International Journal of Computer Applications (0975 - 8887)
Volume 187 - No.130, July 2026

Structured and Compact: A Novel Encoding and Enhancement Paradigm for ML-based SAT Solving

Ziqi Zhang
Dept. of Data Science and Big Data Technology
Capital University of Economics and Business
Beijing, China

Lan Zhang*
Dept. of Data Science and Big Data Technology
Capital University of Economics and Business
Beijing, China

ABSTRACT

Machine Learning for SAT (ML4SAT) offers a data-driven alternative to traditional solvers. However, existing flat-vector paradigms face scalability bottlenecks, feature redundancy, and struggle to exploit the intrinsic logical structures of clauses. To address these deficiencies, a novel encoding and enhancement paradigm is proposed, comprising: (1) a lightweight Prime-Product Clause Encoding (PPCE) scheme leveraging unique prime factorization for mathematically lossless dimensionality compression; (2) a Structure-Aware Enhancement (SAE) module incorporating a localized subnetwork (ClauseNet) to better capture intra-clause logic; and (3) a controlled benchmark evaluating four canonical models—Logistic Regression, Support Vector Machine, Multi-Layer Perceptron, and Transformer—under unified variables. Empirical results on two large-scale CNF datasets (183k and 480k samples) reveal high-dimensional semi-sparsity (S ranging from 73.81% to 75.20%). PPCE reduces feature dimensionality by 75%, while the SAE module can improve accuracy across models without inference overhead. These findings validate specialized encoding and structural modules in ML4SAT, providing practical, data-driven selection guidelines for exploring accuracy-efficiency trade-offs in industrial formal verification.

General Terms

Machine Learning, Boolean Satisfiability

Keywords

Satisfiability, Feature Engineering, Prime-Product Clause Encoding, Structure-Aware Enhancement, Logistic Regression, Support Vector Machine, Multi-Layer Perceptron, Transformer

1. INTRODUCTION

Boolean Satisfiability (SAT) [5], a core NP-complete problem, is fundamental to formal verification, circuit design [2], software validation [18], and combinatorial optimization. The canonical representation of a SAT instance is the Conjunctive Normal Form (CNF):

$$\Phi = \bigwedge_{i=1}^{m} C_i$$

where C_i is a clause and m is the number of clauses.

Traditional SAT solvers rely on logical deduction via resolution [16] and tableau [8]. However, these methods often encounter scalability and efficiency bottlenecks on complex, large-scale CNF instances [1] where hand-crafted heuristics struggle to adapt.

Machine Learning for SAT (ML4SAT) [20] addresses this by learning logical patterns directly from CNF instances, reducing search overhead. These models rely on supervised learning with labels from traditional solvers:

$$\phi(\Phi) = \begin{cases} 1, & \Phi \text{ is satisfiable} \\ 0, & \Phi \text{ is unsatisfiable} \end{cases}$$

Despite these advancements, existing ML4SAT research faces three limitations. (1) Feature encoding scalability is restricted: flattening clauses and literals into a single vector leads to redundancy [19] and fails to adapt to varying instance sizes. (2) Structural boundaries within the flattened input are underutilized, as standard layers do not preserve clause-specific logical constraints. (3) The lack of a controlled benchmark comparing classical and deep learning models prevents reliable model selection for industrial trade-offs.

To address these, this paper introduces:

—The Prime-Product Clause Encoding (PPCE) scheme for input dimensionality compression and feature redundancy mitigation.

—A Structure-Aware Enhancement (SAE) module incorporating ClauseNet to enforce clause boundaries and capture intra-clause semantics.

—A controlled benchmark evaluating four canonical models—Logistic Regression [7], SVM [6], Multi-Layer Perceptron [17], and Transformer [23]—providing data-driven selection guidelines for industrial applications.

Experiments on two large-scale CNF datasets (183k and 480k samples) verify the influence of these encoding and structural strategies. Section 2 reviews related work; Section 3 details the experimental setup; Section 4 analyzes the results.

*Corresponding author: Lan Zhang (Lan@cueb.edu.cn)

2. RELATED WORK

Integrating machine learning (ML) and formal methods has advanced SAT research. While traditional rule-based solvers struggle with complex propositional logic, data-driven ML models have achieved breakthroughs via adaptive feature learning and innovative architectures.

Following NeuroSAT [20], which utilized Graph Neural Networks (GNNs) [9] to capture structural SAT properties, research has focused on mechanism analysis and architectural improvement. Hůla et al. [11] linked GNN message-passing to the classic 2-approximate greedy algorithm [13], demonstrating that GNN decisions follow interpretable logical patterns. Shoham et al. [21] further proposed a "concept learning" framework, proving GNNs perform structured, stepwise reasoning consistent with human-designed algorithms. Furthermore, Liang and Li [15] enhanced NeuroSAT by integrating multi-hop attention, improving information interaction to better capture long-range logical dependencies.

Other neural architectures have also advanced. Chang et al. [4] combined Discrete Hopfield Neural Networks (DHNN) [10] with Weighted C-type Random 2-SAT (WCRAN2SAT) rules and the Binary Artificial Bee Colony (BABC) algorithm [12] for synaptic optimization. Simultaneously, Chang et al. [3] integrated Gated Recurrent Units (GRUs) into the Transformer [23] framework (TG-SAT), utilizing gating mechanisms to capture dynamic relationships and enhance prediction accuracy.

Large Language Models (LLMs) are shifting SAT solving from manual to automated design. Sun et al. [22] introduced AutoSAT, a multi-agent LLM system optimizing heuristic modules within expert-designed solvers. Yu et al. [24] proposed SATLUTION, an agent-based system driving iterative evolution of solver codebases within a correctness loop, achieving competitive results on unseen benchmarks.

In summary, ML4SAT has evolved from black-box GNNs toward transparent mechanism analysis and architectural innovation. Coupled with classical neural optimizations and LLM-driven automation, these advances continue to expand the technical frontiers of satisfiability solving.

3. EXPERIMENTS ON SAT PREDICTION

This section evaluates ML models on CNF formulas to analyze predictability, structural priors, and feature encodings, measuring classification accuracy and computational efficiency under a unified framework.

3.1 Datasets and Data Preprocessing

A propositional formula in Conjunctive Normal Form (CNF) is defined as a conjunction of clauses. Each CNF formula Φ in this study is standardized to consist of $m = 80$ clauses, where each clause contains strictly $n = 4$ literals associated with localized proposition slots. Let C_i denote the i-th clause ($1 \le i \le 80$), and p_j denote the j-th proposition slot within a clause ($1 \le j \le 4$). The formula is expressed as:

$$\Phi = \bigwedge_{i=1}^{80} C_i, \quad C_i = \bigvee_{j=1}^{4} l_{i,j}$$

where $l_{i,j}$ represents the literal corresponding to the j-th proposition slot in the i-th clause.

Each sample corresponds to a complete set of logical clauses C_i, covering both satisfiable (Label 1, $\Phi = 1$) and unsatisfiable (Label 0, $\Phi = 0$) scenarios. The overall class distribution is balanced to prevent empirical bias during training and evaluation.

3.1.1 Dataset Encoding Schemes. To maintain consistency with Prime-Product Clause Encoding (PPCE), the n-tuple encoding scheme is named Tuple Clause Encoding (TCE) in this work, mapping each clause to $n = 4$ consecutive feature dimensions. TCE outputs a 320-column matrix, where every four columns represent one logical clause. Alternatively, PPCE produces an 80-column matrix, with each column representing one logical clause.

3.1.1.1 Tuple Clause Encoding (TCE). In TCE, each clause C_i is represented by a feature vector corresponding to four consecutive propositions (p_1, p_2, p_3, p_4). The status value $v_{i,j}^{\text{TCE}}$ for the j-th proposition in the i-th clause is defined as:

$$v_{i,j}^{\text{TCE}} = \begin{cases} 0 & \text{The proposition is not selected} \\ 1 & \text{The proposition is selected} \\ 2 & \text{The negation of the proposition is selected} \end{cases}$$

For example, the expression $p_1 \vee p_2 \vee \neg p_4$ is encoded as the vector $(1, 1, 0, 2)$.

3.1.1.2 Prime-Product Clause Encoding (PPCE). In PPCE, each proposition in the universal set is assigned a unique prime number: p_1 corresponds to $q_1 = 2$, p_2 to $q_2 = 3$, p_3 to $q_3 = 5$, and p_4 to $q_4 = 7$. The status value $v_{i,j}^{\text{PPCE}}$ is defined as:

$$v_{i,j}^{\text{PPCE}} = \begin{cases} 1 & \text{The proposition is not selected} \\ q_j & \text{The proposition is selected} \\ q_j^2 & \text{The negation of the proposition is selected} \end{cases}$$

The encoding value v_i^{PPCE} of the i-th clause C_i is computed as:

$$v_i^{\text{PPCE}} = \begin{cases} 0, & \text{if } \forall j, v_{i,j}^{\text{PPCE}} = 1 \\ \prod_{j=1}^{4} v_{i,j}^{\text{PPCE}}, & \text{otherwise} \end{cases}$$

Thus, the expression $p_1 \vee p_2 \vee \neg p_4$ is encoded as $2 \times 3 \times 1 \times 7^2 = 294$.

Grounded in the Fundamental Theorem of Arithmetic, PPCE establishes a deterministic multiplicative mapping that injectively projects literal combinations within a clause into a single scalar space. This unique factorization guarantees that the compression is mathematically lossless and uniquely decodable, preserving the truth-assignment constraints. Inactive or empty clauses are assigned a characteristic value of 0, providing a masking effect that nullifies their structural contributions and allows downstream models to isolate active clause blocks.

3.1.2 Dataset Characteristics. A systematic sparsity analysis is conducted on both datasets prior to training. The sparsity rate S of a feature matrix is defined as:

$$S = \frac{N_{\text{zero}}}{N_{\text{total}}} \times 100\%$$

where N_{zero} is the number of zero-valued features and N_{total} is the total number of elements.

The overall sparsity of the 183k-sample dataset is $S = 75.20\%$.

The overall sparsity of the 480k-sample dataset is $S = 73.81\%$.

The empirical results confirm that both are high-dimensional, semi-sparse datasets, which justifies the following design decisions:

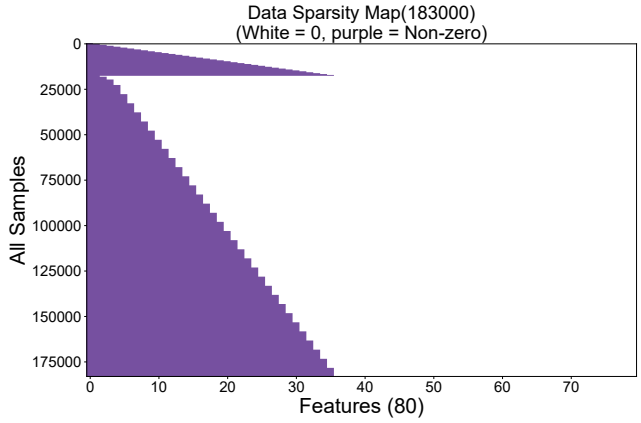

Fig. 1. Sparsity Analysis of the 183k-Sample Dataset

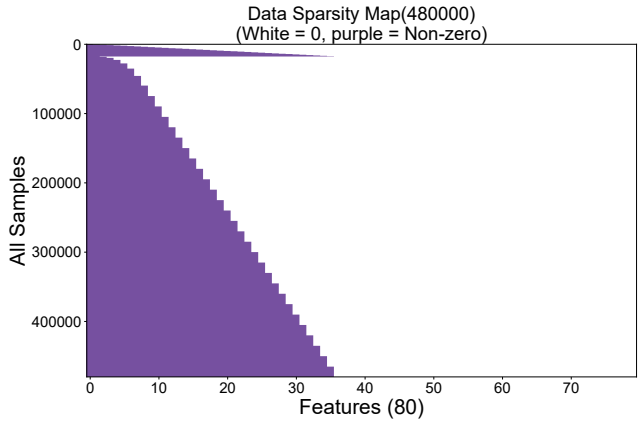

Fig. 2. Sparsity Analysis of the 480k-Sample Dataset

—**Data Standardization Strategy:** Neural models (MLPs and Transformers) require feature normalization to prevent vanishing gradients and training instability. Although classical classifiers like Support Vector Machines (SVMs) are sensitive to feature scales, applying standard scaling to them is counterproductive: it converts zero elements into continuous non-zero values, thereby destroying the native structural sparsity and increasing memory overhead.

—**Structural Enhancement Motivation:** High-dimensional semi-sparsity introduces feature redundancy and disperses effective logical signals. As a result, standard flat-vector models (Non-SAE) struggle to capture sparse assignments efficiently, motivating the introduction of the SAE module to focus on intra- and inter-clause logical relationships.

3.1.3 Data Preprocessing. The dataset is randomly split into a training set $\mathcal{D}_{\text{train}}$ and a test set $\mathcal{D}_{\text{test}}$ with a fixed ratio of 7:3:

$$|\mathcal{D}_{\text{train}}| = \lfloor 0.7N \rfloor, \quad |\mathcal{D}_{\text{test}}| = N - |\mathcal{D}_{\text{train}}|$$

A fixed random seed is maintained across configurations to ensure reproducibility and consistent class distributions.

For neural models, features are standardized via Z-score normalization. For a feature x_j, the standardized feature x'_j is calculated using the dataset mean μ_j and standard deviation σ_j:

$$x'_j = \frac{x_j - \mu_j}{\sigma_j}$$

$$\mu_j = \frac{1}{N}\sum_{k=1}^{N} x_{k,j}$$

$$\sigma_j = \sqrt{\frac{1}{N}\sum_{k=1}^{N}(x_{k,j} - \mu_j)^2}$$

3.2 Experimental Setup and Model Design

The experiment evaluates three primary variables: model type, feature encoding, and structural modeling strategy.

—**Model Types:** Four machine learning models are evaluated: Logistic Regression (LR) [7], Support Vector Machine (SVM) [6], Multi-Layer Perceptron (MLP) [17], and the Transformer.

—**Encodings:** Two configurations are compared: PPCE and TCE.

—**Modeling Strategies:** Non-SAE (treating input features as a continuous flat vector) and SAE (decoupling the input space into independent clause-level fields).

3.2.1 Model Architecture Design. Seven model architectures are evaluated: Non-SAE models (Non-SAE LR, Non-SAE SVM, Non-SAE MLP), SAE-based models (SAE-LR, SAE-SVM, SAE-MLP), and the Transformer.

Architectural configurations match their mathematical properties. Neural models (MLP, Transformer) integrate activation functions and Layer Normalization on standardized data for convergence. Conversely, classical classifiers (LR, SVM) directly ingest raw semi-sparse features to preserve native logical distributions.

3.2.1.1 Non-SAE-based Models

—Non-SAE LR: The baseline linear model mapping the 320-dimensional (TCE) or 80-dimensional (PPCE) flattened inputs directly.

—Non-SAE SVM: A linear-kernel classifier performing binary classification via Hinge Loss, optimized for high-dimensional input spaces.

—Non-SAE MLP: A five-layer fully connected network that autonomously learns high-order nonlinear features from standardized inputs.

3.2.1.2 SAE-based Models. The SAE Module requires tailored structural designs for TCE and PPCE encodings. Figures 3 to 8 detail the integration mechanisms and dimensional transformations for all SAE-based models under both schemes.

—SAE-LR: For TCE-encoded data, the 320-dimensional input is partitioned into 4-dimensional segments $\boldsymbol{x}_i \in \mathbb{R}^4$. As shown in Figure 3, each segment undergoes a localized linear-ReLU transformation:

$$\boldsymbol{h}_i = \text{ReLU}\left(\boldsymbol{W}\boldsymbol{x}_i + \boldsymbol{b}\right)$$

where $\boldsymbol{W} \in \mathbb{R}^{4\times4}$ and $\boldsymbol{b} \in \mathbb{R}^4$. The outputs $\boldsymbol{h}_i$ are concatenated to 320 dimensions for the base LR model, capturing intra-clause relationships.

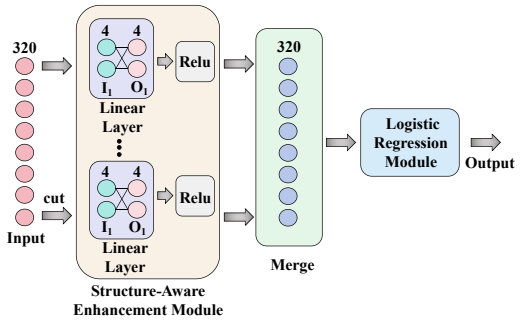

Fig. 3. SAE-LR for TCE-encoded Data

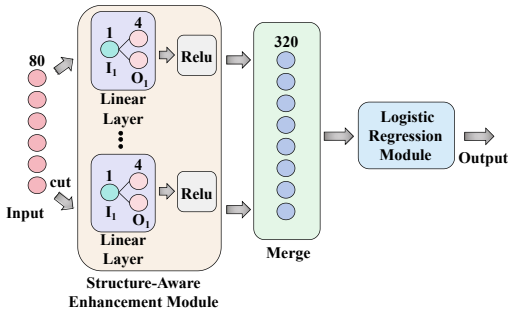

Fig. 4. SAE-LR for PPCE-encoded Data

For PPCE-encoded data (Figure 4), adapting the localized layer requires changing its input dimension to 1:

$$\boldsymbol{h}_i = \text{ReLU}\left(\boldsymbol{W}_P x_i + \boldsymbol{b}_P\right)$$

where $x_i \in \mathbb{R}^1$ is the scalar feature, $\boldsymbol{W}_P \in \mathbb{R}^{4\times 1}$, and $\boldsymbol{b}_P \in \mathbb{R}^4$. The outputs are concatenated into a 320-dimensional vector for the base LR model.

—SAE-SVM: A subnetwork named ClauseNet is integrated to extract intra-clause semantics. It is a feed-forward network with an input dimension of 4, a hidden dimension of 8, and an output dimension of 4:

$$\boldsymbol{z}_i = \text{ReLU}\left(\boldsymbol{W}_1 \boldsymbol{x}_i + \boldsymbol{b}_1\right)$$
$$\boldsymbol{h}_i = \text{ReLU}\left(\boldsymbol{W}_2 \boldsymbol{z}_i + \boldsymbol{b}_2\right)$$

where $\boldsymbol{W}_1 \in \mathbb{R}^{8\times 4}$, $\boldsymbol{W}_2 \in \mathbb{R}^{4\times 8}$, $\boldsymbol{b}_1 \in \mathbb{R}^8$, and $\boldsymbol{b}_2 \in \mathbb{R}^4$. For TCE data (Figure 5), the 320-dimensional input is split into 4-dimensional segments $\boldsymbol{x}_i$, processed by ClauseNet, and concatenated back to 320 dimensions before entering the SVM equipped with a Deep Kernel Mapping Module [14].

For PPCE-encoded data (Figure 6), each 1-dimensional feature $x_i \in \mathbb{R}^1$ is first mapped to a 4-dimensional space via a linear layer:

$$\tilde{\boldsymbol{x}}_i = \boldsymbol{W}_L x_i + \boldsymbol{b}_L$$

where $\boldsymbol{W}_L \in \mathbb{R}^{4\times 1}$ and $\boldsymbol{b}_L \in \mathbb{R}^4$. This expanded representation $\tilde{\boldsymbol{x}}_i$ enters the ClauseNet, utilizing identical weights as the TCE setup for evaluation consistency.

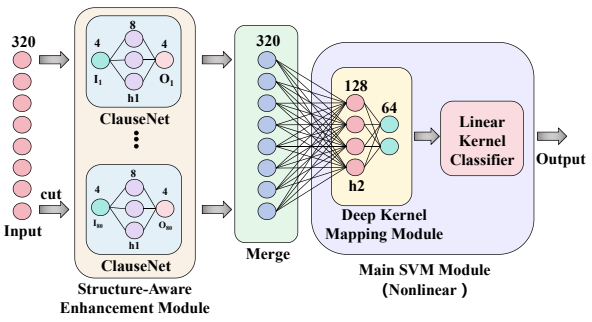

Fig. 5. SAE-SVM for TCE-encoded Data

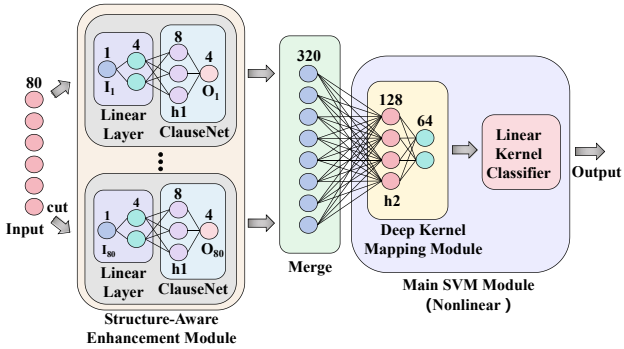

Fig. 6. SAE-SVM for PPCE-encoded Data

—SAE-MLP: This framework employs a deeper ClauseNet (input: 4, hidden: 32 and 16, output: 8). For TCE data (Figure 7), each 4-dimensional segment $\boldsymbol{x}_i$ is processed as:

$$\boldsymbol{z}_i^{(1)} = \text{ReLU}\left(\boldsymbol{W}_1 \boldsymbol{x}_i + \boldsymbol{b}_1\right)$$
$$\boldsymbol{z}_i^{(2)} = \text{ReLU}\left(\boldsymbol{W}_2 \boldsymbol{z}_i^{(1)} + \boldsymbol{b}_2\right)$$
$$\boldsymbol{h}_i = \text{ReLU}\left(\boldsymbol{W}_3 \boldsymbol{z}_i^{(2)} + \boldsymbol{b}_3\right)$$

where $\boldsymbol{W}_1 \in \mathbb{R}^{32\times 4}$, $\boldsymbol{W}_2 \in \mathbb{R}^{16\times 32}$, and $\boldsymbol{W}_3 \in \mathbb{R}^{8\times 16}$. The localized features $\boldsymbol{h}_i$ are concatenated into a 640-dimensional vector for the main MLP, whose depth is reduced to balance the SAE integration.

For PPCE-encoded data (Figure 8), a linear layer $\tilde{\boldsymbol{x}}_i = \boldsymbol{W}_L x_i + \boldsymbol{b}_L$ maps the 1-dimensional feature to a 4-dimensional space. This is subsequently fed into the deep ClauseNet, keeping the structure and weights identical to the TCE configuration.

3.2.2 Hyperparameter Settings. The hyperparameter configurations for all models are summarized in Tables 1 and 2. The loss functions used for training are defined as:

$$\mathcal{L}_{\text{CE}} = -\frac{1}{N}\sum_{k=1}^{N}\left[y_k \log \hat{y}_k + (1-y_k)\log(1-\hat{y}_k)\right]$$

$$\mathcal{L}_{\text{Hinge}} = \frac{1}{N}\sum_{k=1}^{N}\max\left(0, 1 - y_k f(\boldsymbol{x}_k)\right)$$

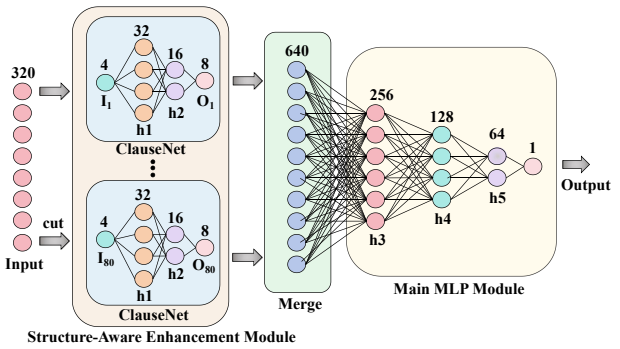

Fig. 7. SAE-MLP for TCE-encoded Data

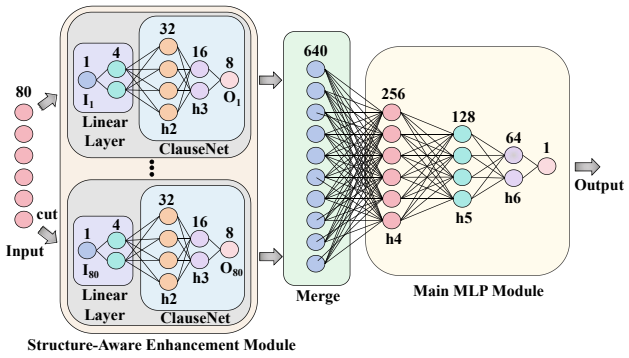

Fig. 8. SAE-MLP for PPCE-encoded Data

Table 1. Hyperparameters for Non-SAE Models and Transformer

Model	Optimizer	Learning Rate	Weight Decay	Scheduler	Loss
LR	SGD	0.00001	0.0001	-	Cross-Entropy
SVM	SGD	0.00001	0.0001	-	Hinge
MLP	SGD	0.01	0.0001	Max Mode Decay 0.5 Patience 5	Cross-Entropy
Transformer	Adam	0.001	-	-	Cross-Entropy

where $y_k \in \{0, 1\}$ represents the ground-truth satisfiability label for the k-th CNF formula, and $\hat{y}_k$ denotes the predicted probability of the formula being satisfiable.

3.2.2.1 Non-SAE Models and Transformer. As shown in Table 1, the non-SAE models employ the SGD optimizer. For linear models (LR and SVM), the learning rate is set to 0.00001 to ensure stable convergence under sparse features. For the MLP, the learning rate is 0.01 to accommodate non-linear mapping. A weight decay of 0.0001 is applied uniformly for regularization, and a scheduler is utilized exclusively for the MLP. The Transformer model utilizes the Adam optimizer with a learning rate of 0.001, bypassing extra weight decay or learning rate scheduling due to its built-in regularization mechanisms.

3.2.2.2 SAE-based Models. As shown in Table 2, SAE-based models utilize the Adam optimizer with a unified learning rate of 0.0005 to accommodate localized clause transformations. Weight decay (0.0001) is retained for SAE-LR and SAE-SVM, but omitted for SAE-MLP to avoid over-regularization on sparse features. The learning rate scheduler is maintained specifically for the SAE-MLP architecture.

Table 2. Hyperparameters for SAE-based Models

Model	Optimizer	Learning Rate	Weight Decay	Scheduler	Loss
SAE-LR	Adam	0.0005	0.0001	-	Cross-Entropy
SAE-SVM	Adam	0.0005	0.0001	-	Hinge
SAE-MLP	Adam	0.0005	-	Max Mode Decay 0.5 Patience 5	Cross-Entropy

3.3 Training Configuration and Evaluation Metrics

All models are implemented in PyTorch and evaluated using GPU acceleration. A fixed random seed of 42 is enforced across all tasks to ensure reproducibility. The batch size is set to 256, and all models are trained for a maximum of 100 epochs. Performance is evaluated across three dimensions: classification accuracy, training time cost, and inference latency.

3.3.1 Classification Accuracy. The classification accuracy (α) measures the proportion of correctly predicted instances within the test subset $\mathcal{D}_{\text{test}}$:

$$\alpha = \frac{N_{\text{correct}}}{N_{\text{total}}}$$

where N_{correct} represents the number of correctly predicted samples and N_{total} is the total number of samples within the test subset.

3.3.2 Total Training Time Cost. The total training time cost (τ_{train}) measures the cumulative time consumed during the training and per-epoch validation phases until empirical convergence:

$$\tau_{\text{train}} = \sum_{e=1}^{E_{\text{converge}}} \left(\mathcal{T}_{\text{train}}^{(e)} + \mathcal{T}_{\text{val}}^{(e)} \right)$$

where $\mathcal{T}_{\text{train}}^{(e)}$ and $\mathcal{T}_{\text{val}}^{(e)}$ denote the training and validation durations at epoch e, and E_{converge} is the specific convergence epoch.

3.3.3 Per-Sample Inference Latency. The per-sample inference latency (τ_{inf}) calculates the average time required to predict the satisfiability label of a single CNF formula post-convergence:

$$\tau_{\text{inf}} = \frac{\mathcal{T}_{\text{total\_inference}}}{N_{\text{test}}}$$

where $\mathcal{T}_{\text{total\_inference}}$ represents the total execution time over the entire test subset $\mathcal{D}_{\text{test}}$.

4. RESULTS AND COMPARATIVE ANALYSIS

This section evaluates experimental results across three dimensions—model selection, encoding design, and structural strategy—to assess their impact on SAT prediction performance and efficiency.

4.1 Performance Comparison of Four Model Types

LR, SVM, MLP, and Transformer architectures are evaluated on SAT prediction across two dataset scales and encoding schemes. As shown in Table 3 and Figure 9, deep learning architectures

Table 3. Comprehensive Experimental Results of the Four Baseline Model Architectures

Scale	Encoding	LR	SVM	MLP	Transformer
183k	PPCE	0.8140/ 1m57s/ 0.0061ms	0.8048/ 2m07s/ 0.0076ms	0.8714/ 2m36s/ 0.0089ms	0.9646/ 18m28s/ 0.0221ms
	TCE	0.8788/ 2m38s/ 0.0063ms	0.8769/ 2m47s/ 0.0077ms	0.9326/ 3m31s/ 0.0086ms	0.9925/ 19m02s/ 0.0227ms
480k	PPCE	0.8035/ 5m11s/ 0.0061ms	0.7980/ 5m38s/ 0.0076ms	0.8672/ 6m49s/ 0.0089ms	0.9895/ 48m37s/ 0.0221ms
	TCE	0.8703/ 7m15s/ 0.0063ms	0.8731/ 7m49s/ 0.0077ms	0.9523/ 7m28s/ 0.0086ms	0.9991/ 50m18s/ 0.0227ms

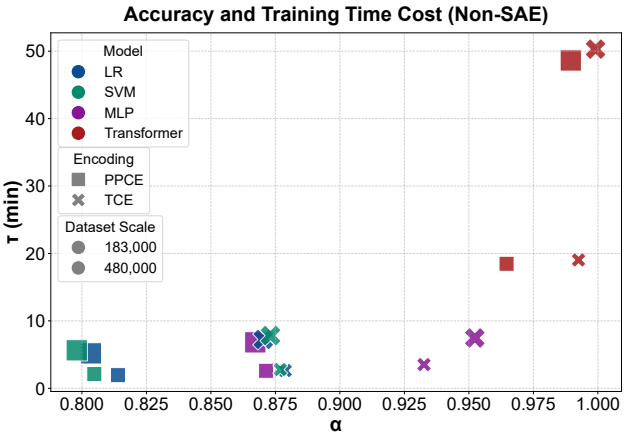

Fig. 9. Accuracy and Training Time Cost of Non-SAE Models

(MLP, Transformer) consistently outperform classical classifiers (LR, SVM), particularly on the 480k TCE dataset.

The Transformer achieves the highest accuracy (α), ranging from 0.9646 to 0.9991, indicating that its attention mechanism effectively captures long-range dependencies and global structural relations within CNF formulas. Furthermore, unlike LR and SVM whose accuracies slightly degrade when scaling from 183k to 480k under PPCE, the Transformer's higher capacity prevents performance saturation, improving accuracy from 0.9646 to 0.9895.

Empirical results identify model architecture as the primary determinant of inference latency (τ_{inf}). As shown in Figure 10, the Transformer exhibits the highest latency compared to other architectures.

The Transformer's τ_{inf} ranges from 0.0221 ms to 0.0227 ms, approximately 2.5–3.7 times higher than alternative models. Specifically, it is 3.5–3.6 times slower than LR ($\tau_{\text{inf}} \approx 0.0063$ ms), 2.9–3.0 times slower than SVM ($\tau_{\text{inf}} \approx 0.0075$ ms), and 2.5–2.6 times slower than MLP ($\tau_{\text{inf}} \approx 0.0087$ ms). This consistent gap indicates that inference latency is inherent to the Transformer's structural complexity.

Regarding training time cost (τ), the Transformer requires the longest duration (up to 50m18s), yet its accuracy improvement over LR and SVM exceeds 0.11 across all configurations. The MLP offers a trade-off between accuracy and efficiency; on the 480k TCE dataset, it achieves an α of 0.9523 in 7m28s—42m50s faster than

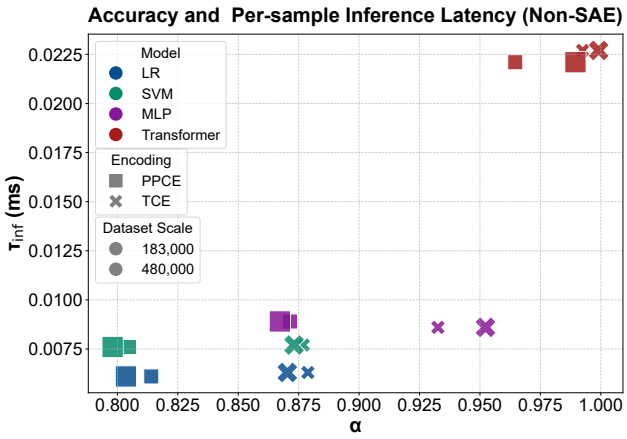

Fig. 10. Accuracy and Per-sample Inference Latency of Non-SAE Models

the Transformer with a moderate accuracy gap of 0.0468. Moreover, LR demonstrates the lowest training overhead (1m57s on 183k PPCE) but yields a maximum α of 0.8140.

Consequently, the Transformer is suitable for scenarios requiring high classification accuracy, whereas the MLP is preferable for latency-sensitive applications.

4.2 Performance Comparison of Two Encoding Schemes

To assess the impact of encoding on SAT prediction, PPCE and TCE are compared. As indicated in Table 3 and Figure 9, TCE yields higher α but longer τ than PPCE. Because TCE structurally aligns with the clausal composition of CNF formulas, models directly process logical relationships without learning feature grouping rules. Conversely, PPCE lacks explicit clause boundaries, forcing models to infer structural dependencies, which increases training cost. This pattern is evident in the MLP results: on the 480k dataset, the TCE-configured MLP achieves an α 0.0851 higher than its PPCE counterpart, while τ increases by 39 seconds (from 6m49s to 7m28s).Conventional machine learning models exhibit even greater sensitivity to this feature compression, transitioning from TCE to PPCE on the 183k dataset drops LR and SVM accuracy by over 0.06, whereas the Transformer decreases by only 0.0279.This difference occurs because LR and SVM rely directly on the full 320 features in TCE, while deep models can still extract useful patterns through non-linear layers even when features are reduced to 80 dimensions in PPCE.

Encoding choice minimally impacts inference speed. As shown in Figure 10, τ_{inf} variations remain within 10% across all models. Transitioning from the 320-dimensional TCE to the 80-dimensional PPCE alters τ_{inf} by less than 5% for LR, 8% for SVM, and 4% for MLP. For example, the MLP's τ_{inf} remains virtually identical between 0.0086 ms (TCE) and 0.0089 ms (PPCE), despite the four-fold dimensionality reduction—demonstrating that PPCE reduces dimensionality without compromising inference efficiency.

In summary, TCE is preferred for maximizing accuracy, particularly in neural models, whereas PPCE serves as an efficient alternative for training-time-constrained applications.

Table 4. Experimental Results of SAE-based Models

Scale	Encoding	SAE-LR	SAE-SVM	SAE-MLP
183k	PPCE	0.8837/ 3m21s/ 0.0068ms	0.9122/ 4m23s/ 0.0078ms	0.9427/ 4m43s/ 0.0085ms
	TCE	0.8947/ 3m21s/ 0.0063ms	0.9516/ 6m25s/ 0.0071ms	0.9727/ 6m57s/ 0.0084ms
480k	PPCE	0.9282/ 9m00s/ 0.0068ms	0.9445/ 12m13s/ 0.0078ms	0.9523/ 15m36s/ 0.0085ms
	TCE	0.8896/ 8m31s/ 0.0063ms	0.9620/ 12m13s/ 0.0071ms	0.9867/ 17m29s/ 0.0084ms

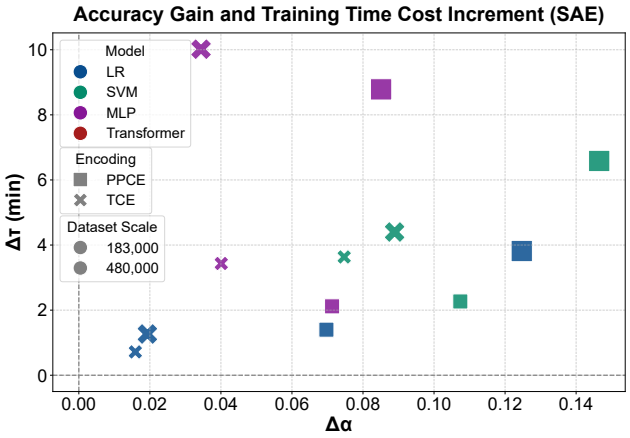

Fig. 11. Accuracy Gain and Training Time Cost Increment of SAE Models

4.3 Performance Comparison: SAE-based vs Non-SAE-based

To quantify the impact of the Structure-Aware Enhancement (SAE) module, three incremental metrics are defined relative to baseline configurations. Let α_0, τ_0, and $\tau_{\text{inf},0}$ denote the accuracy, training time cost, and inference latency of the baseline Non-SAE model, respectively, and let α_{SAE}, τ_{SAE}, and $\tau_{\text{inf,SAE}}$ denote the metrics for the SAE-enhanced model.

The absolute accuracy gain ($\Delta\alpha$) is defined as:

$$\Delta\alpha = \alpha_{\text{SAE}} - \alpha_0$$

The absolute training time cost increment ($\Delta\tau$) is defined as:

$$\Delta\tau = \tau_{\text{SAE}} - \tau_0$$

The absolute per-sample inference latency increment ($\Delta\tau_{\text{inf}}$) is defined as:

$$\Delta\tau_{\text{inf}} = \tau_{\text{inf,SAE}} - \tau_{\text{inf},0}$$

As illustrated in Table 4 and Figure 11, the SAE module consistently yields positive accuracy gains ($\Delta\alpha$) over Non-SAE baselines using TCE encoding.

On the 183k dataset, baseline accuracy α_0 (0.8769–0.9326) improves to 0.8947–0.9727 with SAE. SAE-SVM achieves the maximum $\Delta\alpha$ of 0.0747, followed by SAE-MLP ($\Delta\alpha = 0.0401$) and SAE-LR ($\Delta\alpha = 0.0159$). On the 480k dataset, baseline accuracy (0.8703–0.9523) reaches 0.8896–0.9867 with SAE; SAE-SVM exhibits the largest improvement ($\Delta\alpha = 0.0889$), while SAE-MLP yields a moderate gain ($\Delta\alpha = 0.0344$). The inference overhead introduced by the SAE module is minimal. As shown in Figure 12, $\Delta\tau_{\text{inf}}$ accounts for less than 8% of the baseline latency $\tau_{\text{inf},0}$ across all configurations, remaining below 5% in most cases.

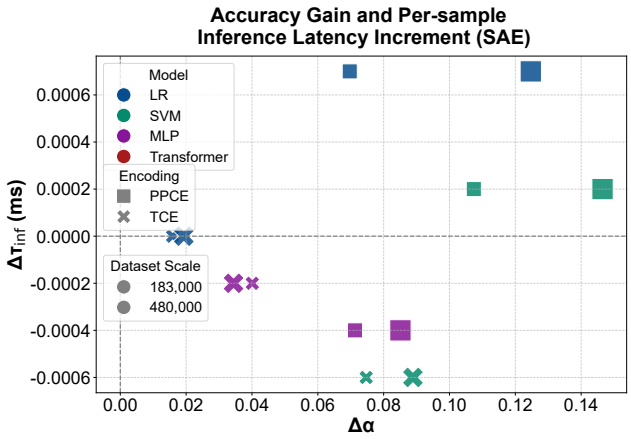

Fig. 12. Accuracy Gain and Per-sample Inference Latency Increment of SAE Models

However, the SAE module increases training overhead; the increment $\Delta\tau$ is most pronounced for SAE-MLP on the 480k TCE dataset, adding approximately 10 minutes to the baseline duration. Empirical observations indicate that conventional machine learning models (LR, SVM) benefit more significantly from the structural priors provided by the SAE module. This improvement pattern extends to PPCE encoding, where structural guidance from the SAE module compensates for the information lost during compression, yielding substantial performance gains. For instance, on the 480k PPCE dataset, SAE boosts LR and SVM accuracy by 0.1247 and 0.1465, respectively. Notably, SAE-MLP with PPCE achieves an accuracy of 0.9523, perfectly matching the baseline MLP configured with the higher-dimensional TCE.

In conclusion, the SAE module effectively embeds clause-level structural information into the learning pipeline, improving prediction accuracy (α) with minimal impact on per-sample inference latency (τ_{inf}). Furthermore, the integration of the SAE module and PPCE encoding maintains high model accuracy while effectively compressing feature dimensionality.

5. CONCLUSION AND FUTURE WORK

This paper investigates three primary challenges in ML4SAT: feature dimensionality, structural underutilization, and the lack of systematic benchmarks. The results yield three conclusions:

—Integrating explicit structural priors improves SAT classification accuracy. The lightweight SAE module delivers consistent gains ($\Delta\alpha$) across LR, SVM, and MLP architectures with negligible impact on inference latency (τ_{inf}), especially benefiting classical classifiers with limited feature extraction capacity.

—PPCE effectively mitigates high-dimensional representation explosion, reducing dimensionality from 320 to 80 compared to TCE. This compression incurs minor computational penal-

ties—with τ_{inf} variations within 10%—demonstrating suitability for resource-constrained deployments.

—PPCE optimizes training efficiency, achieving lower training time costs (τ) than TCE, with performance advantages scaling proportionally with dataset size.

Open challenges remain: PPCE underperforms TCE in deep learning accuracy, and the baseline SAE warrants further optimization for complex clause structures.Future research will focus on:

—Refining encodings and structure-aware architectures to better capture intra- and inter-clause dependencies while preserving logical semantics.

—Exploring alternative representation paradigms, such as vision-based CNF mappings, to reframe SAT prediction for convolutional feature extraction.

—Extending the framework to complex industrial benchmarks and real-world formal verification scenarios to support broader downstream engineering applications.

These research directions aim to advance ML4SAT methodologies, facilitating integration into industrial scenarios and overcoming traditional SAT solver bottlenecks on complex problems.

6. REFERENCES

[1] A. Biere, M. Heule, H. van Maaren, and T. Walsh. *Handbook of Satisfiability*, volume 336 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, Amsterdam, The Netherlands, 2021.

[2] H. K. Büning and T. Lettmann. *Propositional Logic: Deduction and Algorithms*. Cambridge University Press, Cambridge, England, 1999.

[3] W. J. Chang, M. Y. Guo, and J. W. Luo. Predicting the satisfiability of Boolean formulas by incorporating gated recurrent unit (GRU) in the Transformer framework. *PeerJ Computer Science*, 10:e2169, October 2024.

[4] Y. J. Chang, M. S. M. Mohd Kasihmuddin, W. N. A. Ruzai, Y. Guo, and J. Chen. Weighted C-type random 2 satisfiability in discrete Hopfield neural network. *Engineering Applications of Artificial Intelligence*, 160:111760, April 2025.

[5] Stephen A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the Third Annual ACM Symposium on Theory of Computing (STOC)*, pages 151–158, New York, NY, USA, 1971. ACM.

[6] C. Cortes and V. Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–299, September 1995.

[7] D. R. Cox. The regression analysis of binary sequences. *Journal of the Royal Statistical Society: Series B (Methodological)*, 20(2):215–232, June 1958.

[8] R. Goré. Tableau methods for modal and temporal logics. Tech. Rep. TR-ARP-15-95, Australian National University, November 1995.

[9] M. Gori, G. Monfardini, and F. Scarselli. A new model for learning in graph domains. In *Proceedings of the 2005 IEEE International Joint Conference on Neural Networks (IJCNN)*, pages 729–734. IEEE, 2005.

[10] J. J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences*, 79(8):2554–2558, April 1982.

[11] J. Hůla, D. Mojžíšek, and M. Janota. Understanding GNNs for Boolean satisfiability through approximation algorithms. In *Proceedings of the 33rd ACM International Conference on Information and Knowledge Management (CIKM)*, pages 953–961, New York, NY, USA, 2024. ACM.

[12] D. Jia, X. Duan, and M. K. Khan. Binary artificial bee colony optimization using bitwise operation. *Computers & Industrial Engineering*, 76:360–365, October 2014.

[13] D. S. Johnson. Approximation algorithms for combinatorial problems. *Journal of Computer and System Sciences*, 9(3):256–278, September 1974.

[14] Y. J. Li, T. Zhang, and H. H. Hu. Deep kernel mapping support vector machines based on multi-layer perceptron. *Journal of Beijing University of Technology*, 42(11):1652–1661, November 2016.

[15] Y. H. Liang and J. L. Li. Novel message passing network for neural Boolean satisfiability problem solver. *Journal of Computer Applications*, 45(9):2934–2940, September 2025.

[16] J. A. Robinson. A machine-oriented logic based on the resolution principle. *Journal of the ACM*, 12(1):23–41, January 1965.

[17] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, October 1986.

[18] K. Schneider. *Verification of Reactive Systems: Formal Methods and Algorithms*. Springer, Berlin, Germany, 2004.

[19] D. Selsam and N. Bjørner. Guiding high-performance SAT solvers with unsat-core predictions. In *Proceedings of the 22nd International Conference on Theory and Applications of Satisfiability Testing (SAT 2019)*, pages 336–353, Cham, Switzerland, 2019. Springer.

[20] D. Selsam, M. Lamm, B. Bünz, P. Liang, D. L. Dill, and L. de Moura. Learning a SAT solver from single-bit supervision. In *Proceedings of the 7th International Conference on Learning Representations (ICLR)*, 2019. Published online via OpenReview, no numerical pages assigned.

[21] E. Shoham, H. Cohen, K. Wattad, et al. Concept learning for algorithmic reasoning: Insights from SAT-solving GNNs. *Information Sciences*, 726:122754, June 2026.

[22] Y. W. Sun, F. R. Ye, X. Y. Zhang, et al. AutoSAT: Automatically optimize SAT solvers via large language models. arXiv preprint arXiv:2402.10705, February 2024.

[23] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. In *Proceedings of the 31st International Conference on Advances in Neural Information Processing Systems (NIPS 2017)*, pages 5998–6008, Red Hook, NY, USA, 2017. Curran Associates, Inc.

[24] C. X. Yu, R. J. Liang, C. T. Ho, et al. Autonomous code evolution meets NP-completeness. arXiv preprint arXiv:2509.07367, September 2025.