TARGET-AWARE DATA GENERATION FOR SAT LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Learning-based methods for NP-hard problems are becoming increasingly effective, but their scalability is often limited by the cost of obtaining labeled training data. For Boolean satisfiability (SAT), standard dataset construction typically requires generating formulas and then invoking a solver to determine their labels, creating a solver-in-the-loop bottleneck that becomes increasingly expensive as problem size grows. In this work, we introduce a target-aware, solver-free framework for SAT data generation that constructs labeled SAT and UNSAT instances directly, without requiring solver-based verification. The generated formulas are designed to match structural properties of a target benchmark, improving their usefulness for downstream learning. We also introduce a linear-programming-aware graph neural network (LPGNN) that incorporates constraint-violation residuals into message passing, allowing the model to exploit the underlying optimization structure of SAT. Together, these components provide a data-centric approach to SAT learning in which scalable, benchmark-aligned data generation complements model architecture. Empirically, our framework achieves orders-of-magnitude faster data generation while producing synthetic instances that effectively improve GNN-based SAT prediction.

1 INTRODUCTION

Boolean satisfiability (SAT) is a canonical NP-complete problem (Cook, 1971). This is a core primitive in formal verification (Biere et al., 1999), combinatorial optimization (Moskewicz et al., 2001), and cryptanalysis (Soos et al., 2009). Despite the worst-case NP-hardness of SAT, modern conflict-driven clause-learning (CDCL) solvers such as CaDiCaL (Biere et al., 2020) can approximately solve many large practical instances efficiently. At the same time, solver performance remains highly sensitive to instance structure, and the design of effective heuristics remains difficult and domain-specific. This has motivated increasing interest in machine learning methods for SAT, with the broader goal of using learned structure to improve performance on hard combinatorial problems.

Recent works have shown that Graph Neural Networks (GNNs) can learn useful representations of SAT formulas from graph-based encodings. One major thread explores alternative graph constructions and literal-incidence encodings to better capture the underlying community and algebraic structure of formulas (Ansótegui et al., 2012; Cheng et al., 2024; Peltonen & Wattenhofer, 2026). Another focuses on providing learned search guidance, ranging from local search heuristics to improving CDCL solver decisions (Yolcu & Póczos, 2019; Wang et al., 2021). Some works focus on optimization-aware architectures that incorporate algebraic feasibility signals, such as linear programming residuals or energy-based functions, directly into message passing (Khalil et al., 2022; Eliasof & Haber, 2024; Cardillo et al., 2025). Together with comprehensive benchmark suites such as G4SATBench (Li et al., 2024), these advances have established SAT as a primary testbed for scaling machine learning methods on NP-hard problems.

A basic premise of machine learning is that performance depends jointly on the architecture, the size of the model, and the quality and scale of the training data. The recent history of language modeling illustrates this point: progress accelerated once next-token prediction provided a scalable self-supervised objective, allowing increasingly large architectures to be trained on massive corpora (Kaplan et al., 2020; Hoffmann et al., 2022; Radford et al., 2019; Brown et al., 2020). Without a scalable source of supervision, however, architectural improvements alone are difficult to exploit.

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.

For SAT learning, recent work has focused on graph representations and GNN architectures, and we also introduce architectural improvements below. However, SAT does not naturally come with a next-token-prediction analogue: supervision is typically obtained by generating formulas and then calling a complete SAT solver to determine whether each instance is SAT (satisfiable formula) or UNSAT (unsatisfiable formula). As formulas become larger and harder, this solver-in-the-loop labeling process becomes increasingly costly, limiting the amount of usable supervision and tying learning progress to exact solver runtime. Recent works have begun to highlight this issue explicitly (Serrano et al., 2024), and we discuss additional related literature in Section 2.

In this work, we address this data bottleneck directly. We propose a target-aware, solver-free data-generation framework that constructs SAT and UNSAT formulas with certified labels by construction while matching structural properties of a target benchmark. The goal is not merely to generate more data, but to generate data that is useful for downstream learning. In particular, we show that synthetic SAT data can be generated with meaningful benchmark-aligned structure. Such data, when combined with GNN architectures, improves predictive performance. This supports a data-centric view: progress depends not only on better architectures, but also on scalable mechanisms for producing informative training data. Although synthetic data augmentation carries an inherent risk of introducing distributional bias, prior work suggests it can act as an effective regularizer, often improving model robustness and downstream generalization (Dao et al., 2019; Shorten & Khoshgoftaar, 2019).

The starting point of our approach is a simple asymmetry. While solving SAT is NP-hard, which implies that finding a satisfying assignment for a given formula is difficult in the worst case; constructing a formula around a known assignment is easier. This is analogous in spirit to the method of manufactured solutions in numerical partial differential equations, where one first chooses an exact solution and then constructs a problem of this kind for which that solution is known (Salari & Knupp, 2000). We exploit the same inverse viewpoint for SAT data generation: rather than sampling a formula and then invoking a solver to determine its label, we first construct a certificate, a satisfying assignment for SAT instances or an explicit contradiction for UNSAT instances, and then generate a Conjunctive Normal Form (CNF) formula consistent with that certificate.

Our generation method is therefore CNF-native and solver-free. For SAT instances, we construct clauses around a planted assignment. For UNSAT instances, we embed a certified contradiction and add benchmark-aligned filler clauses. In both cases, the label is known by construction, avoiding repeated calls to a complete solver during data generation. At the same time, target-aware matching reduces the distributional mismatch that would arise from generic planted constructions.

In addition, we connect this data pipeline to an optimization-aware model design. Using the standard binary linear encoding of CNF formulas, we represent SAT as a feasibility problem and derive a Linear Programming (LP)-aware residual mechanism that injects constraint-violation information into message passing. This exposes algebraic feasibility structure to the GNN and complements the proposed generation framework, following the general direction of optimization-aware graph learning (Eliasof & Haber, 2024; Khalil et al., 2022). Overall, this work advances a simple claim: for machine learning on NP-hard problems, scalable and task-aligned data generation can be as important as model architecture.

Contributions.

- We propose a **target-aware solver-free data-generation framework** for SAT learning, motivated by the broader challenge of scaling machine learning methods for NP-hard problems.
- We introduce a **CNF-native formula-generation strategy** that produces correctly labeled SAT and UNSAT formulas by construction.
- We develop a **benchmark-aligned augmentation approach** that matches generated formulas to structural and algebraic properties of a target distribution, making synthetic data more useful for downstream learning than generic generated instances.
- We formulate the learning pipeline through a **binary linear feasibility view** that connects CNF formulas, generated instances, and graph-based learning in a common algebraic framework.

- We introduce an **LP-aware residual GNN architecture** that incorporates feasibility-related signals into message passing, complementing the proposed data-generation framework with an optimization-aware model component.

2 Related Work

Graph representations for SAT. A central modeling decision in SAT learning is how to represent a CNF formula as a graph. Bipartite constructions dominate the recent literature. NeuroSAT (Selsam et al., 2019) and G4SATBench represent formulas using literal–clause graphs, whereas MILP-SAT-GNN (Cardillo et al., 2025) builds a variable-constraint graph from a mixed-integer formulation. These encodings preserve clause structure and expose the incidence pattern between variables and constraints, but they also imply that variables co-occurring in a clause remain separated by at least two hops in the message-passing graph. Alternative non-bipartite constructions include the variable interaction graph and related variants (Ansótegui et al., 2012). Empirical comparisons, however, generally favor bipartite encodings for satisfiability prediction: SATGL (Cheng et al., 2024) reports a clear advantage for literal–clause graphs, and Fu et al. (2025) similarly find non-bipartite structures to be substantially weaker for this task.

Data augmentation in machine learning. Beyond SAT-specific formula generation, our work connects to the broader paradigm of data augmentation, which is a cornerstone of robust representation learning (Shorten & Khoshgoftaar, 2019; Cubuk et al., 2020). While continuous domains such as computer vision and graphics readily admit semantics-preserving perturbations (e.g., cropping, rotation), augmenting discrete, graph-structured data is inherently more challenging (Ding et al., 2022; Zhao et al., 2022). Graph data augmentation techniques often rely on random node dropping, feature masking, or edge perturbation (Rong et al., 2020; You et al., 2020). However, applying these generic perturbations to combinatorial problems like SAT is highly destructive, as altering a single edge can inadvertently change the satisfiability status of the underlying formula. Consequently, augmentation for NP-hard problems requires structure-preserving techniques. Our framework bypasses destructive perturbations entirely by generating valid, benchmark-aligned formulas from the ground up, providing a domain-native augmentation strategy that scales combinatorial learning safely.

Optimization-aware signals in graph learning. A related line of work enriches graph neural networks with optimization-derived information. BPGNN (Eliasof & Haber, 2024) incorporates a Quadratic Binary Optimization (QUBO) residual, thereby coupling message passing to an energy-based feasibility signal. MIP-GNN (Khalil et al., 2022) propagates linear-constraint residual information to support branching decisions in mixed-integer optimization, and related learning-to-branch approaches use LP slack information as node features (Gasse et al., 2019). Our setting differs from these works in both problem formulation and learning objective. We study SAT classification under a CNF-native ILP encoding and use the resulting residual quantities to enrich the graph representation itself, rather than to imitate branching or other solver decisions.

Training data generation and supervision. Most supervised SAT-GNN pipelines rely on solver-generated labels, making dataset construction dependent on the runtime of complete SAT solvers. A common *generate-and-label* pipeline samples random k-CNF formulas, labels them using a CDCL solver (CaDiCaL (Biere et al., 2020)), and retains only instances with the desired label, discarding the rest. As solver runtime grows near the phase transition, this process becomes increasingly expensive with problem size. Alternative generation methods, including graph generative models such as HardSATGEN (Li et al., 2023), and HardCore (Cotnareanu et al., 2024), focus on producing realistic or computationally challenging benchmarks. However, these methods typically rely on neural generation, solver-in-the-loop verification, or other computationally intensive procedures, limiting their scalability for large-scale GNN pretraining. In contrast, our approach employs a solver-free generation strategy designed for efficient large-scale data generation.

3 Preliminaries

Boolean Satisfiability and CNF. SAT is the decision problem of determining whether a Boolean formula admits an assignment of truth values that makes the formula true. Throughout this paper, we consider formulas in *conjunctive normal form* (CNF), where a formula φ is a conjunction of m

clauses over n binary variables $x_j \in \{0,1\}$. Each clause C_i is a disjunction of these variables or their negations. Thus:

$$\varphi = \bigwedge_{i=1}^{m} C_i. \tag{1}$$

A formula is satisfiable (SAT) if there exists an assignment $\mathbf{x} \in \{0,1\}^n$ under which every clause evaluates to true. For example, $(x_1 \vee \neg x_2) \wedge (x_2 \vee x_3)$ is SAT (e.g., $x_1 = 1, x_2 = 1$), while $(x_1) \wedge (\neg x_1)$ is UNSAT.

Binary Linear Encoding. A CNF formula admits a standard binary linear encoding. By associating each Boolean variable with a binary decision variable, a clause with positive variables P and negative variables N is satisfied if and only if

$$\sum_{j \in P} x_j - \sum_{j \in N} x_j \geq 1 - N. \tag{2}$$

For example, under this encoding, a clause such as $x_1 \vee \neg x_2 \vee x_3$ can be written and rearranged as:

$$x_1 + (1 - x_2) + x_3 \geq 1 \implies x_1 - x_2 + x_3 \geq 0. \tag{3}$$

Applying this conversion clause by clause maps a CNF formula $\varphi = \bigwedge_{i=1}^{m} C_i$ to an integer system of inequalities $A\mathbf{x} \geq \mathbf{b}$, with $A \in \{-1, 0, 1\}^{m \times n}$ and $\mathbf{b} \in \mathbb{Z}^m$. Satisfiability is therefore exactly equivalent to binary feasibility: the formula is SAT if and only if there exists $\mathbf{x} \in \{0,1\}^n$ satisfying all inequalities.

Slack Variables and Clause Semantics. For both analysis and model design, it is convenient to rewrite the inequality system in equality form by introducing a slack vector $\mathbf{s}$ with components $s_i \in \{0, \ldots, k_i - 1\}$ for each clause C_i of width k_i, yielding $A\mathbf{x} - \mathbf{s} = \mathbf{b}$. The slack variable s_i is not merely an algebraic artifact; it has a direct semantic interpretation. It records the number of literals satisfying clause C_i beyond the single literal required for feasibility. Consequently, $s_i = 0$ exactly when the clause is satisfied by a single literal, whereas $s_i > 0$ when it is satisfied by two or more literals.

4 Target-Aware Solver-Free Data Generation

The data-generation pipeline is the primary practical component of our framework. Its starting point is the *solution-first* viewpoint introduced in BPGNN (Eliasof & Haber, 2024): rather than solving each instance to obtain a label, one first constructs an instance around a known solution or contradiction. However, this idea must respect *CNF semantics*. Once the variables and polarities of a clause are fixed, its induced LP inequality is fixed as well; in particular, the LP right-hand side is not a free continuous parameter that can be chosen independently. For this reason, our generator is *CNF-native*: we generate clauses directly in formula space and only afterwards map the resulting CNF into an integer linear program (ILP). This design preserves logical validity at the data-creation stage while also matching the representation consumed by the downstream GNN.

4.1 SAT Instance Generation

The SAT generation process is designed to match the induced LP slack distribution of a target benchmark. Unlike purely structural statistics (e.g., clause width), the LP slack captures how tightly each clause is satisfied under the reference assignment. Specifically, a slack of zero indicates that a clause is satisfied by exactly one literal, whereas larger slacks correspond to increasing redundancy in the satisfying assignment. By matching the slack distribution, the generated formulas inherit a key statistical property associated with the constraint tightness of the target benchmark. Once $\mathbf{x}^\star$ and a clause C_i are fixed, the induced slack is determined:

$$s_i^\star = A_i \mathbf{x}^\star - b_i = \#\{\text{literals in } C_i \text{ satisfied by } \mathbf{x}^\star\} - 1. \tag{4}$$

The slack-planted generator eliminates any remaining mismatch by deterministically constructing each clause to match a target slack $s_i^\star$ drawn from the empirical benchmark SAT slack distribution.

Algorithm 1 SAT Generation

Require: Size n, clauses m, target benchmark statistics $\mathcal{T}$, SAT slack dist. Π_s
1: Sample $\mathbf{x}^\star \in \{0,1\}^n$ uniformly at random
2: **for** $i = 1$ to m **do**
3: Sample width k_i from width distribution in $\mathcal{T}$
4: Sample k_i distinct variables, weighted by occurrence skew in $\mathcal{T}$
5: Sample $s_i^\star \in \{0, \ldots, k_i - 1\}$ from Π_s
6: Set $(s_i^\star + 1)$ literals satisfied and the rest violated under $\mathbf{x}^\star$
7: Add resulting clause C_i to φ
8: **end for**
9: **return** $(\varphi, \mathbf{x}^\star, \mathbf{s}^\star, \text{SAT})$

For a clause of width k_i with target slack $s_i^\star$, we sample k_i distinct variables weighted by the target benchmark's occurrence skew. We then designate $(s_i^\star + 1)$ of them to be *satisfied* (setting their polarity to match $\mathbf{x}^\star$) and the remaining $(k_i - s_i^\star - 1)$ to be *violated* (setting their polarity opposite to $\mathbf{x}^\star$). To prevent the planted assignment from being revealed through polarity statistics, the satisfied and violated roles are assigned by balancing each variable's positive and negative occurrences across the formula. This generation process avoids the costly rejection loops inherent to solver-in-the-loop pipelines. Since the assignments are applied independently to each of the m clauses up to width k, the computational complexity of producing a perfectly aligned, guaranteed-SAT instance is strictly $\mathcal{O}(mk)$. Algorithm 1 gives the complete procedure.

Remark 1 (Non-uniqueness of satisfying assignments). A planted witness does not imply a unique satisfying assignment. For SAT instances, existence of *some* satisfying assignment is sufficient. The witness is used only to generate a correctly labeled formula and to characterize the induced slack assignment presented in Equation 4.

4.2 UNSAT Instance Generation

For UNSAT instances, there is no globally satisfying planted witness. Instead, we certify unsatisfiability by construction. Each instance embeds a *contradiction core* within a larger set of slack-planted *filler* clauses. The core guarantees the UNSAT label; adding arbitrary clauses to an already-UNSAT formula preserves unsatisfiability. The contradiction core is defined as an unsatisfiable subformula over a subset of variables. Its construction is not uniquely determined, as different unsatisfiable cores can provide equivalent unsatisfiability certificates while inducing different computational properties.

Slack-planted filler. Just as the SAT generator plants the slack distribution $\mathbf{s}^\star$, the UNSAT filler generator aligns with the benchmark UNSAT slack distribution by sampling a target slack $s_i^\star$ from the benchmark UNSAT slack distribution. Our method is presented in Algorithm 2.

Remark 2 (Topological Structure of the UNSAT Core). To ensure the network learns generalized constraint satisfaction, we integrate the contradiction core into the global formula. The core is constructed to be structurally similar to the rest of the graph, and the surrounding filler clauses are sampled to match the target domain. This target-aware method embeds the contradiction within a statistically similar topology, ensuring that it acts as an inherent part of the global formula.

5 Linear-Programming Graph Neural Network (LPGNN)

Given a CNF formula φ with n Boolean variables and m clauses, we first map it to the standard binary linear feasibility problem

$$A\mathbf{x} \geq \mathbf{b}, \qquad \mathbf{x} \in \{0,1\}^n, \tag{5}$$

where each row of $A \in \{-1, 0, 1\}^{m \times n}$ corresponds to one clause. We then introduce nonnegative slack variables and write the system in equality form

$$\widehat{A}\mathbf{z} = \mathbf{b}, \qquad \widehat{A} = [\, A \mid -I \,], \qquad \mathbf{z} = \begin{bmatrix} \mathbf{x} \\ \mathbf{s} \end{bmatrix}. \tag{6}$$

Algorithm 2 UNSAT Generation

Require: Size n, clauses m, target benchmark statistics $\mathcal{T}$, UNSAT slack dist. Π_u
1: Generate a contradiction core $\mathcal{C}$ and a reference assignment $\mathbf{x}^\star$
2: Add the core clauses $\mathcal{C}$ to ψ
3: **for** $i = 1$ to $m - |\mathcal{C}|$ **do**
4: Sample width k_i from width distribution in $\mathcal{T}$
5: Sample k_i distinct variables, weighted by occurrence skew in $\mathcal{T}$
6: Sample $s_i^\star \in \{0, \ldots, k_i - 1\}$ from Π_u
7: Set $(s_i^\star + 1)$ literals satisfied and the rest violated under $\mathbf{x}^\star$
8: Add resulting clause C_i to ψ
9: **end for**
10: **return** $(\psi, \mathbf{x}^\star, \mathbf{s}^\star, \text{UNSAT})$

The LPGNN is built on the augmented variable vector $\mathbf{z} = [\mathbf{x}; \mathbf{s}]$. Thus, the graph contains two node types:

$$V = V_x \cup V_s, \qquad |V_x| = n, \qquad |V_s| = m. \tag{7}$$

The node $v_j \in V_x$ represents Boolean variable x_j, and the node $u_i \in V_s$ represents the slack variable s_i associated with clause i. We introduce an edge (u_i, v_j) whenever $A_{ij} \neq 0$, with edge attribute A_{ij}.

Each node is initialized from simple structural features. For variable nodes, these include degree, positive occurrence count, and negative occurrence count. For slack nodes, these include clause width, right-hand side b_i, and literal-composition statistics. Let $\mathbf{q}_p \in \mathbb{R}^{d_{\text{in}}}$ denote the raw feature vector of node $p \in V$. The initial hidden state is

$$\mathbf{h}_p^{(0)} = \psi_{\text{init}}(\mathbf{q}_p) \in \mathbb{R}^d, \qquad p = 1, \ldots, n + m, \tag{8}$$

where ψ_{init} is an MLP shared across nodes, possibly with node-type embeddings.

Layer update. At layer ℓ, the hidden states are collected in

$$H^{(\ell)} = \begin{bmatrix} (\mathbf{h}_1^{(\ell)})^\top \\ \vdots \\ (\mathbf{h}_{n+m}^{(\ell)})^\top \end{bmatrix} \in \mathbb{R}^{(n+m)\times d}. \tag{9}$$

The layer has two components: a standard graph message-passing update and an LP residual update. First, each node embedding is projected to a scalar continuous surrogate for the corresponding coordinate of $\mathbf{z}$:

$$\tilde{z}_p^{(\ell)} = \sigma\Big(\alpha^{(\ell)}(\mathbf{h}_p^{(\ell)})\Big), \qquad p = 1, \ldots, n + m, \tag{10}$$

where $\alpha^{(\ell)} : \mathbb{R}^d \to \mathbb{R}$ is a learned scalar projection and σ is the activation function. In vector form,

$$\tilde{\mathbf{z}}^{(\ell)} = \sigma\Big(\alpha^{(\ell)}(H^{(\ell)})\Big) \in \mathbb{R}^{n+m}. \tag{11}$$

The vector $\tilde{\mathbf{z}}^{(\ell)}$ is not interpreted as a valid discrete assignment. It is a continuous latent surrogate used only to compute feasibility-related features. The clause-level residual is

$$\mathbf{r}^{(\ell)} = \widehat{A}\tilde{\mathbf{z}}^{(\ell)} - \mathbf{b} \in \mathbb{R}^m. \tag{12}$$

This residual lives in constraint space. Since the GNN updates node embeddings associated with the coordinates of $\mathbf{z}$, we map the residual back to node space using the transpose:

$$\mathbf{g}^{(\ell)} = \widehat{A}^\top \mathbf{r}^{(\ell)} \in \mathbb{R}^{n+m}. \tag{13}$$

The pth entry $g_p^{(\ell)}$ aggregates the signed residuals of all constraints in which node p participates. Equivalently, if one temporarily relaxes $\mathbf{z}$ to a continuous variable, then

$$\widehat{A}^\top \mathbf{r}^{(\ell)} = \nabla_{\mathbf{z}} \frac{1}{2} \|\widehat{A}\mathbf{z} - \mathbf{b}\|_2^2 \Big|_{\mathbf{z}=\tilde{\mathbf{z}}^{(\ell)}}. \tag{14}$$

Table 1: Wall-clock time (ms) to generate one correctly labeled SAT/UNSAT pair at the phase transition. **Naive** enumerates all 2^n assignments (capped at a 30s timeout). **CaDiCaL** represents the standard solver-in-the-loop pipeline (generate-and-test). **Ours** evaluates the proposed solver-free, target-aware generation. We present median over 200 repetitions. For $n \geq 500$, CaDiCaL times are extrapolated via log-linear fit from measured data.

n	m	Naive (ms)	CaDiCaL (ms)	Ours (ms)	Speedup
15	64	272	0.66	0.55	$1.2\times$
20	85	6,188	0.87	0.70	$1.2\times$
50	213	N/A	2.26	1.62	$1.4\times$
75	319	N/A	4.47	2.42	$1.8\times$
100	426	N/A	8.98	3.24	$2.8\times$
150	639	N/A	66.1	4.76	$14\times$
200	852	N/A	525	6.43	$82\times$
250	1064	N/A	4,080	8.09	$504\times$
500	2129	N/A	$\sim 10^{8\dagger}$	17.1	$\sim 10^{6}\times$
1000	4258	N/A	$\sim 10^{16\dagger}$	35.0	$\sim 10^{14}\times$

We note that Equation (14) is not a gradient of the original discrete SAT problem, nor does the network perform strict gradient descent on the LP relaxation. The projected residual $\widehat{A}^\top \mathbf{r}^{(\ell)}$ serves as a dynamic feasibility feature injected into the non-linear message-passing mechanism, providing a structured measure of constraint violation. The subsequent neural network layers are trained to filter the noise introduced by the integrality gap, extract a useful directional signal, and map it toward the discrete domain. The scalar node residual is lifted to the hidden dimension by another learned map:

$$\mathbf{e}_p^{(\ell)} = \rho^{(\ell)}\left(g_p^{(\ell)}\right) \in \mathbb{R}^d, \qquad \rho^{(\ell)} : \mathbb{R} \to \mathbb{R}^d. \tag{15}$$

We then combine this LP-aware signal with standard message passing. Let $\mathcal{N}(p)$ denote the neighbors of node p, and let a_{pq} denote the edge attribute, e.g. A_{ij} for variable–slack edges. A generic LPGNN layer is

$$\begin{aligned} \mathbf{m}_p^{(\ell)} &= \sum_{q \in \mathcal{N}(p)} \phi_{\text{msg}}^{(\ell)}\left(\mathbf{h}_p^{(\ell)}, \mathbf{h}_q^{(\ell)}, a_{pq}\right), \\ \bar{\mathbf{h}}_p^{(\ell)} &= \mathbf{h}_p^{(\ell)} + \mathbf{e}_p^{(\ell)}, \qquad \mathbf{h}_p^{(\ell+1)} = \phi_{\text{upd}}^{(\ell)}\left(\bar{\mathbf{h}}_p^{(\ell)}, \mathbf{m}_p^{(\ell)}\right). \end{aligned} \tag{16}$$

Here $\phi_{\text{msg}}^{(\ell)}$ and $\phi_{\text{upd}}^{(\ell)}$ are learned neural networks. In practice, $\phi_{\text{upd}}^{(\ell)}$ may be implemented as an MLP-style update, residual block, or attention-based update. The essential feature of LPGNN is not the specific message-passing primitive, but the additional node-space feasibility signal $\widehat{A}^\top \mathbf{r}^{(\ell)}$ injected at every layer. Equivalently, in matrix notation, one may write a layer abstractly as

$$H^{(\ell+1)} = \text{MPNN}^{(\ell)}\left(H^{(\ell)} + \rho^{(\ell)}\left(\widehat{A}^\top\left[\widehat{A}\sigma\left(\alpha^{(\ell)}(H^{(\ell)})\right) - \mathbf{b}\right]\right), G\right), \tag{17}$$

where $\rho^{(\ell)}$ is applied row-wise to the scalar node residuals, and MPNN stands for the message passing neural network framework.

Slack nodes versus constraint nodes. In a standard bipartite SAT graph implementation, constraint nodes act as summaries of clauses. In contrast, in LPGNN the slack node s_i is not a representation of clause i, but an explicit component of the augmented variable vector $\mathbf{z} = [\mathbf{x}; \mathbf{s}]$. As a result, slack nodes participate directly in the residual computation Equation 12 and receive feasibility signals in node space via Equation 13. This design induces a tight coupling between message passing and constraint satisfaction. The architecture updates the latent representations of both variables and slacks, which together determine the constraint residuals.

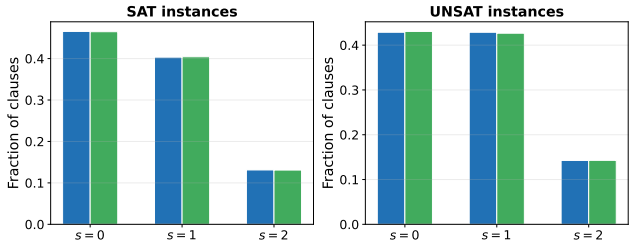


Figure 1: Induced slack distributions of G4SATBench Medium 3-SAT dataset (**blue** bars) and our target-aware synthetic instances (**green** bars), for SAT clauses (left) and UNSAT (right). The generated data mirrors the target.

Table 2: Generation quality on Medium 3-SAT (300 instances per group). Our generated data follows the target across all measures.

	Target	**Gen.**
Clause width	3	3
Modularity	0.16	0.18
Clustering	0.27	0.25
Communities	6.2	6.5
Median CDCL dec.	998	918
Median solve (ms)	3.7	3.7

Table 3: SATLIB results with both SAT and UNSAT instances: baseline vs. baseline+ target-aware generated data. Using generated data improves performance across various metrics and families.

	BA ↑		**AUC** ↑		**CSR** ↑		k/m ↓	
Benchmark set	Baseline	+Gen.	Baseline	+Gen.	Baseline	+Gen.	Baseline	+Gen.
uf/uuf 50-218	60.1±0.8	64.8±2.3	64.0±1.7	72.2±1.6	88.2±0.1	96.2±0.2	12.0±0.2	4.3±0.2
uf/uuf 75-325	60.8±2.4	60.8±2.4	64.3±3.2	68.1±9.3	88.0±0.6	96.2±0.2	12.2±1.4	4.0±0.2
uf/uuf 100-430	61.7±0.4	70.6±1.2	65.7±3.0	80.2±1.1	88.1±0.4	96.6±0.1	11.7±0.5	3.8±0.2
uf/uuf 175-753	64.2±1.2	67.5±2.0	72.7±1.0	73.2±3.4	87.7±0.3	96.4±0.0	12.3±0.2	3.8±0.0
uf/uuf 200-860	58.3±5.9	58.3±5.1	63.5±1.0	65.7±5.8	86.9±1.4	96.4±0.1	12.7±1.1	4.1±0.2
uf/uuf 225-960	58.3±4.7	58.3±7.2	68.5±7.1	70.1±2.9	88.1±0.5	96.7±0.1	11.3±0.7	3.7±0.1
uf/uuf 250-1065	60.0±4.1	65.8±2.4	64.5±6.3	80.8±1.4	87.3±1.0	96.4±0.2	12.8±1.4	3.9±0.2

6 Experiments

We evaluate the proposed framework along three principal dimensions. First, *generation efficiency*: the computational cost of synthesizing instances relative to solver-in-the-loop pipelines. Second, *generation fidelity*: we show that our synthetic data follows the structure of the targeted original benchmark. Third, *downstream utility and scalability*: the impact of data augmentation on neural solver performance. To establish the generalizability of our approach, we conduct these experiments across diverse SAT families from G4SATBench (Li et al., 2024) and SATLIB (Hoos & Stützle, 2000), utilizing multiple neural architectures, including our LPGNN, NLocalSAT (Zhang et al., 2020), and QuerySAT (Ozolins et al., 2022). To demonstrate the stability of our method, we report the mean and standard deviation across three random seeds.

Generator Efficiency. We evaluate the computational efficiency of our solver-free generation method against a standard solver-in-the-loop pipeline (Table 1). The *naive* baseline demonstrates theoretical worst-case exponential scaling. The CaDiCaL (Biere et al., 2020) baseline represents the conventional approach: repeatedly sampling random 3-CNF formulas near the satisfiability phase transition (Crawford & Auton, 1996) and applying a solver to verify labels. Our proposed target-aware generator scales near-linearly with the number of clauses, as satisfiability or unsatisfiability is guaranteed by construction without solver verification. While initialization overhead yields times comparable to CaDiCaL for small formulas, our method demonstrates significant scaling stability at benchmark-relevant sizes where the solver-in-the-loop approach becomes infeasible. To further demonstrate the efficiency of our method, we provide in Appendix A.1 additional comparisons with recent SAT generators HardSATGEN (Li et al., 2023) and HardCore (Cotnareanu et al., 2024).

Data Generation Quality. Figure 1 shows that our synthetic instances reproduce the G4SATBench Medium 3-SAT slack distribution. We additionally compare the generated data in terms of community structure (modularity, clustering, number of communities) and the solver-perceived hardness (CDCL decisions and CaDiCaL solve time, (Garzón et al., 2022)). Table 2 shows that the generated instances

Table 4: Scaling results for architectures trained on synthetic data, evaluated on the Medium 3-SAT test set. Both models gain higher assignment quality from the generated data. NL denotes **NLocalSAT** and QS denotes **QuerySAT**.

Train Volume	**Medium 3-SAT**			
	CSR (%) ↑		k/m (%) ↓	
	NL	QS	NL	QS
500	97.0±0.3	93.2±0.9	3.6±0.4	7.1±0.6
1k	97.2±0.7	94.0±0.6	3.4±0.9	6.5±0.5
5k	97.3±0.6	94.6±0.1	3.3±0.7	6.0±0.1
10k	97.9±0.3	94.3±0.7	2.5±0.3	6.1±0.6

Table 5: Classification scaling of LPGNN from original baseline to target-aware synthetic instances, on Medium (M) and Easy (E) 3-SAT (3-S) and `ps` families. Using the generated data yields stronger classification results.

	BA ↑		**AUC ↑**	
Family	Base.	+Gen.	Base.	+Gen.
3-S M	66.2±2.8	70.2±0.8	72.8±2.6	79.3±0.2
3-S E	61.5±1.1	65.0±2.9	68.4±0.9	72.2±1.8
`ps` M	88.0±1.2	89.0±1.2	95.5±0.5	95.6±1.0
`ps` E	82.5±2.3	84.8±1.0	90.9±1.3	92.3±1.0

match the target on every axis, reproducing both the structural shape and the difficulty of the benchmark. Corresponding validation with additional families is presented in Section A.2.

Scaling Behavior. We utilize the G4SATBench and SATLIB datasets for our experiments. Assignment quality is quantified via the Constraint Satisfaction Rate (CSR) of SAT instances and the violated-clause fraction (k/m) of UNSAT instances, where k is the number of violated clauses in the predicted assignment, and m represents the total number of clauses in the formula. Classification quality is measured using Balanced Accuracy (BA) and AUC, which captures the model's overall ability to distinguish between SAT and UNSAT instances.

Table 3 presents results on the SATLIB dataset when target-aware synthetic instances are added to the original data. We observe that augmentation improves over the original baseline across all metrics. The proposed data-generation framework is not restricted to a specific model, and it is compatible with any desired architecture. Table 4 contains results for two architectures: NLocalSAT and QuerySAT. As shown in Table 5, augmenting the real training set with target-aware synthetic instances improves LPGNN's balanced accuracy and AUC over the baseline. Note that random 3-SAT formulas generated at the phase transition are designed to lack trivial structural shortcuts, making them difficult to solve without complete algorithmic search. To further evaluate our framework, we provide additional empirical results in Appendix B, including SATLIB results, additional G4SATBench families and generalization tests. Additionally, we present an ablation study in Appendix C, and comprehensive dataset details and further experimental settings are provided in Appendix D.

7 Conclusion

Learning-based approaches to NP-hard problems are often bottlenecked not by model capacity, but by the cost of obtaining labeled data. We have introduced a target-aware, solver-free data generation framework that eliminates the reliance on expensive solver-in-the-loop labeling, enabling the efficient production of large-scale, correctly labeled SAT data. By constructing instances with certified labels and aligning them to benchmark structure, our approach provides both computational efficiency and task-relevant supervision. Empirically, this translates into orders-of-magnitude speedups in data generation and clear performance gains when training GNNs.

More broadly, our results support a data-centric perspective on learning NP-hard problems, highlighting that efficient, distribution-aware data generation is a key enabler for unlocking the potential of modern machine learning methods. Building on this foundation, future work will explore expanding this generation framework to capture the complex, hidden community structures required for specialized industrial SAT applications, as well as adapting similar target-aware methodology to synthesize training data for other NP-hard combinatorial optimization problems. Ultimately, this work demonstrates that scalable, task-aligned data generation is just as critical as model architecture in advancing deep learning for complex combinatorial reasoning.

AI USE STATEMENT

In this work, we used generative AI tools for grammar correction and improving the clarity and readability of the manuscript. We have not used generative AI tools for generating research ideas, developing the methodology, analyzing or interpreting results, or generating scientific claims. We have reviewed all AI-assisted work and verified that it accurately reflects the intended meaning and scientific content of the work. We take responsibility for the final content of this work.

ETHICS STATEMENT

The ability to efficiently train scalable, learning-based heuristics for NP-hard problems carries dual-use implications. On the positive side, accelerating Boolean satisfiability directly benefits formal verification, software reliability, and hardware design, contributing to the development of safer and more robust computing systems. However, since SAT solvers are highly generalizable, advancements in this domain could theoretically be leveraged for adversarial purposes, such as cryptanalysis and breaking cryptographic protocols. We mitigate these concerns by focusing our evaluation exclusively on standard, open-source benchmarks and clearly presenting our data generation framework to promote transparent, defensive research. Our study does not involve human subjects or introduce privacy, bias, or fairness concerns.

REPRODUCIBILITY STATEMENT

To ensure the reproducibility of our results, we have explicitly outlined the procedures for our target-aware, solver-free data generation pipeline in Algorithm 1 (SAT Generation) and Algorithm 2 (UNSAT Generation). Comprehensive details regarding the benchmark datasets are provided in Appendix D. The architectural details, objective functions, and training hyperparameters for LPGNN and other evaluated baseline models are documented in Appendix D.3. To ensure statistical reliability, all empirical evaluations are reported with the mean and standard deviation across three random seeds. Our code will be made publicly available upon acceptance.

REFERENCES

Carlos Ansótegui, Jesús Giráldez-Cru, and Jordi Levy. The community structure of SAT formulas. In *Theory and Applications of Satisfiability Testing (SAT)*, pp. 410–423, 2012.

Armin Biere, Alessandro Cimatti, Edmund Clarke, and Yunshan Zhu. Symbolic model checking without bdds. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, pp. 193–207, 1999. doi: 10.1007/3-540-49059-0_14.

Armin Biere, Katalin Fazekas, Mathias Fleury, and Maximilian Heisinger. Cadical, kissat, paracooba, plingeling and treengeling entering the sat competition 2020. In *Proc. of SAT Competition 2020 - Solver and Benchmark Descriptions*, volume vol. B-2020-1 of *Department of Computer Science Report Series B*, pp. 50–53. University of Helsinki, 2020.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, volume 33, pp. 1877–1901, 2020.

Franco Alberto Cardillo, Hamza Khyari, and Umberto Straccia. MILP-SAT-GNN: Yet another neural SAT solver. *arXiv preprint arXiv:2507.01825*, 2025.

HongTao Cheng, Jiawei Liu, Jianwang Zhai, Mingyu Zhao, Cheng Yang, and Chuan Shi. Satgl: An open-source graph learning toolkit for boolean satisfiability. In *2024 2nd International Symposium of Electronics Design Automation (ISEDA)*, pp. 746–751. IEEE, 2024.

Stephen A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing (STOC)*, pp. 151–158, 1971.

Joseph Cotnareanu, Zhanguang Zhang, Hui-Ling Zhen, Yingxue Zhang, and Mark Coates. Hardcore generation: Generating hard unsat problems for data augmentation. *Advances in Neural Information Processing Systems*, 37:62409–62431, 2024.

James M Crawford and Larry D Auton. Experimental results on the crossover point in random 3-sat. *Artificial Intelligence*, 81(1-2):31–57, 1996.

Ekin D Cubuk, Barret Zoph, Jonathon Shlens, and Quoc V Le. Randaugment: Practical automated data augmentation with a reduced search space. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 3008–3017, 2020.

Tri Dao, Albert Gu, Alexander Ratner, Virginia Smith, Chris De Sa, and Christopher Re. A kernel theory of modern data augmentation. In Kamalika Chaudhuri and Ruslan Salakhutdinov (eds.), *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 1528–1537. PMLR, 09–15 Jun 2019.

Kaize Ding, Zhe Xu, Hanghang Tong, and Huan Liu. Data augmentation for deep graph learning: A survey. *ACM SIGKDD Explorations Newsletter*, 24(2):61–77, 2022.

Moshe Eliasof and Eldad Haber. Quadratic binary optimization with graph neural networks. *arXiv preprint arXiv:2404.04874*, 2024.

Yi Fu, Anthony Tompkins, Yang Song, and Maurice Pagnucco. Structure based sat dataset for analysing gnn generalisation. 2025.

Iván Garzón, Pablo Mesejo, and Jesús Giráldez-Cru. On the performance of deep generative models of realistic sat instances. In *25th International Conference on Theory and Applications of Satisfiability Testing (SAT 2022)*, pp. 3–1. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2022.

Maxime Gasse, Didier Chételat, Nicola Ferroni, Laurent Charlin, and Andrea Lodi. Exact combinatorial optimization with graph convolutional neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*, 2022.

Holger H Hoos and Thomas Stützle. Satlib: An online resource for research on sat. *Sat*, 2000: 283–292, 2000.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

Elias B. Khalil, Christopher Morris, and Andrea Lodi. MIP-GNN: A data-driven framework for guiding combinatorial solvers. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2022.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, 2015.

Yang Li, Xinyan Chen, Wenxuan Guo, Xijun Li, Wanqian Luo, Junhua Huang, Hui-Ling Zhen, Mingxuan Yuan, and Junchi Yan. Hardsatgen: Understanding the difficulty of hard sat formula generation and a strong structure-hardness-aware baseline. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 4414–4425, 2023.

Zhaoyu Li, Jinpei Guo, and Xujie Si. G4SATBench: Benchmarking and advancing SAT solving with graph neural networks. *Transactions on Machine Learning Research*, 2024.

Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an efficient sat solver. In *Proceedings of the 38th Annual Design Automation Conference (DAC)*, pp. 530–535, 2001. doi: 10.1145/378239.379017.

Emils Ozolins, Karlis Freivalds, Andis Draguns, Eliza Gaile, Ronalds Zakovskis, and Sergejs Kozlovics. Goal-aware neural sat solver. In *2022 International joint conference on neural networks (IJCNN)*, pp. 1–8. IEEE, 2022.

Saku Peltonen and Roger Wattenhofer. On the expressive power of GNNs for boolean satisfiability. In *International Conference on Learning Representations (ICLR)*, 2026.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. *OpenAI technical report*, 2019.

Yu Rong, Wenbing Huang, Tingyang Xu, and Junzhou Huang. Dropedge: Towards deep graph convolutional networks on node classification. In *International Conference on Learning Representations (ICLR)*, 2020.

Kambiz Salari and Patrick Knupp. Code verification by the method of manufactured solutions. Technical Report SAND2000-1444, Sandia National Laboratories, 2000.

Daniel Selsam, Matthew Lamm, Benedikt Bünz, Percy Liang, Leonardo de Moura, and David L. Dill. Learning a SAT solver from single-bit supervision. In *International Conference on Learning Representations (ICLR)*, 2019.

Christopher R Serrano, Jonathan Gallagher, Kenji Yamada, Alexei Kopylov, and Michael A Warren. Self-satisfied: An end-to-end framework for sat generation and prediction. *arXiv preprint arXiv:2410.14888*, 2024.

Connor Shorten and Taghi M Khoshgoftaar. A survey on image data augmentation for deep learning. *Journal of Big Data*, 6(1):1–48, 2019.

Mate Soos, Karsten Nohl, and Claude Castelluccia. Extending sat solvers to cryptographic problems. *Lecture Notes in Computer Science*, 5584:244–257, 2009. doi: 10.1007/978-3-642-02777-2_24.

Grigori S Tseitin. On the complexity of derivation in propositional calculus. In *Automation of reasoning: 2: Classical papers on computational logic 1967–1970*, pp. 466–483. Springer, 1983.

Alasdair Urquhart. Hard examples for resolution. *Journal of the ACM (JACM)*, 34(1):209–219, 1987.

Wenxi Wang, Yang Hu, Mohit Tiwari, Sarfraz Khurshid, Kenneth McMillan, and Risto Miikkulainen. Neuroback: Improving cdcl sat solving using graph neural networks. 2021.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *International Conference on Learning Representations (ICLR)*, 2019.

Emre Yolcu and Barnabás Póczos. Learning local search heuristics for boolean satisfiability. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

Yuning You, Tianlong Chen, Yongduo Sui, Ting Chen, Zhangyang Wang, and Yang Shen. Graph contrastive learning with augmentations. In *Advances in neural information processing systems*, volume 33, pp. 5812–5823, 2020.

Wenjie Zhang, Zeyu Sun, Qihao Zhu, Ge Li, Shaowei Cai, Yingfei Xiong, and Lu Zhang. Nlocalsat: Boosting local search with solution prediction. 2020.

Tong Zhao, Wei Jin, Yozen Liu, Yingheng Wang, Gang Liu, Stephan Günnemann, Neil Shah, and Meng Jiang. Graph data augmentation for graph machine learning: A survey. *arXiv preprint arXiv:2202.08871*, 2022.

A GENERATION QUALITY

To establish the quality and fidelity of the generated data, we further evaluate the instances across multiple settings. A comparison with recent generative baselines is presented in Appendix A.1, and per-family quantitative results are provided in Appendix A.2.

A.1 COMPARISON WITH GENERATIVE BASELINES

We compare our method with recent learned SAT generators, specifically HardSATGEN (Li et al., 2023) and HardCore (Cotnareanu et al., 2024). While these baselines employ neural generative models and solver-in-the-loop certification to synthesize small sets of hard evaluation instances (Garzón et al., 2022), our framework targets the complementary objective of generating massive volumes of target-aligned training data cheaply. This dictates fundamentally different mechanisms. HardSATGEN relies on an inference-heavy neural split-merge procedure, and HardCore embeds GNN-detected cores into formulas sampled via the power-law engine, both incur substantial per-instance computational costs. In contrast, our generator bypasses learned models and solvers entirely: utilizing pre-extracted target statistics, it plants a satisfying assignment or a contradiction core with statistically matched filler in an $\mathcal{O}(mk)$ pass.

We evaluate the generation time of the above methods on identical target dimensions (n, m), for `mp1` and `sp4` instances (Li et al., 2023). The empirical results demonstrate that our approach obtains a significant timing gain, as presented in Table 6.

Table 6: Per-instance generation time (seconds) on identical target dimensions

Instance (n, m)	**HardSATGEN**	**HardCore**	**Ours**
mp1			
(400, 1737)	65	0.77	**0.007**
(851, 2731)	138	2.53	**0.012**
(932, 2994)	151	3.15	**0.012**
sp4			
(819, 4423)	224	4.42	**0.017**
(786, 5793)	362	5.37	**0.023**
(2052, 3959)	252	9.48	**0.018**

A.2 STRUCTURAL AND HARDNESS FIDELITY

We characterize structural properties of our generated data instances. We summarize our results with three measures: the *modularity*, which quantifies how cleanly the graph decomposes into such groups (higher score meaning a more pronounced community structure); the *clustering coefficient*, which measures local link density, i.e. how often two neighbors of a variable are themselves adjacent; and the *number of communities* recovered by the partition. Computational hardness is measured directly, as the median number of CDCL decisions and the median CaDiCaL (Biere et al., 2020) solve time, following (Garzón et al., 2022).

As demonstrated in Table 7 (Easy 3-SAT) and Table 8 (`ps`), the generated and target instances exhibit strong alignment across all evaluated metrics. They share identical clause-width distributions and demonstrate agreement in graph-theoretic properties, including modularity, clustering coefficient, and community count, alongside analogous CDCL decision counts and solve times. Consequently, the generator captures not merely the clause statistics of the benchmarks, but also their underlying structural topologies and solver-perceived hardness.

B ADDITIONAL RESULTS

B.1 DATA SCALING ON G4SATBENCH 3-SAT

Table 9 reports the scaling results on G4SATBench Medium and Easy 3-SAT tiers. We show results for LPGNN trained on the baseline training set, augmented with added 1k synthetic instances, and

Table 7: Generation quality on Easy 3-SAT (300 instances): our generated instances match the target benchmark on structure and solver-perceived hardness.

	Easy 3-SAT	
	Target	**Generated**
Clause width	3	3
Modularity	0.07	0.09
Clustering	0.68	0.63
Communities	3.0	3.0
Median CDCL decisions	14	14
Median solve time (ms)	< 0.1	< 0.1

Table 8: Generation quality on ps (300 instances), a structured pseudo-industrial family: our generated instances match the target community structure and solver hardness at both tiers.

	Medium ps		Easy ps	
	Target	**Generated**	**Target**	**Generated**
Modularity	0.02	0.01	0.00	0.00
Clustering	0.94	0.94	1.00	1.00
Communities	2.0	2.0	1.0	1.0
Median CDCL decisions	30	50	12	12
Median solve time (ms)	0.1	0.1	< 0.1	< 0.1

evaluated on the corresponding G4SATBench test sets. Augmentation improves over the baseline across all metrics.

Table 9: Scaling of LPGNN on the Medium and Easy 3-SAT families as target-aware synthetic instances are added to the original training set (evaluated on the G4SATBench test set). Across both structural difficulty tiers, augmenting the training corpus with generated synthetic data consistently improves downstream performance for both satisfiability classification and assignment quality.

Target Family	**Data**	**BA** $\uparrow$	**AUC** $\uparrow$	**CSR (%)** $\uparrow$	k/m **(%)** $\downarrow$
Medium 3-SAT	Baseline	66.2 ± 2.8	72.8 ± 2.6	97.0 ± 0.2	3.5 ± 0.2
	+Gen.	72.7 ± 0.8	79.4 ± 0.3	97.5 ± 0.1	3.0 ± 0.1
Easy 3-SAT	Baseline	61.5 ± 1.1	68.4 ± 0.9	96.9 ± 0.5	4.4 ± 0.3
	+Gen.	64.7 ± 3.0	71.1 ± 3.2	97.0 ± 0.1	4.1 ± 0.1

B.2 Additional SATLIB Results

We apply our method using additional families from the SATLIB benchmark suite. For each family we compare a baseline LPGNN trained only on the family's real instances with baseline + generated, which adds target-aware synthetic instances equal to five times the family's size, using three seeds and an $80/20$ train/test split. The families fall into two groups by the labels they provide. Families with both satisfiable and unsatisfiable instances support four metrics: balanced accuracy (BA), AUC, CSR, and the violated-clause fraction k/m, with results presented in Table 3. Results for satisfiable-only families are evaluated by CSR and presented in Table 10.

B.3 LPGNN Generalization Tests

Table 11 reports an inference-time evaluation. We utilize the models trained on 3-SAT Easy, without any re-training, to the test sets of the other 3-SAT tiers and of the SR family at all three tiers. We observe that using a model that was additionally trained using synthetic data improves generalization performance.

Table 10: Satisfiable-only SATLIB results: baseline vs. baseline+ target-aware generated data, evaluated by constraint-satisfaction rate (CSR). Generated data yields stronger CSR rate.

	CSR (%) ↑	
Family	Baseline	+Gen.
Backbone-controlled `cbs_b10`	87.3±0.2	97.5±0.1
Backbone-controlled `cbs_b30`	88.4±1.2	97.2±0.0
Backbone-controlled `cbs_b50`	88.2±1.0	97.4±0.0
Backbone-controlled `cbs_b70`	87.5±0.0	97.5±0.1
Backbone-controlled `cbs_b90`	87.5±0.0	97.2±0.2
Backbone-minimal `rti`	87.2±0.0	97.4±0.0
Backbone-minimal `bms`	87.4±0.0	96.1±0.4

Table 11: Generalization of the LPGNN model trained on Easy 3-SAT. Across all settings, using synthetic data yields stronger performance.

	CSR (%) ↑		k/m (%) ↓	
Target set	Baseline	+Gen.	Baseline	+Gen.
3-SAT Medium	96.8±0.3	97.2±0.1	3.7±0.3	3.2±0.2
3-SAT Hard	96.7±0.3	97.2±0.1	3.6±0.3	3.1±0.1
SR Easy	91.0±1.1	92.5±0.6	9.1±1.2	7.4±0.7
SR Medium	90.9±1.4	92.9±0.5	9.1±1.3	7.2±0.5
SR Hard	90.9±1.4	93.2±0.4	9.1±1.4	6.8±0.4

C Ablation Studies

We ablate our framework to check our target-aware data-generation method and the LP-residual architectural mechanism.

C.1 Data Generation

To demonstrate the necessity of target-aware statistics for downstream performance, we train LPGNN on 10k synthetic instances and evaluate it on the real Medium 3-SAT test set. We compare two distinct generation regimes: a *Generic* baseline, utilizing uniform variable sampling and no target matching, and our *Full Method*, which aligns the synthetic data with the target benchmark's occurrence skew and induced LP slack distributions. As detailed in Table 12, incorporating target-aware statistics via our Full Method improves downstream generalization. These results illustrate that generic synthetic instances are insufficient for optimal model training, and that embedding the structural and marginal properties of the target distribution into the generation process is important for maximizing neural solver efficacy.

Table 12: Comparison of generic versus target-aware synthetic data on Medium 3-SAT.

Generation Regime	**CSR (%) ↑**	k/m **(%) ↓**
Generic (no matching)	94.6	6.1
Full Method	**97.3**	**3.0**

C.2 LP Residual

To isolate the contribution of the linear programming (LP) residual mechanism, we compare the LPGNN architecture against a baseline that uses only the underlying GIN backbone. Both models are trained on the G4SATBench Medium 3-SAT training set and evaluated on the Medium 3-SAT test set over three seeds. Table 13 reports the comparison, where we demonstrate the effect of our architecture mechanism.

Table 13: Contribution of the LP residual: LPGNN in comparison with a GIN-backbone baseline, both trained on Medium 3-SAT training set and evaluated on Medium 3-SAT test set. LPGNN mechanism shows improvement over the baseline

Metric	LPGNN	GIN Backbone
CSR (%) ↑	97.0 ± 0.2	95.1 ± 0.1
k/m (%) ↓	3.5 ± 0.2	5.5 ± 0.1

D Experimental Setup

D.1 G4SATBench Details

We utilize the G4SATBench benchmark (Li et al., 2024), which provides random 3-SAT instances categorized into three difficulty levels, alongside additional formula families designed for out-of-distribution (OOD) evaluation. We note that the slack distribution of SAT and UNSAT instances described in Algorithms 1 and 2 refers to the training sets of the benchmark.

3-SAT Instances. The core 3-SAT dataset is constructed near the theoretical satisfiability phase transition, exhibiting a clause-to-variable ratio of $\alpha = m/n \approx 4.27$. These instances are partitioned by difficulty into Easy, Medium, and Hard tiers, with each tier maintaining an equal balance of satisfiable and unsatisfiable formulas. Both the Easy and Medium tiers contain 1,600 training instances, 200 validation instances, and 200 test instances. The Easy instances range from $n = 10$ to 40 variables ($m \in [55, 175]$) with an average α of 4.59. The Medium instances range from $n = 40$ to 200 variables ($m \in [175, 850]$) with an average α of 4.28. The Hard tier is reserved exclusively for testing, comprising 200 instances with $n \in [200, 300]$ variables and $m \in [853, 1278]$ clauses, yielding an average α of 4.26.

ps Instances. The `ps` family provides structured pseudo-industrial instances. We use the Easy and Medium tiers, each with an equal balance of satisfiable and unsatisfiable formulas and the same 1,600 training, 200 validation, and 200 test split. The Easy instances range from $n = 10$ to 40 variables ($m \in [59, 319]$) with an average α of 6.96, and the Medium instances from $n = 40$ to 100 variables ($m \in [240, 781]$) with α of 6.92.

Additional Families. To assess out-of-distribution generalization, we utilize the SR family provided by the benchmark. Consistent with the 3-SAT testing protocol, the SR evaluation sets contain 200 test instances (100 SAT and 100 UNSAT) for each of the three difficulty levels. For the satisfiable subsets, the SR instances exhibit a mean variable and clause count of $n = 27$, $m = 151$ for the Easy tier; $n = 123$, $m = 680$ for the Medium tier; and $n = 302$, $m = 1640$ for the Hard tier.

D.2 Contradiction Core Construction

The UNSAT core must satisfy two distinct requirements: it must analytically certify the UNSAT label without a solver, and it must induce refutations as hard as the target distribution. We construct the core as a Tseitin formula (Tseitin, 1983) on a random 3-regular graph. By assigning vertices an odd total charge, the system becomes unsatisfiable by a parity argument: summing the XOR constraints counts every edge twice, yielding an even sum that contradicts the odd right-hand side. Tseitin formulas over expander graphs require exponential-size resolution refutations (Urquhart, 1987).

To integrate the core, we pad it with $m - |\mathcal{C}|$ satisfiable filler clauses to match the target size m. A near-satisfying reference assignment $\mathbf{x}^\star$, which violates exactly one core clause, is generated in linear time by propagating charges along a spanning tree. While the target's width, occurrence, and slack distributions are preserved, the core's graph size is calibrated per family to match the original data solve cost and hardness spread. We sample uniformly from $[60, 92]$ for the phase-transition Medium 3-SAT family, $[4, 18]$ for Easy 3-SAT and Easy `ps`, and $[8, 26]$ for Medium `ps`.

D.3 Implementation Details

All of our runs are single-GPU on NVIDIA RTX 4090 (24 GB). In the following, we elaborate the experimental details.

D.3.1 LPGNN

Our experiments use the LPGNN model with a GIN (Xu et al., 2019) backbone, with per-layer residual connections. Table 14 lists the full configuration. Training minimizes a binary cross-entropy classification loss together with an assignment loss.

Table 14: LPGNN training configuration.

Hyperparameter	Value
Hidden dimension d	128
Layers	16
Optimiser	Adam (Kingma & Ba, 2015)
Learning rate	10^{-3}
Weight decay	10^{-6}
Batch size (3-SAT easy / medium)	256 / 64
Max epochs	100
Early-stopping patience	15
Loss	BCE for classification and assignment
Scheduler	ReduceLROnPlateau (factor=0.5, patience=5)

D.3.2 Additional Architecture

NLocalSAT (Zhang et al., 2020). NLocalSAT extends NeuroSAT (Selsam et al., 2019) with a per-variable assignment head that predicts a soft variable assignment at each inference step. The model operates on the Literal–Clause Graph (LCG): $2n$ literal nodes and m clause nodes exchange messages via an LSTM recurrent cell for T steps, sharing complement information between positive and negative literals after each iteration. The per-variable assignment logits are derived by averaging the positive and negative literal embeddings for each variable.

QuerySAT (Ozolins et al., 2022). QuerySAT likewise operates on the LCG with LSTM-based message passing, but introduces a *query gradient mechanism*: at each step the network produces a trial variable assignment, evaluates it against a differentiable clause-loss (fraction of unsatisfied clauses), and computes the gradient of that loss with respect to the trial assignment. This gradient vector is concatenated back into the literal embeddings as additional input to the next LSTM step, giving the network explicit first-order feedback about which variables are most responsible for unsatisfied clauses.

Table 15 summarises the configurations used for the NLocalSAT and QuerySAT. Both models operate on the Literal–Clause Graph (LCG) and are trained with the same continuous-volume pipeline used for LPGNN.

Table 15: Training configuration for NLocalSAT and QuerySAT. All hyperparameters not listed here match the LPGNN configuration in Table 14.

Hyperparameter	NLocalSAT	QuerySAT
Message-passing steps	16	32
Hidden dimension	128	128
Query dimension	—	128
Optimiser	Adam (Kingma & Ba, 2015)	Adam (Kingma & Ba, 2015)
Learning rate	10^{-3}	10^{-3}
Weight decay	10^{-6}	10^{-6}
Batch size (easy / medium)	256 / 64	256 / 64
Max epochs	100	100
Early-stopping patience	15	15