NeuroSelect: Learning to Select Clauses in SAT Solvers*

Hongduo Liu
CUHK

Peng Xu
CUHK

Yuan Pu
CUHK

Lihao Yin
Huawei Noah's Ark Lab

Hui-Ling Zhen
Huawei Noah's Ark Lab

Mingxuan Yuan
Huawei Noah's Ark Lab

Tsung-Yi Ho
CUHK

Bei Yu
CUHK

Abstract

Modern SAT solvers depend on conflict-driven clause learning to avoid recurring conflicts. Deleting less valuable learned clauses is a crucial component of modern SAT solvers to ensure efficiency. However, a single clause deletion policy cannot guarantee optimal performance on all SAT instances. This paper introduces a new clause deletion metric to diversify existing clause deletion policies. Then, we propose to use machine learning to evaluate and select clause deletion policies adaptively based on the input instance. We show that our method can reduce the runtime of the state-of-the-art SAT solver Kissat by 5.8% on large industry benchmarks.

1 Introduction

The Boolean satisfiability (SAT) problem consists of determining a satisfying variable assignment for a Boolean function or proving that no such assignment exists. It has wide applications in various fields like circuit verification, test pattern generation, and automatic theorem proving. SAT is the first problem proven NP-complete [1], which means it is at least as hard as any other problem in the class NP.

Clause deletion, a critical technique in modern SAT solvers, manages memory and computational resources by removing redundant or less useful learned clauses. While effective, traditional heuristics for clause deletion often struggle with the variability inherent in different SAT instances. Therefore, machine learning (ML) presents a promising avenue for enhancing adaptability by automating the selection of clauses based on their characteristics.

However, evaluating the usefulness of learned clauses using ML models, as illustrated in Figure 1(a), faces significant challenges: **(1)** ML models excel at identifying patterns in static data. However, the state of a SAT solver is constantly changing as it navigates the solution space, adding and deleting clauses. **(2)** The usefulness of a learned clause is not only decided by itself but also determined by other selected clauses, adding additional complexity to the decision-making process. **(3)** Evaluating clauses directly requires model inferences for each clause in every clause deletion phase, which requires

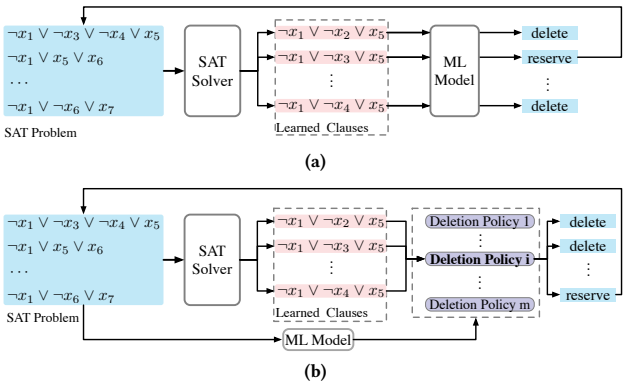


Figure 1: Two learning aided clause deletion mechanisms. (a) Evaluate learned clauses directly; (b) Evaluate clause deletion policies.

acceleration of GPUs. The computational demand is even more significant than SAT solving.

In an attempt to tackle these challenges, Vaezipoor *et al.* [2] proposed a method for training clause deletion heuristics using reinforcement learning. The trained policy outputs an integer literal block instance (LBD) threshold and deletes all clauses with LBD values above the threshold. However, this preliminary study primarily focuses on conceptual development, lacking comprehensive results on real SAT instances.

Based on the above observations, our paper shifts the focus from individual clause evaluation to assessing the overall quality of clause deletion policies using ML, as shown in Figure 1(b). We propose an ML model that dynamically selects the most suitable clause deletion policy based on the characteristics of the input SAT problem.

This approach acknowledges that the effectiveness of clause deletion depends not only on the characteristics of individual clauses but also on the overarching strategy guiding their elimination. It allows a better understanding of the cumulative impact of removing multiple clauses over time, offering a more comprehensive view of their long-term effectiveness. Moreover, evaluating the deletion policy only requires a one-time inference of the ML model before SAT solving, which can be efficient even on CPUs.

The initial step of our method involves the generation of complementary clause deletion policies. Generally, modern SAT solvers use a single clause deletion policy that considers factors such as clause activity, size, and glue value. Although it works well for many cases, it's not universally optimal for all SAT instances. We note that certain variables are more involved in Boolean constraint propagation, making clauses containing them more efficient at narrowing the search space. By integrating this propagation involvement metric into clause deletion policies, we can significantly reduce solving

*This work is partially supported by The Research Grants Council of Hong Kong SAR (No. CUHK14210723).

DAC '24, June 23–27, 2024, San Francisco, CA, USA

ACM ISBN 979-8-4007-0601-1/24/06…$15.00
https://doi.org/10.1145/3649329.3656250

time for certain SAT instances. The new policy, guided by variable propagation frequency, enhances existing strategies, offering our ML model a broader spectrum of options.

The second step focuses on selecting the most effective clause deletion policy. This involves leveraging the advantages of various policies while mitigating their drawbacks. However, classifying large-scale industrial SAT instances containing thousands of variables and clauses presents considerable challenges for conventional methods. Previous message-passing neural networks can only capture local formula information, failing to model long-term variable interactions during solving. To address this, we propose a global attention network that captures implicit inter-dependencies between variables not embodied in the original clauses. Transformers enable global attention but often scale quadratically in time and space complexity. Therefore, our model utilizes linear attention, substantially reducing complexity and facilitating efficient training and inference. The global view and reduced complexity from linear attention empower our model to classify industrial SAT instances effectively.

In conclusion, our main contributions are as follows.

- We propose a new clause deletion criterion complementing existing clause deletion metrics.
- We present a learning-aided clause deletion algorithm capable of adaptively selecting the optimal clause deletion policy for any given SAT instance. This method requires only a one-time inference, making it efficient even on CPUs.
- To classify large industrial SAT instances, we propose a graph transformer network with linear attention to capturing both local and global information of SAT problems.
- Compared to the leading SAT solver Kissat, our approach reduces average runtime by 5.8% for industrial cases.

2 Preliminaries

SAT Problem. The satisfiability problem is a fundamental problem in computer science and addresses whether a given propositional logic formula can be satisfied by assigning truth values to its variables. A formula is considered satisfiable if an assignment exists that makes it evaluated as true and unsatisfiable otherwise. A propositional logic formula can be transformed into a conjunctive normal form (CNF) to facilitate this analysis. In CNF, the formula is expressed as a conjunction ("and") of disjunctions ("or") of literals. A literal can be either a Boolean variable, denoted as x, or its negation, denoted as $\neg x$. For instance, $(x_1 \vee x_2) \wedge (\neg x_2 \vee x_3)$ is an example of a CNF, with x_1, x_2, $\neg x_2$, and x_3 all being literals. The disjunctions $(x_1 \vee x_2)$ and $(\neg x_2 \vee x_3)$ within the CNF are also clauses. By assigning values of $x_1 = \mathit{True}$, $x_2 = \mathit{False}$, and $x_3 = \mathit{True}$, the CNF can be satisfied.

Conflict Driven Clause Learning Algorithm. The conflict-driven clause learning (CDCL) algorithm is a prominent technique in modern SAT solvers. The algorithm's flow is depicted in Figure 2. It begins by selecting a variable and assigning it a value, triggering a sequence of Boolean Constraint Propagation (BCP) and potentially more variable assignments. If a conflict arises, indicating an unsatisfiable set of assignments, the algorithm analyzes the conflict and learns a new clause to prevent the solver from repeating the same mistake. If the conflict occurs at the top level and remains unresolved, it signifies the unsatisfiability of the given problem. Otherwise, the algorithm backtracks to a point where it can resolve the conflict and

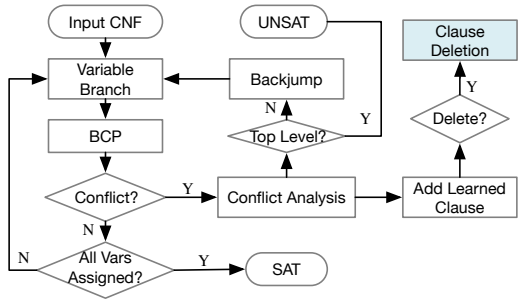


Figure 2: The flow of conflict-driven clause learning (CDCL) algorithm.

resume from there. This iterative process continues until all variables are assigned (yielding a satisfying solution) or unsatisfiability is established.

As the algorithm progresses, it accumulates a considerable number of learned clauses. The growth in the number of clauses can significantly impact memory usage and potentially slow down the solver. Clause deletion is a necessary strategy to manage resources effectively. Hence, certain learned clauses are selectively removed when either the number of learned clauses or the total number of clauses surpasses a predefined threshold. The process of clause deletion typically incorporates heuristics that determine the least useful clauses for removal. These heuristics consider factors such as the clause's activity (how frequently it has been involved in conflict analysis), its size (number of literals), and its glue value (a measure of the diversity of decision levels within the clause).

Machine Learning for SAT. Recently, machine learning has been harnessed in SAT solving, with the approaches broadly falling into two categories. The first category comprises end-to-end solvers, while the second focuses on enhancing modern SAT solvers' performance by incorporating neural network predictions. End-to-end solvers [3, 4] directly find a satisfiable solution or predict unsatisfiability through neural networks. However, they exhibit a lack of completeness, which means they cannot provide proof of unsatisfiability. Moreover, their effectiveness has primarily been demonstrated on small SAT instances encompassing up to hundreds of variables, failing to show any possibilities to surpass model SAT solvers. In contrast, learning-aided solvers incorporate neural networks to refine search heuristics in modern SAT solvers, and the completeness of the solver will not be influenced. Notable improvements have been achieved in various aspects, including branch heuristics [5, 6], prediction of satisfying assignments [7], glue variable prediction [8], and restart policies [9].

GNN and Graph Transformers. Graph Neural Networks (GNNs) are specialized neural networks for analyzing data represented as graphs [10, 11]. These graphs are expressed as $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, with $\mathcal{V}$ and $\mathcal{E}$ symbolizing nodes and edges, respectively. GNNs aim to learn functions mapping nodes to vectors, which is useful for tasks like graph classification and link prediction. The fundamental concept in GNNs involves updating a node's representation using its neighbors' information. Specifically, for any node $v \in \mathcal{V}$, the message passing at layer ℓ in a GNN is given by:

$$\mathbf{h}_v^{\ell} = \text{UPDATE}\left(\mathbf{h}_v^{\ell-1}, \text{AGG}\left(\left\{\mathbf{m}_{uv}^{\ell} : u \in \mathcal{N}(v)\right\}\right)\right), \tag{1}$$

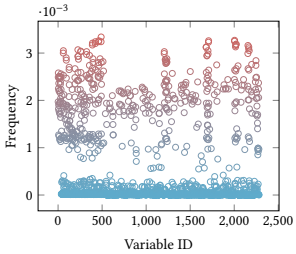


Figure 3: Distribution of propagation frequency.

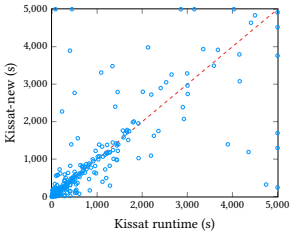


Figure 4: Default vs. new clause deletion policy.

where $\mathcal{N}(v)$ represents nodes connected to v in the graph $\mathcal{G}$. $\mathbf{m}_{uv}^{\ell}$ is the message passed from the node u to node v. The function $\texttt{AGG}(\cdot)$ aggregates messages from all neighboring nodes into a single vector, and $\texttt{UPDATE}(\cdot)$ modifies $\mathbf{h}_v^{\ell-1}$ to $\mathbf{h}_v^{\ell}$ using this vector. This process of iteratively propagating information enables GNNs to effectively capture the graph's relational structure.

Beyond the traditional local neighborhood message passing, there's increasing interest in using Transformers for graph encoding [12, 13]. These models employ global attention, aggregating embeddings from all nodes to update each node's representation. This global attention can be seen as extending GNN message passing to a fully connected graph, where every node is considered a neighbor to every other (i.e., $\forall v \in \mathcal{V}, \mathcal{N}(v) = \mathcal{V}$). By incorporating global attention, these models enhance their ability to understand long-range interactions between nodes, as well as potential links within the graph. As its ability to detect implicit dependencies, graph transformers have been applied to learning-based SAT solvers [14, 15] to enhance the performance.

3 A New Clause Deletion Metric

3.1 Motivation

As depicted in Figure 2, each time a variable is assigned a value, Boolean constraint propagation (BCP), also known as unit propagation, is triggered. During BCP, identification of a unit clause (a clause with only one unassigned literal) leads to setting the truth value of that literal to satisfy the clause. This assignment can spark further propagations, as it may result in the emergence of additional unit clauses. This iterative process continues until no more unit clauses are available or a conflict is encountered. Learned clauses containing the propagated variable are crucial at each step for deducing new assignments or learned clauses, thereby rendering them useful.

Figure 3 presents the frequency of variable propagation when solving a SAT instance from SAT competition 2022 using the Kissat solver [16]. We observe that some variables are propagated significantly more frequently than others, indicating that the learned clauses containing these variables are referred to more often. This implies their greater involvement in deriving new assignments or conflicts.

Current clause deletion policies rank clauses based on conflict analysis participation, clause size, and glue value. Our approach proposes a deletion criterion based on a clause's propagation frequency, hypothesizing that clauses containing frequently propagated variables are valuable in narrowing the search space. We suggest tracking each variable's propagation times since the last deletion to determine clause importance.

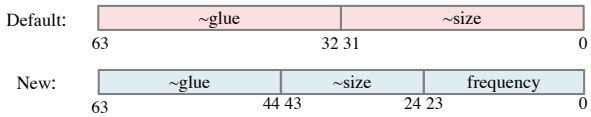


Figure 5: The default clause scoring algorithm in Kissat and our new clause scoring algorithm considering variable propagation frequency.

3.2 Injection into Kissat

To evaluate the effectiveness of propagation frequency in guiding clause deletion, we integrated the new criterion into the state-of-the-art SAT solver Kissat [16].

Clause Deletion in Kissat.. In Kissat, learned clauses are classified as either non-reducible or reducible. Non-reducible clauses, distinguished by their low glue values, are consistently retained within the solver. Reducible clauses, in contrast, undergo periodic deletion. This deletion process employs a scoring mechanism that favors the elimination of clauses with lower scores. The scoring is primarily decided by the glue value of a clause, with its size serving as a secondary criterion. Therefore, the one with a lower glue value is assigned a higher score among the two learned clauses. If the glue values are identical, the clause with the smaller size receives the higher score.

New Clause Deletion Policy.. Our new clause deletion policy incorporates variable propagation frequency as an additional metric. For a given clause `c`, we denote this new criterion as `c.frequency`, calculated using the equation:

$$\text{c.frequency} = \sum_{v \in c} (f_v > \alpha f_{\max}). \tag{2}$$

Here, f_v indicates the frequency of variable v used to trigger propagation since the last clause deletion, and $f_{\max}$ represents the maximum propagation frequency among all variables. The factor α is an adjustable parameter set to 4/5, according to our empirical studies. Figure 5 shows the distinctions between the original and our new clause scoring algorithms, illustrating how various metrics form a 64-bit score for a learned clause. The symbol ~ represents element-wise negation, emphasizing that lower values translate to higher scores.

Performance Comparisons.. Figure 4 compares runtimes on SAT competition 2022 instances using a standard 5,000s timeout, contrasting the default clause deletion policy in Kissat and our new clause deletion policy (Kissat-new). Instances that remained unsolved by both policies are not included. Each dot represents an instance, with its position reflecting the solution time under the respective policies: the x-coordinate for the default clause deletion policy and the y-coordinate for our new policy. Dots on the 5,000-second boundary signify timeouts under either policy. Instances falling below the dotted diagonal line indicate where our new policy outperforms the default policy used in Kissat. Conversely, instances above this line show where our policy is less effective. These results suggest the need to adaptively select the most suitable clause deletion policy based on the specific instance.

4 Learning-aided Deletion Policy Selection

4.1 Overall Flow

Figure 6 illustrates the NeuroSelect algorithm, our learning-aided approach to select clause deletion policies. The algorithm begins

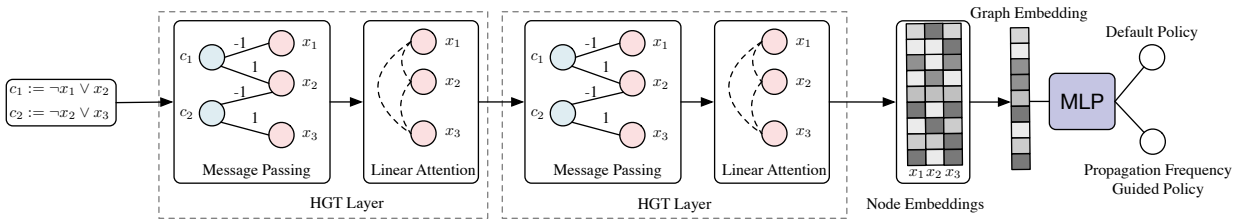


Figure 6: Overview of NeuroSelect.

with transforming the input CNF formula into a compact graph representation. This graph is then processed through a series of tailored neural network modules, termed the Hybrid Graph Transformer (HGT) layer, which is responsible for generating node embeddings. These embeddings are subsequently aggregated to form a graph embedding, encapsulating the information of the SAT instance. The final step involves employing a multi-layer perceptron (MLP) to produce a prediction. This prediction determines the applicability of either the default clause deletion policy or the propagation frequency-guided policy.

The HGT layer uniquely integrates a message-passing network with a linear attention mechanism. The message-passing aspect effectively captures the local connectivity and interactions between variables and clauses, which is crucial for understanding the structural information of the CNF formula. GNNs are known for their proficiency in capturing immediate, local relationships within graph structures. However, the challenge arises in representing potential dependencies between variables, a common occurrence in Boolean constraint propagation. For instance, assigning a truth value to one variable in a clause can significantly impact the truth values of variables in remotely connected clauses due to the intricately interconnected nature of the constraints. This complexity is further heightened with the introduction of learned clauses. While adding deeper layers can enhance information aggregation, it may lead to the over-smoothing problem, where node features become increasingly homogenized with those of their neighbors, losing the distinctiveness of the network.

To address these challenges, our model incorporates a global attention mechanism to account for long-term interactions between variables, enabling the consideration of relationships between any two variables, regardless of their distance. This global attention mechanism adeptly models interactions between distantly located variables, a task that traditional graph-based models typically struggle with. However, conventional self-attention mechanisms introduce quadratic complexity, which is impractical for graphs derived from real SAT instances that may contain millions of variables. To circumvent this, we employ a linear attention scheme, offering linear complexity relative to the number of nodes. This combination of a local message-passing network and a global linear attention mechanism empowers the NeuroSelect model to effectively comprehend both local structures (through message passing) and global variable interactions (via global attention). Furthermore, its reduced computational complexity allows the model to scale to very large SAT instances.

4.2 Graph Representations of CNFs

In our work, we adopt an undirected bipartite graph representation for SAT instances as proposed in NeuroComb [17], which offers a more compact format compared to the one used in NeuroSAT [4]. This representation employs two distinct node types to signify variables and clauses. Specifically, we denote a SAT instance by a bipartite graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{W})$, where $\mathcal{V}$ is the set of vertices divided into two disjoint subsets $\mathcal{V}_1$ and $\mathcal{V}_2$. Here $\mathcal{V}_1$ represents variables and $\mathcal{V}_2$ represents clauses. The edge set $\mathcal{E} \subseteq \mathcal{V}_1 \times \mathcal{V}_2$ comprises edges that link a vertex in $\mathcal{V}_1$ with a vertex in $\mathcal{V}_2$. Every edge $e \in \mathcal{E}$ has a corresponding weight $w(e)$ in the edge weight set $\mathcal{W}$.

In the graph representation, if a variable x_i from $\mathcal{V}_1$ appears in a clause c_j from $\mathcal{V}_2$, the edge weight $w(x_i, c_j)$ is assigned a value 1 if $x_i \in c_j$ and -1 if $\neg x_i \in c_j$. The initial embedding for a node in $\mathcal{V}_1$ is set to 1, and for a node in $\mathcal{V}_2$, it is set to 0.

4.3 Hybrid Graph Transformer of NeuroSelect

We combine local message passing and global attention in the HGT layer to model interactions between variables. The proposed hybrid graph transformer layer empowers our NeuroSelect model to comprehend both local structures and global variable interactions. We also manage computational complexity using a linear attention scheme, resulting in efficient scalability to large SAT instances.

HGT Layer. Consider the input node features $\mathbf{X}^{\ell}(\mathcal{V})$ of the ℓ-th HGT layer, represented in $\mathbb{R}^{|\mathcal{V}| \times d}$, where d denotes the dimensionality of these features. Initially, these node features are processed by a message-passing neural network (MPNN), which encodes the structural information inherent in the original CNF formula:

$$(\mathbf{X}_M^{\ell+1}(\mathcal{V}_1), \mathbf{X}_M^{\ell+1}(\mathcal{V}_2)) = \text{MPNN}^{\ell}\left(\mathbf{X}^{\ell}(\mathcal{V}), \mathcal{E}, \mathcal{W}\right). \tag{3}$$

In this context, $\mathbf{X}_M^{\ell+1}(\mathcal{V}_1)$ and $\mathbf{X}_M^{\ell+1}(\mathcal{V}_2)$ represent the MPNN's output variable and clause features, respectively. Subsequently, a linear attention (LinearAttn) layer is employed to capture the long-range relationships among variables:

$$\mathbf{X}^{\ell+1}(\mathcal{V}_1) = \text{LinearAttn}^{\ell}\left(\mathbf{X}_M^{\ell+1}(V_1)\right). \tag{4}$$

The attention mechanism is applied solely to variable nodes for two reasons. Primarily, the complete embedding of the CNF is derived from variable node features alone, with clause nodes mainly facilitating message passing between variables. Moreover, as the number of clauses in a CNF typically surpasses that of variables, focusing attention on variable nodes effectively reduces computational and memory demands. Finally, the new variable node features, combined with clause node features derived from the MPNN, constitute the input for the subsequent HGT layer:

$$\mathbf{X}^{\ell+1}(\mathcal{V}) = (\mathbf{X}^{\ell+1}(\mathcal{V}_1), \mathbf{X}_M^{\ell+1}(\mathcal{V}_2)). \tag{5}$$

MPNN. As shown in Equation (1), every message-passing layer includes two operations: aggregation and update. For a specific layer ℓ, the aggregation function is implemented by

$$\mathbf{m}_{uv}^{\ell} = \frac{1}{|\mathcal{N}(v)|} \sum_{u \in \mathcal{N}(v)} w_{uv} \cdot \text{MLP}(\mathbf{h}_u^{\ell-1}), \tag{6}$$

where $\mathbf{m}_{uv}^{\ell}$ is the message ready to pass from u to v and w_{uv} is the weight between nodes u and v. MLP is a single linear layer. The update function is realized by

$$\mathbf{h}_v^{\ell} = \sigma(\text{MLP}(\mathbf{m}_{uv}^{\ell} + \text{MLP}(\mathbf{h}_v^{\ell-1}))), \tag{7}$$

where σ is an activate function (e.g. ReLU).

Linear Attention Layer. We incorporate the global interactions between variables in our model by using an all-pair attention unit, which calculates the influence between any two variable nodes in the graph. Assume the input node embedding of the linear attention layer is denoted by $\mathbf{Z} \in \mathbb{R}^{N \times d}$. The linear attention function [18] is defined as

$$\begin{aligned} \mathbf{Q} &= f_Q(\mathbf{Z}), \qquad \tilde{\mathbf{Q}} = \frac{\mathbf{Q}}{\|\mathbf{Q}\|_F}, \quad \mathbf{V} = f_V(\mathbf{Z}), \\ \mathbf{K} &= f_K(\mathbf{Z}), \qquad \tilde{\mathbf{K}} = \frac{\mathbf{K}}{\|\mathbf{K}\|_F}, \quad \mathbf{D} = \text{diag}\left(\mathbf{1} + \frac{1}{N}\tilde{\mathbf{Q}}\left(\tilde{\mathbf{K}}^{\top}\mathbf{1}\right)\right), \end{aligned} \tag{8}$$

where $f_Q, f_K, and f_V$ are linear feed-forward layers to encode the query, key, and value matrix. $\|\cdot\|_F$ denotes the Frobenius norm and $\mathbf{1}$ is an N-dimensional all-one column vector. The `diag` operation changes the N-dimensional column vector into a $N \times N$ diagonal matrix. Subsequently, we have the output of the global attention layer in the format of

$$\mathbf{Z}^{out} = \text{LinearAttn}(\mathbf{Z}) = \mathbf{D}^{-1}\left[\mathbf{V} + \frac{1}{N}\tilde{\mathbf{Q}}\left(\tilde{\mathbf{K}}^{\top}\mathbf{V}\right)\right]. \tag{9}$$

Complexity Analysis. The computational complexity of an MPNN layer within an HGT layer is $\mathcal{O}(|\mathcal{E}|)$, where message passing occurs exclusively between connected nodes by an edge. Here, $|\mathcal{E}|$ represents the total number of edges in the transformed graph. Importantly, as linear attention is constrained to variable nodes, it necessitates $\mathcal{O}(|\mathcal{V}_1|)$ operations, where $|\mathcal{V}_1|$ denotes the number of variables in the CNF formula. Consequently, the overall complexity of the process can be expressed as $\mathcal{O}(|\mathcal{E}| + |\mathcal{V}_1|)$. Due to the inherent sparsity in the variable-clause connections, the model can scale linearly with the CNF's size, which significantly enhances the efficiency of both training and inference.

4.4 Mapping Function

Assume the NeuroSelect model comprises L HGT layers. In this model, the graph representation $h_{\mathcal{G}}$ is derived through a global mapping function, `READOUT`$(\cdot)$, which operates on the embeddings of variable nodes. This process is mathematically given by

$$\mathbf{h}_{\mathcal{G}} = \text{READOUT}\left(\left\{\mathbf{h}_v^L : v \in \mathcal{V}_1\right\}\right). \tag{10}$$

Afterward, the computed graph representation $\mathbf{h}_{\mathcal{G}}$ is input into an MLP. This MLP calculates the probability $\hat{y} \in \{0, 1\}$, determining the selection between two clause deletion policies.

For training the model, we employ a binary cross-entropy loss function. This loss function, essential for optimizing the model's parameters, is defined as follows:

$$\mathcal{L} = -(y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})), \tag{11}$$

where y represents the ground truth label of the CNF formula.

Table 1: Statistics of the Training and Test Datasets

Data Type	Year	# CNFs	# Variables	# Clauses
Training	2016	74	16,649	86,186
	2017	124	12,863	99,896
	2018	148	13,407	93,094
	2019	131	12,237	68,900
	2020	123	16,921	85,808
	2021	136	16,219	97,434
Test	2022	144	19,807	104,472

5 Experimental Results

5.1 Dataset

Table 1 presents the dataset statistics used for training and evaluating our NeuroSelect Model. The training dataset, comprising 736 CNF instances, is sourced from the main track of SAT competitions held between 2016 and 2021. Due to the typically large scale of CNF formulas in these competitions, often featuring millions of variables and clauses, we have applied a filtering criterion: any formula whose graph conversion exceeds 400,000 nodes is excluded to adhere to GPU memory limitations. These instances are labeled using two clause deletion policies: Kissat's default clause deletion policy and our proposed propagation frequency-driven approach, and each is applied to the same SAT instance. Our primary aim is to determine the clause deletion policy that most effectively reduces runtime. However, due to the variability of CPU time, we focus on the total number of propagations required to solve the SAT problem, a more reliable and deterministic measure. A SAT instance is labeled as '1' if it sees at least a 2% reduction in propagations with the new deletion policy compared to the default policy in Kissat; otherwise, it is labeled as '0'. The test dataset includes SAT formulas from the 2022 main track of the SAT Competition. Following the same criteria as the training dataset, we exclude large CNF formulas with graph conversions exceeding 400,000 nodes.

5.2 Implementation Details

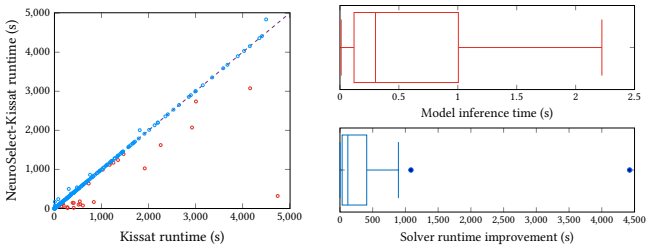
Our NeuroSelect model integrated three message-passing layers within a single HGT layer and incorporated two such layers. The model features a hidden dimension set to 32. For training, network parameter optimization was done using the Adam optimizer, with a learning rate set to 10^{-4}. We trained the model for 400 epochs with a batch size of 1. The version of Kissat is 3.1.0 in our evaluation. The model inference is done on the CPU when using NeuroSelect to guide the clause deletion policy selection.

5.3 Effectiveness of NeuroSelect Model

To evaluate the performance of our NeuroSelect model on classifying CNF formulas, we initially compared it against existing SAT instance classification networks. The first work is NeuroSAT [4], which encodes CNF formulas using a literal-clause graph and relies on multi-layer perceptrons and LSTMs for message passing. Additionally, we considered G4SATBench [19], incorporating various SAT benchmarks and GNN models for different prediction tasks. Our evaluation employed the Graph Isomorphism Network (GIN) [11],

Table 2: Performance of different SAT classification models.

	precison	recall	F1	accuracy
NeuroSAT [4]	47.27%	44.07%	45.61%	56.94%
G4SATBench [19]	43.48%	33.90%	38.10%	54.86%
NeuroSelect w/o attention	56.45%	**58.33%**	57.38%	63.89%
NeuroSelect	**66.00%**	55.00%	**60.50%**	**69.44%**



(a) (b)

Figure 7: Performance of NeuroSelect-Kissat. (a) Comparisons with Kissat on SAT competition 2022 instances; (b) NeuroSelect model inference time and runtime improvement over Kissat.

Table 3: Runtime statistics of Kissat and NeuroSelect-Kissat on SAT competition 2022 instances.

	solved	median (s)	average (s)
Kissat [16]	274	307.02	713.28
NeuroSelect-Kissat	**274**	**271.34**	**671.73**

utilizing a variable-clause graph representation for CNF formulas. As G4SATBench already includes an implementation of NeuroSAT, we use this existing implementation in our assessment. As detailed in Table 2, our NeuroSelect model demonstrates a considerable performance advantage over both NeuroSAT and G4SATBench.

To further validate the efficacy of the attention block in NeuroSelect, we conducted a comparative analysis of the model with and without this feature. The findings reveal that including the attention block leads to more than 5% improvement in accuracy. This enhancement is attributed to the model's improved global view of the CNF formula, highlighting the importance of the global attention mechanism in our architecture.

5.4 Effectiveness of Adaptive Policy Selection

To verify the effectiveness of NeuroSelect in enhancing solver performance, we integrated it into Kissat [16], creating NeuroSelect-Kissat. The inference of the NeuroSelect model was made sequentially, while the SAT solving was tested in parallel on 20 processes, each aiming to solve a SAT instance within a 5,000-second timeout. From the 400 instances in the main track of the 2022 SAT competition, NeuroSelect guided the clause deletion policy for 144 instances with graph conversions below 400,000 nodes. The remaining instances used the default policy. NeuroSelect-Kissat's runtime includes both model inference and SAT-solving durations. Figure 7(a) illustrates the comparative runtimes of NeuroSelect-Kissat and Kissat across the 400 SAT competition instances. Instances positioned below the diagonal line indicate a reduced runtime with NeuroSelect-Kissat. Although there are instances above the diagonal caused by wrong reductions of NeuroSelect, they are fewer and relatively closer to the diagonal line. More detailed statistics regarding Kissat and NeuroSelect-Kissat performance are listed in Table 3. Both NeuroSelect-Kissat and Kissat solved 274 instances within the time limit. However, NeuroSelet-Kissat has a shorter median runtime, leading to a 5.8% improvement. This suggests that NeuroSelect-Kissat successfully profits from the benefits of both clause deletion policies.

Additionally, Figure 7(b) presents box and whisker plots illustrating NeuroSelect's inference times and the runtime enhancements achieved over Kissat. Once trained, NeuroSelect requires only a single call to determine the optimal clause deletion policy, which can be executed on a CPU. Inference times vary between 0.01 and 2.22 seconds, while runtime improvements reach up to 4425 seconds. This demonstrates that NeuroSelect significantly enhances Kissat's performance with a negligible inference cost.

6 Conclusion

This work advances SAT solving by introducing a new clause deletion criterion and a learning-aided clause selection algorithm. We also present a tailored network with a linear transformer to classify large SAT instances, outperforming previous GNN approaches. Our techniques yield more than 5% reduction in runtime compared to Kissat on the SAT Competition 2022 benchmark.

References

[1] S. COOK, "The complexity of theorem proving procedure," in *Proc. STOC*, 1971.
[2] P. Vaezipoor, G. Lederman, Y. Wu, R. Grosse, and F. Bacchus, "Learning clause deletion heuristics with reinforcement learning," in *AITP*, 2020.
[3] B. Bünz and M. Lamm, "Graph neural networks and boolean satisfiability," *arXiv preprint arXiv:1702.03592*, 2017.
[4] D. Selsam, M. Lamm, B. Benedikt, P. Liang, L. de Moura, D. L. Dill *et al.*, "Learning a SAT solver from single-bit supervision," in *Proc. ICLR*, 2018.
[5] D. Selsam and N. Bjørner, "Guiding high-performance SAT solvers with unsat-core predictions," in *Proc. SAT*, 2019.
[6] V. Kurin, S. Godil, S. Whiteson, and B. Catanzaro, "Can Q-learning with graph networks learn a generalizable branching heuristic for a sat solver?" *Proc. NIPS*, vol. 33, pp. 9608–9621, 2020.
[7] W. Zhang, Z. Sun, Q. Zhu, G. Li, S. Cai, Y. Xiong, and L. Zhang, "Nlocalsat: Boosting local search with solution prediction," *arXiv preprint arXiv:2001.09398*, 2020.
[8] J. M. Han, "Enhancing sat solvers with glue variable predictions," *arXiv preprint arXiv:2007.02559*, 2020.
[9] J. H. Liang, C. Oh, M. Mathew, C. Thomas, C. Li, and V. Ganesh, "Machine learning-based restart policy for CDCL SAT solvers," in *Proc. SAT*, 2018.
[10] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *IEEE Transactions on Neural Networks (TNN)*, vol. 20, no. 1, pp. 61–80, 2008.
[11] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" in *Proc. ICLR*, 2018.
[12] V. P. Dwivedi and X. Bresson, "A generalization of transformer networks to graphs," *arXiv preprint arXiv:2012.09699*, 2020.
[13] D. Kreuzer, D. Beaini, W. Hamilton, V. Létourneau, and P. Tossou, "Rethinking graph transformers with spectral attention," *Proc. NIPS*, vol. 34, pp. 21 618–21 629, 2021.
[14] F. Shi, C. Lee, M. K. Bashar, N. Shukla, S.-C. Zhu, and V. Narayanan, "Transformer-based machine learning for fast sat solvers and logic synthesis," *arXiv preprint arXiv:2107.07116*, 2021.
[15] Z. Shi, M. Li, Y. Liu, S. Khan, J. Huang, H. Zhen, M. Yuan, and Q. Xu, "Satformer: Transformer-based UNSAT core learning," in *Proc. ICCAD*, 2023.
[16] A. Biere and M. Fleury, "Gimsatul, IsaSAT and Kissat entering the SAT Competition 2022," in *SAT Competition*, 2022.
[17] W. Wang, Y. Hu, M. Tiwari, S. Khurshid, K. McMillan, and R. Miikkulainen, "Neurocomb: Improving SAT solving with graph neural networks," *arXiv preprint arXiv:2110.14053*, 2021.
[18] Q. Wu, W. Zhao, C. Yang, H. Zhang, F. Nie, H. Jiang, Y. Bian, and J. Yan, "Sgformer: Simplifying and empowering transformers for large-graph representations," in *Proc. NIPS*, 2023.
[19] Z. Li, J. Guo, and X. Si, "G4SATBench: Benchmarking and advancing sat solving with graph neural networks," *arXiv preprint arXiv:2309.16941*, 2023.