) &= \sum_{i=1}^{n} \left(1 - (2p_i^s - 1)^\alpha\right), \\ \sigma(\mathbf{P}) &= \sum_{i=1}^{n} \max\left\{ \sqrt{\frac{\sum_{s=1}^{S} (p_i^s - \mu_i(\mathbf{P}))^2}{S}}, 10^{-8} \right\}, \end{aligned} \tag{4}$$

where $\mu_i(\mathbf{P}) = \sum_{s=1}^{S} p_i^s / S$ is the coordinate-wise population mean. The hyperparameter α is an even integer, and the scalar γ is annealed over $T_{\max}$ steps of a run along an increasing schedule $\gamma_1, \dots, \gamma_{T_{\max}}$. At $\gamma < 0$, the term rewards $p_i \approx \frac{1}{2}$, convexifying the landscape and smoothing away spurious minima. As γ crosses zero, it penalizes fractional values and drives $\mathbf{p} \to \{0,1\}^n$, so that first-order iterations land near discrete solutions. A final thresholding, $x_i = \mathbf{1}[p_i \ge \frac{1}{2}]$, produces each candidate solution. The diversity term $\sigma(\mathbf{P})$ sums population standard deviations over the coordinates. Its negative coefficient in Eq. (3) rewards chains for occupying different regions of the search space, and ρ controls the strength of this interaction. PQQA applies an AdamW (Loshchilov & Hutter, 2019) descent step, adds Langevin noise, and projects each chain onto the hypercube using operator Π:

$$\mathbf{p}^s \leftarrow \Pi_{[0,1]}\big(\mathrm{AdamW}(\mathbf{p}^s, \nabla_{\mathbf{p}^s} E) + \sqrt{2\eta T}\boldsymbol{\xi}^s\big) \tag{5}$$

where $\boldsymbol{\xi}^s \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, η is the AdamW learning rate, and T is the Langevin temperature. PQQA is *per-instance* and *training-free*, fitting no policy and requiring no labeled solutions. Aside from thresholding, it also requires no separate feasibility-repair heuristic.

3 METHODOLOGY

CLEBO extends the energy-based sampling method from Section 2 to solve binary constraint learning problems. Specifically, we develop a reliable gradient estimator for the learned component to iterate over the relaxed domain $\mathbf{p} \in [0,1]^n$, even when the learned component has been trained on discrete inputs. We detail the method for handling a learned objective f_θ as in (1). The learned constraint is handled similarly as a penalty of the differentiable surrogate $c(\mathbf{p})$ (see Appendix B.1 for details).

3.1 ENERGY-BASED OPTIMIZATION FOR CL

CLEBO instantiates the per-chain cost $c(\mathbf{p})$ of Eq. (3) with the penalty construction of Eq. (2), but with the trained network as the objective, i.e.,

$$c(\mathbf{p}) = f_\theta(\mathbf{p}) + \sum_{j=1}^{m} \lambda_j \, [g_j(\mathbf{p})]_+, \tag{6}$$

with f_θ replaced by $-f_\theta$ for maximization problems. Replacing the analytic objective f with the learned f_θ is what distinguishes this setting from Eq. (2). The analytic terms take meaningful values everywhere on the cube, whereas f_θ, although defined there, is typically fit only on $\{0,1\}^n$. Eq. (6) therefore queries the network off its training support, which is the issue addressed in Section 3.2.

The gradient of the population energy w.r.t. a chain can be computed as

$$\nabla_{\mathbf{p}} E = \underbrace{\widehat{\nabla} f_\theta(\mathbf{p})}_{\text{learned}} + \underbrace{\textstyle\sum_{j=1}^{m} \lambda_j \nabla_{\mathbf{p}} [g_j(\mathbf{p})]_+}_{\text{analytic (pathwise)}} + \gamma \nabla_{\mathbf{p}} \phi(\mathbf{p}, \alpha) - S\rho \nabla_{\mathbf{p}} \sigma(\mathbf{P}), \tag{7}$$

where the analytic penalties, sharpening term ϕ, and diversity term σ are evaluated on the relaxed variables and differentiated directly. Gradients are taken with respect to the coordinates of the chain being updated. The variants of CLEBO differ in how the gradient of the learned component $\widehat{\nabla} f_\theta$ is estimated. $\text{CLEBO}_{\text{base}}$ differentiates it pathwise on the relaxation, whereas the sampled variants estimate it from discrete samples to avoid evaluation outside the training manifold.

3.2 GRADIENT ESTIMATORS

The two estimators arise from two interpretations of $\mathbf{p}$. Under the relaxation view, $\mathbf{p}$ is a continuous solution and the estimator is the network gradient evaluated directly at $\mathbf{p}$. Under the distributional view, $\mathbf{p}$ parameterizes a distribution $q_{\mathbf{p}}$ over $\{0,1\}^n$, corresponding to independent Bernoulli variables, with probability mass function $q_{\mathbf{p}}(\mathbf{x}) = \prod_{i=1}^{n} p_i^{x_i}(1-p_i)^{1-x_i}, \mathbf{x} \in \{0,1\}^n$, and the gradient is estimated using samples drawn from $q_{\mathbf{p}}$.

Relaxed estimator ($\text{CLEBO}_{\text{base}}$). In this case, we set the gradient $\widehat{\nabla} f_\theta(\mathbf{p}) = \nabla_{\mathbf{p}} f_\theta(\mathbf{p})$. It requires only a forward and backward pass. However, it provides useful guidance for the discrete problem only when the network's behavior on fractional inputs matches its behavior on binary inputs, which is a condition unlikely to hold when $\mathbf{p}$ lies outside the model's training distribution.

Sampled estimator (CLEBO_{re}, $\text{CLEBO}_{\text{loo}}$). The distributional view evaluates f_θ only on discrete samples and does not differentiate through the network. For each chain, we draw K samples and baseline each one against the mean of the other $K-1$, i.e., a leave-one-out (LOO) baseline (Kool et al., 2019):

$$\widehat{\nabla} f_\theta(\mathbf{p}) = \frac{1}{K} \sum_{k=1}^{K} \Big(f_\theta(\mathbf{x}^{(k)}) - b_k \Big) \nabla_{\mathbf{p}} \log q_{\mathbf{p}}(\mathbf{x}^{(k)}), \tag{8}$$

with $\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(K)}$ drawn independently from $q_{\mathbf{p}}$ and $b_k = \frac{1}{K-1} \sum_{\ell \neq k} f_\theta(\mathbf{x}^{(\ell)})$ for $K \geq 2$. This gives the LOO variant ($\text{CLEBO}_{\text{loo}}$), which spends K forward passes per chain per step to reduce variance. Because each b_k is independent of $\mathbf{x}^{(k)}$, the LOO estimate is unbiased for $\nabla_{\mathbf{p}} \mathbb{E}_{\mathbf{x} \sim q_{\mathbf{p}}}[f_\theta(\mathbf{x})]$, the gradient of the expected discrete objective. CLEBO_{re} instead uses REINFORCE (Williams, 1992). Each chain draws a single sample $\mathbf{x}^s \sim q_{\mathbf{p}^s}$ and the baseline is the mean reward over the population, $\bar{b} = \frac{1}{S} \sum_{s'=1}^{S} f_\theta(\mathbf{x}^{s'})$, so that $\widehat{\nabla} f_\theta(\mathbf{p}^s) = \big(f_\theta(\mathbf{x}^s) - \bar{b}\big) \nabla_{\mathbf{p}} \log q_{\mathbf{p}^s}(\mathbf{x}^s)$. It costs one forward pass per chain per step. By contrast, Gumbel–softmax (Jang et al., 2017; Maddison et al., 2017) and

straight-through (Bengio et al., 2013) estimators still differentiate through the network at relaxed inputs and at a vertex, respectively, and thus depend on its continuous extension, which the sampled estimators never query.

Substituting Eq. (8) into Eq. (7) gives the gradient each chain follows under CLEBO_{re} and $\text{CLEBO}_{\text{loo}}$,

$$\widehat{\nabla}_{\mathbf{p}} E \;=\; \underbrace{\tfrac{1}{K}\sum_{k=1}^{K}\big(f_\theta(\mathbf{x}^{(k)}) - b_k\big)\nabla_{\mathbf{p}} \log q_{\mathbf{p}}(\mathbf{x}^{(k)})}_{\text{sampled (LOO)}} + \underbrace{\textstyle\sum_{j=1}^{m} \lambda_j \, \nabla_{\mathbf{p}}[g_j(\mathbf{p})]_+}_{\text{pathwise}} + \gamma \nabla_{\mathbf{p}} \phi(\mathbf{p}, \alpha) - S\rho \nabla_{\mathbf{p}} \sigma(\mathbf{P}),$$

with the sharpening and diversity terms unchanged from Eq. (7). Only the learned term is sampled. The analytic penalties keep their exact pathwise gradient, which is deterministic and nonzero whenever $\mathbf{p}$ violates a constraint, and scoring them as well would add sampling variance to a gradient already known exactly. The sampled variants therefore descend $\mathbb{E}_{\mathbf{x}\sim q_{\mathbf{p}}}[f_\theta(\mathbf{x})]$ plus the remaining relaxed terms of Eq. (6). Algorithm 1 summarizes the resulting method.

Comparing the estimators. The relaxed estimator can fail when the network's continuous extension does not preserve the ordering of nearby discrete solutions. For example, an off-manifold slope at a fractional $\mathbf{p}$ may point toward decreasing $f_\theta(\mathbf{p})$ while shifting $q_{\mathbf{p}}$ toward worse binary solutions. The optimizer then improves an artifact of the relaxation rather than the discrete objective. The sampled estimator avoids this failure mode because it compares network values only at binary samples. Better samples receive greater relative probability through the score-function gradient, even when the relaxed slope between them points in the wrong direction. This robustness comes at a cost, since LOO requires K forward evaluations per chain and its gradient carries sampling variance. The relaxed estimator, by contrast, needs only one forward and backward pass and is deterministic. It is therefore preferable when the network's continuous extension agrees with the discrete objective, as can happen for smooth models trained with good domain coverage. Sampled gradients are most useful when this agreement is doubtful or absent. Appendix D isolates this effect on a synthetic benchmark whose relaxation reliability ranges from near-perfect to adversarial, measured as the rank correlation between the network's values at fractional points and at their roundings.

Algorithm 1 CLEBO.

Require: Network f_θ, constraints $g_1, \ldots, g_m$, weights $\lambda_1, \ldots, \lambda_m$, chains S, steps $T_{\max}$, schedule $(\gamma_t)_{t=1}^{T_{\max}}$, estimator $\in \{\text{CLEBO}_{\text{base}}, \text{CLEBO}_{\text{re}}, \text{CLEBO}_{\text{loo}}\}$, samples K
Ensure: best feasible binary solution $\mathbf{x}^\star$

$\mathbf{p}^s \leftarrow \Pi_{[0,1]}\big(\frac{1}{2}\mathbf{1} + 0.01\,\boldsymbol{\xi}^s\big)$ for $s = 1, \ldots, S$ ▷ initialize population
for $t = 1, \ldots, T_{\max}$ **do**
  for each chain s **in parallel do**
    if estimator is $\text{CLEBO}_{\text{base}}$ **then**
      $\widehat{\nabla} f_\theta(\mathbf{p}^s) \leftarrow \nabla_{\mathbf{p}} f_\theta(\mathbf{p}^s)$ ▷ backpropagate at $\mathbf{p}^s$
    else if estimator is CLEBO_{re} **then**
      Sample $\mathbf{x}^s \sim q_{\mathbf{p}^s}$ and set $b \leftarrow \frac{1}{S}\sum_{s'} f_\theta(\mathbf{x}^{s'})$ ▷ population mean
      $\widehat{\nabla} f_\theta(\mathbf{p}^s) \leftarrow \big(f_\theta(\mathbf{x}^s) - b\big)\nabla_{\mathbf{p}} \log q_{\mathbf{p}^s}(\mathbf{x}^s)$
    else
      Sample $\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(K)} \sim q_{\mathbf{p}^s}$ and set $b_k \leftarrow \frac{1}{K-1}\sum_{\ell \neq k} f_\theta(\mathbf{x}^{(\ell)})$ ▷ leave-one-out
      $\widehat{\nabla} f_\theta(\mathbf{p}^s) \leftarrow \frac{1}{K}\sum_k \big(f_\theta(\mathbf{x}^{(k)}) - b_k\big)\nabla_{\mathbf{p}^s} \log q_{\mathbf{p}^s}(\mathbf{x}^{(k)})$
    $\nabla_{\mathbf{p}^s} E \leftarrow \widehat{\nabla} f_\theta(\mathbf{p}^s) + \sum_j \lambda_j \nabla_{\mathbf{p}^s}[g_j(\mathbf{p}^s)]_+ + \gamma_t \nabla_{\mathbf{p}^s} \phi(\mathbf{p}^s, \alpha) - S\rho \nabla_{\mathbf{p}^s} \sigma(\mathbf{P})$
  $\mathbf{p}^s \leftarrow \Pi_{[0,1]}\big(\text{AdamW}(\mathbf{p}^s, \nabla_{\mathbf{p}^s} E) + \sqrt{2\eta T}\,\boldsymbol{\xi}^s\big)$ for all s ▷ Eq. (5)
$\mathbf{x}^s \leftarrow \mathbf{1}[\mathbf{p}^s \geq \frac{1}{2}]$ for all s ▷ round
return the feasible $\mathbf{x}^s$ with the best objective

3.3 IMPLEMENTATION DETAILS

Selecting the penalty weights $\lambda_1, \ldots, \lambda_m$ is critical for balancing objective quality and feasibility, yet tuning them separately for each instance can be costly. We address this challenge systematically through scale normalization and parallel computation over a set of hyperparameters.

Scale normalization. Objective values and constraint violations can differ by orders of magnitude. For example, the knapsack capacity slack is on the order of thousands, whereas the molecular-design objective is on the order of tens. A single unnormalized penalty weight therefore cannot balance objective quality and feasibility consistently across constraints and instances. We instead scale each constraint penalty to the magnitude of the objective. Let β_0 and β_j denote the standard deviations of the objective and the j-th constraint, respectively. We set $\lambda_j = \lambda\,\beta_0/\beta_j$, where $\lambda \in \mathbb{R}_{>0}$ becomes a single parameter that controls the objective–feasibility trade-off for all constraints. The same normalized weight applies to a penalty whether it is differentiated pathwise or, for a learned constraint, estimated through the score function (Appendix B.1).

Each scale is estimated on inputs for which the corresponding component (objective or constraint) is reliable. For learned components, we use the standard deviation measured on the training distribution, thereby avoiding off-support evaluations on arbitrary fractional or infeasible inputs. For analytic components, we use a deterministic, seeded sample of binary inputs. If a constraint is nearly constant on this sample, its scale cannot be estimated reliably, and we retain the unnormalized weight λ.

Parallel penalty grid. With scale normalization, a single λ controls the trade-off between objective quality and feasibility, and its best value can vary across instances. To avoid per-instance tuning, we evaluate a fixed set of penalty weights $\Lambda = \{\lambda^{(1)}, \ldots, \lambda^{(|\Lambda|)}\}$. At termination, the solution from each chain is rounded via $x_i = \mathbf{1}[p_i \geq 0.5]$, and we report the best feasible rounded solution over all chains and penalty values. We evaluate all penalty values in a single GPU batch, partitioning the chains into independent blocks, one per $\lambda \in \Lambda$. Each block holds S chains, and diversity, gradient clipping, and estimator baselines are computed within each block only. Every block therefore retains the behavior of a separate fixed-λ solve while benefiting from GPU parallelism. Section 4.1 and Table 2 (Appendix A) give the grid values and the remaining solver configuration.

Structural constraints. Some constraints require a group of binary variables to take exactly one active index, as in assignment problems. If each variable is sampled independently, the sampler wastes effort on patterns that violate them. We instead couple each such group through a softmax over its coordinates, so every sample activates exactly one index per group and rounding takes the largest coordinate. These constraints therefore hold by construction and never enter the penalty (Appendix B.2).

4 EXPERIMENTS

We evaluate CLEBO by addressing three research questions.

- **Q1**. How effectively does CLEBO optimize an unconstrained neural objective, and how does its performance scale with the size of a fully connected network?
- **Q2**. Can CLEBO retain high solution quality when the formulation includes hard combinatorial constraints or a learned constraint?
- **Q3**. Does CLEBO extend beyond fully connected networks to learned objectives represented by graph neural networks, including architectures for which no exact MILP encoding is available?

Together, these questions isolate the three capabilities central to our approach, namely scalable optimization of a learned function, penalty-based handling of feasibility, and independence from a particular neural architecture.

4.1 COMPUTATIONAL SETUP

All of our experiments were run on a Linux machine with an NVIDIA H100 GPU with 20 GB of memory, 6 CPU cores, and 16 GB of memory. The software environment uses Python 3.11, PyTorch 2.11 with CUDA 12.8, Gurobi 12, gurobi-machinelearning 1.5.6, and OMLT 1.2.2.

Benchmarks. We consider three benchmarks, each selected to address a distinct research question. Full benchmark details are provided in Appendix C.

- The *random-network* benchmark isolates unconstrained optimization of a neural objective, $\max_{\mathbf{x}\in\{0,1\}^n} f_\theta(\mathbf{x})$, and follows the standard setting used to evaluate local search over neural objectives (Perakis & Tsiourvas, 2022; Tong et al., 2024; 2026). We use $n \in \{100, 1000\}$ and

ReLU *Multi-Layer Perceptrons (MLPs)* with depth–width configurations 2×100, 2×500, 4×100, and 4×500. Each configuration contains 10 independently generated networks. We also include an $n{=}100$, 1×50 configuration as an anchor.

- The *neural-constrained knapsack* benchmark augments a strongly correlated binary knapsack problem (Pisinger, 2005) with a learned ReLU MLP constraint. It therefore combines an analytic capacity constraint, a learned feasibility condition, and a linear objective. We again use $n \in \{100, 1000\}$, the four MLP configurations above, and 10 instances per configuration.
- Finally, the *computer-aided molecular design* (CAMD) benchmark (Zhang et al., 2024) maximizes a GNN prediction of banana odor while enforcing combinatorial constraints that encode molecular validity. We consider $N \in \{4, 6, 8, 10, 12\}$ molecular fragments and evaluate both a GraphSAGE (Hamilton et al., 2017) objective and a single-head graph attention network (GAT) (Veličković et al., 2018) objective. Here, acyclicity and connectivity imply that the adjacency is a spanning tree, which we exploit by letting each node choose one parent among the earlier nodes, so those constraints hold by construction and are removed from the penalty (Appendices B.2 and C.3).

Baselines. We compare CLEBO with MILP, which embeds each network exactly into a mixed-integer linear formulation and solves it with Gurobi under a one-hour limit, reporting its incumbent at termination. The ReLU MLPs are embedded via gurobi-machinelearning (Gurobi Optimization, LLC, 2023) and the CAMD GNN via OMLT (Ceccon et al., 2022), with the full constraint set and symmetry-breaking constraints of Zhang et al. (2024) (Appendix C.3). The GAT objective has no exact MILP encoding, so that experiment has no MILP baseline.

Methods. We evaluate $\text{CLEBO}_{\text{base}}$, CLEBO_{re}, and $\text{CLEBO}_{\text{loo}}$ with $K \in \{4, 8, 16, 32\}$ samples, which differ only in the estimator $\widehat{\nabla} f_\theta$ of Section 3.2. $\text{CLEBO}_{\text{loo}K}$ denotes $\text{CLEBO}_{\text{loo}}$ with K samples. Every run uses 3,000 AdamW steps and $S{=}1000$ chains per penalty block. Constrained problems solve the grid $\Lambda = \{0.5, 0.7, 1, 1.5, 3, 10\}$ in one batch (Section 3.3), and the unconstrained random-network benchmark uses a single block. At termination, every chain is rounded, its objective and every constraint, learned or analytic, are evaluated at the binary solution in double precision, and the run returns the best feasible solution. We inherit the remaining hyperparameters from Ichikawa & Arai (2025) and list them in Table 2 (Appendix A).

Metrics and evaluation. We report two metrics. (1) *Gap to the best-known solution.* Per instance, the best-known solution is the best feasible objective found by any method, including every LOO budget, and a method's gap is $100 \cdot (\text{best} - \text{obj})/|\text{best}|$. We average gaps over the instances of a configuration; for MILP, we average only over those where it finds a feasible solution, and we also count the instances on which each method attains the best-known solution. (2) *Runtime.* Sampler runtime is the wall-clock time of the single batched solve over the whole penalty grid, and MILP runtime is the Gurobi solve time. Beyond the gap and time metrics reported in the main paper, Appendix F shows how objective quality and feasibility vary with the penalty weight λ, and Appendix G reports the distribution of gap and runtime across instances, the raw objectives, per-method timings, and how long Gurobi needs to first reach the objective each variant returns.

4.2 RESULTS

Table 1 reports gaps to the best-known solution and wall-clock times across all benchmarks. Figure 1 reports the distributional results of the gap and time across the largest instances of the random benchmark and knapsack problems.

Q1: Unconstrained neural objectives. The random-network results show that the sampled estimator remains effective as both the input dimension and the network size increase. On the exactly solvable 1×50 configuration, MILP proves optimality and $\text{CLEBO}_{\text{loo}32}$ is within 0.05% of that solution. For every larger $n{=}100$ network, MILP reaches the one-hour limit, whereas $\text{CLEBO}_{\text{loo}32}$ has a mean gap of at most 0.27% and attains the best-known solution on at least seven of the ten networks. The difference is more pronounced for $n{=}1000$. $\text{CLEBO}_{\text{loo}32}$ gaps range from 0.00% to 0.60%, while the MILP gaps range from 7.36% to 100% (100% indicating a trivial all-zero incumbent). $\text{CLEBO}_{\text{loo}32}$ gives the lowest displayed gap on every non-anchor configuration and attains the best-known solution on 8–10 of the 10 networks at $n{=}1000$. The relaxed estimator, at 0.65–5.9%, is the next best. Thus, parallel sampling provides consistently strong solutions in the unconstrained setting and degrades gradually with network size, unlike the MILP embedding. Figure 1 (left) illustrates the quality–

Table 1: Gap to the best-known solution (%) and runtime (seconds), computed as described in Section 4.1, for MILP and CLEBO with the relaxed ($\text{CLEBO}_{\text{base}}$), REINFORCE ($\text{CLEBO}_{\text{re}}$), and LOO ($K$=32) estimators. For random-network and knapsack, gaps are means over 10 instances, and parentheses count the instances on which a method attains the best-known solution. Knapsack labels give the number of instances on which MILP found a feasible solution, and its gap averages over those only. CAMD uses the GraphSAGE objective and a single instance per size. *Solved to proven optimality by Gurobi on every instance. **Bold** marks the lowest gap per row.

	Gap to best known (%)				Time (s)			
Instance	MILP	$\text{CLEBO}_{\text{base}}$	CLEBO_{re}	$\text{CLEBO}_{\text{loo32}}$	MILP	$\text{CLEBO}_{\text{base}}$	CLEBO_{re}	$\text{CLEBO}_{\text{loo32}}$
Random Network								
n=100, 1×50*	**0.00** (10)	0.65 (0)	0.75 (0)	0.05 (5)	46	7	8	9
n=100, 2×100	7.43 (0)	0.91 (1)	2.72 (0)	**0.00** (10)	3600	7	8	11
n=100, 2×500	100.00 (0)	1.24 (1)	2.41 (0)	**0.00** (10)	3601	7	8	17
n=100, 4×100	100.00 (0)	2.89 (3)	6.02 (0)	**0.27** (7)	3600	8	8	11
n=100, 4×500	100.00 (0)	2.36 (0)	4.99 (0)	**0.02** (9)	3602	8	9	28
n=1000, 2×100	7.36 (0)	5.93 (0)	30.39 (0)	**0.00** (10)	3600	7	9	59
n=1000, 2×500	100.00 (0)	1.16 (1)	31.50 (0)	**0.01** (9)	3600	8	9	71
n=1000, 4×100	47.55 (0)	5.51 (1)	35.12 (0)	**0.60** (9)	3601	8	9	60
n=1000, 4×500	100.00 (0)	2.35 (2)	38.46 (0)	**0.09** (8)	3602	8	9	82
Neural-Constrained Knapsack								
n=100, 2×100 (10/10)	**0.00** (9)	0.86 (3)	0.48 (1)	0.52 (0)	621	8	9	42
n=100, 2×500 (9/10)	**0.00** (6)	0.55 (6)	0.20 (2)	0.22 (1)	1806	9	8	81
n=100, 4×100 (7/10)	**0.34** (4)	0.42 (4)	0.38 (3)	0.43 (2)	1865	9	8	47
n=100, 4×500 (8/10)	0.14 (3)	0.24 (4)	**0.02** (5)	0.06 (5)	2602	16	10	144
n=1000, 2×100 (10/10)	**0.00** (8)	0.80 (1)	0.59 (1)	0.54 (0)	12	14	22	338
n=1000, 2×500 (10/10)	**0.00** (10)	0.58 (0)	0.41 (0)	0.49 (0)	448	21	23	411
n=1000, 4×100 (10/10)	**0.00** (9)	0.59 (1)	0.51 (0)	0.53 (0)	161	14	21	343
n=1000, 4×500 (9/10)	**0.00** (9)	0.53 (1)	0.67 (0)	0.59 (0)	2066	28	25	474
Molecular Design (CAMD)								
N=4	**0.00**	**0.00**	**0.00**	**0.00**	4	19	38	78
N=6	**0.00**	3.39	**0.00**	3.39	154	22	53	139
N=8	**0.00**	8.58	**0.00**	**0.00**	1157	24	67	187
N=10	**0.00**	48.21	**0.00**	**0.00**	3601	27	82	266
N=12	12.80	59.99	**0.00**	**0.00**	3600	29	99	319

runtime tradeoff on the largest configuration (n=1000, 4×500). Increasing the LOO sample budget reduces the gap at the cost of longer runtime; $\text{CLEBO}_{\text{loo32}}$ achieves near-zero gaps while the MILP finds no solution better than the all-zero solution within the one-hour time limit.

Q2: Hard and learned constraints. The knapsack experiment tests whether the framework can enforce feasibility without a symbolic reformulation of the learned constraint. The analytic objective and capacity constraint use pathwise gradients, while the score-function estimator applies only to the learned constraint. CLEBO finds a feasible solution on every instance. In contrast, MILP fails to find a feasible incumbent on one to three of the 10 instances in half of the configurations. CLEBO also achieves near-optimal objectives in substantially less time. On the largest configuration, MILP takes an average of 2,066 s, while $\text{CLEBO}_{\text{base}}$ reaches an objective within 0.53% of the best-known value in 28 s. Across all configurations, all three estimators remain within 0.9% of the best-known objective, with gaps of 0.02–0.67% for CLEBO_{re} and 0.06–0.59% for $\text{CLEBO}_{\text{loo32}}$. Gurobi often requires hundreds of seconds even to match the objective attained by $\text{CLEBO}_{\text{base}}$ (Appendix G.4). Figure 1 (right) shows a different tradeoff on the largest knapsack configuration. The sampling methods achieve similar gaps, but $\text{CLEBO}_{\text{base}}$ and CLEBO_{re} require substantially less time than the leave-one-out variants and MILP. These strongly correlated instances (Pisinger, 2005) have weak LP relaxations and are challenging for branch-and-bound. The experiment therefore tests whether the framework can enforce a learned constraint through a penalty alone while solving an inherently difficult combinatorial problem.

Q3: Graph neural objectives. The CAMD experiment combines hard molecular-validity constraints with a learned GraphSAGE objective (Table 1). For $N \in \{4, 6, 8\}$, MILP proves optimality. CLEBO_{re} matches the optimum at every size, while $\text{CLEBO}_{\text{loo32}}$ does so except at $N = 6$, where its gap is 3.39%. The relaxed estimator falls behind at $N = 6$ and $N = 8$, with gaps of 3.39% and 8.58%,

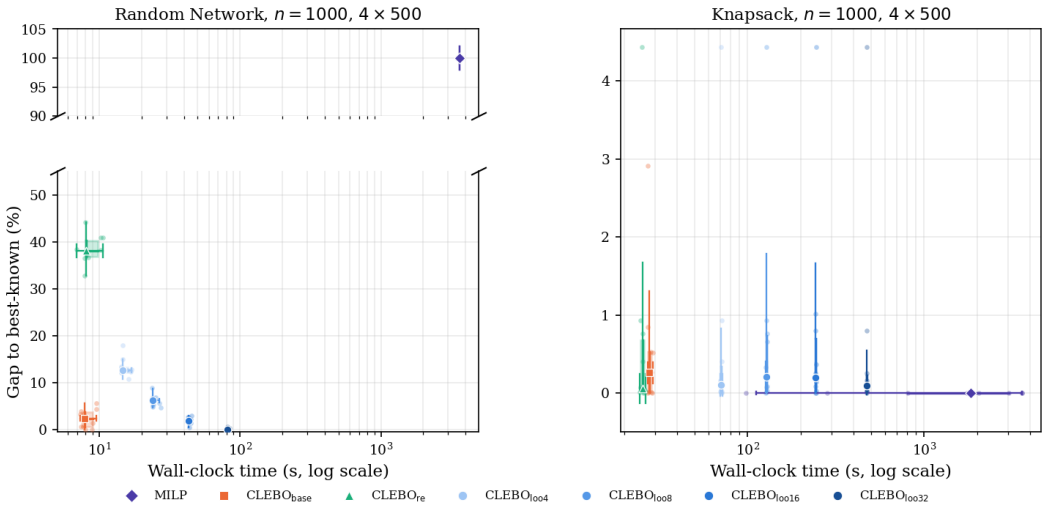

Figure 1: Gap to the best-known solution (%) against wall-clock time (s, log scale) on the largest configuration (n=1000, 4×500) of the random-network (left) and knapsack (right) benchmarks. Each method is summarized by a box spanning the interquartile range on both axes, with cross-hairs at the medians over the 10 instances, and individual instances are overlaid as points. The left axis is broken to show the MILP incumbent at 100%.

respectively. The advantage of the sampled estimators grows with problem size. At $N = 10$, both match the incumbent found by MILP within the one-hour limit, while the relaxed estimator has a gap of 48%. At $N = 12$, both attain the best-known objective. The MILP incumbent after one hour is 12.80% below this value, and the relaxed estimator has a gap of 59.99%. Replacing GraphSAGE with a graph attention objective, for which no exact MILP encoding is available, requires no change to the algorithm. The framework continues to produce feasible molecules at every tested size (Appendix E).

Limitations. Three limitations qualify these results. (1) CLEBO is a heuristic. It certifies neither optimality nor feasibility, since constraints are enforced by penalty, whereas the MILP proves optimality whenever it finishes. In practice, its scalability to networks beyond the reach of exact embedding offsets this, and every run in our experiments returned a feasible solution. (2) The sampled estimators cost K forward passes per chain and step, so CLEBO$_{\text{loo32}}$ is 10–17$\times$ slower than CLEBO$_{\text{base}}$ on the largest configurations. (3) CLEBO only handles binary decision variables.

5 CONCLUSION

We presented a GPU-native approach to constraint learning with binary variables, built on energy-based sampling and annealing with penalty-based constraint handling and a reliable gradient estimator for the learned components. Our algorithm uses only forward and backward passes through the network and requires no symbolic MILP representation of it, which makes it agnostic to the network architecture and its activation functions. Across three benchmarks, it scales to networks well beyond the reach of exact embedding, and on some problem classes returns better solutions than the MILP solver. Future work includes extending beyond binary problems, applying the framework to real-world learned constraint models, and integrating sampling with training: the solutions a sampler visits are precisely the regions where a surrogate's reliability matters most, so leveraging them in training, i.e., active learning, could improve that reliability.

AI USE STATEMENT

We used generative AI tools for feedback on the design of the method and experiments, to implement parts of the code, and to summarize experimental output. We did not use these tools to formulate or prove mathematical claims or to generate data, and the remaining required-disclosure tasks do not apply to this work. We also used them to draft and polish parts of the text, most of which the authors then reviewed and rewrote. Every benchmark instance is either adopted from the literature or generated from a seed by the released code. We reviewed the AI-assisted work: the code is covered by the released test suite, we regenerated every reported number from the released result files, and the authors checked the text. We take responsibility for the final content of this work, including text, claims, and artifacts produced with the aid of generative AI.

ETHICS STATEMENT

Constraint learning is used in applications where part of an optimization model is learned from data rather than written down, including molecular and materials design, process engineering, and planning problems with learned cost or feasibility models. By making optimization over trained networks with binary decisions more scalable and less dependent on the network's architecture, our work may help such systems use richer learned models than exact reformulations allow. At the same time, the same capability could be applied to search learned models in domains where the designed objects or decisions are sensitive, such as the design of harmful compounds. As with many optimization technologies, the broader societal impact depends on the application context and deployment choices. This work involves no human subjects or personal data.

REPRODUCIBILITY STATEMENT

All code, trained models, result files, and instructions are available in an anonymous repository at `https://anonymous.4open.science/r/clebo-B90E/README.md`. Appendix A lists the solver configuration, and Appendix C lists the benchmarks.

ACKNOWLEDGMENTS

REFERENCES

Ross Anderson, Joey Huchette, Will Ma, Christian Tjandraatmadja, and Juan Pablo Vielma. Strong mixed-integer programming formulations for trained neural networks. *Mathematical Programming*, 183:3–39, 2020.

Egon Balas. *On the facial structure of scheduling polyhedra*, pp. 179–218. Springer, Berlin, Heidelberg, 1985.

Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.

David Bergman, Teng Huang, Philip Brooks, Andrea Lodi, and Arvind U. Raghunathan. JANOS: An integrated predictive and prescriptive modeling framework. *INFORMS Journal on Computing*, 34 (2):807–816, 2022.

Francesco Ceccon, Jordan Jalving, Joshua Haddad, Alexander Thebelt, Calvin Tsay, Carl D. Laird, and Ruth Misener. OMLT: Optimization and machine learning toolkit. *Journal of Machine Learning Research*, 23(349):1–8, 2022.

Chih-Hong Cheng, Georg Nührenberg, and Harald Ruess. Maximum resilience of artificial neural networks. In *International Symposium on Automated Technology for Verification and Analysis (ATVA)*, 2017.

Michele Conforti, Gérard Cornuéjols, and Giacomo Zambelli. *Integer Programming*. Springer, Cham, 2014.

Oscar Dowson, Robert B Parker, and Russell Bent. Mathoptai. jl: Embed trained machine-learning predictors into jump models. *INFORMS Journal on Computing*, 2026.

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, 2024.

Matthias Fey and Jan Eric Lenssen. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

Matteo Fischetti and Jason Jo. Deep neural networks and mixed integer linear optimization. *Constraints*, 23(3):296–309, 2018.

Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, San Francisco, 1979.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pp. 249–256. JMLR Workshop and Conference Proceedings, 2010.

Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2011.

Gurobi Optimization, LLC. Gurobi machine learning, 2023. URL `https://github.com/Gurobi/gurobi-machinelearning`.

William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

Yixin Huang, Shengyu Feng, and Yiming Yang. Unsupervised neural Langevin sampler for mixed integer linear programming. In *International Conference on Machine Learning (ICML)*, 2026.

Yuma Ichikawa. Controlling continuous relaxation for combinatorial optimization. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

Yuma Ichikawa and Yamato Arai. Optimization by parallel quasi-quantum annealing with gradient-based sampling. In *International Conference on Learning Representations (ICLR)*, 2025.

Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with Gumbel-softmax. In *International Conference on Learning Representations (ICLR)*, 2017.

Narendra Karmarkar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4 (4):373–395, 1984.

Richard M Karp. Reducibility among combinatorial problems. In *Complexity of Computer Computations: Proceedings of a symposium on the Complexity of Computer Computations, held March 20–22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York, and sponsored by the Office of Naval Research, Mathematics Program, IBM World Trade Corporation, and the IBM Research Mathematical Sciences Department*, pp. 85–103. Springer, 1972.

Leonid G. Khachiyan. Polynomial algorithms in linear programming. *USSR Computational Mathematics and Mathematical Physics*, 20(1):53–72, 1980.

Samuel Kolb. Learning constraints and optimization criteria. In *AAAI Workshop on Declarative Learning-Based Programming*. AAAI Press, 2016. URL `https://aaai.org/papers/aaaiw-ws0235-16-12651/`.

Wouter Kool, Herke van Hoof, and Max Welling. Buy 4 REINFORCE samples, get a baseline for free! In *Deep Reinforcement Learning Meets Structured Prediction, ICLR Workshop*, 2019.

Hongpei Li, Hui Yuan, Han Zhang, Jianghao Lin, Dongdong Ge, Mengdi Wang, and Yinyu Ye. FMIP: Joint continuous-integer flow for mixed-integer linear programming. In *The Fourteenth International Conference on Learning Representations*, 2026. URL `https://openreview.net/forum?id=kyvW6S0u3z`.

Michele Lombardi and Michela Milano. Boosting combinatorial problem modeling with machine learning. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 5472–5478, 2018.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=Bkg6RiCqY7.

Chris J. Maddison, Andriy Mnih, and Yee Whye Teh. The concrete distribution: A continuous relaxation of discrete random variables. In *International Conference on Learning Representations (ICLR)*, 2017.

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, 2025.

Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In *International Conference on Machine Learning (ICML)*, 2010.

Georgia Perakis and Asterios Tsiourvas. Optimizing objective functions from trained relu neural networks via sampling. *arXiv:2205.14189*, 2022.

David Pisinger. Where are the hard knapsack problems? *Computers & Operations Research*, 32(9): 2271–2284, 2005.

Buser Say, Ga Wu, Yu Qing Zhou, and Scott Sanner. Nonlinear hybrid planning with deep net learned transition models and mixed-integer linear programming. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 2017.

Alexander Schrijver. *Theory of Linear and Integer Programming*. John Wiley & Sons, Chichester, 1986.

Artur M. Schweidtmann and Alexander Mitsos. Deterministic global optimization with artificial neural networks embedded. *Journal of Optimization Theory and Applications*, 180(3):925–948, 2019.

Thiago Serra, Christian Tjandraatmadja, and Srikumar Ramalingam. Bounding and counting linear regions of deep neural networks. In *International Conference on Machine Learning (ICML)*, 2018.

Zi-Song Shen, Feng Pan, Yao Wang, Yi-Ding Men, Wen-Biao Xu, Man-Hong Yung, and Pan Zhang. Free-energy machine for combinatorial optimization. *Nature Computational Science*, 5(4):322–332, 2025.

Haoran Sun, Katayoon Goshvadi, Azade Nova, Dale Schuurmans, and Hanjun Dai. Revisiting sampling for combinatorial optimization. In *International Conference on Machine Learning (ICML)*, 2023.

Zhiqing Sun and Yiming Yang. DIFUSCO: Graph-based diffusion solvers for combinatorial optimization. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

Jiatai Tong, Junyang Cai, and Thiago Serra. Optimization over trained neural networks: Taking a relaxing walk. In *International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research (CPAIOR)*, 2024.

Jiatai Tong, Yilin Zhu, Thiago Serra, and Samuel Burer. Optimization over trained neural networks: Going large with gradient-based algorithms. In *International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research (CPAIOR)*, 2026.

Calvin Tsay, Jan Kronqvist, Alexander Thebelt, and Ruth Misener. Partition-based formulations for mixed-integer optimization of trained ReLU neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *International Conference on Learning Representations (ICLR)*, 2018.

Juan Pablo Vielma, Shabbir Ahmed, and George Nemhauser. Mixed-integer models for nonseparable piecewise-linear optimization: Unifying framework and extensions. *Operations Research*, 58(2): 303–315, 2010.

Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3–4):229–256, 1992.

Laurence A. Wolsey. *Integer Programming*. John Wiley & Sons, New York, 1998.

Shiqiang Zhang, Juan S. Campos, Christian Feldmann, Frederik Sandfort, Miriam Mathea, and Ruth Misener. Augmenting optimization-based molecular design with graph neural networks. *Computers & Chemical Engineering*, 186:108684, 2024.

A CLEBO CONFIGURATION

Table 2 reports the settings shared by the three estimators.

Table 2: Sampler configuration shared by $\text{CLEBO}_{\text{base}}$, CLEBO_{re}, and $\text{CLEBO}_{\text{loo}}$. Defaults marked † are inherited from Ichikawa & Arai (2025).

Setting	Value
Optimizer	AdamW, learning rate 0.01, weight decay $0.01^\dagger$
Steps per run	$3{,}000$
Langevin temperature T	$0.001^\dagger$
Sharpening exponent α	$4^\dagger$
Annealing schedule γ	linear from -2 to $0.1^\dagger$
Diversity strength ρ	$3.0^\dagger$
Chains per penalty block S	1000
Penalty grid Λ (constrained)	$\{0.5, 0.7, 1, 1.5, 3, 10\}$
Penalty (random network)	λ=1
Gradient clipping	Frobenius norm ≤ 10 per block
Initialization	$\text{clip}(0.5 + 0.01\boldsymbol{\xi}, 0, 1)$, $\boldsymbol{\xi} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
LOO sample budget K	$\{4, 8, 16, 32\}$
LOO logit clip ϵ	10^{-6}
REINFORCE probability clip	$[10^{-4}, 1 - 10^{-4}]$
Softmax temperature τ (CAMD)	5

B EXTENSIONS

B.1 THE LEARNED CONSTRAINT CASE

When the learned network is a constraint $g_\theta(\mathbf{x}) \le 0$ rather than the objective, it enters the per-chain cost of Eq. (6) through its own hinge penalty,

$$c(\mathbf{p}) = f(\mathbf{p}) + \sum_{j=1}^{m} \lambda_j \, [g_j(\mathbf{p})]_+ + \lambda_g \, [g_\theta(\mathbf{p})]_+, \tag{9}$$

with λ_g scale-normalized exactly as the analytic weights (Section 3.3). The sharpening and diversity terms, the annealing schedule, and the λ-grid are unchanged. What changes is only which term the estimator $\widehat{\nabla} g_\theta$ applies to. The learned quantity is now $\lambda_g[g_\theta(\cdot)]_+$, and the objective f is analytic and therefore differentiated pathwise.

Relaxed estimator (CLEBO$_{\text{base}}$). Every term of Eq. (9) is evaluated on the relaxed $\mathbf{p}$ and differentiated by autograd, so $\widehat{\nabla} g_\theta(\mathbf{p}) = \lambda_g \nabla_{\mathbf{p}} \, [g_\theta(\mathbf{p})]_+$.

Sampled estimators (CLEBO$_{\text{re}}$, CLEBO$_{\text{loo}}$). Here g_θ is evaluated only on discrete samples $\mathbf{x}^{(k)} \sim q_{\mathbf{p}}$ and is never differentiated. Each sample is scored by the learned penalty alone,

$$c_g(\mathbf{x}) = \lambda_g \, [g_\theta(\mathbf{x})]_+, \tag{10}$$

which excludes the analytic objective f and the analytic constraints g_j: both admit an exact pathwise gradient. The gradient each chain follows is then

$$\begin{aligned} \widehat{\nabla}_{\mathbf{p}} E \;=\; & \underbrace{\tfrac{1}{K} \sum_{k=1}^{K} \big(c_g(\mathbf{x}^{(k)}) - b_k\big) \nabla_{\mathbf{p}} \log q_{\mathbf{p}}(\mathbf{x}^{(k)})}_{\text{sampled}} \\ & + \underbrace{\nabla_{\mathbf{p}} f(\mathbf{p}) + \textstyle\sum_{j=1}^{m} \lambda_j \nabla_{\mathbf{p}} \, [g_j(\mathbf{p})]_+}_{\text{pathwise}} \\ & + \; \gamma \nabla_{\mathbf{p}} \phi(\mathbf{p}, \alpha) - S\rho \nabla_{\mathbf{p}} \sigma(\mathbf{P}). \end{aligned} \tag{11}$$

The two sampled variants differ only in how b_k is formed, and both are computed on the same reward c_g, so that the baseline is a control variate for the learned penalty's contribution to the variance. CLEBO$_{\text{loo}}$ draws $K \ge 2$ samples per chain and baselines each against the mean of the other $K-1$ from that same chain, $b_k = \frac{1}{K-1} \sum_{\ell \ne k} c_g(\mathbf{x}^{(\ell)})$, a proper control variate because all K samples share one $\mathbf{p}$. Similar to the learned objective case, CLEBO$_{\text{re}}$ draws a single sample per chain, for which no within-chain baseline exists, and instead subtracts the mean reward over the S chains of the same penalty block.

B.2 STRUCTURAL REPARAMETERIZATION

The construction below applies to any encoding in which groups of binary variables encode a single choice.

Consider constraints $\sum_{i \in G_c} x_i = 1$, $c \in \mathcal{C}$, over disjoint index groups $G_c \subseteq \{1, \dots, n\}$. In an assignment problem, for example, each job forms one group consisting of its indicator for every machine, and the constraint requires exactly one machine per job. If each variable in the group is sampled independently, the sampler explores all $2^{|G_c|}$ patterns, although only the $|G_c|$ patterns with a single active index are feasible. We instead couple each group through a softmax over its coordinates,

$$q_{\mathbf{p}}(\mathbf{x}_{G_c} = \mathbf{e}_i) = \pi_i(\mathbf{p}) = \frac{\exp(\tau p_i)}{\sum_{i' \in G_c} \exp(\tau p_{i'})}, \qquad i \in G_c, \tag{12}$$

where $\mathbf{e}_i$ activates index i alone, $\boldsymbol{\pi}_{G_c}(\mathbf{p})$ collects these probabilities over the group, and the inverse temperature τ (Table 2) controls how sharply the group concentrates on its largest coordinate. Each group thus becomes a single choice among $|G_c|$ alternatives. Variables outside any group, collected in $\mathcal{B}$, are sampled independently as before, and the sampling distribution factorizes as

$$q_{\mathbf{p}}(\mathbf{x}) = \prod_{c \in \mathcal{C}} q_{\mathbf{p}}(\mathbf{x}_{G_c}) \prod_{b \in \mathcal{B}} p_b^{x_b} (1 - p_b)^{1 - x_b}. \tag{13}$$

Every sample selects exactly one index per group, so these constraints hold by construction and never enter the penalty.

Gradients. The softmax is applied wherever the solution is evaluated, and the rest of the method is unchanged. The relaxed estimator evaluates the network and the analytic terms at $\boldsymbol{\pi}_{G_c}(\mathbf{p})$ in place of $\mathbf{p}_{G_c}$ and backpropagates through the softmax. The sampled estimators replace the Bernoulli score in Eq. (8) by the score of the drawn index i,

$$\nabla_{\mathbf{p}_{G_c}} \log q_{\mathbf{p}}(\mathbf{x}_{G_c} = \mathbf{e}_i) = \tau\big(\mathbf{e}_i - \boldsymbol{\pi}_{G_c}(\mathbf{p})\big),$$

which shifts probability toward the sampled alternative when its advantage is positive and away from it otherwise. The sharpening term, the diversity term, and the projection still act on $\mathbf{p}$, and at termination each group is rounded to its largest coordinate.

Generalizations. The same idea extends to choices that depend on one another. In CAMD (Appendix C.3), each node picks a parent among the earlier nodes, each edge a weight of 1 or 2, and each node a type, and each choice is drawn only among the options still valid given the earlier ones. Every sample is then a valid tree whose probability is the product of these restricted probabilities, and only the remaining constraints enter the penalty.

C BENCHMARK DETAILS

C.1 RANDOM NETWORK

This benchmark places the learned network in the *objective* and imposes no constraints:

$$\max_{\mathbf{x}\in\{0,1\}^n} \quad f_\theta(\mathbf{x}), \tag{14}$$

where f_θ is a random ReLU network with Xavier initialization Glorot & Bengio (2010) with L hidden layers of width H. The benchmark reports the maximized utility f_θ, while CLEBO minimizes $c = -f_\theta$. Unlike the other benchmarks, f_θ is not trained: it is a generic nonlinear objective, so the benchmark isolates the difficulty of optimizing over a network purely as a function of its size, independent of any learning task. It is the binary restriction of the continuous random-ReLU maximization used by Perakis & Tsiourvas (2022) and Tong et al. (2024).

Because the raw initialized output is $O(1)$ while the sharpening term is $O(n)$, we rescale the objective by a positive constant so that its standard deviation over random binary inputs equals n. This leaves the argmax unchanged and is applied identically to all methods and the baseline. We use $n \in \{100, 1000\}$ with architectures (layers×width) 2×100, 2×500, 4×100, and 4×500, plus a small 1×50 network at n=100 that Gurobi solves exactly. For the MILP baseline, the network is embedded exactly via big-M reformulations and maximized over $\{0,1\}^n$. Every binary vector is feasible, so Gurobi always has an incumbent, but the embedding becomes intractable as n and the width grow.

C.2 NEURAL-CONSTRAINED BINARY KNAPSACK

The binary knapsack benchmark considers:

$$\begin{aligned} \max_{\mathbf{x}\in\{0,1\}^n} \quad & \textstyle\sum_{i=1}^n v_i x_i \\ \text{s.t.} \quad & \textstyle\sum_{i=1}^n w_i x_i \le W, \\ & g_\theta(\mathbf{x}) \le 0, \end{aligned} \tag{15}$$

where the integer values and weights follow the strongly-correlated instance class (Pisinger, 2005): $w_i \sim \mathrm{Uniform}\{1, R\}$ and $v_i = w_i + R/10$ with $R = 1000$, the capacity is $W = \lfloor 0.5 \cdot \sum_i w_i \rfloor$, and g_θ is a trained ReLU MLP. The reported utility is $f(\mathbf{x}) = \sum_i v_i x_i$, while the sampler uses the minimization cost $c(\mathbf{x}) = -f(\mathbf{x})$. The strong value–weight correlation makes profit-to-weight ratios nearly identical, so the LP relaxation is weak, and branch-and-bound is hard. This adds genuine combinatorial difficulty on top of the learned constraint, since with uncorrelated coefficients Gurobi closes the knapsack at the root, making the network the only source of hardness.

The ground-truth constraint is a random pairwise interaction: $g^*(\mathbf{x}) = \frac{1}{n}\sum_{i\neq j} C_{ij} x_i x_j$ where $C_{ij} \sim \mathcal{N}(0,1)$ with zero diagonal. The MLP is trained on 50,000 binary samples with labels centered at the training-set median, so $g_\theta(\mathbf{x}) \le 0$ defines a region containing $\sim$50% of binary vectors.

C.3 COMPUTER-AIDED MOLECULAR DESIGN

We follow the molecular design problem of Zhang et al. (2024). The goal is to assemble N molecular fragments into a molecule that a trained graph neural network predicts to smell of banana. We train this classifier with the released code of Zhang et al. (2024), which uses two SAGEConv layers with sum aggregation and 16 hidden units, mean pooling, and a linear output, implemented in PyTorch Geometric (Fey & Lenssen, 2019). A molecule is encoded by binary node features X, which describe each fragment, and binary adjacency matrices, which describe how the fragments are bonded. The objective $f_\theta(X, A)$ is the network's banana score, which the sampler minimizes as $-f_\theta$. Table 3 summarizes the constraints and how CLEBO handles each, and Zhang et al. (2024, Sections 4 and 5) give their full definitions together with the symmetry-breaking constraints (B1) and (C27) used in the MILP baseline. We build the MILP with OMLT (Ceccon et al., 2022) rather than gurobi-machinelearning, since OMLT can encode a GNN whose graph is itself a decision variable, and solve it with Gurobi.

CLEBO representation. Instead of deciding every entry of X and the adjacency matrices, CLEBO builds each molecule node by node. Every node after the first attaches to one earlier node, so the

graph is always a tree. Each edge then receives a weight of 1 or 2, and each node receives one of four types, each of which limits the total edge weight a node can hold. All three choices are sampled from softmax distributions (Appendix B.2), and each is made only among the options that keep every node within its limits. Every sample is therefore a valid molecule. Thirteen of the 23 constraints hold by the representation, and the remaining ten enter the penalty (Table 3). Four of the ten (C4, C10, C15, C17) are satisfied by every decoded molecule, so they act only on the fractional molecule of the relaxed estimator, and six are enforced by the penalty alone. The relaxed estimator does not sample. It evaluates the network on a fractional molecule built from the same softmax probabilities, without the validity restrictions, in which each adjacency entry is the probability that the corresponding edge is chosen and each type entry is the probability of that type. At termination, each chain is rounded by making the same choices in the same order, taking the most probable allowed option at each step.

Table 3: CAMD constraints of Zhang et al. (2024) and how CLEBO handles them. Thirteen hold by the representation and ten enter the penalty, of which C4, C10, C15 and C17 are satisfied by every decoded molecule. The MILP includes all 23 together with the symmetry-breaking constraints (B1) and (C27).

Constraint	Meaning	Handled by
C0, C1	all fragments exist, fragments 0 and 1 are bonded	representation
C3	bonds are symmetric	representation
C5, C25	connected, no non-aromatic rings (a tree)	representation
C6, C7	double bonds are symmetric, none on the diagonal	representation
C11–C13	one type, neighbor count, and hydrogen count per fragment	representation
C14	neighbor count matches degree	representation
C19′	double-bond flag equals double-bond count	representation
C21	covalence equals neighbors plus hydrogens plus double bonds	representation
C4	bonds only between existing fragments	penalty
C10	double bonds only on bonds	penalty
C15	a double bond requires both endpoints flagged	penalty
C17	at most $\lfloor \mathrm{cov}/2 \rfloor$ double bonds per fragment	penalty
C22, C23	at most $N/4$ oxygens and $N/4$ double bonds	penalty
B3	no double bond on the benzene fragment	penalty
B4	no oxygen–oxygen bond	penalty
B5	at most two aromatic fragments	penalty
Req. 5.1.1	at least one aromatic fragment or carbonyl oxygen	penalty

Dense GNN forward pass. Because the relaxed estimator evaluates the network on a fractional adjacency matrix, we implement the trained SAGEConv layers with dense matrix products, which reproduce the original network exactly on binary graphs.

D RELAXATION RELIABILITY BENCHMARK

The relaxed estimator is useful only when the objective's values at fractional points rank solutions the same way as its values at the nearby binary points. This appendix measures that agreement and shows how the relaxed estimator degrades as it breaks down, on a synthetic problem where the agreement can be set directly, independent of any network architecture or training distribution.

Relaxation reliability. We measure agreement by the rank correlation

$$r = \mathrm{Spearman}\Big(\big(f(\mathbf{p}^{(k)})\big)_{k=1}^{M},\ \big(f(\mathrm{round}(\mathbf{p}^{(k)}))\big)_{k=1}^{M}\Big), \qquad \mathbf{p}^{(k)} \sim \mathrm{Unif}([0,1]^n), \tag{16}$$

with $M = 10{,}000$. A value near one means relaxed values predict discrete ones, a value near zero means they carry no information, and a negative value means they point the wrong way.

Problem. We maximize $f(\mathbf{x})$ subject to $\mathbf{w}^\top \mathbf{x} \le W$ over $\mathbf{x} \in \{0,1\}^n$, with $n = 100$, integer weights $w_i \sim \mathrm{Unif}\{1,\dots,100\}$, and $W = \frac{1}{2}\sum_i w_i$. The objective is

$$f(\mathbf{x}) = f_{\mathrm{base}}(\mathbf{x}) + \beta\,\eta(\mathbf{x}), \qquad f_{\mathrm{base}}(\mathbf{x}) = \mathbf{x}^\top C\mathbf{x} + \mathbf{c}^\top \mathbf{x}, \tag{17}$$

where C is a random symmetric matrix with zero diagonal and entries of order $1/n$, and $\mathbf{c} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$. The distortion η is multiplied by $\delta(\mathbf{x}) = 1 - \frac{1}{n}\sum_i |2x_i - 1|$, which is zero at every binary point, so the discrete problem is identical for every β and only the relaxation changes. We use two distortions,

$$\eta_{\mathrm{UG}}(\mathbf{x}) = \delta(\mathbf{x}) \textstyle\sum_{j=1}^{10} \sin\big(20\,(P^\top \mathbf{x})_j\big), \qquad \eta_{\mathrm{AG}}(\mathbf{x}) = -\delta(\mathbf{x})\, f_{\mathrm{base}}(\mathbf{x}), \tag{18}$$

where $P \in \mathbb{R}^{n\times 10}$ is a random projection. The uninformative distortion (UG) adds high-frequency noise and drives r toward zero. The adversarial distortion (AG) reverses the objective in the interior and drives r below zero.

Setup. We use the solver of Section 3 with a single penalty weight $\lambda = 1$, $S = 100$ chains, and 500 steps, which suffices for this small problem, and compare CLEBO$_{\mathrm{base}}$ with CLEBO$_{\mathrm{loo}}$ (K=16). The objective is learned, and the capacity constraint is analytic, so CLEBO$_{\mathrm{loo}}$ evaluates f only at binary samples and differentiates the constraint pathwise for both estimators. Similarly to the random network benchmark, we rescale f so that its standard deviation over random binary points is n, which leaves the optimum unchanged and is the same for every β. We report the best feasible objective in original units, averaged over 5 seeds, each with a new instance. Because η vanishes at binary points, CLEBO$_{\mathrm{loo}}$ is unaffected by β by construction, and the result of interest is the CLEBO$_{\mathrm{base}}$ column.

Results. CLEBO$_{\mathrm{base}}$ matches CLEBO$_{\mathrm{loo}}$ when the relaxation is reliable, falls behind clearly once $r \approx 0.4$, and reaches 16.8 against 42.0 at $r = 0.05$. When the interior gradients are reversed, it is driven toward the anti-optimum, ending near -30 at $r = -0.84$. Relaxed gradients can therefore be systematically misleading, not merely noisy.

Table 4: Best feasible objective on the synthetic benchmark (mean $\pm$ std over 5 seeds) as the relaxation reliability r decreases. UG makes interior gradients uninformative, and AG reverses them. CLEBO$_{\mathrm{loo}}$ is unaffected by construction.

Distortion	β	r	CLEBO$_{\mathrm{base}}$	CLEBO$_{\mathrm{loo}}$
None	0	0.86	42.2 ± 3.8	42.0 ± 3.8
UG	2	0.68	39.8 ± 4.0	42.0 ± 3.8
	5	0.39	34.6 ± 4.1	42.0 ± 3.8
	10	0.22	27.4 ± 2.9	42.0 ± 3.8
	20	0.11	22.4 ± 2.8	42.0 ± 3.8
	50	0.05	16.8 ± 3.1	42.0 ± 3.8
AG	1	0.85	35.5 ± 3.2	42.0 ± 3.8
	2	0.02	-13.9 ± 21.4	42.0 ± 3.8
	5	-0.84	-30.3 ± 6.3	42.0 ± 3.8

E ARCHITECTURE-AGNOSTIC EVALUATION

The MILP baseline requires an exact mixed-integer encoding of the network, as exists for ReLU layers, which rules out architectures such as attention. CLEBO only needs forward and backward passes, so we replace the GraphSAGE with a single-head GAT and leave the method unchanged. Table 5 reports the results on the same banana-odor benchmark. The experiment establishes applicability rather than ranking the estimators, but the sampled estimators match or improve on $\text{CLEBO}_{\text{base}}$ at every N.

Table 5: CAMD with a learned GAT objective. The table reports the best feasible objective obtained by each estimator for a single seed.

N	$\text{CLEBO}_{\text{base}}$	CLEBO_{re}	$\text{CLEBO}_{\text{loo4}}$	$\text{CLEBO}_{\text{loo8}}$	$\text{CLEBO}_{\text{loo16}}$	$\text{CLEBO}_{\text{loo32}}$
4	3.10	10.50	10.50	10.50	10.50	10.50
6	7.29	9.82	9.82	9.82	9.82	9.82
8	16.72	16.72	16.72	16.72	16.72	16.72
10	12.88	16.25	16.25	16.25	16.12	16.25
12	8.42	15.95	11.36	12.23	9.39	9.87

F ALGORITHM ANALYSIS

This appendix shows how solution quality and feasibility change with the penalty weight λ (Figures 2–5). The per-instance numbers behind Table 1 are in Appendix G. On the knapsack, a larger λ yields more feasible chains, and at n=1000 also steadily larger gaps, while all estimators behave almost identically. The best solution therefore sits near the smallest feasible λ. On CAMD, the sampled estimators beat the relaxed estimator at every λ from N=8 onward, so retuning λ cannot close the difference. From N=6 onward, they also return more feasible chains.

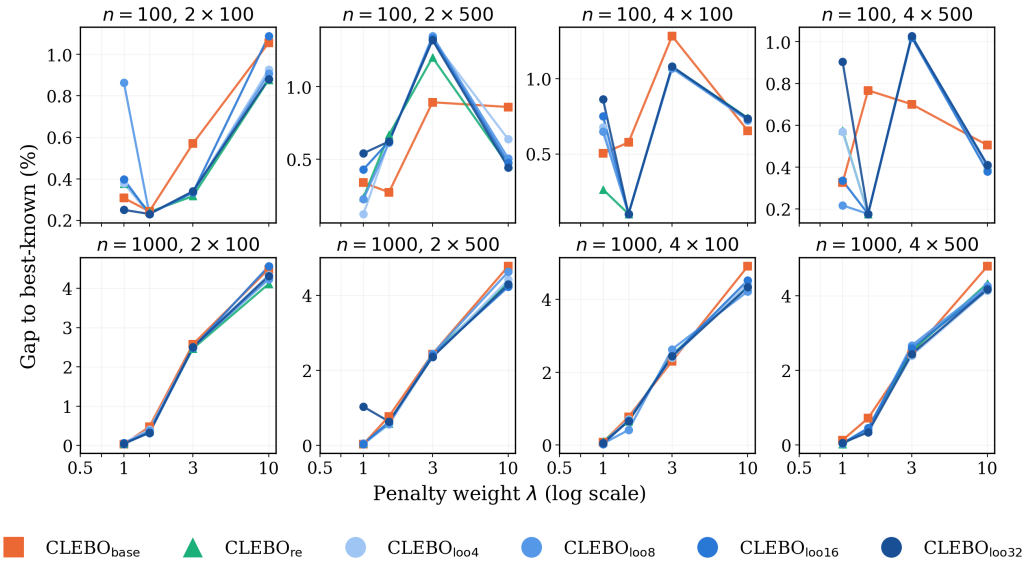

Figure 2: *Knapsack*: mean gap to the best-known solution (%) at each λ, over the instances feasible at that λ. Hollow points rest on fewer than half of the 10 instances.

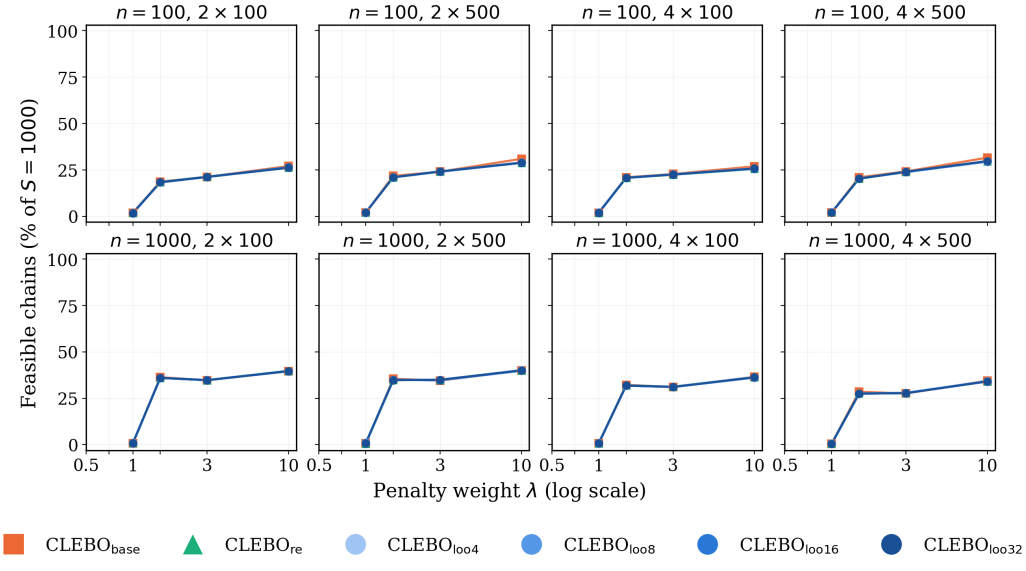

Figure 3: *Knapsack*: feasible chains (% of S=1000) at each λ, averaged over the 10 instances.

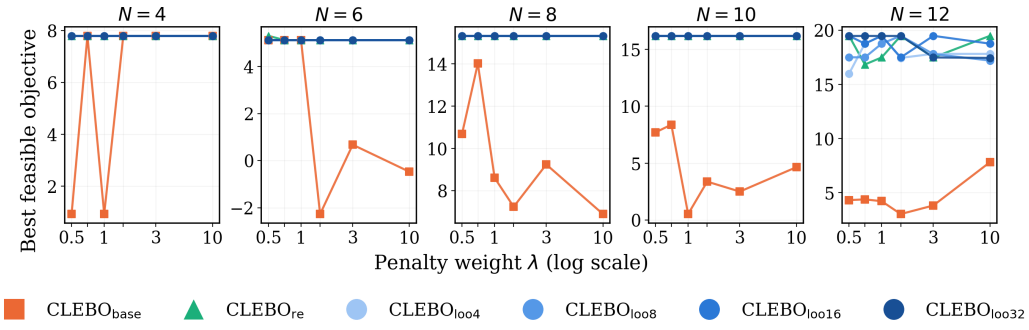

Figure 4: *Molecular design*: best feasible objective at each λ (higher is better).

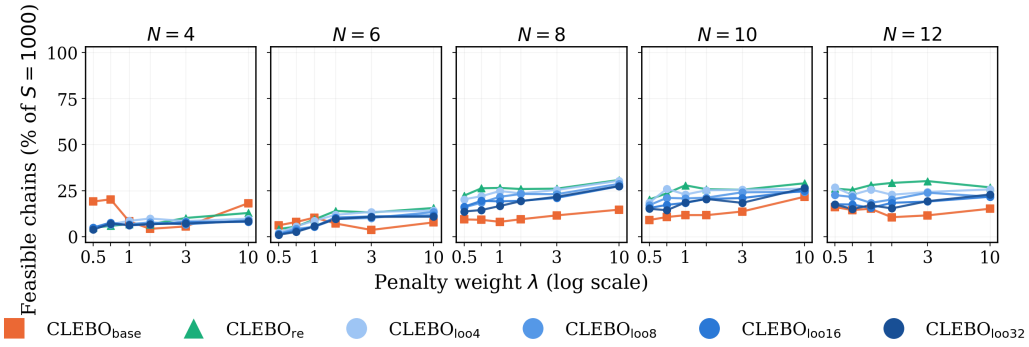

Figure 5: *Molecular design*: feasible chains (% of S=1000) at each λ.

G Detailed Numerical Results

This appendix further details the downstream performance summarized in Table 1: the distribution of gap and runtime across the instances of each configuration (Appendix G.1), the raw objectives (Appendix G.2), wall-clock times (Appendix G.3), and how long Gurobi needs to reach the objective each variant returns (Appendix G.4).

G.1 Gap and Runtime Across Instances

Figures 6 and 7 show gap against runtime for every configuration. On random networks, MILP finds no useful solution on five of the nine configurations, while $\text{CLEBO}_{\text{loo}}$ stays close to the best known and improves with K. On the knapsack, MILP is best when it finds a feasible solution, and every estimator has median gaps below 0.3%, with $\text{CLEBO}_{\text{base}}$ and CLEBO_{re} taking under 30 s.

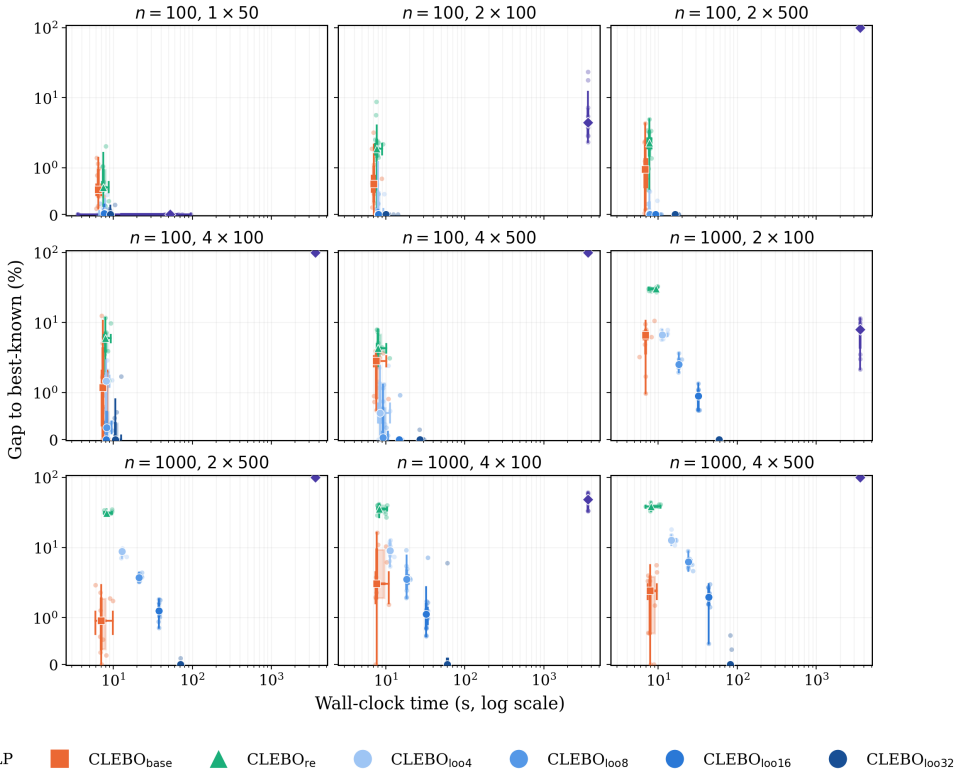

Figure 6: *Random network*: gap to the best-known solution (%, symlog) against runtime (s, log). Boxes span the interquartile range over the 10 networks on both axes, with medians as cross-hairs.

G.2 Objectives

Tables 6–8 give the raw objectives behind Table 1: the best feasible objective over the λ-grid, averaged over the 10 instances per configuration for the random-network and knapsack benchmarks.

G.3 Wall-Clock Time

Table 9 gives wall-clock times. $\text{CLEBO}_{\text{base}}$ and CLEBO_{re} are the cheapest, and $\text{CLEBO}_{\text{loo}}$ grows roughly linearly with K on the larger instances.

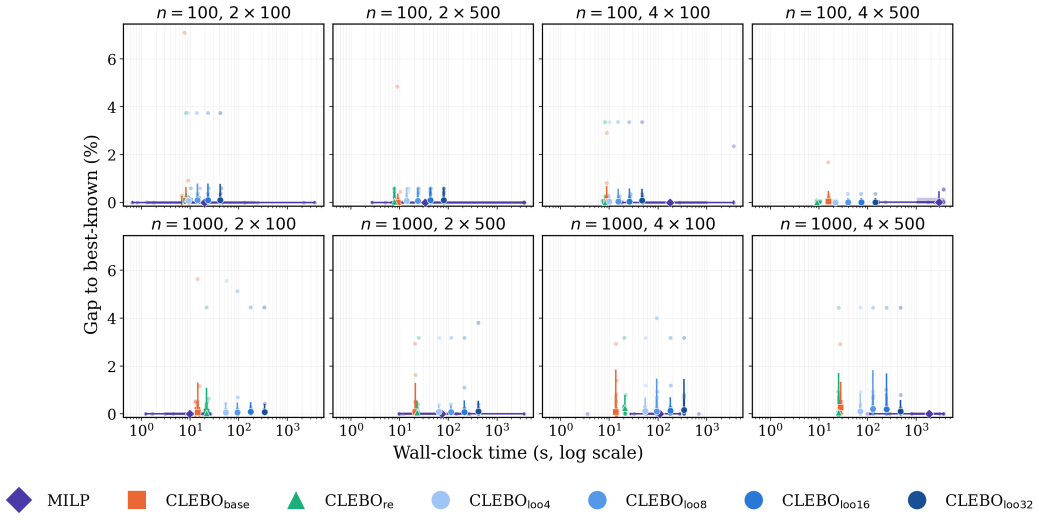

Figure 7: *Knapsack*: as Figure 6, on a linear gap axis. MILP is summarized over the instances where it found a feasible solution.

Table 6: Random-network objectives (mean over 10 networks). † time-limit incumbent. * solved to proven optimality on every network. Bold marks the best per row.

Instance	MILP	CLEBO$_{base}$	CLEBO$_{re}$	CLEBO$_{loo4}$	CLEBO$_{loo8}$	CLEBO$_{loo16}$	CLEBO$_{loo32}$
n=100, 1 × 50*	**926**	920	920	925	926	926	926
n=100, 2 × 100	816†	862	849	867	868	868	**869**
n=100, 2 × 500	0†	893	882	902	903	**903**	903
n=100, 4 × 100	0†	944	920	959	967	**969**	968
n=100, 4 × 500	0†	1074	1047	1089	1094	1097	**1098**
n=1000, 2 × 100	28094†	28532	21122	28336	29576	30110	**30382**
n=1000, 2 × 500	0†	30072	20852	27844	29302	30034	**30421**
n=1000, 4 × 100	18181†	32910	22619	31427	33142	33876	**34245**
n=1000, 4 × 500	0†	30565	19328	27175	29360	30657	**31231**

Table 7: Knapsack objectives (mean over 10 instances). MILP averages over the $k/10$ instances where it found a feasible solution, and † marks a configuration with at least one instance not proven optimal. Bold marks the best per row.

Instance	MILP	CLEBO$_{base}$	CLEBO$_{re}$	CLEBO$_{loo4}$	CLEBO$_{loo8}$	CLEBO$_{loo16}$	CLEBO$_{loo32}$
n=100, 2 × 100 (10/10)	**32668**†	32404	32518	32518	32508	32508	32507
n=100, 2 × 500 (9/10)	**32682**†	32475	32582	32583	32576	32573	32574
n=100, 4 × 100 (7/10)	32475†	32485	**32500**	32499	32492	32489	32486
n=100, 4 × 500 (8/10)	**32615**†	32540	32606	32605	32595	32595	32595
n=1000, 2 × 100 (10/10)	**322861**	320267	320950	320749	320823	321112	321091
n=1000, 2 × 500 (10/10)	**322855**†	320965	321508	321539	321545	321260	321258
n=1000, 4 × 100 (10/10)	**322862**	320945	321200	321160	320882	321160	321142
n=1000, 4 × 500 (9/10)	**322972**†	321077	320641	320806	320436	320494	320893

G.4 GUROBI TIME TO MATCH OUR METHOD

Table 10 gives the time Gurobi needs to first reach each variant's objective, read from its incumbent log. On the knapsack it reaches each variant on 4 to 10 of the 10 instances, within 14 minutes at most. On random networks it never reaches CLEBO$_{loo16}$ or CLEBO$_{loo32}$ outside the 1×50 networks. On CAMD it needs 1093 s at N=8 and 3404 s at N=10, and it never reaches any sampled estimator at N=12. The gaps in Table 1 are therefore not a matter of Gurobi needing more time.

Table 8: CAMD objectives (GraphSAGE, single seed). † not proven optimal within 1 h. Bold marks the best per row.

Instance	MILP	CLEBO$_{\text{base}}$	CLEBO$_{\text{re}}$	CLEBO$_{\text{loo4}}$	CLEBO$_{\text{loo8}}$	CLEBO$_{\text{loo16}}$	CLEBO$_{\text{loo32}}$
N=4	**7.8**	**7.8**	**7.8**	**7.8**	**7.8**	**7.8**	**7.8**
N=6	**5.3**	5.1	**5.3**	5.1	5.1	5.1	5.1
N=8	**15.3**	14.0	**15.3**	**15.3**	**15.3**	**15.3**	**15.3**
N=10	**16.2**†	8.4	**16.2**	**16.2**	**16.2**	**16.2**	**16.2**
N=12	17.0†	7.8	**19.5**	**19.5**	**19.5**	**19.5**	**19.5**

Table 9: Wall-clock time (s) of the full λ-grid solve. MILP has a 1 h limit. Random-network and knapsack rows are means over 10 instances.

Instance	MILP	CLEBO$_{\text{base}}$	CLEBO$_{\text{re}}$	CLEBO$_{\text{loo4}}$	CLEBO$_{\text{loo8}}$	CLEBO$_{\text{loo16}}$	CLEBO$_{\text{loo32}}$
Random Network							
n=100, 1 × 50*	46	7	8	8	8	8	9
n=100, 2 × 100	3600	7	8	8	8	9	11
n=100, 2 × 500	3601	7	8	8	8	10	17
n=100, 4 × 100	3600	8	8	8	8	8	11
n=100, 4 × 500	3602	8	9	9	10	15	28
n=1000, 2 × 100	3600	7	9	12	18	32	59
n=1000, 2 × 500	3600	8	9	13	22	38	71
n=1000, 4 × 100	3601	8	9	11	19	33	60
n=1000, 4 × 500	3602	8	9	15	25	44	82
Neural-Constrained Knapsack							
n=100, 2 × 100	621	8	9	10	14	23	42
n=100, 2 × 500	1806	9	8	14	24	43	81
n=100, 4 × 100	1865	9	8	10	16	26	47
n=100, 4 × 500	2602	16	10	22	40	75	144
n=1000, 2 × 100	12	14	22	54	94	175	338
n=1000, 2 × 500	448	21	23	63	113	212	411
n=1000, 4 × 100	161	14	21	55	95	178	343
n=1000, 4 × 500	2066	28	25	71	128	243	474
Molecular Design (GraphSAGE)							
N=4	4	19	38	38	42	54	78
N=6	154	22	53	55	65	91	139
N=8	1157	24	67	71	87	120	187
N=10	3601	27	82	91	114	164	266
N=12	3600	29	99	108	135	198	319

Table 10: Time (s) for Gurobi to first reach each variant's objective (1 h limit). "−" means never reached. For random-network and knapsack, entries average over the k instances on which the value was reached.

Instance	CLEBO$_{\text{base}}$	CLEBO$_{\text{re}}$	CLEBO$_{\text{loo4}}$	CLEBO$_{\text{loo8}}$	CLEBO$_{\text{loo16}}$	CLEBO$_{\text{loo32}}$
Random Network						
n=100, $1 \times 50^{*}$	7 (10)	7 (10)	17 (10)	17 (10)	17 (10)	18 (10)
n=100, 2×100	–	–	–	–	–	–
n=100, 2×500	–	–	–	–	–	–
n=100, 4×100	–	–	–	–	–	–
n=100, 4×500	–	–	–	–	–	–
n=1000, 2×100	1053 (4)	181 (10)	993 (3)	2184 (1)	–	–
n=1000, 2×500	–	–	–	–	–	–
n=1000, 4×100	–	2768 (3)	–	–	–	–
n=1000, 4×500	–	–	–	–	–	–
Neural-Constrained Knapsack						
n=100, 2×100	39 (9)	52 (9)	52 (9)	47 (10)	52 (9)	47 (10)
n=100, 2×500	563 (6)	782 (9)	781 (9)	780 (9)	780 (9)	780 (9)
n=100, 4×100	154 (4)	94 (6)	107 (5)	78 (6)	78 (6)	78 (6)
n=100, 4×500	789 (4)	640 (5)	293 (4)	281 (5)	281 (5)	281 (5)
n=1000, 2×100	7 (9)	8 (8)	6 (10)	5 (9)	7 (9)	6 (9)
n=1000, 2×500	63 (10)	72 (10)	73 (10)	72 (10)	70 (10)	72 (10)
n=1000, 4×100	71 (9)	63 (10)	64 (10)	60 (10)	63 (10)	60 (10)
n=1000, 4×500	667 (9)	843 (9)	831 (9)	651 (9)	679 (9)	843 (9)
Molecular Design (GraphSAGE)						
N=4	1	1	1	1	1	1
N=6	123	145	123	123	123	123
N=8	403	1093	1093	1093	1093	1093
N=10	350	3404	3404	3404	3404	3404
N=12	327	–	–	–	–	–