# Learning Better Representations From Less Data For Propositional Satisfiability

**Mohamed Ghanem**
CISPA Helmholtz Center for Information Security, Saarbrücken, Germany

**Frederik Schmitt**
CISPA Helmholtz Center for Information Security, Saarbrücken, Germany

**Julian Siber**
CISPA Helmholtz Center for Information Security, Saarbrücken, Germany

**Bernd Finkbeiner**
CISPA Helmholtz Center for Information Security, Saarbrücken, Germany
Technical University of Munich, Munich, Germany

## Abstract

Training neural networks on NP-complete problems typically demands very large amounts of training data and often needs to be coupled with computationally expensive symbolic verifiers to ensure output correctness. In this paper, we present NeuRes, a neuro-symbolic approach to address both challenges for propositional satisfiability, being the quintessential NP-complete problem. By combining certificate-driven training and expert iteration, our model learns better representations than models trained for classification only, with a much higher data efficiency – requiring orders of magnitude less training data. NeuRes employs propositional resolution as a proof system to generate proofs of unsatisfiability and to accelerate the process of finding satisfying truth assignments, exploring both possibilities in parallel. To realize this, we propose an attention-based architecture that autoregressively selects pairs of clauses from a dynamic formula embedding to derive new clauses. Furthermore, we employ expert iteration whereby model-generated proofs progressively replace longer teacher proofs as the new ground truth. This enables our model to reduce a dataset of proofs generated by an advanced solver by ∼32% after training on it with no extra guidance. This shows that NeuRes is not limited by the optimality of the teacher algorithm owing to its self-improving workflow. We show that our model achieves far better performance than NeuroSAT in terms of both correctly classified and proven instances.

**2012 ACM Subject Classification** Computing methodologies → Learning latent representations; Theory of computation → Automated reasoning

**Keywords and phrases** Satisfiability, Neuro-symbolic learning, Resolution proofs, Expert iteration, Graph neural networks, Attention

**Digital Object Identifier** 10.4230/LIPIcs.CVIT.2016.23

**Related Version** *Conference Version*: `NeurIPS 2024 proceedings`

**Acknowledgements** This work was supported by the European Research Council (ERC) Grant HYPER (No. 101055412).

## 1 Introduction

Boolean satisfiability (SAT) is a fundamental problem in computer science. For theory, this stems from SAT being the first problem proven NP-complete [13]. For practice, this is due to many highly-optimized SAT solvers being used as flexible reasoning engines in a variety of tasks such as model checking [12, 47], software verification [17], planning [27], and mathematical proof search [24]. Recently, SAT has also served as a litmus test for assessing the symbolic reasoning capabilities of neural models and a promising domain for

neuro-symbolic systems [43, 42, 1, 10, 36]. So far, neural models only provide limited, if any, justification for unsatisfiability predictions. NeuroCore [42], for example, predicts an unsatisfiable core, the verification of which can be as hard as solving the original problem. No certificates at all or certificates that are hard to check limit neural methods in a domain where correctness is critical and prevents close integrations with symbolic methods. Therefore, we propose a neuro-symbolic model that utilizes *resolution* to solve SAT problems by generating easy-to-check certificates.

A resolution proof is a sequence of case distinctions, each involving two clauses, that ends in the empty clause (falsum). This technique can also be used to prove satisfiability by exhaustively applying it until no further new resolution steps are possible and the empty clause has not been derived. Generating such a proof is an interesting problem from a neuro-symbolic perspective because unlike other discrete combinatorial problems that have been considered before [46, 7, 28, 30, 11], it requires selecting compatible *pairs* of clauses from the dynamically growing pool, as newly derived clauses are naturally considered for derivation in subsequent steps. In this work, we devise three attention-based mechanisms to perform this pair-selection needed for generating resolution proofs. In addition, we augment the architecture to efficiently handle *sat* (satisfiable) formulas with an assignment decoding mechanism that assigns a truth value to each literal. We hypothesize that, despite their final goals being in complete opposition, resolution and *sat* assignment finding can form a mutually beneficial collaboration. On the one hand, clauses derived by resolution incrementally inject additional information into the network, e.g., deriving a single-literal clause by resolution directly implies that literal should be true in any possible *sat* assignment. On the other hand, finding a *sat* assignment absolves the resolution network from having to prove satisfiability by exhaustion. On that basis, given an input formula, NeuRes proceeds in two parallel tracks: (1) finding a *sat* assignment, and (2) deriving a resolution proof of unsatisfiability. Both tracks operate on a shared representation of the problem state. Depending on which track succeeds, NeuRes produces the corresponding SAT verdict which is guaranteed to be sound by virtue of its certificate-based design. Since both of our certificate types are efficient to check, we can afford to perform these symbolic checks at each step. When comparing NeuRes with NeuroSAT [43], which has been trained to predict satisfiability with millions of samples, we demonstrate that NeuRes achieves a higher accuracy while providing a proof and requires only thousands of training samples.

As for most problems in theorem proving we are not only interest in finding any proof but a short proof. Resolution proofs can vary largely in their size depending on the resolution steps taken. Being able to efficiently check the proof, also allows us to adapt the proof target while training the model. In particular, we explore an expert iteration mechanism [2, 39] that pre-rolls the resolution proof of the model and replaces the target proof whenever the pre-rolled proof is shorter. We demonstrate that this bootstrapping mechanism iteratively shortens the proofs of our training dataset while further improving the overall performance of the model.

We make the following contributions:

1. We introduce novel architectures which combine graph neural networks with attention mechanisms for generating resolution proofs and assignments for CNF formulas (Section 4).
2. We show that for propositional logic, learning to prove rather than predict satisfiability results in better representations and requires far less training samples (Section 6 and 7).
3. We devise a bootstrapped training procedure where our model progressively produces shorter resolution proofs than its teacher (Section 6.2) boosting the model's overall performance.

The implementation of our framework can be found at `https://github.com/Oschart/NeuRes`.

## 2 Related Work

### SAT Solving and Certificates.

We refer to the annual SAT competitions [5] for a comprehensive overview on the ever-evolving landscape of SAT solvers, benchmarks, and proof checkers. SAT solvers are complex systems with a documented history of bugs [9, 26], hence proof certificates have been partially required in this competition since 2013 [3]. Unlike satisfiable formulas, there are several ways to certify unsatisfiable formulas [23]. Resolution proofs [52, 20] are easy to verify [15], but non-trivial to generate from modern solvers based on the paradigm of conflict-driven clause learning [34]. Clausal proofs, e.g., in DRAT format [50], are easier to generate and space-efficient, but hard to validate. Verifying the proofs can take longer than their discovery [22] and requires highly optimized algorithms [31].

### Deep Learning for SAT Solving.

NeuroSAT [43] was the first study of the Boolean satisfiability problem as an end-to-end learning task. Building upon the NeuroSAT architecture, a simplified version has been trained to predict unsatisfiable cores and successfully integrated as a branching heuristic in a state-of-the-art SAT solver [42]. Recent work has employed a related architecture as a phase selection heuristic [49]. It has been shown that both the NeuroSAT architecture and a newly introduced deep exchangeable architecture can outperform SAT solvers on instances of 3-SAT problems [10]. The NeuroSAT architecture has also been applied on special classes of crypto-analysis problems [44]. In addition to supervised learning, unsupervised methods have been proposed for solving SAT problems. For Circuit-SAT a deep-gated DAG recursive neural architecture has been presented together with a differentiable training objective to optimize towards solving the Circuit-SAT problem and finding a satisfying assignment [1]. For Boolean satisfiability, a differentiable training objective has been proposed together with a query mechanism that allows for recurrent solution trials [36].

### Deep Learning for Formal Proof Generation.

In formal mathematics, deep learning has been integrated with theorem proving for clause selection [33, 18], premise selection [25, 48, 6, 35], tactic prediction [51, 37] and whole proof searches [40, 19]. For SMT formulas specifically, deep reinforcement learning has been applied to tactic prediction [4]. In the domain of quantified boolean formulas, heuristics have been learned to guide search algorithms in proving the satisfiability and unsatisfiability of formulas [32]. For temporal logics, deep learning has been applied to prove the satisfiability of linear-time temporal logic formulas and the realizability of specifications [21, 41, 14].

## 3 Proofs of (Un-)Satisfiability

We start with a brief review of certifying the (un-)satisfiability of propositional formulas in conjunctive normal form. For a set of Boolean variables $V$, we identify with each variable $x \in V$ the *positive literal* $x$ and the *negative literal* $\neg x$ denoted by $\bar{x}$. A *clause* corresponds to a disjunction of literals and is abbreviated by a set of literals, e.g., $\{\bar{1}, 3\}$ represents $(\neg x_1 \vee x_3)$. A formula in *conjunctive normal form (CNF)* is a conjunction of clauses and is

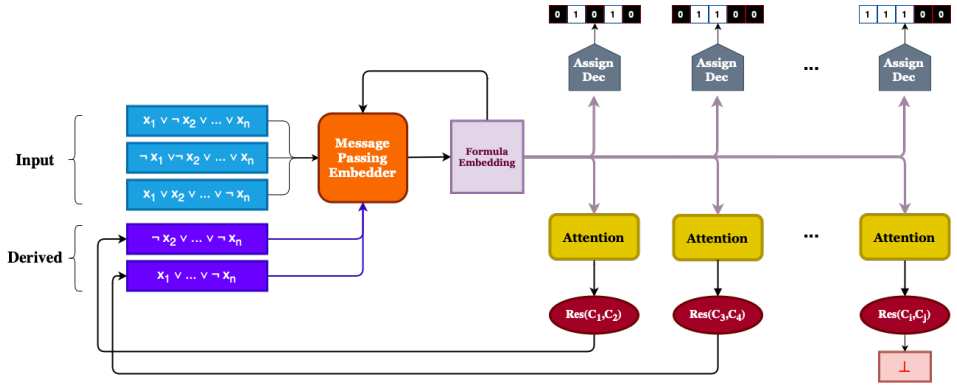



**Figure 1** Overall NeuRes architecture.

abbreviated by a set of clauses, e.g., $\{\{\bar{1}, 3\}, \{1, 2, \bar{4}\}\}$ represents $(\neg x_1 \vee x_3) \wedge (x_1 \vee x_2 \vee \neg x_4)$. Any Boolean formula can be converted to an equisatisfiable CNF formula in polynomial time, for example with Tseitin transformation [45].

A CNF formula is *satisfiable* if there exists an *assignment* $\mathcal{A} : V \to \{\top, \bot\}$ such that all clauses are satisfied, i.e., each clause contains a positive literal $x$ such that $\mathcal{A}(x) = \top$ or a negative literal $\bar{x}$ such that $\mathcal{A}(x) = \bot$. If no such assignment exists we call the formula *unsatisfiable*. To prove unsatisfiability we rely on resolution, a fundamental inference rule in satisfiability testing [16]. The resolution rule (*Res*) picks clauses with two opposite literals and performs the following inference:

$$\frac{C_1 \cup \{x\} \qquad C_2 \cup \{\bar{x}\}}{C_1 \cup C_2}\ Res$$

Resolution effectively performs a case distinction on the value of variable $x$: Either it is assigned to *false*, then $C_1$ has to evaluate to *true*, or it is assigned to *true*, then $C_2$ has to evaluate to *true*. Hence, we may infer the clause $C_1 \cup C_2$. A *resolution proof* for a CNF formula is a sequence of applications of the *Res* rule ending in the empty clause.

# 4 Models

## 4.1 General Architecture

NeuRes is a neural network that takes a CNF formula as a set of clauses and outputs either a satisfying truth assignment or a resolution proof of unsatisfiability. As such, our model comprises a formula embedder connected to two downstream heads: (1) an attention network responsible for selecting clause pairs, and (2) a truth assignment decoder. See Figure 1 for an overview of the NeuRes architecture. After obtaining the initial clause and literal embeddings (representing the input formula), we continue with the iterative certificate generation phase. At each step, the model selects a clause pair which gets resolved into a new clause to append to the current formula graph while decoding a candidate truth assignment in parallel. The model keeps deriving new clauses until the empty clause is found (marking resolution proof completion), a satisfying assignment is found (marking a certified *sat* verdict), or the limit on episode length is reached (marking timeout).

## 4.2 Message-Passing Embedder

Similar to NeuroSAT, we use a message-passing GNN to obtain clause and literal embeddings by performing a predetermined number of rounds. Our formula graph is also constructed in a similar fashion to NeuroSAT graphs where clause nodes are connected to their constituent literal nodes and literals are connect to their complements (cf. Appendix A). For a formula in $m$ variables and $n$ clauses, the outputs of this GNN are two matrices: $E^L \in \mathbb{R}^{m \times d}$ for literal embeddings and $E^C \in \mathbb{R}^{n \times d}$ for clause embedding, where $d \in \mathbb{N}^+$ is the embedding dimension. Here we have two key differences from NeuroSAT. Firstly, NeuroSAT uses these embeddings as voters to predict satisfiability through a classification MLP. In our case, we use these embeddings as clause tokens for clause pair selection and literal tokens for truth value assignment. Secondly, since our model derives new clauses with every resolution step, we need to embed these new clauses, as well as update existing embeddings to reflect their relation to the newly inferred clauses. Consequently, we need to introduce a new phase to the message-passing protocol, for which we explore two approaches: *static embeddings* and *dynamic embeddings*.

In a *static* approach, we do not change the embeddings of initial clauses upon inferring a new clause. Instead, we exchange local messages between the node corresponding to the new clause and its literal nodes, in both directions. The main advantage of this approach is its low cost. A major drawback is that initial clauses never learn information about their relation to newly inferred clauses.

In a *dynamic* approach, we do not only generate a new clause and its embedding, we also update the embeddings of all other clauses. This accounts for the fact that the utility of an existing clause may change with the introduction of a new clause. We perform one message-passing round on the mature graph for every newly derived clause, which produces the new clause embedding and updates other clause embeddings. Since message-passing rounds are parallel across clauses, a single update to the whole embedding matrix is reasonably efficient.

## 4.3 Selector Networks

After producing clause and literal embeddings, NeuRes enters the derivation stage. At each step, our model needs to select two clauses to resolve, produce the resultant clause, and add it to the current formula. To realize our clause-pair selection mechanism, we employ three attention-based designs.

### 4.3.1 Cascaded Attention (Casc-Attn)

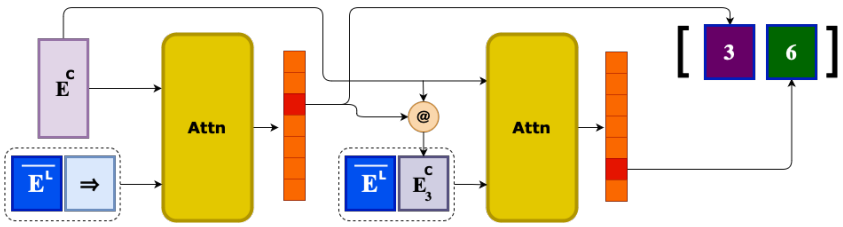



**Figure 2** Cascaded attention.

In this design, pairs are selected by making two consecutive attention queries on the clause pool. We condition the second attention query on the outcome (i.e., the clause) of the first query. Figure 2 shows this scheme where we perform the first query using the mean

of the literal embeddings $\overline{E^L}$ concatenated with a zero vector while performing the second query using the mean of the literal embeddings concatenated with the embedding vector $E^C_{c_1}$ of the clause selected in the first query. Formally, Casc-Attn selects a clause index pair $(c_1, c_2)$ as follows:

$$c_i = \underset{j}{\textbf{argmax}} \left[u^T \tanh(W_1 q_i + W_2 E^C_j)\right] \quad \text{with} \quad q_i = \begin{cases} \overline{E^L} \parallel \mathbf{0} & \text{if } i = 1 \\ \overline{E^L} \parallel E^C_{c_1} & \text{if } i = 2 \end{cases} \tag{1}$$

where $W_1 \in \mathbb{R}^{2d \times d}, W_2 \in \mathbb{R}^{d \times d}, u \in \mathbb{R}^d$ are trainable network parameters.

The advantage of this design is that it is not limited to pair selection and can be used to select a tuple of arbitrary length. The main downside, however, is that this design chooses $c_1$ independently from $c_2$, which is undesirable because the utility of a resolution step is determined by both clauses simultaneously (not sequentially).

### 4.3.2 Full Self-Attention (Full-Attn)

To address the downside of independent clause selection, this variant performs self-attention between all clauses to obtain a matrix $S \in \mathbb{R}^{n \times n}$ where $S_{i,j}$ represents the attention score of the clause pair $(c_i, c_j)$ as shown in Figure 3. The model selects clause pairs by choosing the cell with the maximal score. In this attention scheme, the clause embeddings are used as both queries and keys.

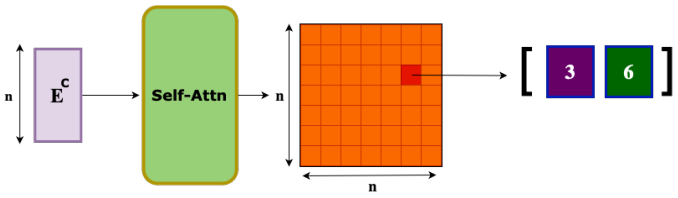



**Figure 3** Full self-attention.

Formally, Full-Attn selects a clause index pair $(c_1, c_2)$ as follows:

$$(c_1, c_2) = \underset{(i,j)}{\textbf{argmax}} \; S_{i,j} \quad \text{with} \quad Q = E^C W_Q; \quad K = E^C W_K; \quad S = \frac{QK^T}{\sqrt{d}} \tag{2}$$

where $W_Q \in \mathbb{R}^{d \times d}, W_K \in \mathbb{R}^{d \times d}$ are trainable network parameters. Since $S$ contains many cells that correspond to invalid resolution steps (i.e., clause pairs that cannot be resolved), we mask out the invalid cells from the attention grid in ensure the network selection is valid at every step.

### 4.3.3 Anchored Self-Attention (Anch-Attn)

In Full-Attn, the attention grid grows quadratically with the number of clauses. In this variant, we relax this cost by exploiting a property of binary resolution where each step targets a single variable in the two resolvent clauses. This allows us to narrow down candidate clause pairs by first selecting a variable as an anchor on which our clauses should be resolved. As such, we do not need to consider the full clause set at once, only the clauses containing the chosen variable $v$. We further compress the attention grid by lining clauses containing

the literal $v$ on rows while lining clauses containing the literal $\neg v$ on columns. This reduces the redundancy of the attention grid since clauses containing the variable $v$ with the same parity cannot be resolved on $v$, so there is no point in matching them. In this scheme, we have two attention modules: one attention network to choose an anchor variable followed by a self-attention network to produce the anchored score grid.

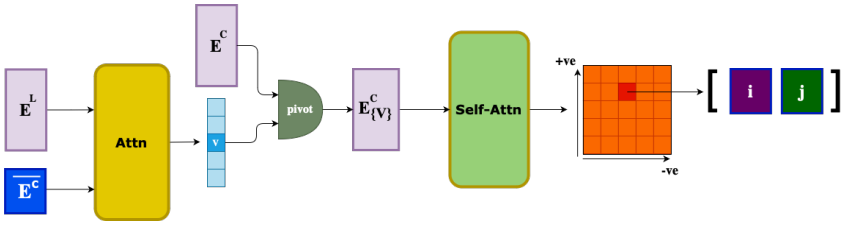



**Figure 4** Anchored self-attention.

In light of Figure 4, this approach combines structural elements from Casc-Attn and Full-Attn; however, both elements are used differently in Anch-Attn. Firstly, the attention mechanism in Casc-Attn is used to select clauses whereas Anch-Attn uses it to select variables. Secondly, self-attention in Full-Attn matches any pair of clauses $(c_i, c_j)$ in both directions as the row and column dimensions in the attention score grid reflect the same clauses (all clauses). By contrast, Anch-Attn computes self-attention scores for clause pairs in only one order (positive instance to negative instance). Formally, Anch-Attn selects an anchor variable $v$ as follows:

$$v = \underset{i}{\mathbf{argmax}} \left[ u^T \tanh(W_1 \overline{E^C} + W_2(E_i^{L^+} + E_i^{L^-})) \right] \tag{3}$$

where $W_1 \in \mathbb{R}^{d \times d}, W_2 \in \mathbb{R}^{d \times d}, u \in \mathbb{R}^d$ are trainable network parameters. The clause index pair $(c_1, c_2)$ is then selected according to the same equations of Full-Attn (Eq. 2) using the $v$-anchored set of clause embeddings.

## 4.4 Assignment Decoder

To extract satisfying assignments, we use a sigmoid-activated MLP $\psi$ on top of the literal embeddings $E^L$ to assign a truth value $\hat{\mathcal{A}}(l_i)$ to a literal $l_i$ as shown in Eq. 4.

$$\hat{\mathcal{A}}(l_i) = \sigma(\psi(E_i^L)) \tag{4}$$

Note that since for each variable, we have a positive and a negative literal embeddings, we can construct two different truth assignments at a time using this method. However, supervising both assignments did not improve the performance compared to only supervising the positive assignment (on positive literal embeddings). Thus, to simplify our loss function, we only derive truth assignments from the positive literal embeddings at train time while extracting both at test time. Interestingly, at test time, we found using negative literals (in addition to the positive ones) sometimes produces satisfying assignments before the positive branch despite receiving no direct supervision during training. Our intuition regarding this observation attributes it to the fact that the formula graph has no explicit notion of positive and negative literals, it only represents connections to clauses (positive and negative literals are connected by an undirected edge that does not distinguish their parity). As such, both literal nodes have a different local view into the rest of formula, which could result in one of them leading to a satisfying assignment faster than the other.

# 5 Training and Hyperparameters

## 5.1 Dataset

For our training and testing data, we adopt the same formula generation method as NeuroSAT [43], namely $\mathbf{SR}(n)$ where $n$ is the number of variables in the formula. This method was designed to generate a generalized formula distribution that is not limited to a particular domain of SAT problems. To control our data distributions, we vary the range on the number of Boolean variables involved in each formula. For our training data, we use formulas in $\mathbf{SR}(U(10, 40))$ where $U(10, 40)$ denotes the uniform distribution on integers between 10 and 40 (inclusive). To generate our teacher certificates comprising resolution proofs and truth assignments, we use the BooleForce solver [8] on the formulas generated on the $\mathbf{SR}$ distribution.

## 5.2 Loss Function

We train our model in a supervised fashion using teacher-forcing on solver certificates. During *unsat* episodes, teacher actions (clause pairs) are imposed over the whole run. The length of the teacher proof dictates the length of the respective episode, denoted as $T$. Model parameters $\theta$ maximize the likelihood of teacher choices $y_t$ thereby minimizing the resolution loss $\mathcal{L}_{Res}$ shown in Eq 5.

$$\mathcal{L}_{Res} = -\frac{1}{T}\sum_{t}\log(p(y_t;\theta))\cdot\gamma^{(T-t)} \tag{5}$$

During *sat* episodes, we minimize $\mathcal{L}_{sat}$ computed as the binary cross-entropy loss between the sigmoid-activated outputs of assignment decoder $\hat{\mathcal{A}} : V \to [0, 1]$ and the teacher assignment $\mathcal{A} : V \to \{0, 1\}$ as shown in Eq. 6.

$$\mathcal{L}_{sat} = \frac{1}{T}\sum_{t}\left[\frac{\gamma^{(T-t)}}{|V|}\sum_{v}^{V}\mathrm{BCE}(\hat{\mathcal{A}}(v), \mathcal{A}(v))\right] \tag{6}$$

In both types of episodes, step-wise losses are weighted by a time-horizon discounting factor $\gamma < 1.0$ over the whole episode. The main rationale behind this is that later losses should have higher weights as the formula tends to get easier to solve with each new clause inferred by resolution.

## 5.3 Hyperparameters

NeuRes has several hyperparameters that influence network size, depth, and loss weighting. In the experiments we fix the embedding dimension to 128. We train our models with a batch size of 1 and the Adam optimizer [29] for 50 epochs which took about six days on a single NVIDIA A100 GPU. We linearly anneal the learning rate from $5 \times 10^{-5}$ to zero over the training episodes. This empirically yields better results than using a constant learning rate. We use a time discounting factor $\gamma = 0.99$ for the episodic loss. We apply global-norm gradient clipping with a ratio of 0.5 [38].

# 6 Generating Resolution Proofs

NeuRes uses resolution as the core reasoning technique for certificate generation, both in the *unsat* and *sat* cases. Hence, we start with an in-depth comparative evaluation of several

internal variants for resolution only. In particular, we evaluate the success rate (i.e., problems solved before timeout) and proof length relative to the teacher, denoted by p-Len $= \frac{|\mathcal{P}_{\text{NeuRes}}|}{|\mathcal{P}_{\text{teacher}}|}$. We use a limit of 4 on this ratio as a timeout to avoid simply brute-forcing a resolution proof. Note that we measure p-Len only for solved formulas to avoid diluting the average with resolution trails that timed out. For experiments in this section, we train on 8K *unsat* formulas in $\mathbf{SR}(U(10, 40))$ and test our models on 10K unseen formulas belonging to the same distribution. We use more formulas than the model was trained on to more reliably demonstrate its learning capacity.

**Table 1** Performance of all attention variants on *unsat* $\mathbf{SR}(U(10, 40))$ test problems.

| Variant | Static-Embed | | Dynamic-Embed | |
|---|---|---|---|---|
| | **Proven (%)** | **p-Len** | **Proven (%)** | **p-Len** |
| Casc-Attn | 14.72 | 1.87 | 37.33 | 1.79 |
| Full-Attn | 25.38 | **1.61** | **95.2** | **1.67** |
| Anch-Attn | **28.72** | 2.12 | 60.5 | 2.28 |

## 6.1 Attention Variants

To assess the basic resolution performance of NeuRes, we evaluate each attention variant using both static and dynamic embeddings. For this experiment, we perform 32 rounds of message-passing for each input formula. As shown in Table 1, dynamic embedding is decisively better for all three attention variants, thereby confirming its conceptual merit. While anchored attention leads over other variants under static embeddings, full attention performs significantly better for dynamic embeddings, albeit at the cost of longer proofs on average. We believe that Anch-Attn's better performance in the static setting can be explained through the full connectivity of its attention grid (proven in Appendix B). Since dynamic-embedding Full-Attn is the best-performing configuration over in-distribution test settings, we will demonstrate the remaining evaluation experiments exclusively on this variant.

## 6.2 Shortening Teacher Proofs with Bootstrapping

During our initial experiments, we discovered proofs produced by NeuRes that were shorter than the corresponding teacher proofs in the training data. Although teacher proofs were generated by a traditional SAT solver, they are not guaranteed to be size-optimal. The size of resolution proofs is their only real drawback, hence any method that can reduce this size would be immensely useful. Upon closer inspection we find that, on average, our previous best performer trained with regular teacher-forcing manages to shorten ∼18% of teacher proofs by a notable factor (cf. Appendix C). This inspired us to devise a bootstrapped training procedure to capitalize on this feature: We pre-roll each input problem using model actions only, and whenever the model proof is shorter than the teacher's, it replaces the teacher's proof in the dataset. In other words, we maximize the likelihood of the shorter proof. In doing so iteratively, the model progressively becomes its own teacher by exploiting redundancies in the teacher algorithm.

**Table 2** Bootstrapped training data reduction statistics. Reduction statistics are computed on the $\mathbf{SR}(U(10, 40))$ training set while p-Len and success rate are computed on a test set of the same distribution.

| | |
|---|---|
| Reduction Depth | **max**: 23, **avg**: 6.6 |
| Proof Reduction (%) | **max**: 86.11, **avg**: 33.51 |
| Proofs Reduced (%) | 90.08 |
| Total Reduction (%) | 31.85 |
| p-Len | 1.15 |
| Success Rate (%) | 100.0 |

The outcome of this bootstrapped training process is summarized in Table 2. We find that bootstrapping results in notable gains in terms of both success rate and optimality. The sharp decline in proof length (relatively quantified by p-Len) at test time shows that the models transfers the bootstrapped knowledge to unseen test formulas, as opposed to merely overfitting on training formulas. In addition to success rate and p-Len, we inspect the reduction statistics of our bootstrapped variant (first three rows of Table 2). Since the bootstrapped model performs multiple reduction scans over the training dataset, we add a metric for reduction depth computed as the number of progressive reductions made to a proof. To further quantify this effect, we report the maximum and average reduction ratios of reduced proofs relative to teacher proofs. Finally, we report the total reduction made to the dataset size in terms of total number of proof steps.

In Appendix C, we have compiled additional statistics (cf. Table 5) on proof shortening during the training process, as well as an example proof reduced by the bootstrapped NeuRes (Figure 7). We only include a small reduction example (from 20 steps to 10 steps) for space constraints, but we observed many more examples of much larger reductions (e.g., over 400 steps).

## 7 Resolution-Aided SAT Solving

In this section, we evaluate the performance of our fully integrated model trained on a hybrid dataset comprising 8K unsatisfiable formulas (and their resolution proofs) and 8K satisfiable formulas (and their satisfying assignments). For the unsatisfiable formulas, timeout ($4 \times |\mathcal{P}_{\text{teacher}}|$) and optimality (p-Len) are measured similarly to previous experiments. For satisfiable formulas, we set the timeout (maximum #trials) to $2 \times |V|$. Ultimately, this section aims to investigate the effect of incorporating a certificate-driven downstream head on the quality of the learnt representations through its impact on the performance of the complementary task, i.e., proving/predicting satisfiability. We use NeuroSAT as our baseline as it employs the same formula embedding architecture. Since NeuroSAT proves *sat* but only *predicts unsat*, we train a classification MLP on top of our trained NeuRes model to further showcase the benefit of our representations on prediction accuracy.

Table 3 confirms this main hypothesis. In essence, this result points to the fact that learning signals obtained from training on *unsat* certificates largely enhance the ability of the neural network to extract useful information from the input formula. This is doubly promising considering NeuroSAT was trained on *millions* of formulas while NeuRes was trained on only *16K* formulas. Lastly, we find that augmenting *sat* formulas by resolution

**Table 3** Performance of full solver mode tested on **SR**(40) problems and trained on **SR**($U(10, 40)$) problems where PREDICTED refers to the satisfiability prediction without certificate.

| Model | Proven (%) | | | Predicted (%) | | |
|---|---|---|---|---|---|---|
| | SAT | UNSAT | TOTAL | SAT | UNSAT | TOTAL |
| NEURES | **96.8** | **99.6** | **98.2** | **84.28** | **99.2** | **91.65** |
| NEUROSAT [43] | 70 | - | - | 73 | 96 | 85 |

derivations results in relative improvements ($\sim 2.3\%$) in success rate even though these derivations are attempting to prove unsatisfiability.

## 8 Utilizing Model Fan-Out

In our Full-Attention module, we compute $n^2$ scores and only perform the top-score resolution step. This greedy approach arguably underutilizes the attention grid computations as it ignores other high-scoring steps that might lead to a shorter proof thereby improving the success rate in addition to reducing the number of queries to the model. The latter leads to an overall runtime reduction since performing an extra symbolic resolution step is much faster than a forward model pass. As such, we experiment with performing the top $k$ steps of the attention grid after each forward pass. It should be noted, however, that this yields diminishing returns as it leads to a faster growth of the clause base which in turn inflates the attention grid. For $k > 1$, after deriving the empty clause, only clauses that connect to it in the resolution graph are kept in the final proof. This post-processing step is linear in the proof length and eliminates redundant resolution steps resulting from the higher fan-out. Table 4 shows that taking the top 3 steps already yields a massive reduction in proof lengths along with a significant boost to the success rate.

**Table 4** Performance of different model fan-outs on **SR**(40) test data. Proof length (p-Len) and #Model Calls are both normalized by the length of the teacher proof.

| Full-Attn Fan-out | p-Len | #Model Calls | Total Proven (%) |
|---|---|---|---|
| **Top-1** | 1.15 | 1.15 | 98.2 |
| **Top-3** | 0.57 | 0.49 | 99.9 |
| **Top-5** | **0.52** | **0.43** | **100.0** |

One way to offset the attention grid inflation with higher fan-out would be to keep a saliency map for all clauses then discarding $k$ clauses with the least saliency scores after each forward pass. One simple way to compute this saliency score for a clause would be the sum/mean/max of its respective row in the attention scores grid. Another proxy for saliency could be the recency score reflecting how many steps have elapsed since the last time a given clause was used.

## 9 Generalizing to Larger Problems

In order to test our model's out-of-distribution performance, we evaluate our NeuRes model on five datasets comprising formulas with up to 5 times more variables than encountered during training. We use the same distributions reported by NeuroSAT and we run our model for the same maximum number of iterations (1000).

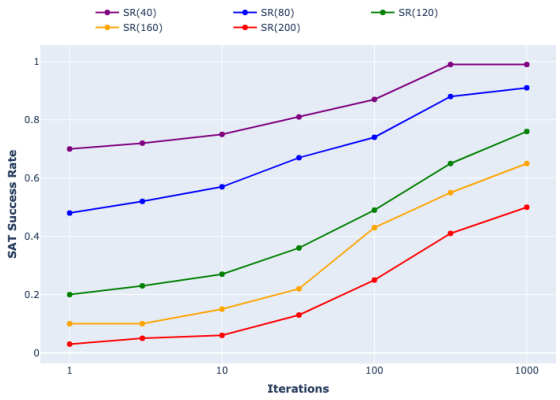



**Figure 5** SAT success rate over iterations.

Figure 5 shows the scalability of NeuRes to larger problems by letting it run for more iterations. Compared to NeuroSAT [43], NeuRes scores a much higher first-try success rate on all 5 problem distributions, and a higher final success rate on all of them except for **SR**(40) on which both models nearly score 100%. Particularly, NeuRes shows higher first-try success on the 3 largest problem sizes where NeuroSAT solves zero or near-zero problems on the first try.

## 10 Conclusion

In this paper, we introduced a deep learning approach for proving and predicting propositional satisfiability. We proposed an architecture that combines graph neural networks with attention mechanisms to generate resolution proofs of unsatisfiability. Unlike methods that merely predict unsatisfiability, our models provide easily verifiable certificates for their verdicts. We demonstrated that our certificate-based training and resolution-aided mode of operation surpass previous approaches in terms of performance and data efficiency, which we attribute to learning better representations.

Despite its promising benchmark performance, our model cannot solely outperform highly engineered industrial solvers, as is currently the case for all neural methods as standalone tools. The gap between neural networks and symbolic algorithms is still rather large, and our hope is to bring deep learning methods one concrete step closer to filling this gap. For NeuRes, this step is recognizing the immense value of carefully integrating certificates into the model design and training as opposed to using shallow supervision labels. Last but not least, it is worth noting that even at their present state, neural networks stand great potential to advance traditional solvers by combining them into hybrid solvers that utilize the deep long-range dependencies captured by neural networks along with the exploration speed of symbolic algorithms. Moreover, we demonstrated a unique potential to advance SAT solving through proof reduction, as proof size is a major challenge in certifying the

results of traditional solvers. This proof reduction is facilitated by a bootstrapped training procedure that uses teacher proofs as a guide as opposed to a golden standard.

## References


1 Saeed Amizadeh, Sergiy Matusevych, and Markus Weimer. Learning to solve circuit-sat: An unsupervised differentiable approach. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL: https://openreview.net/forum?id=BJxgz2R9t7.

2 Thomas Anthony, Zheng Tian, and David Barber. Thinking fast and slow with deep learning and tree search. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 5360–5370, 2017. URL: https://proceedings.neurips.cc/paper/2017/hash/d8e1344e27a5b08cdfd5d027d9b8d6de-Abstract.html.

3 Adrian Balint, Anton Belov, Marijn J.H. Heule, and Matti Järvisalo, editors. *Proceedings of SAT Competition 2013: Solver and Benchmark Descriptions*, volume B-2013-1 of *Department of Computer Science Series of Publications B*. University of Helsinki, Finland, 2013.

4 Mislav Balunovic, Pavol Bielik, and Martin T. Vechev. Learning to solve SMT formulas. In Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pages 10338–10349, 2018. URL: https://proceedings.neurips.cc/paper/2018/hash/68331ff0427b551b68e911eebe35233b-Abstract.html.

5 Tomas Balyo, Marijn Heule, Markus Iser, Matti Järvisalo, and Martin Suda, editors. *Proceedings of SAT Competition 2023: Solver, Benchmark and Proof Checker Descriptions*. Department of Computer Science Series of Publications B. Department of Computer Science, University of Helsinki, Finland, 2023.

6 Kshitij Bansal, Sarah M. Loos, Markus N. Rabe, Christian Szegedy, and Stewart Wilcox. Holist: An environment for machine learning of higher order logic theorem proving. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 454–463. PMLR, 2019. URL: http://proceedings.mlr.press/v97/bansal19a.html.

7 Irwan Bello, Hieu Pham, Quoc V. Le, Mohammad Norouzi, and Samy Bengio. Neural combinatorial optimization with reinforcement learning. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Workshop Track Proceedings*. OpenReview.net, 2017. URL: https://openreview.net/forum?id=Bk9mxlSFx.

8 Armin Biere. Booleforce sat solver. https://fmv.jku.at/booleforce/, 2010.

9 Robert Brummayer, Florian Lonsing, and Armin Biere. Automated testing and debugging of SAT and QBF solvers. In Ofer Strichman and Stefan Szeider, editors, *Theory and Applications of Satisfiability Testing - SAT 2010, 13th International Conference, SAT 2010, Edinburgh, UK, July 11-14, 2010. Proceedings*, volume 6175 of *Lecture Notes in Computer Science*, pages 44–57. Springer, 2010. doi:10.1007/978-3-642-14186-7_6.

10 Chris Cameron, Rex Chen, Jason S. Hartford, and Kevin Leyton-Brown. Predicting propositional satisfiability via end-to-end learning. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, New York, NY, USA, February 7-12, 2020*, pages 3324–3331. AAAI Press, 2020. doi:10.1609/aaai.v34i04.5733.

11 Quentin Cappart, Didier Chételat, Elias B. Khalil, Andrea Lodi, Christopher Morris, and Petar Velickovic. Combinatorial optimization and reasoning with graph neural networks. *J. Mach. Learn. Res.*, 24:130:1–130:61, 2023. URL: http://jmlr.org/papers/v24/21-0449.html.

12 Edmund M. Clarke, Armin Biere, Richard Raimi, and Yunshan Zhu. Bounded model checking using satisfiability solving. *Formal Methods Syst. Des.*, 19(1):7–34, 2001. doi:10.1023/A:1011276507260.

13 Stephen A. Cook. The complexity of theorem-proving procedures. In Michael A. Harrison, Ranan B. Banerji, and Jeffrey D. Ullman, editors, *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing, May 3-5, 1971, Shaker Heights, Ohio, USA*, pages 151–158. ACM, 1971. doi:10.1145/800157.805047.

14 Matthias Cosler, Frederik Schmitt, Christopher Hahn, and Bernd Finkbeiner. Iterative circuit repair against formal specifications. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023. URL: https://openreview.net/pdf?id=SEcSahl0Ql.

15 Ashish Darbari, Bernd Fischer, and João Marques-Silva. Industrial-strength certified SAT solving through verified SAT proof checking. In Ana Cavalcanti, David Déharbe, Marie-Claude Gaudel, and Jim Woodcock, editors, *Theoretical Aspects of Computing - ICTAC 2010, 7th International Colloquium, Natal, Rio Grande do Norte, Brazil, September 1-3, 2010. Proceedings*, volume 6255 of *Lecture Notes in Computer Science*, pages 260–274. Springer, 2010. doi:10.1007/978-3-642-14808-8_18.

16 Martin Davis and Hilary Putnam. A computing procedure for quantification theory. *J. ACM*, 7(3):201–215, 1960. doi:10.1145/321033.321034.

17 Leonardo Mendonça de Moura and Nikolaj S. Bjørner. Satisfiability modulo theories: introduction and applications. *Commun. ACM*, 54(9):69–77, 2011. doi:10.1145/1995376.1995394.

18 Vlad Firoiu, Eser Aygün, Ankit Anand, Zafarali Ahmed, Xavier Glorot, Laurent Orseau, Lei M. Zhang, Doina Precup, and Shibl Mourad. Training a first-order theorem prover from synthetic data. *CoRR*, abs/2103.03798, 2021. URL: https://arxiv.org/abs/2103.03798, arXiv:2103.03798.

19 Emily First, Markus N. Rabe, Talia Ringer, and Yuriy Brun. Baldur: Whole-proof generation and repair with large language models. *CoRR*, abs/2303.04910, 2023. arXiv:2303.04910, doi:10.48550/arXiv.2303.04910.

20 Evguenii I. Goldberg and Yakov Novikov. Verification of proofs of unsatisfiability for CNF formulas. In *2003 Design, Automation and Test in Europe Conference and Exposition (DATE 2003), 3-7 March 2003, Munich, Germany*, pages 10886–10891. IEEE Computer Society, 2003. URL: https://doi.ieeecomputersociety.org/10.1109/DATE.2003.10008, doi:10.1109/DATE.2003.10008.

21 Christopher Hahn, Frederik Schmitt, Jens U. Kreber, Markus Norman Rabe, and Bernd Finkbeiner. Teaching temporal logics to neural networks. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. URL: https://openreview.net/forum?id=dOcQK-f4byz.

22 Marijn Heule, Warren A. Hunt Jr., and Nathan Wetzler. Bridging the gap between easy generation and efficient verification of unsatisfiability proofs. *Softw. Test. Verification Reliab.*, 24(8):593–607, 2014. doi:10.1002/stvr.1549.

23 Marijn J. H. Heule. Proofs of unsatisfiability. In Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability - Second Edition*, volume 336 of *Frontiers in Artificial Intelligence and Applications*, pages 635–668. IOS Press, 2021. doi:10.3233/FAIA200998.

24 Marijn J. H. Heule and Oliver Kullmann. The science of brute force. *Commun. ACM*, 60(8):70–79, 2017. doi:10.1145/3107239.

25 Geoffrey Irving, Christian Szegedy, Alexander A. Alemi, Niklas Eén, François Chollet, and Josef Urban. Deepmath - deep sequence models for premise selection. In Daniel D. Lee, Masashi Sugiyama, Ulrike von Luxburg, Isabelle Guyon, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona,*

*Spain*, pages 2235–2243, 2016. URL: https://proceedings.neurips.cc/paper/2016/hash/f197002b9a0853eca5e046d9ca4663d5-Abstract.html.

26 Matti Järvisalo, Marijn Heule, and Armin Biere. Inprocessing rules. In Bernhard Gramlich, Dale Miller, and Uli Sattler, editors, *Automated Reasoning - 6th International Joint Conference, IJCAR 2012, Manchester, UK, June 26-29, 2012. Proceedings*, volume 7364 of *Lecture Notes in Computer Science*, pages 355–370. Springer, 2012. doi:10.1007/978-3-642-31365-3_28.

27 Henry A. Kautz and Bart Selman. Planning as satisfiability. In Bernd Neumann, editor, *10th European Conference on Artificial Intelligence, ECAI 92, Vienna, Austria, August 3-7, 1992. Proceedings*, pages 359–363. John Wiley and Sons, 1992.

28 Elias B. Khalil, Hanjun Dai, Yuyu Zhang, Bistra Dilkina, and Le Song. Learning combinatorial optimization algorithms over graphs. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 6348–6358, 2017. URL: https://proceedings.neurips.cc/paper/2017/hash/d9896106ca98d3d05b8cbdf4fd8b13a1-Abstract.html.

29 Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

30 Wouter Kool, Herke van Hoof, and Max Welling. Attention, learn to solve routing problems! In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL: https://openreview.net/forum?id=ByxBFsRqYm.

31 Peter Lammich. Efficient verified (UN)SAT certificate checking. *J. Autom. Reason.*, 64(3):513–532, 2020. doi:10.1007/s10817-019-09525-z.

32 Gil Lederman, Markus N. Rabe, Sanjit Seshia, and Edward A. Lee. 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. URL: https://openreview.net/forum?id=BJluxREKDB.

33 Sarah M. Loos, Geoffrey Irving, Christian Szegedy, and Cezary Kaliszyk. Deep network guided proof search. In Thomas Eiter and David Sands, editors, *LPAR-21, 21st International Conference on Logic for Programming, Artificial Intelligence and Reasoning, Maun, Botswana, May 7-12, 2017*, volume 46 of *EPiC Series in Computing*, pages 85–105. EasyChair, 2017. URL: https://doi.org/10.29007/8mwc, doi:10.29007/8MWC.

34 João Marques-Silva, Inês Lynce, and Sharad Malik. Conflict-driven clause learning SAT solvers. In Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability - Second Edition*, volume 336 of *Frontiers in Artificial Intelligence and Applications*, pages 133–182. IOS Press, 2021. doi:10.3233/FAIA200987.

35 Maciej Mikula, Szymon Antoniak, Szymon Tworkowski, Albert Qiaochu Jiang, Jin Peng Zhou, Christian Szegedy, Lukasz Kucinski, Piotr Milos, and Yuhuai Wu. Magnushammer: A transformer-based approach to premise selection. *CoRR*, abs/2303.04488, 2023. URL: https://doi.org/10.48550/arXiv.2303.04488, arXiv:2303.04488, doi:10.48550/ARXIV.2303.04488.

36 Emils Ozolins, Karlis Freivalds, Andis Draguns, Eliza Gaile, Ronalds Zakovskis, and Sergejs Kozlovics. Goal-aware neural SAT solver. In *International Joint Conference on Neural Networks, IJCNN 2022, Padua, Italy, July 18-23, 2022*, pages 1–8. IEEE, 2022. doi:10.1109/IJCNN55064.2022.9892733.

37 Aditya Paliwal, Sarah M. Loos, Markus N. Rabe, Kshitij Bansal, and Christian Szegedy. Graph representations for higher-order logic and theorem proving. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, New York, NY, USA, February 7-12, 2020*, pages 2967–2974. AAAI Press, 2020. URL: https://doi.org/10.1609/aaai.v34i03.5689, doi:10.1609/AAAI.V34I03.5689.

565 **38** Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. Understanding the exploding gradient
566 problem. *CoRR, abs/1211.5063*, 2(417):1, 2012.

567 **39** Stanislas Polu, Jesse Michael Han, Kunhao Zheng, Mantas Baksys, Igor Babuschkin, and
568 Ilya Sutskever. Formal mathematics statement curriculum learning. In *The Eleventh Interna-*
569 *tional Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*.
570 OpenReview.net, 2023. URL: https://openreview.net/pdf?id=-P7G-8dmSh4.

571 **40** Stanislas Polu and Ilya Sutskever. Generative language modeling for automated theorem
572 proving. *CoRR*, abs/2009.03393, 2020. URL: https://arxiv.org/abs/2009.03393, arXiv:
573 2009.03393.

574 **41** Frederik Schmitt, Christopher Hahn, Markus N. Rabe, and Bernd Finkbeiner. Neural
575 circuit synthesis from specification patterns. In Marc'Aurelio Ranzato, Alina Beygelzi-
576 mer, Yann N. Dauphin, Percy Liang, and Jennifer Wortman Vaughan, editors, *Ad-*
577 *vances in Neural Information Processing Systems 34: Annual Conference on Neural*
578 *Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*,
579 pages 15408–15420, 2021. URL: https://proceedings.neurips.cc/paper/2021/hash/
580 8230bea7d54bcdf99cdfe85cb07313d5-Abstract.html.

581 **42** Daniel Selsam and Nikolaj S. Bjørner. Guiding high-performance SAT solvers with unsat-
582 core predictions. In Mikolás Janota and Inês Lynce, editors, *Theory and Applications of*
583 *Satisfiability Testing - SAT 2019 - 22nd International Conference, SAT 2019, Lisbon, Portugal,*
584 *July 9-12, 2019, Proceedings*, volume 11628 of *Lecture Notes in Computer Science*, pages
585 336–353. Springer, 2019. doi:10.1007/978-3-030-24258-9_24.

586 **43** Daniel Selsam, Matthew Lamm, Benedikt Bünz, Percy Liang, Leonardo de Moura, and
587 David L. Dill. Learning a SAT solver from single-bit supervision. In *7th International*
588 *Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*.
589 OpenReview.net, 2019. URL: https://openreview.net/forum?id=HJMC_iA5tm.

590 **44** Ling Sun, David Gérault, Adrien Benamira, and Thomas Peyrin. Neurogift: Using a machine
591 learning based sat solver for cryptanalysis. In Shlomi Dolev, Vladimir Kolesnikov, Sachin
592 Lodha, and Gera Weiss, editors, *Cyber Security Cryptography and Machine Learning - Fourth*
593 *International Symposium, CSCML 2020, Be'er Sheva, Israel, July 2-3, 2020, Proceedings*,
594 volume 12161 of *Lecture Notes in Computer Science*, pages 62–84. Springer, 2020. doi:
595 10.1007/978-3-030-49785-9_5.

596 **45** Grigori S Tseitin. On the complexity of derivation in propositional calculus. *Automation of*
597 *reasoning: 2: Classical papers on computational logic 1967–1970*, pages 466–483, 1983.

598 **46** Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. *Advances in neural*
599 *information processing systems*, 28, 2015.

600 **47** Yakir Vizel, Georg Weissenbacher, and Sharad Malik. Boolean satisfiability solvers and their
601 applications in model checking. *Proc. IEEE*, 103(11):2021–2035, 2015. doi:10.1109/JPROC.
602 2015.2455034.

603 **48** Mingzhe Wang, Yihe Tang, Jian Wang, and Jia Deng. Premise selection for theorem
604 proving by deep graph embedding. In Isabelle Guyon, Ulrike von Luxburg, Samy Ben-
605 gio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, ed-
606 itors, *Advances in Neural Information Processing Systems 30: Annual Conference on*
607 *Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA,*
608 *USA*, pages 2786–2796, 2017. URL: https://proceedings.neurips.cc/paper/2017/hash/
609 18d10dc6e666eab6de9215ae5b3d54df-Abstract.html.

610 **49** Wenxi Wang, Yang Hu, Mohit Tiwari, Sarfraz Khurshid, Kenneth L. McMillan, and Risto
611 Miikkulainen. Neuroback: Improving CDCL SAT solving using graph neural networks. In
612 *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna,*
613 *Austria, May 7-11, 2024*. OpenReview.net, 2024. URL: https://openreview.net/forum?id=
614 samyfu6G93.

615 **50** Nathan Wetzler, Marijn Heule, and Warren A. Hunt Jr. Drat-trim: Efficient checking and
616 trimming using expressive clausal proofs. In Carsten Sinz and Uwe Egly, editors, *Theory*

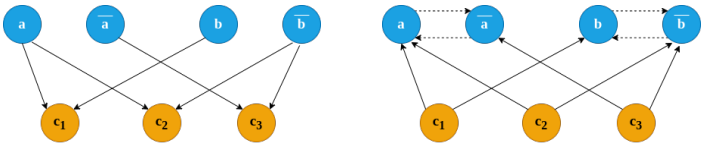
*and Applications of Satisfiability Testing - SAT 2014 - 17th International Conference, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 14-17, 2014. Proceedings*, volume 8561 of *Lecture Notes in Computer Science*, pages 422–429. Springer, 2014. doi:10.1007/978-3-319-09284-3_31.

51 Kaiyu Yang and Jia Deng. Learning to prove theorems via interacting with proof assistants. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 6984–6994. PMLR, 2019. URL: http://proceedings.mlr.press/v97/yang19a.html.

52 Lintao Zhang and Sharad Malik. Validating SAT solvers using an independent resolution-based checker: Practical implementations and other applications. In *2003 Design, Automation and Test in Europe Conference and Exposition (DATE 2003), 3-7 March 2003, Munich, Germany*, pages 10880–10885. IEEE Computer Society, 2003. URL: https://doi.ieeecomputersociety.org/10.1109/DATE.2003.10014, doi:10.1109/DATE.2003.10014.

# Appendix

## A NeuroSAT Formula Graph Construction




**(a)** Literal-to-Clause Phase **(b)** Clause-to-Literal Phase

**Figure 6** Two-phase message-passing round on NeuroSAT formula graph.

NeuroSAT-style formula graphs have two designated node types: clause nodes connected to the literal nodes corresponding to their constituent literals [43]. For example, in Figure 6, the clause contents are as follows: $c_1 = (a \vee b)$, $c_2 = (a \vee \bar{b})$, $c_3 = (\bar{a} \vee \bar{b})$. Each message-passing round involves two exchange phases: (1) Literal-to-Clause, and (2) Clause-to-Literal (and implicitly Literal-to-Complement). This construction is particularly efficient as it allows the message-passing protocol to cover the entire graph connectivity in at most $|V|+1$ rounds where $V$ is the set of variables in the formula.

## B Clause Connectivity Under Static Embeddings

In Section 4.2, we stated that under static embeddings for a derived clause, as the embedder creates its embedding, it only updates the representations of the variables involved in it – leaving other clause embeddings intact. This might present a problem for Full-Attn where the attention grid contains all clauses including disconnected[1] pairs. An example of such a pair would be two derived clauses that do not share a variable. This could potentially lower the efficacy of the Full-Attn mechanism as it tries to match clauses that are unaware of each other. Interestingly, despite being a relaxation on Full-Attn, Anch-Attn has a distinct edge over Full-Attn under static embeddings in form of the following property:

▶ **Lemma 1.** *Clauses in the variable-anchored attention grid of Anch-Attn are guaranteed to be connected under both static and dynamic embeddings.*

**Proof.** Let $v$ be a variable in the input formula, and the set of clauses of a $v$-anchored attention grid be $A$. We show that we always have at least one clause $A_i \in A$ that reaches all other clauses in $A$ on the formula graph. We make two case distinctions:

**Case 1:** All clauses in $A$ are input clauses (in the original formula). Here, the lemma follows trivially since all these clause were connected during the input-phase message-passing protocol as they share at least one variable $v$.

**Case 2:** $A$ contains derived clauses. Let $A_i$ be the most recently derived clause in $A$. Since $A_i$ shares variable $v$ with all other clauses in $A$, then $A_i$ would be connected to them all during the derivation-phase message-passing protocol immediately after $A_i$ was derived. This is because $A_i$ receives a message from $V$ (under both static and dynamic embeddings) containing information about all other clauses containing $v$, which is precisely $A \setminus \{A_i\}$. Therefore, the lemma holds. ◀

[1] We use the terms ***connected*** and ***disconnected*** here to refer to the fact of whether two nodes have exchanged messages (in either direction) or not, respectively.

## C Teacher Proof Reduction

One rather interesting observation on Table 5 is that the model appears to be marginally better at producing shorter proofs for unseen (test) formulas than for training formulas. While we would normally expect the opposite, a fair speculation would be that the trained model was teacher-forced to match teacher proofs during training over multiple epochs while the same does not hold for unseen formulas where the bias towards teacher behavior is significantly lower. Definitively confirming this would require a more in-depth investigation.

**Table 5** Teacher proof reduction statistics of non-bootstrapped model trained on unreduced $\mathbf{SR}(U(10, 40))$ dataset. Note that all rows, except for Total Reduction, are computed over the *reduced portion* of the dataset, i.e., the proofs that were successfully shortened by NeuRes.

| (%) | Train | Test |
|---|---|---|
| Proofs Reduced | 17.82 | 18.29 |
| Max. Reduction | 86.11 | 76.4 |
| Avg. Reduction | 23.55 | 23.65 |
| Total Reduction | 3.07 | 3.15 |

## D Runtime Comparison with Traditional Solver

In the following table, we compare the average runtimes of our top-1 Full-Attn, top-3 Full-Attn, and the traditional solver we used as a teacher (BooleForce) on our main $\mathbf{SR}(40)$ test dataset. For both Full-Attn models, we use our Python prototype implementation; for BooleForce, we use an official C implementation.

**Table 6** Average time (ms) to solve an instance by neural model vs. teacher solver.

| Solver | SAT (ms) | UNSAT (ms) | Total (ms) |
|---|---|---|---|
| Full-Attn Top-1 | 2.3 | 88 | 45.15 |
| Full-Attn Top-3 | 3 | 54.4 | 28.7 |
| BooleForce | 4 | 5 | 4.5 |

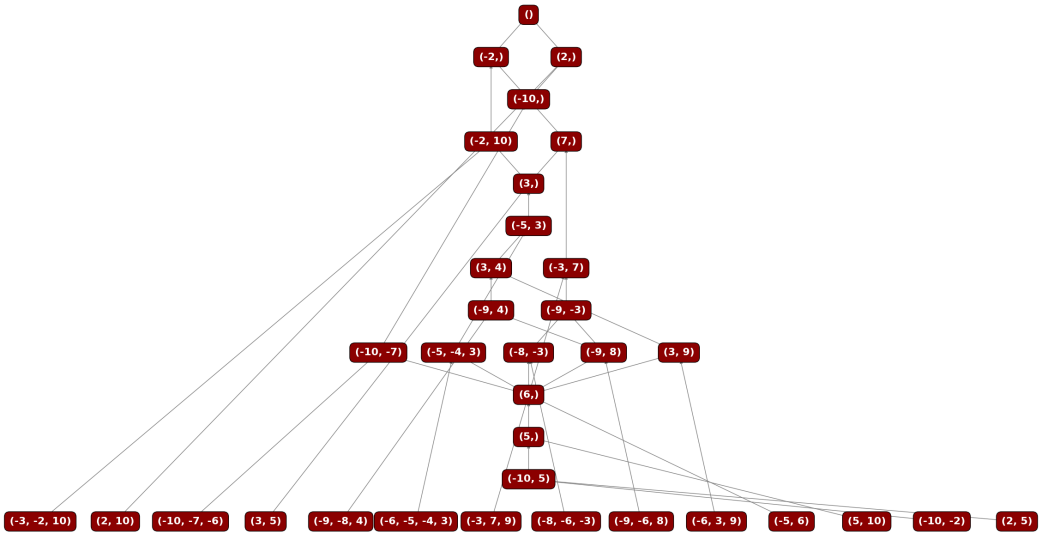
## E Model Size Comparison with NeuroSAT

In terms of the model architecture, both NeuRes and NeuroSAT models can be broken down to:

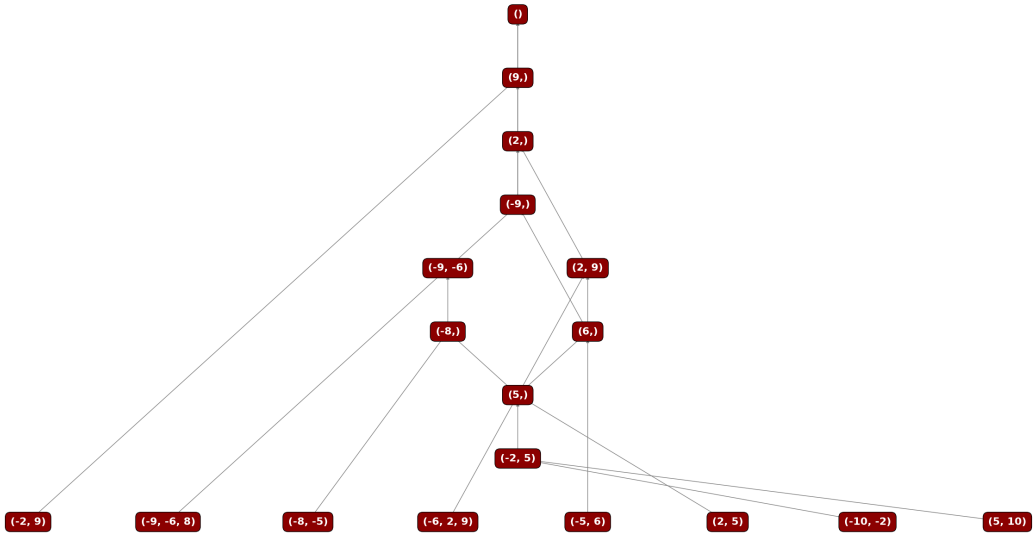
- **Embedding/Representation Network:** for both models, this network is an LSTM-based GNN that embeds the formula graph by message-passing. We use the exact same architecture and model size to ensure that our improved representations are a result of our fully certificate-based learning objective as opposed to a tweak in the model architecture. This GNN has $429,824$ parameters in total.
- **Downstream Networks:** NeuroSAT: uses a 3-layer MLP applied on the literal embeddings (width $= 128$) to extract the literal votes to predict if the formula is satisfiable or

not. This MLP has $128 \times 128 \times 3 = 49,152$ parameters. NeuRes (Full-Attn): uses an attention module to select clause pairs. This attention network is composed of two 1-layer MLPs for the query and key transformations on the clauses embeddings (width = 128). The whole attention module has $128 \times 128 \times 2 = 32,768$ parameters. To decode the variable assignments, NeuRes uses a 2-layer scalar MLP with $128 \times 128 + 128 = 16,512$ parameters

Total NeuroSAT size $= 429,824 + 49,152 = 478,976$ parameters
Total NeuRes size $= 429,824 + 49,280 = 479,104$ parameters
All in all, NeuRes only learns 128 more parameters.




**(a)** Teacher Proof




**(b)** NeuRes Proof

**Figure 7** Teacher Proof Reduction Example