

---

# Augment with Care: Contrastive Learning for Combinatorial Problems

---

Haonan Duan<sup>\*12</sup> Pashootan Vaezipoor<sup>\*12</sup> Max B. Paulus<sup>3</sup> Yangjun Ruan<sup>12</sup> Chris J. Maddison<sup>12</sup>

## Abstract

Supervised learning can improve the design of state-of-the-art solvers for combinatorial problems, but labelling large numbers of combinatorial instances is often impractical due to exponential worst-case complexity. Inspired by the recent success of contrastive pre-training for images, we conduct a scientific study of the effect of augmentation design on contrastive pre-training for the Boolean satisfiability problem. While typical graph contrastive pre-training uses label-agnostic augmentations, our key insight is that many combinatorial problems have well-studied invariances, which allow for the design of *label-preserving augmentations*. We find that label-preserving augmentations are critical for the success of contrastive pre-training. We show that our representations are able to achieve comparable test accuracy to fully-supervised learning while using only 1% of the labels. We also demonstrate that our representations are more transferable to larger problems from unseen domains. Our code is available at <https://github.com/h4duan/contrastive-sat>.

## 1. Introduction

Combinatorial problems, e.g., Boolean satisfiability (SAT) or mixed-integer linear programming (MILP), have many applications in the industry and can encode many fundamental computational tasks. These problems are NP-complete, so solvers that perform efficiently in the worst case are not within reach. However, learning can be used to improve the average complexity of solvers on the population of combinatorial problems found in the wild (Nair et al., 2020).

---

<sup>\*</sup>Equal contribution <sup>1</sup>University of Toronto <sup>2</sup>Vector Institute <sup>3</sup>ETH Zürich. Correspondence to: Haonan Duan <haonand@cs.toronto.edu>.

Supervised learning is a promising approach to combinatorial solver design (e.g., Selsam & Bjørner, 2019; Nair et al., 2020). Unfortunately, the need for labels is a severe limitation. Many real-world problems, such as cryptography SAT instances, are extremely hard or, in some cases, even impossible to solve (Nejati & Ganesh, 2019). Computing expert branching labels for large-scale MILPs requires sophisticated parallel solvers (Nair et al., 2020).

In order to scale learning for combinatorial problems, we ask: how much can we learn from *unlabelled* combinatorial instances? In this work, we consider a contrastive learning approach, which begins by creating multiple “views” of every unlabelled instance, a process called augmentation. An encoder is trained to maximize the similarity between the representations of augmentations that come from the same instance, while minimizing the similarity between those of distinct ones (Chen et al., 2020a). This has been successful in computer vision: contrastive representations can be used with linear predictors to achieve competitive accuracies on ImageNet using a fraction of the labelled instances (Chen et al., 2020a; He et al., 2020; Chen et al., 2020b).

Our key insight is that combinatorial problems have well-studied invariances that can be used to design extremely effective augmentations for contrastive learning. Contrastive learning theory indicates that augmentations should be (roughly) label-preserving in order to confer guarantees on downstream prediction (Arora et al., 2019; Tosh et al., 2021; HaoChen et al., 2021). This is in contrast to the majority of label-agnostic graph contrastive frameworks (e.g., You et al., 2020; Hassani & Khasahmadi, 2020). For some combinatorial problems, label-preserving transformations are available from subroutines of existing solvers, e.g., variable elimination modifies SAT formulas while preserving their satisfiability. Our augmentations produce new formulas by randomly applying such satisfiability-preserving transformations. Crucially, our augmentations do not require full solves and are much cheaper to compute than the labels.

We study data augmentations for contrastive learning for Boolean satisfiability. We demonstrate that:

- Augmentation design is critical for contrastive pre-training on combinatorial problems. In particular, our

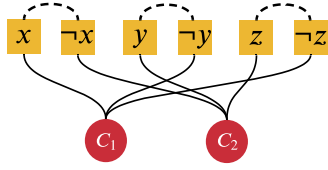


Figure 1. A bipartite graph representation ( $LIG^+$ ) of a SAT formula  $\phi := (x \vee \neg y \vee \neg z) \wedge (\neg x \vee y \vee z)$ . The subgraph without  $---$  edges is the LIG of  $\phi$ .

augmentations are sufficient for strong performance, while existing graph augmentations are not.

- Our contrastive method matches the best supervised baseline while using  $100\times$  fewer training labels.
- Our contrastive representations transfer better to new problem types than supervised representations.

## 2. Background

**SAT.** A Boolean formula in propositional logic consists of Boolean variables composed by logical operators “and” ( $\wedge$ ), “or” ( $\vee$ ) and “not” ( $\neg$ ). A *literal* is a variable  $v$  or its negation  $\neg v$ . A *clause* is a disjunction of literals  $\bigvee_{i=1}^n l_i$ . A Boolean formula is in *Conjunctive Normal Form* (CNF) if it is a conjunction of clauses. We assume that all formulas are in CNF. A variable assignment satisfies a clause when at least one of its literals is satisfied. A formula  $\phi$  is satisfied under  $\pi$  when all of its clauses are satisfied, and  $\pi$  is a *satisfying assignment* for  $\phi$ . SAT is the problem of deciding, for a given formula  $\phi$ , if there exists a satisfying assignment (SAT) or not (UNSAT).

We can represent a SAT formula  $\phi$  by a bipartite graph called *literal-clause incidence graph* (LIG) (Figure 1). The graph contains a node for every clause and literal of  $\phi$ . An edge connects a clause node and a literal node iff the clause contains that literal. We define  $LIG^+$  when the literal nodes of the same variable in LIG are connected.

**NeuroSAT.** Selsam et al. (2018) proposed NeuroSAT to classify satisfiability of Boolean formulas. NeuroSAT consists of two parts: **1) Encoder:** a special *Graph Neural Network* (GNN) that takes in the  $LIG^+$  representation of a formula and produces an embedding for its literals; **2) Aggregator:** A function that maps each literal representation to a vote and then aggregates them into a prediction about satisfiability. More details can be found in Appendix B.

## 3. Related Work

**Graph Contrastive Learning.** Existing graph contrastive frameworks can, in principle, be used to

learn representations for combinatorial problems. Common augmentations in graph contrastive frameworks include: 1) perturbing structures, such as, node dropping, subgraph sampling or graph diffusion, and 2) perturbing features, such as, masking or adding noise to the node features (Hassani & Khasahmadi, 2020). These augmentations have achieved success in multiple graph-level tasks (You et al., 2020; Hassani & Khasahmadi, 2020), as well as node-level tasks (Zhu et al., 2020; Wan et al., 2021; Tong et al., 2021).

**Supervised Learning for Combinatorial Optimization (CO).** Almost all modern approaches to machine learning for CO, use variations of a GNN architecture. On the supervised front, the work of Nowak et al. (2018) on *Quadratic Assignment Problem*, Joshi et al. (2019) and Prates et al. (2019) on *Travelling Salesman Problem* (TSP) have shown encouraging results. Gasse et al. (2019) trained a branching heuristic for MILPs by imitating an expert policy. Later, Nair et al. (2020) extended those ideas to make them scalable to substantially larger instances. In SAT, Selsam et al. (2018) proposed a way to train GNNs to solve SAT problems in an end-to-end fashion. The same architecture was used in Selsam & Bjørner (2019) to guide variable branching across conventional solvers.

**Reinforcement Learning for CO.** Supervised learning is bottlenecked by the need for labels. Consequently, many have explored the use of *Reinforcement Learning* (RL). Node selection policy of Dai et al. (2017) for TSP and Kool et al. (2018) for *Vehicle Routing Problem* are examples of that effort. Lederman et al. (2020) and Yolcu & Póczos (2019) used REINFORCE to train variable branching heuristic for *quantified Boolean formulas* and local search algorithm WalkSAT, respectively. Kurin et al. (2019) used DQN for SAT and Vaezipoor et al. (2021) improved a SOTA #SAT solver via *Evolution Strategy*. We refer to (Cappart et al., 2021; Bengio et al., 2021) for more comprehensive record of efforts in this area.

**Unsupervised Learning for CO.** Toenshoff et al. (2021) proposed an unsupervised approach to solve constrained optimization problems on graphs by minimizing a problem dependent loss function. Amizadeh et al. (2019a;b) solved SAT and CircuitSAT problems by minimizing an energy function. Lastly, inspired by probabilistic method, Karalias & Loukas (2020) trained a GNN in an unsupervised way to act as a distribution over possible solutions of a given problem, by minimizing a probabilistic penalty loss. Our method is distinct from other unsupervised techniques in that we do not minimize a problem-dependent loss function and rather learn problem representations through contrastive learning. To the best of our knowledge this is the first attempt at applying contrastive learning in the domain of combinatorial optimization.

## 4. Framework Overview

Similar to contrastive algorithms in other domains (e.g., images), we learn representations by contrasting augmented views of the same instance against negative samples. Our framework (Figure 2) consists of four major components:

**Augmentations.** Given a combinatorial instance  $\phi$ , a stochastic augmentation is applied to form a pair of positive samples, denoted by  $(\hat{\phi}_1, \hat{\phi}_2)$ . Our key insight is the tailored design of augmentations for combinatorial problems should preserve the label of the problem, e.g., satisfiability for SAT, as detailed in Section 5.

**Format Transformation ( $\mathcal{E}$ ).** Formulas  $(\hat{\phi}_1, \hat{\phi}_2)$  are transformed into (e.g.,  $\text{LIG}^+$ ) graphs  $(G_1, G_2)$ .

**Encoder.** A neural encoder is trained to extract graph-level representations  $\mathbf{z}_1, \mathbf{z}_2 \in \mathbb{R}^d$  from the augmented graphs  $G_1$  and  $G_2$ . This encoder can be any GNN such as GCN (Kipf & Welling, 2016) and the encoder proposed by NeuroSAT.

**Contrastive Loss.** In the end, SimCLR’s (Chen et al., 2020a) contrastive loss  $\mathcal{L}$  is applied to the graph representations  $\{\mathbf{z}_i\}_{i=1}^{2n}$ , obtained from a mini-batch of  $n$  instances. We follow Chen et al. (2020a) and use an MLP projection head to project each  $\mathbf{z}_i$  to  $\mathbf{m}_i$ . For a positive pair of projected representations  $(\mathbf{m}_i, \mathbf{m}_j)$ , the loss  $\mathcal{L}_{i,j}$  is a cross-entropy loss of differentiating the positive pair from the other  $2(n-1)$  negative samples (i.e., augmented samples of other instances):

$$\mathcal{L}_{i,j} = -\log \frac{\exp(\text{sim}(\mathbf{m}_i, \mathbf{m}_j) / \tau)}{\sum_{k=1}^{2N} \mathbf{1}_{k \neq i} \exp(\text{sim}(\mathbf{m}_i, \mathbf{m}_k) / \tau)}, \quad (1)$$

where  $\tau$  is the temperature parameter,  $\mathbf{1}$  is the indicator function, and  $\text{sim}$  measures the similarity between two representations:  $\text{sim}(\mathbf{m}_i, \mathbf{m}_j) := \mathbf{m}_i^T \mathbf{m}_j / \|\mathbf{m}_i\| \|\mathbf{m}_j\|$ . The final loss  $\mathcal{L}$  is the average of  $\mathcal{L}_{i,j}$  over all positive pairs. After training is completed, we only keep the encoder to extract the representation  $\mathbf{z}_i$  for downstream tasks.

## 5. Augmentations for Combinatorial Problems

### 5.1. Definition and Motivation

Intuitively, contrastive learning leads to representations that are invariant across augmentations. Thus, to guarantee downstream predictive performance, augmentations should (mostly) preserve the labels of downstream tasks (Arora et al., 2019; Tosh et al., 2021; HaoChen et al., 2021; Dubois et al., 2021). The augmentations found in computer vision, e.g., cropping and color jittering of images, typically preserve the labels of classification tasks. This is not the case for most previous graph contrastive frameworks (You et al., 2020; Hassani & Khasahmadi, 2020), where simple graph augmentations like node dropping and link perturbation are used without consideration of the

downstream task. For combinatorial problems, these *label-agnostic augmentations* (LAAs) are likely to produce many false positive pairs. For example, in the SR dataset in NeuroSAT, each SAT can be turned into UNSAT by flipping one literal in one clause.

Thus, we aim to design *label-preserving augmentations* (LPAs), which are transformations that preserve the instance label. Formally, given a problem family  $\Phi$  and a labelling function  $f$ , an augmentation distribution  $A$  is an LPA iff:

$$f(\hat{\phi}) = f(\phi), \quad \forall \hat{\phi} \in \text{supp}(A(\cdot|\phi)), \forall \phi \in \Phi,$$

where  $\text{supp}$  denotes the support of a distribution.

Importantly, LPAs are well-studied for many combinatorial problems and are much cheaper to obtain than labels. For SAT prediction, common preprocessing techniques from SAT solvers such as variable elimination can be used as LPAs, as discussed in Section 5.2. There are also LPAs for other combinatorial problems, such as, adding cuts (Achterberg et al., 2020) for MILPs and deleting dominant vertices (Akiba & Iwata, 2016) for Minimum Vertex Cover.

The type of LPAs is also crucial. Intuitively, LPAs that make more significant changes to an instance create harder positives from which better representations can be learned. Indeed, LPAs that lead to larger augmentation support  $\text{supp}(A(\cdot|\phi))$  will split  $\Phi$  into coarser equivalences classes, and thus (roughly speaking) a classifier can be learned with fewer labelled instances.

### 5.2. Label-preserving Augmentations for SAT

The LPAs for SAT preserve satisfiability of any Boolean formula. In other words, a (*un*)satisfiable instance remains (*un*)satisfiable after the applications of LPAs. We review some common LPAs for SAT below, with examples and time complexity results provided in Appendix A.

**Unit Propagation (UP).** A clause is a *unit clause* if it contains only one literal. If an instance  $\phi$  contains a unit clause  $\ell$ , we can 1) remove all clauses in  $\phi$  containing the literal  $\ell$  and 2) delete  $\neg\ell$  from all other clauses.

**Add Unit Literal (AU).** The inverse of UP: 1) construct a unit clause from a new literal  $\ell$ , 2) add its negation  $\neg\ell$  to some other clauses and 3) create new clauses containing  $\ell$ .

**Pure Literal Elimination (PL).** A variable  $v$  is called pure if it occurs with only one polarity in  $\phi$ . We can delete all clauses in  $\phi$  containing  $v$ .

**Subsumed Clause Elimination (SC).** If a clause  $c_1$  is a subset of  $c_2$ , i.e., all literals in  $c_1$  are also in  $c_2$ , then deleting  $c_2$  does not change satisfiability of  $\phi$ .

**Clause Resolution (CR).** Resolution produces a new clause

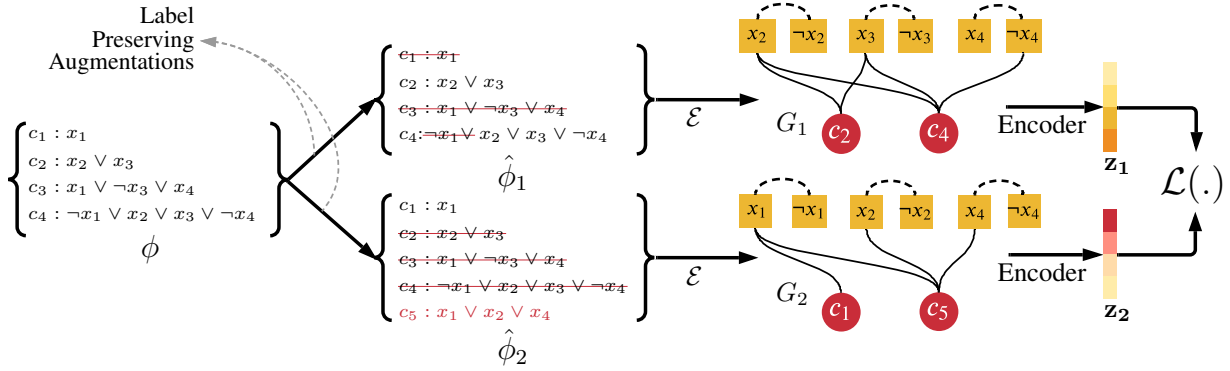


Figure 2. Our contrastive learning framework for combinatorial problems. Given an instance  $\phi$ , a pair of augmented samples ( $\hat{\phi}_1, \hat{\phi}_2$ ) are formed using label-preserving augmentations. ( $\hat{\phi}_1, \hat{\phi}_2$ ) are then transformed to graph formats ( $G_1, G_2$ ). An encoder is used to extract graph representations ( $z_1, z_2$ ). Lastly, a standard contrastive loss is applied over a mini-batch of instances.

implied by two clauses containing complementary literals:

$$\frac{\ell \vee a_1 \vee \dots \vee a_n, \neg \ell \vee b_1 \vee \dots \vee b_m}{a_1 \vee \dots \vee a_n \vee b_1 \vee \dots \vee b_m} \quad (2)$$

The new clause  $c$  is called the *resolvent* of  $c_1, c_2$ :  $c = c_1 \otimes c_2$ . Adding  $c$  to  $\phi$  does not change satisfiability.

**Variable Elimination (VE).** Let  $S_\ell$  be a set of clauses containing the literal  $\ell$ , and  $S_{-\ell}$  be a set of clauses containing its negation  $\neg \ell$ . Then a new set  $S$  is obtained by pairwise resolving on clauses of  $S_\ell$  and  $S_{-\ell}$ :  $S = \{c_1 \otimes c_2 | c_1 \in S_\ell, c_2 \in S_{-\ell}\}$ . Replacing  $S_\ell \cup S_{-\ell}$  with  $S$  does not change satisfiability (Eén & Biere, 2005).

Note that some augmentations, such as VE, have the worst-case exponential complexity if run until convergence. However, in our paper, we only eliminate a small and fixed number of variables. Therefore, all LPAs listed here are cheap and have polynomial-time complexity.

## 6. Empirical Study of Augmentations for SAT

Our first experiments study the role of augmentations in contrastive learning for SAT prediction. We followed the standard linear evaluation protocol to evaluate representations (Chen et al., 2020a), i.e., we report test accuracy of a linear classifier trained on top of frozen representations.

**Architecture.** We primarily used the encoder of NeuroSAT (Selsam et al., 2018) as the GNN architecture. We also re-ran a small number of experiments with another type of GNN in Appendix E. The dimension of literal representations was chosen to be 128. We discarded the aggregator of NeuroSAT and obtained the graph-level representations by average-pooling over all literal representations. All design details of the encoder followed the original NeuroSAT paper unless otherwise specified.

**Experimental Setting.** We used the contrastive loss in Equation 1 with the temperature 0.5. For the projection head, we used a 2-layer MLP, with the dimension of hidden and output layer being 64. Appendix F also shows some experiments with other contrastive objectives. We used Adam optimizer with learning rate  $2 \times 10^{-4}$  and weight decay  $10^{-5}$ . The batch size was 128 and the maximum training epoch was 5000. The generator produced a set of new unlabelled instances for each batch. We used sklearn’s (Pedregosa et al., 2011) logistic regression model for linear evaluation. We generated 100 separate labelled instances to train our linear evaluators, and another 500 as the validation set to pick the hyperparameters (ranging from  $10^{-3}$  to  $10^3$ ) of  $L_2$  regularization. The test set consisted of  $10^4$  instances.

**Datasets.** We experimented using four generators: SR (Selsam et al., 2018), Power Random 3SAT (PR) (Ansótegui et al., 2009), Double Power (DP) and Popularity Similarity (PS) (Giráldez-Cru & Levy, 2017). SR and PR are the synthetic generators. DP and PS are pseudo-industrial generators producing instances that mimic real-world problems. We generated instances of 10 variables for SR and PR, and 20 for DP and PS. The number inside the parenthesis denotes the number of variables per instance. We also tweaked the parameters of the generators so that they produce roughly balanced SAT and UNSAT. More details are in Appendix D.

**Augmentations.** We used four of the LPAs from Section 5.2, namely: AU, SC, CR and VE. The augmentations UP and PL were not studied because most of our instances originally do not contain unit clauses or pure literals.

For our baseline we adopted four LAAs from GraphCL (You et al., 2020) with some adjustments. The issue with the original augmentations was that they operate directly on the graph. However, NeuroSAT requires the input to be in

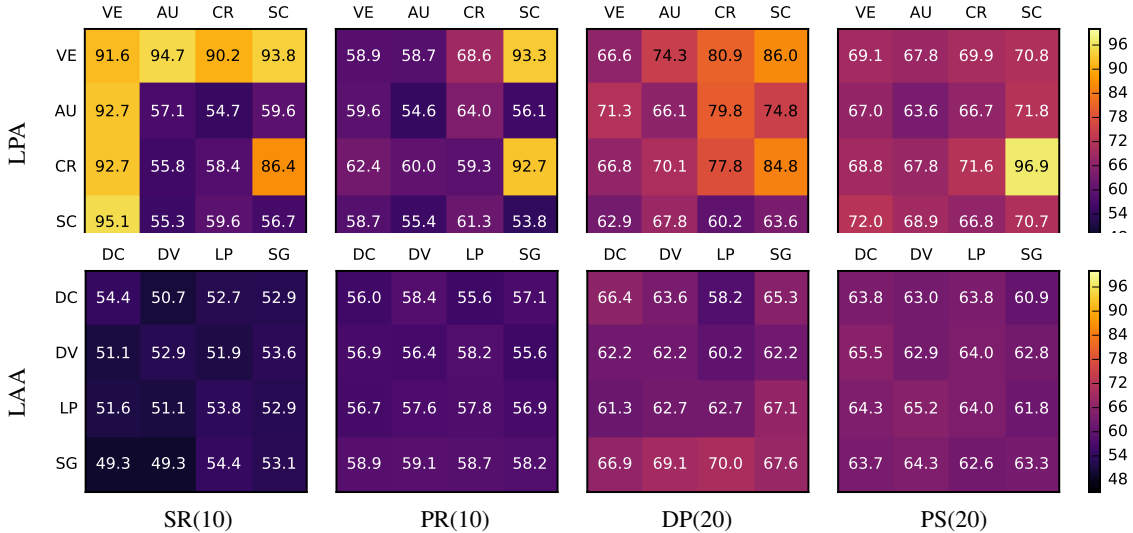


Figure 3. Contrastive learning with LPAs learns much better representations for SAT predictions than with LAAs. We evaluated the linear classification performance of SSL models trained with different single augmentations (diagonal) or paired combinations (off-diagonal). The four heatmaps on the first row show results using LPAs, and the ones on the second row using LAAs. Each column denotes different datasets. For off-diagonal entries on each heatmap, the row corresponds to the first augmentation applied.

Table 1. Adding SAT-preserving clauses (Resolution) leads to much higher accuracy than adding random ones (Random). The number of clauses added is controlled to be the same for both.

Type \ Dataset	SR	PR	DP	PS
Random	55.6	52.8	64.7	68.9
Resolution (ours)	<b>86.4</b>	<b>92.6</b>	<b>84.8</b>	<b>96.9</b>

the  $LIG^+$  format, and blindly applying these augmentations might break that structure. Thus, we adapted GraphCL augmentations to maintain the  $LIG^+$  structure, resulting in the following augmentations: *drop clauses* (DC), *drop variables* (DV), *link perturbation* (LP) and *subgraph* (SG). DC and DV correspond to *node dropping*. LP corresponds to *edge perturbation*, where we randomly add or remove links between literals and clauses on a SAT instance. Lastly, SG is similar to the one in GraphCL where we do a random walk on an instance and keep the resulting subgraph.

All 8 augmentations except SC are parameterized by  $p$  which controls the intensity of perturbations. For SC, we eliminate all subsumed clauses. In this section, we chose the  $p$  that achieved the best linear evaluation performance when the corresponding augmentation was applied alone. The results for tuning  $p$  are shown in Appendix C.

### 6.1. Label-Preserving Augmentations are Necessary

**Do LPAs learn better representations than LAAs?** To investigate the effect of using different augmentations,

we evaluated the linear classification performance of SSL models trained with different single augmentations or paired combinations. As Figure 3 shows, the accuracy (%) for the best LPA pair is: 95.1 for SR, 93.3 for PR, 86.0 for DP and 96.9 for PS, while the corresponding number (%) for LAAs is much lower: 54.4, 59.1, 70.0 and 65.5. These results show that SSL models trained with LPAs learn significantly better representations than those with LAAs.

**Do our gains simply come from adding clauses?** The four LAAs from GraphCL do not add new clauses to the original instance. To investigate whether this explains the performance gap, we compared CR against adding the same number of randomly generated clauses (*adding random clauses*). We eliminated subsumed clauses (SC) after both augmentations. As shown in Table 1, adding SAT-preserving resolvents achieved at least 20% higher accuracy than adding randomly generated clauses.

**Does combining LPAs and LAAs hurt performance?** In Table 2, we trained SSL models with VE followed by 7 different augmentations on SR(10). We compared the accuracy change with the model trained with VE only. From Table 2, adding LAAs resulted in 30% drop in accuracy, while LPAs decreased the accuracy by at most 3%.

In summary, the performance gap between LPAs and LAAs meets our intuition. LAAs do not guarantee preserving satisfiability, which could result in false positive pairs that hurt the SAT prediction performance.

Table 2. Adding LAAs to LPAs significantly hurts the performance. Cells represent the difference of linear evaluation accuracy between training with VE followed by another aug and with VE alone on SR(10). Yellow indicates improved accuracy.

LPAs			LAAs			
AU	CR	SC	DC	DV	LP	SG
-1.5	0.7	1.2	-34.4	-32.4	-33.7	-35.9

## 6.2. Type, Order, and Strengths of LPAs are Crucial

Previous studies of contrastive learning for image (Chen et al., 2020a) and graph (You et al., 2020) have shown that the quality of learned representations relies heavily on finding the right type and composition of augmentations. We observe a similar pattern in SAT. In Figure 3, the accuracy gap between the best LPA combination and the worst is between 20 – 40% for all datasets.

The conjecture in SimCLR (Chen et al., 2020a) is that stronger augmentations lead to better representations. Intuitively, weaker augmentations often create very correlated positive examples, providing shortcuts for neural networks to cheat in the contrastive task without learning meaningful representations. Based on this conjecture, we study what type, order, and strengths of LPAs induce harder positive pairs and thus better representation quality.

### Resolution-based augmentations are the most powerful.

In Figure 3, the best pair for each dataset includes either CR or VE. Both of them are based on the resolution rule in Equation 2. Resolution is a powerful inference rule in propositional logic. In fact, we can build a sound and complete propositional theorem prover with only resolutions (Genesereth & Kao, 2013). The Davis–Putnam algorithm (Davis & Putnam, 1960), the basis of practical SAT solvers, iteratively applies resolution until reaching satisfiability certificates. Resolution-based augmentations in contrastive learning may help the neural work learn the essence of resolution, which leads to better satisfiability prediction.

### Composing different augmentations is beneficial across datasets.

The highest accuracy in Figure 3 always comes from off-diagonal entries, i.e., composition of different augmentations. In PR, every singular augmentation failed to obtain accuracy higher than 60% by itself. While singular augmentations achieved decent accuracy on SR and DP, combinations further improved the accuracy. Similar to image and graph, composing different augmentations resulted in harder positives and better representations.

### Eliminating subsumed clauses after adding resolvents is particularly helpful.

CR followed by SC consistently produced high-quality representations across all datasets. Using CR alone tended to perform substantially worse. The

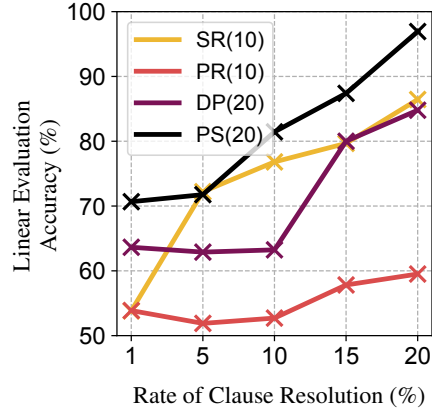


Figure 4. Adding more resolvents improves the performance across all datasets. Each point represents linear evaluation accuracy of an SSL model trained with CR of the specified rate (i.e., # of added resolvents divided by # of original clauses) followed by SC.

accuracy drop without SC is: 28.1% for SR, 33.4% for PR, 7.0% for DP, 25.3% for PS. Without SC, all original clauses are preserved by the augmentation, leading to trivial positive pairs. In other words, without SC the GNN may have solved the contrastive task by finding a common subgraph, not by learning anything meaningful about resolution rules.

Interestingly, swapping the order of this pair also decreased accuracy by 20 – 30% for all datasets. We conjecture that this may be an artefact of our specific generators. The percent of subsumed clauses in the original instances (over 1000 samples) was 43, 0, 76, 69 for SR, PR, DP and PS. Applying SC first had no effect on PR due to no subsumed clauses present. For the other three, SC may have eliminated too many clauses, which hurt the diversity of resolvents created in the next phase.

### More resolvents lead to better representations.

Figure 4 studies the effect of the number of resolvents added in CR. In general, adding more resolvents improved the linear evaluation performance across all datasets. More resolvents not only add more new clauses, but also may help SC delete more original clauses, which create harder positives.

## 6.3. Do Our Augmentations Reveal the True Label?

As mentioned before, CR and VE are powerful enough to build a sound and complete solver. Because our instances are relatively small, it is important to ask if our augmentations “accidentally” reveal the true label(SAT/UNSAT) to the model. We do not believe this is the case for the following reasons.

Our augmentations (polytime complexity) cannot in general determine whether a formula is satisfiable (exponential in the worst case). When applied to our datasets, VE only

eliminated between 1-4 variables, and CR added between 10-20% of the total number of original clauses. This amount of work alone is not enough to solve SAT.

To assess whether our augmentations are effectively solving our *specific* instances, we measured the number of decision steps of CryptoMiniSat solvers (Soos et al., 2009) on different datasets before and after our LPAs. If our LPAs were close to solving the instance, the decision steps should have decreased dramatically after LPAs were applied. However, Table 6 in Appendix I shows the decision steps remain relatively stable after augmentations.

## 7. Comparison with Other Methods

The second set of experiments compared our proposed framework with other baselines in the setups of linear evaluation, fine-tuning and few-shot transfer learning. In addition to our model (SSL with LPA), we studied 4 baselines: SSL with LAA, supervised models trained without augmentations, supervised models trained with LPA or LAA. The details of training SSL models followed Section 6. For supervised models, we trained the encoder and aggregator of NeuroSAT end-to-end. The hyperparameters for supervised models were the same as SSL in Section 6, except the learning rate was chosen to be  $2 \times 10^{-5}$ , following Selsam et al. (2018). For each dataset, we chose the best augmentation combination for LPA and LAA according to Figure 3. The degree parameter associated with each augmentation was tuned separately for SSL and supervised models. We used 200 instances as validation sets for early stopping of all methods.

Unless otherwise specified, all datasets used in this section have 40 variables per instance; in the Appendix H we report results for the same experiments on smaller instances. For SR datasets, following NeuroSAT, we trained on SR(U(10, 40)) and tested on SR(40). Training procedures for all models were the same as Section 7.1, except the batch size was 80 for SR(40) to avoid memory issues. When fine-tuning NeuroSAT, we used different learning rates for the encoder and aggregator, which were separately tuned for different models on each dataset.

### 7.1. Linear Evaluation

We first evaluated the linear evaluation accuracy of all methods following the procedure in Section 6. We varied the number of labelled instances from 2 to  $10^4$ .

As shown in the first row of Figure 5, our method achieved substantially higher accuracy than others across all datasets in the low-label regime. For example, with 10 training labelled instances, the improvement (%) of ours compared from the second best method was: 9.42 for SR, 14.07 for PR, 30.81 for DP, 22.18 for PS. Our model’s accuracy was

also on par with fully-supervised models that had access to significantly more labels: for all datasets, the accuracy of our models trained with 100 labels matched or exceeded the accuracy of our best supervised baselines trained with  $10^4$  labels, a  $100\times$  reduction in the number of labels needed.

On the other hand, SSL models trained with LAAs were not much better than random-initialized ones under linear evaluation. We also found that using LAAs for supervised models even hurt performance, possibly because LAAs add label noise. For example, with 10000 labels on DP(40), adding LAAs gave 20.21% lower accuracy than supervised without augmentations.

### 7.2. Fine-tuning

We evaluated the fine-tuning accuracy of different SSL models compared to the supervised baselines. Specifically, we took the pre-trained SSL models and optimized them end-to-end on labelled data for a few epochs.

As shown in the second row of Figure 5, the fine-tuning results resemble those of linear evaluation. In the low-label regime, our model (SSL + LPA) dominated across all datasets. In contrast, SSL with LAAs did not improve much from supervised models that were randomly initialized. As with linear evaluation, we observed a reduction in sample complexity of at least  $100\times$ .

### 7.3. Few-shot Transfer Learning

We investigated how well the SSL models trained in Section 7.1 perform in 10-shot transfer to unseen and larger datasets, Uniform Random 3SAT (UR) and Community Attachment (CA) (Giráldez-Cru & Levy, 2015). UR is a synthetic generator, and CA is pseudo-industrial. We trained a logistic regression classifier on the fixed representations with 10 SAT and 10 UNSAT instances from the target dataset. We compared transferability of representations from SSL with those from supervised representations trained on the same source dataset. In particular, the supervised baseline was trained using 100,000 labelled source instances with the same augmentations as SSL. Only the encoder of the supervised models was used for extracting representations on target datasets.

#### Do our representations transfer better than supervised?

As shown in Figure 6, we found that our SSL models generally transferred better than the supervised baseline. This improvement also tended to be larger when the source and target domain were more distinct, such as from pseudo-industrial instances (DP, PS) to random 3 SAT (UR), and from random 3SAT (PR) to pseudo-industrial instances (CA). This meets our intuition, because SSL does not leverage labels in the source dataset, and thus reduces overfitting on source labels (Yang et al., 2020).

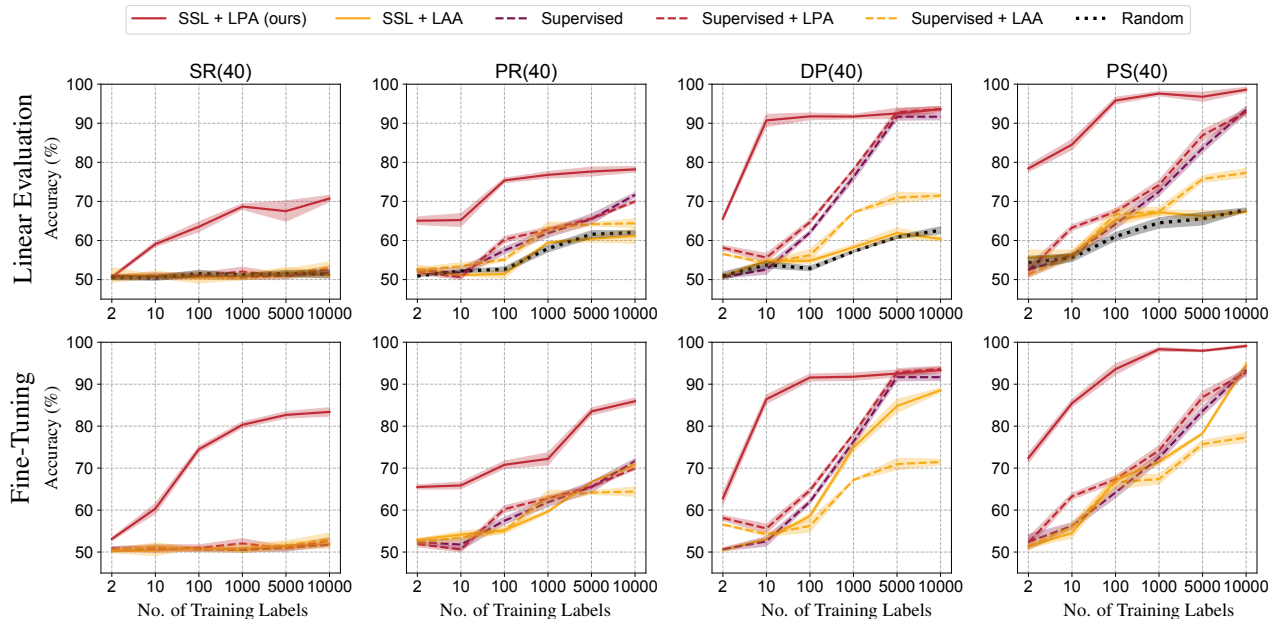


Figure 5. Our method (SSL + LPA) achieves significantly higher accuracy after linear evaluation and fine-tuning than all baselines in low-label regime and is comparable to supervised models that have been given more labels. We vary the number of training labelled instances from 2 to  $10^4$  and report the average accuracy and standard error over 3 trials for all methods.

**Does transferability vary with the source dataset?** We also found that transferability of representations depended heavily on the source datasets. Generally speaking, models trained on pseudo-industrial generators, DP and PS, transferred worse than the synthetic generators, SR and PR. We conjecture that small industrial instances are not challenging enough for models to learn generalizable representation of unseen problem domains. In Appendix G, we also performed an experiment to show how the model trained on SR, PR, DP and PS transfer to each other, which supports similar conclusions: models trained on SR and PR transferred better to DP and PS than vice versa.

## 8. Conclusions and Outlook

We studied the effect of data augmentations on contrastive learning for the Boolean satisfiability problem. We designed label-preserving augmentations using well-studied transformations, e.g., clause resolution or variable elimination, and confirmed the hypothesis that data augmentations should be label-preserving to help in downstream prediction. The design of our augmentations was critical; we found that resolution-based augmentations, which produced more distinct augmentations, were necessary for strong results. Our contrastive method was able to learn strong SAT predictors (at least as strong as our best supervised baselines) with  $100\times$  fewer labelled training instances.

Although our results are restricted to Boolean satisfiability, they hold lessons for solver design more broadly. For example, the convex hull of MILP feasible sets is invariant to cuts, which suggests that contrastive pre-training could be used to improve Neural Diving (Nair et al., 2020). In general, our results strongly suggest that studying and exploiting invariances can dramatically improve the sample complexity of heuristics learned via imitation learning.

The study of combinatorial problems is fruitful for the broader machine learning community, because these problems are non-trivial and so much is known about their invariances. In particular, experiments can be designed that exactly satisfy the assumptions of invariant learning theory, making combinatorial problems fantastic test beds for the emerging field of contrastive and self-supervised learning.

## Acknowledgments

We thank Roger Grosse and Guiliang Liu for valuable discussions and insights. Resources used in preparing this research were provided, in part, by the Province of Ontario, the Government of Canada through CIFAR, and companies sponsoring the Vector Institute. We acknowledge the support of the Natural Sciences and Engineering Research Council of Canada (NSERC), RGPIN-2021-03445.



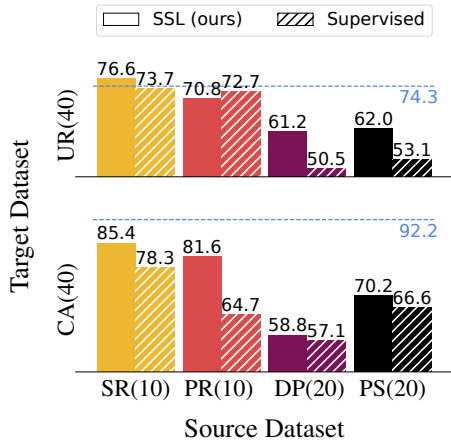


Figure 6. The representations of our model (solid bars) transfer better from source to unseen target dataset than those of supervised (hatched bars) trained on the same source. 10-shot transfer accuracy (%) was evaluated for each setup. The blue line is the accuracy of fully-supervised models trained on each target dataset with  $10k$  labels.

## References

- Achterberg, T., Bixby, R. E., Gu, Z., Rothberg, E., and Wenginger, D. Presolve reductions in mixed integer programming. *INFORMS Journal on Computing*, 32(2):473–506, 2020.
- Akiba, T. and Iwata, Y. Branch-and-reduce exponential/fpt algorithms in practice: A case study of vertex cover. *Theoretical Computer Science*, 609:211–225, 2016.
- Amizadeh, S., Matuskevych, S., and Weimer, M. Learning to solve circuit-sat: An unsupervised differentiable approach. In *ICLR*, 2019a.
- Amizadeh, S., Matuskevych, S., and Weimer, M. Pdp: A general neural framework for learning constraint satisfaction solvers. *arXiv preprint arXiv:1903.01969*, 2019b.
- Ansótegui, C., Bonet, M. L., and Levy, J. Towards industrial-like random sat instances. In *Twenty-First International Joint Conference on Artificial Intelligence*, 2009.
- Arora, S., Khandeparkar, H., Khodak, M., Plevrakis, O., and Saunshi, N. A theoretical analysis of contrastive unsupervised representation learning. *arXiv preprint arXiv:1902.09229*, 2019.
- Bardes, A., Ponce, J., and LeCun, Y. Vicreg: Variance-invariance-covariance regularization for self-supervised learning. *arXiv preprint arXiv:2105.04906*, 2021.
- Bengio, Y., Lodi, A., and Prouvost, A. Machine learning for combinatorial optimization: a methodological tour d’horizon. *European Journal of Operational Research*, 290(2):405–421, 2021.
- Cappart, Q., Chételat, D., Khalil, E., Lodi, A., Morris, C., and Veličković, P. Combinatorial optimization and reasoning with graph neural networks. *arXiv preprint arXiv:2102.09544*, 2021.
- Chen, T., Kornblith, S., Norouzi, M., and Hinton, G. A simple framework for contrastive learning of visual representations. In *International conference on machine learning*, pp. 1597–1607. PMLR, 2020a.
- Chen, X., Fan, H., Girshick, R., and He, K. Improved baselines with momentum contrastive learning. *arXiv preprint arXiv:2003.04297*, 2020b.
- Dai, H., Khalil, E. B., Zhang, Y., Dilkina, B., and Song, L. Learning combinatorial optimization algorithms over graphs. *arXiv preprint arXiv:1704.01665*, 2017.
- Davis, M. and Putnam, H. A computing procedure for quantification theory. *Journal of the ACM (JACM)*, 7(3):201–215, 1960.
- Dubois, Y., Bloem-Reddy, B., Ullrich, K., and Maddison, C. J. Lossy compression for lossless prediction. In *NeurIPS*, 2021.
- Eén, N. and Biere, A. Effective preprocessing in sat through variable and clause elimination. In *International conference on theory and applications of satisfiability testing*, pp. 61–75. Springer, 2005.
- Gasse, M., Chételat, D., Ferroni, N., Charlin, L., and Lodi, A. Exact combinatorial optimization with graph convolutional neural networks. *arXiv preprint arXiv:1906.01629*, 2019.
- Genesereth, M. and Kao, E. Introduction to logic. *Synthesis Lectures on Computer Science*, 4(1):1–165, 2013.
- Giráldez-Cru, J. and Levy, J. A modularity-based random sat instances generator. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*, 2015.
- Giráldez-Cru, J. and Levy, J. Locality in random sat instances. *International Joint Conferences on Artificial Intelligence*, 2017.
- HaoChen, J. Z., Wei, C., Gaidon, A., and Ma, T. Provable guarantees for self-supervised deep learning with spectral contrastive loss. *arXiv preprint arXiv:2106.04156*, 2021.
- Hassani, K. and Khasahmadi, A. H. Contrastive multi-view representation learning on graphs. In *International Conference on Machine Learning*, pp. 4116–4126. PMLR, 2020.

- He, K., Fan, H., Wu, Y., Xie, S., and Girshick, R. Momentum contrast for unsupervised visual representation learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 9729–9738, 2020.
- Joshi, C. K., Laurent, T., and Bresson, X. An efficient graph convolutional network technique for the travelling salesman problem. *arXiv preprint arXiv:1906.01227*, 2019.
- Karalias, N. and Loukas, A. Erdos goes neural: an unsupervised learning framework for combinatorial optimization on graphs. In *NeurIPS 2020 34th Conference on Neural Information Processing Systems*, 2020.
- Kipf, T. N. and Welling, M. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.
- Kool, W., Van Hoof, H., and Welling, M. Attention, learn to solve routing problems! *arXiv preprint arXiv:1803.08475*, 2018.
- Kurin, V., Godil, S., Whiteson, S., and Catanzaro, B. Improving SAT Solver Heuristics with Graph Networks and Reinforcement Learning. *CoRR*, abs/1909.11830, 2019.
- Lederman, G., Rabe, M. N., Seshia, S., and Lee, E. A. Learning Heuristics for Quantified Boolean Formulas through Reinforcement Learning. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020.
- Nair, V., Bartunov, S., Gimeno, F., von Glehn, I., Lichocki, P., Lobov, I., O’Donoghue, B., Sonnerat, N., Tjandraatmadja, C., Wang, P., et al. Solving mixed integer programs using neural networks. *arXiv preprint arXiv:2012.13349*, 2020.
- Nejati, S. and Ganesh, V. Cdcl (crypto) sat solvers for cryptanalysis. In *Proceedings of the 29th Annual International Conference on Computer Science and Software Engineering*, pp. 311–316, 2019.
- Nowak, A., Villar, S., Bandeira, A. S., and Bruna, J. Revised note on learning quadratic assignment with graph neural networks. In *2018 IEEE Data Science Workshop (DSW)*, pp. 1–5. IEEE, 2018.
- Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.
- Prates, M., Avelar, P. H., Lemos, H., Lamb, L. C., and Vardi, M. Y. Learning to solve np-complete problems: A graph neural network for decision tsp. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pp. 4731–4738, 2019.
- Selsam, D. and Bjørner, N. Guiding high-performance sat solvers with unsat-core predictions. In *International Conference on Theory and Applications of Satisfiability Testing*, pp. 336–353. Springer, 2019.
- Selsam, D., Lamm, M., Benedikt, B., Liang, P., de Moura, L., Dill, D. L., et al. Learning a sat solver from single-bit supervision. In *International Conference on Learning Representations*, 2018.
- Soos, M., Nohl, K., and Castelluccia, C. Extending sat solvers to cryptographic problems. In *International Conference on Theory and Applications of Satisfiability Testing*, pp. 244–257. Springer, 2009.
- Toenshoff, J., Ritzert, M., Wolf, H., and Grohe, M. Graph neural networks for maximum constraint satisfaction. *Frontiers in artificial intelligence*, 3:98, 2021.
- Tong, Z., Liang, Y., Ding, H., Dai, Y., Li, X., and Wang, C. Directed graph contrastive learning. *Advances in Neural Information Processing Systems*, 34, 2021.
- Tosh, C., Krishnamurthy, A., and Hsu, D. Contrastive learning, multi-view redundancy, and linear models. In *Algorithmic Learning Theory*, pp. 1179–1206. PMLR, 2021.
- Vaezipoor, P., Lederman, G., Wu, Y., Maddison, C., Grosse, R. B., Seshia, S. A., and Bacchus, F. Learning branching heuristics for propositional model counting. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(14): 12427–12435, May 2021.
- Wan, S., Zhan, Y., Liu, L., Yu, B., Pan, S., and Gong, C. Contrastive graph poisson networks: Semi-supervised learning with extremely limited labels. *Advances in Neural Information Processing Systems*, 34, 2021.
- Yang, X., He, X., Liang, Y., Yang, Y., Zhang, S., and Xie, P. Transfer learning or self-supervised learning? a tale of two pretraining paradigms. *arXiv preprint arXiv:2007.04234*, 2020.
- Yolcu, E. and Póczos, B. Learning Local Search Heuristics for Boolean Satisfiability. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, 8-14 December 2019, Vancouver, BC, Canada*, pp. 7990–8001, 2019.

You, Y., Chen, T., Sui, Y., Chen, T., Wang, Z., and Shen, Y. Graph contrastive learning with augmentations. *Advances in Neural Information Processing Systems*, 33:5812–5823, 2020.

Zhu, Y., Xu, Y., Yu, F., Liu, Q., Wu, S., and Wang, L. Deep graph contrastive representation learning. *arXiv preprint arXiv:2006.04131*, 2020.

## A. LPAs for SAT

Table 3. Examples of LPAs for SAT. UP: remove  $c_1, c_3$  and  $\neg x$  from  $c_4$ . AU: add unit literal  $\neg x_5$ , add  $x_5$  to  $c_1$  and create a new random clauses containing  $\neg x_5$ . SC: remove  $c_4$  because  $c_2 \subset c_4$ . CR: add  $c_2 \otimes c_3$ . VE: eliminate  $x_3$  by adding  $c_2 \otimes c_3$  and  $c_4 \otimes c_3$ .

Original	UP
$c_1 : x_1$	<del><math>x_1</math></del>
$c_2 : x_2 \vee x_3$	$x_2 \vee x_3$
$c_3 : x_1 \vee \neg x_3 \vee x_4$	<del><math>x_1 \vee \neg x_3 \vee x_4</math></del>
$c_4 : \neg x_1 \vee x_2 \vee x_3 \vee \neg x_4$	<del><math>\neg x_1 \vee x_2 \vee x_3 \vee \neg x_4</math></del>
AU	SC
<del><math>\neg x_5</math></del>	$x_1$
$x_5 \vee x_1$	$x_2 \vee x_3$
$x_2 \vee x_3$	$x_1 \vee \neg x_3 \vee x_4$
$x_1 \vee \neg x_3 \vee x_4$	<del><math>\neg x_1 \vee x_2 \vee x_3 \vee \neg x_4</math></del>
$\neg x_1 \vee x_2 \vee x_3 \vee \neg x_4$	<del><math>\neg x_1 \vee x_2 \vee x_3 \vee \neg x_4</math></del>
<del><math>\neg x_5 \vee x_1 \vee \neg x_2 \vee x_3</math></del>	
CR	VE
$x_1$	$x_1$
$x_2 \vee x_3$	<del><math>x_2 \vee x_3</math></del>
$x_1 \vee \neg x_3 \vee x_4$	<del><math>x_1 \vee \neg x_3 \vee x_4</math></del>
$\neg x_1 \vee x_2 \vee x_3 \vee \neg x_4$	<del><math>\neg x_1 \vee x_2 \vee x_3 \vee \neg x_4</math></del>
$x_1 \vee x_2 \vee x_4$	$x_1 \vee x_2 \vee x_4$

## B. NeuroSAT

**Encoder** We use  $L^n$  and  $C^n$  to denote the embeddings for literals and clauses at the message passing round  $n$ . In addition, we have hidden states for literals and clauses, denoted by  $L_h^n, C_h^n$ . Let  $M$  be the bipartite adjacency matrix of the  $LIG^+$ . NeuroSAT encoder is parameterized by two MLPs ( $L_{msg}, C_{msg}$ ) and two layer-norm LSTMs ( $L_u, C_u$ ), which are shared for all rounds. Then for round  $n$ , embeddings are updated as:

$$\begin{aligned} (C^{n+1}, C_h^{n+1}) &\leftarrow C_u([C_h^n, M^T L_{msg}(L^n)]) \\ (L^{n+1}, L_h^{n+1}) &\leftarrow L_u([L_h^n, FLIP(L^n), MC_{msg}(C^{n+1})]), \end{aligned}$$

where  $FLIP$  swaps the literal and its negation in the embedding.

In this work, we always set the number of message passing rounds to 26. We use  $L^{26}$  as the final literal embedding. The graph-level representation can be computed as an average of the embeddings for all literals.

**Aggregator** Given the embeddings  $L^N$  from the encoder, we project each literal representation to a single scalar using an MLP  $L_{vote}$ :  $V \leftarrow L_{vote}(L^N)$ . Then we minimize the cross-entropy loss between the true label and  $\text{sigmoid}(\text{mean}(L_{vote}))$

## C. Augmentation Rates for LAAs and LPAs

We also investigated the effect of augmentation rates for LAAs and LPAs, with results presented in Figure 7 and 8.

## D. Datasets

### D.1. Brief Description

- **SR (Selsam et al., 2018)** A random SAT generator proposed as a challenge for neural networks to learn intrinsic properties about satisfiability without cheating on some miscellaneous statistics about the dataset.
- **PR (Ansótegui et al., 2009)** A random k-SAT generator where the frequency of each variable is sampled from a power-law distribution.

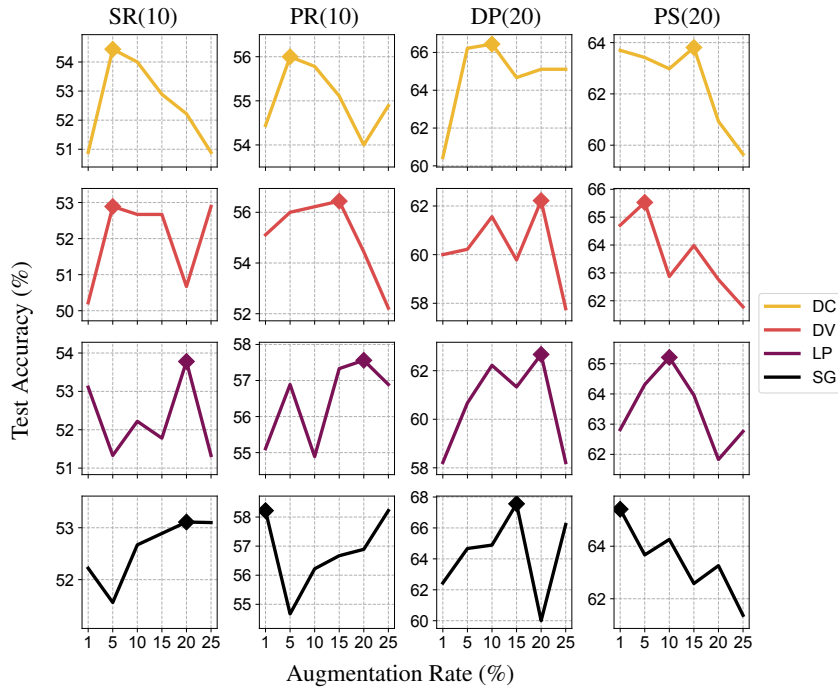


Figure 7. The effect of rate of LAAs on the accuracy across datasets. The diamond markers indicate the maximum achieved accuracy which was the rate that was subsequently chosen to produce the heatmap of Figure 3.

- **DP (Ansótegui et al., 2009)** A pseudo-industrial generator based on PR with varying clause length
- **PS (Giráldez-Cru & Levy, 2017)** A pseudo-industrial generator based on the notion of locality.
- **UR** A random k-SAT generator where each variable is sampled uniformly.
- **CA (Giráldez-Cru & Levy, 2015)** A pseudo-industrial generator based on the notion of modularity.

## D.2. Parameters

For the purpose of reproducibility, we show the parameters of each generator used in the paper below:

- **SR(10) / SR(40)** All parameters follow NeuroSAT.
- **PR(10)** Number of variables: 10. Number of clauses: 41. Variable per clause: 3. Power-law exponents of variables: 1.7.
- **PR(40)** Number of variables: 40. Number of clauses: 147. Variable per clause: 3. Power-law exponents of variables: 2.5.
- **DP(20)** Number of variables: 20. Number of clauses: 34. Average variables per clause: 4. Power-law exponents of variables: 1.7.
- **DP(40)** Number of variables: 40. Number of clauses: 75. Average variables per clause: 5. Power-law exponents of variables: 1.7.
- **PS(20)** Number of variables: 20. Number of clauses: 58. Min variable per clause: 2. Average variables per clause: 4.
- **PS(40)** Number of variables: 40. Number of clauses: 73. Min variable per clause: 2. Average variables per clause: 5.

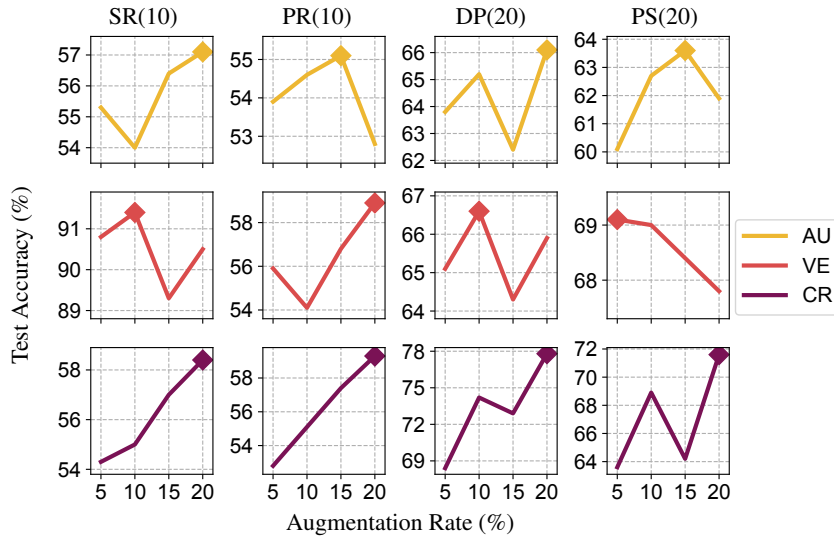


Figure 8. The effect of rate of our LPA augmentations on the accuracy across datasets. The diamond markers indicate the maximum achieved accuracy which was the rate that was subsequently chosen to produce the heatmap of Figure 3.

### E. Does our method work with other GNN architectures?

We also studied if our framework could be used with more common architectures, such as, graph convolutional networks (GCNs). We chose the GCN architecture in NeuralDiver (Nair et al., 2020) without the residual connection. The number of layers is set to be 10. Table 4 shows that SSL + LPA still dominates in the low-label regime.

Table 4. Linear evaluation performance of different methods with GCNs on SR(10).

	2	10	100	1000	5000	10000
Supervised	49.82	50.08	50.68	50.23	51.25	78.82
SSL + LPA	60.35	70.26	73.17	75.29	75.27	77.54
SSL + LAA	51.32	51.29	48.93	50.72	50.41	63.24

### F. Does our method work with other contrastive loss functions?

We replaced the SimCLR’s loss function in Equation 1 with the VICReg loss (Bardes et al., 2021). We set  $\lambda = 15$ ,  $\mu = 1$ ,  $\nu = 1$  for the hyperparameters of VICReg. As shown in Table 5, linear evaluation performance for both objective functions are quite close.

Table 5. Linear evaluation performance of our methods with SimCLR and VICReg loss on SR(10)

	2	10	100	1000	5000	10000
Ours + SimCLR	79.53	88.32	92.23	93.32	93.01	95.12
Ours + VICReg	76.38	89.15	91.27	93.98	94.01	94.77

### G. More Transfer Learning Results

See Figure 9.

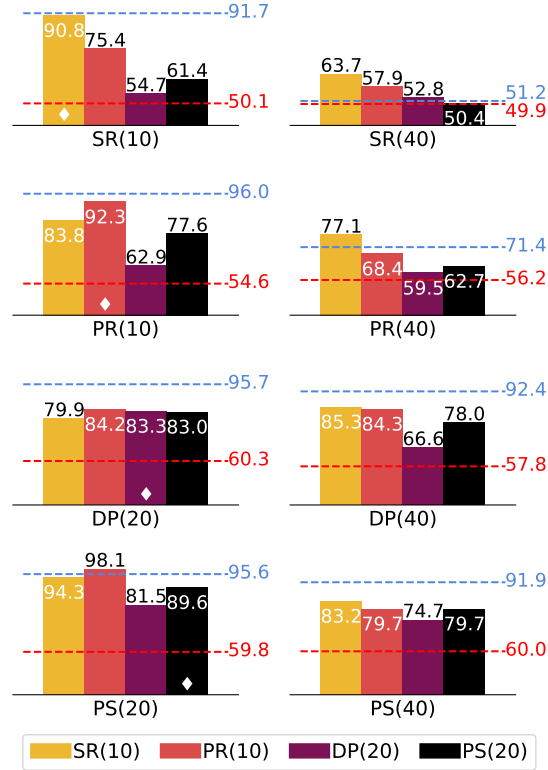


Figure 9. Each bar represents the SSL model trained on a different source dataset, and each subplot is the target dataset 4 SSL models are evaluated on. The number on top of each bar is the linear evaluation accuracy with 20 labels from the target dataset. The blue line is the fully-supervised baseline trained on 20 labels on the target baseline, and the red line is the supervised baseline trained on 10000 labels. The ◆ symbol emphasizes that the train and test datasets are from the same distribution. *Left*: Transfer to unseen problems of similar size. *right*: Transfer to unseen problems of larger size.

## H. Evaluation on Smaller Datasets

We also performed linear evaluation and fine-tuning experiments for the smaller datasets in Section 6. The results are shown in Figure 10.

## I. Decision step

We measured the decision steps of CryptoMiniSat (Soos et al., 2009) solvers before and after our augmentations. The result is shown in Table 6.

Table 6. Decision steps of CryptoMiniSat (Soos et al., 2009) solvers before and after augmentations used in Figure 5. We use the same augmentation as the corresponding SSL model in Figure 5. The statistics is computed over 1000 instances for each dataset.

	SR(40)		PR(40)		DP(40)		PS(40)	
	SAT	UNSAT	SAT	UNSAT	SAT	UNSAT	SAT	UNSAT
Before	17.54	13.25	18.21	12.14	32.78	0.27	33.26	7.37
After	16.36	16.40	21.61	12.64	28.47	0.36	34.83	6.92

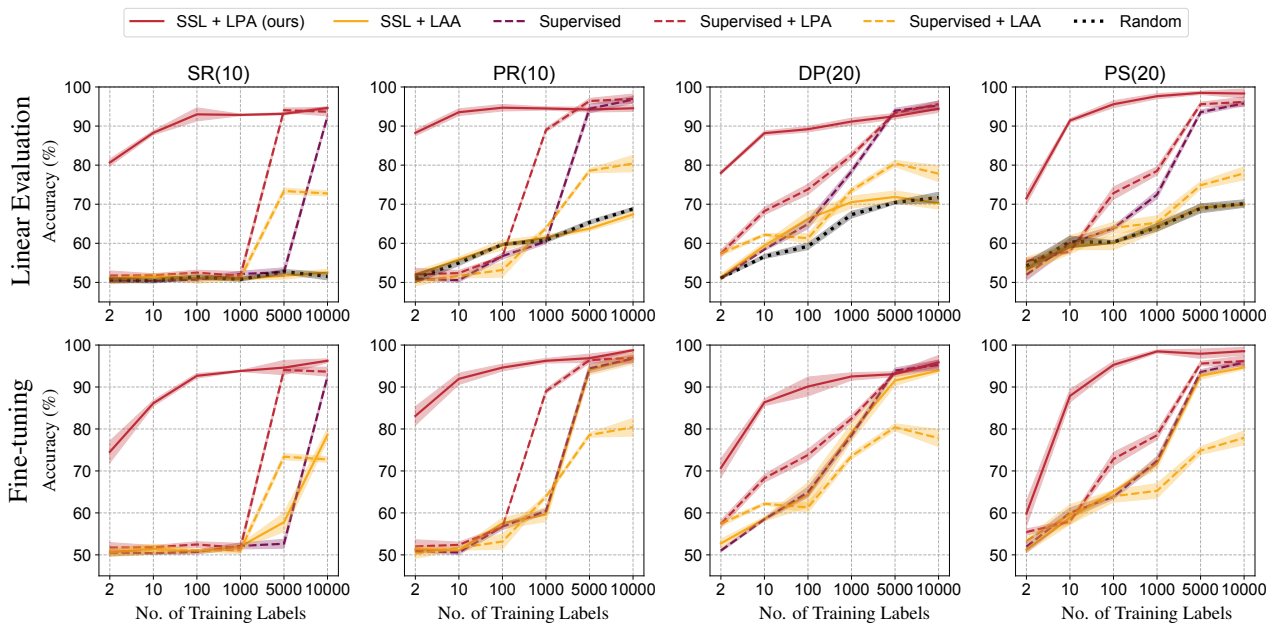


Figure 10. Our method (SSL + LPA) achieves significantly higher accuracy after linear evaluation and fine-tuning than baselines in low-label regime and is comparable to supervised models that have been given more labels. We vary the number of training labelled instances from 2 to  $10^4$  and report the average accuracy and standard error over 3 trials for all methods.