CEUR-WS.org/Vol-4039/paper18.pdf

Recurrent Neural Networks for Guiding Proof Search in Propositional Logic

Gianluca Amato[1], Nicola Balestra[1], Marco Maggesi[2] and Maurizio Parton[1]

[1]*Computational Logic and Artificial Intelligence lab, Università degli Studi "Gabriele d'Annunzio" di Chieti e Pescara*
[2]*Università degli Studi di Firenze*

Abstract

DeepSAT is an ongoing research project investigating the use of deep reinforcement learning for automated theorem proving in propositional logic. In contrast to traditional SAT solvers, which focus on satisfiability checking, our aim is to construct formal proofs of validity within the sequent calculus framework. As a preliminary step toward this goal, we have focused on training recurrent neural networks to predict whether a given propositional formula is a tautology. These models will form the basis of the value function in the planned reinforcement learning architecture. Although in its early stages, DeepSAT lays the groundwork for a logic-agnostic, explainable, and energy-efficient alternative to existing neural approaches based on large language models, which require substantial computational resources.

Keywords

Propositional logic, Automated and interactive theorem proving, Deep learning, Recurrent neural networks

1. Introduction

Automated theorem proving is a longstanding challenge at the intersection of logic, artificial intelligence, and symbolic reasoning. In propositional logic, the state of the art is dominated by SAT solvers [1], which reduce the task of proving validity to checking Boolean satisfiability. These tools are remarkably fast and effective.

In contrast, neural approaches to reasoning have gained increasing attention for their ability to learn inference patterns directly from data [2, 3]. These models do not follow explicit logical rules, but instead learn internal representations that approximate logical behavior, offering a complementary and potentially more flexible approach.

Our aim is to develop a system for automated proof search in propositional logic. This paper presents a hybrid system that combines a neural model with a symbolic proof engine based on the sequent calculus. As a first step, we train a Tree Long Short-Term Memory (Tree-LSTM) [4] classifier to predict whether a given propositional formula is a tautology. This classifier is initially developed and evaluated independently as a supervised model. In a second phase, it is embedded into a Monte Carlo Tree search (MCTS) [5] agent, where it serves as a policy-value function to guide deductive proof search. Since the ultimate goal of the search is to construct a valid proof, estimating the likelihood that a formula is a tautology offers a meaningful heuristic for both action selection and value estimation within the MCTS framework.

The architecture of the Tree-LSTM classifier leverages the hierarchical structure of logic syntax: formulas are represented as trees, enabling this model to learn compositional embeddings aligned with their syntactic structure. The symbolic component, in turn, applies inference rules over sequents to construct formal derivations. We evaluate this model both as a standalone classifier and as a heuristic guide within the MCTS agent.

ICTCS 2025: Italian Conference on Theoretical Computer Science, September 10–12, 2025, Pescara, Italy

gianluca.amato@unich.it (G. Amato); nicola.balestra@phd.unich.it (N. Balestra); marco.maggesi@unifi.it (M. Maggesi); maurizio.parton@unich.it (M. Parton)
https://www.sci.unich.it/~amato (G. Amato); https://sites.google.com/unifi.it/maggesi/ (M. Maggesi)
0000-0002-6214-5198 (G. Amato); 0009-0007-1893-3406 (N. Balestra); 0000-0003-4380-7691 (M. Maggesi); 0000-0003-4905-3544 (M. Parton)

© 2025 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

CEUR Workshop Proceedings ceur-ws.org ISSN 1613-0073

The Python code, the dataset used for the experiments and the trained models can all be found on our git repository at the URL https://github.com/CLAI-UdA/deepsat-experiments/tree/ictcs2025.

2. The DeepSAT Project

DeepSAT is a broader research initiative in which the present work is situated. Its central idea is to frame theorem proving as a sequential decision process, where each state corresponds to a collection of sequents, and each action applies an inference rule to advance the proof. Neural networks provide heuristics in the form of a policy — guiding the choice of inference steps — and a value function — estimating the provability of a sequent. These components are integrated into a tree search procedure such as Monte Carlo Tree Search.

While we acknowledge that this approach is unlikely to compete with SAT solvers in terms of raw performance, *DeepSAT* offers several potential advantages in comparison to subsymbolic neural approaches, particularly large-scale language models:

Logic-agnosticism. The architecture is not tied to classical propositional logic and can, in principle, be adapted to non-classical logics such as intuitionistic or linear logic without major structural changes.

Extensibility. The system can be extended to support first-order logic, potentially serving as an intelligent assistant in interactive proof development.

Transparency and accessibility. Unlike solutions developed within large-scale industrial research frameworks (e.g., by OpenAI or DeepMind), *DeepSAT* is designed around principles of openness and explainability.

These features are not intended to outperform traditional symbolic solvers, which remain the most efficient and reliable tools for large-scale automated reasoning. Rather, *DeepSAT* is conceived as a structured and interpretable alternative to purely neural approaches, particularly those based on large language models. The goal is not to replace symbolic reasoning, but to complement it with learned guidance that remains transparent, adaptable, and logically grounded.

Specifically, the project embraces the following key principles:

Open-source. All software components, trained models, and data will be made publicly available.

Energy efficiency. The system is designed to operate with a fraction of the computational resources typically required for training and inference in larger models.

Explainability. Although human-readability of the produced proofs is not our primary goal, we aim to design architectures whose internal functioning is more interpretable, owing to the need to operate under constrained computational budgets. This will require novel strategies for the representation and manipulation of formulas and sequents, tailored to the structure of logical reasoning tasks.

This paper represents a first step towards the broader goal of the *DeepSAT* project. We focus in particular on the implementation of the value function.

3. RNN for Predicting Truth Values of Formula

We present a method for classifying propositional formulas as tautologies or non-tautologies using a neural network architecture. This approach is based on a Tree-LSTM model, which is specifically designed to operate on tree-structured data, making it well-suited for the recursive and compositional nature of logical formulas.

The central idea is to represent each formula as a binary syntax tree that captures its hierarchical structure, and to process this tree using a recursive neural network that can learn to recognize various logical patterns. The Tree-LSTM model, following the architecture proposed by Tai et al. [4], computes vector representations for each node of the syntax tree in a bottom-up manner, combining the information of the subformulas through a gated mechanism derived from standard LSTM cells [6].

To train and evaluate the model, we construct a synthetic dataset of well-formed propositional formulas labeled as tautologies or non-tautologies by means of exhaustive truth table evaluation. To address class imbalance and ensure that well-known tautologies are well represented during training, we perform targeted data augmentation using instantiated tautology schemata.

3.1. Formula Representation and Dataset Generation

In our framework, each formula is recursively defined in terms of atomic components (propositional variables and logical constants) and logical connectives, according to the formal syntax of propositional logic.

Atomic formulas correspond to individual propositional variables, each assigned a unique numerical identifier. Compound formulas are constructed recursively by applying logical connectives — such as negation ($\neg$) and binary connectives ($\wedge, \vee, \rightarrow$) — to one or more subformulas.

To train and evaluate the model, we constructed a synthetic dataset of normalized propositional formulas, generated through a controlled recursive process, with bounded depth and a limited set of propositional variables.

The generation follows a recursive pattern: at each level, the formula may be the Boolean constant false ($\bot$), a propositional variable (A_i), or a compound formula.

Once generated, a normalization procedure is applied to each formula, renaming propositional variables in order of first appearance with sequential indices starting from 0. This step reduces superficial syntactic redundancy, allowing the model to focus on logical structure rather than on arbitrary variable names.

Each formula is labeled either as a tautology or non-tautology by exhaustively evaluating all possible truth assignments. A formula is labeled as a tautology if it is evaluated true under every possible assignment of truth values. Given the limited number of propositional variables, this method is computationally feasible. For future extensions involving a larger variable space, we plan to adopt SAT solvers for more scalable and efficient labeling. The final dataset consists of $10,000$ distinct formulas, each paired with its corresponding Boolean label.

However, the initial distribution of tautologies was highly imbalanced: only 4.18% of the generated formulas were tautologies. This imbalance stems from the random generation process itself: since tautologies constitute only a small fraction of the space of well-formed formulas, they are rarely produced by chance. Such imbalance risks biasing the model during training, potentially leading it to ignore the minority class and underperform on tautological instances. Furthermore, an analysis of the dataset revealed the absence of canonical tautologies such as the Law of the Excluded Middle and Law of Non-Contradiction.

To make the dataset more representative, we introduced a data augmentation phase based on the generation of new examples from known tautological schemata. For this purpose, we defined a collection of tautological schemata containing metavariables. These schemata include the following examples from classical logic: Reflexivity ($A \rightarrow A$), Law of the Excluded Middle, Law of Non-Contradiction, De Morgan's Law for Conjunction, De Morgan's Law for Disjunction, De Morgan's Law for Disjunction, Distribution of Conjunction over Disjunction, Distribution of Disjunction over Conjunction.

To generate concrete instances, each metavariable in the schemata is replaced with a randomly generated formula, and the result is finally normalized. The generation process produced formulas with structural properties consistent with those of randomly generated examples in the dataset. This ensures structural consistency between instantiated and randomly generated formulas.

In total $3,000$ new tautological formulas were generated and included as independent examples in the dataset, increasing the proportion of tautologies to 26.29%. Even with the additions of these formulas, the dataset was still unbalanced. To address this challenge, we adopted an asymmetric loss function, described in Section 3.4.

3.2. Preprocessing Data

After dataset generation, each propositional formula is converted into a binary syntax tree that reflects its compositional logic structure. In this representation, each node n corresponds to a subformula $f \in \mathcal{F}$, where $\mathcal{F}$ is the set of well-formed formulas. The children of each node represent the arguments of the main connective in f. This tree structure is required by the Tree-LSTM model, which processes formulas recursively from leaves to root.

To enable numerical processing by the model, to each node n is associated an index $id(n) \in \mathbb{N}$, which depends on the main connective of the subformula f it contains, via a tokenization function T:

- For each propositional letter A_k, assign the token

$$T(A_k) = k + 1, \qquad \text{with} \quad k \geq 0$$

- For each logical connective $\Box \in \{\neg, \wedge, \vee, \to\}$, assign a token

$$T(\Box) = \begin{cases} 100, & \text{if } \Box \text{ is } \neg \\ 101, & \text{if } \Box \text{ is } \wedge \\ 102, & \text{if } \Box \text{ is } \vee \\ 103, & \text{if } \Box \text{ is } \to \end{cases}$$

- For the Boolean constant $\bot$, assign the token

$$T(\bot) = 105$$

The index $id(n)$ is stored in the *embedding_index* field of the node n, and subsequently passed to a learnable embedding layer that maps symbolic identifiers to dense vector representations. Specifically, the embedding layer is parameterized by a matrix $\mathbb{E} \in \mathbb{R}^{V \times E}$, where V denotes the vocabulary size (i.e., the number of distinct tokens) and E is the embedding dimension. Given an index $i = id(n)$, the model retrieves the corresponding embedding vector by selecting the i-th row of $\mathbb{E}$, denoted $x_n = \mathbb{E}[i] \in \mathbb{R}^E$. This vector x_n is then used as input to the subsequent Tree-LSMT cell at node n.

The embedding matrix $\mathbb{E}$ is composed of learnable parameters and participates in gradient-based optimization. At each training step, only the rows corresponding to input symbols are updated.

An example of an embedding-annotated tree is shown in Figure 1.

```
└── [Implication] A0 ∧ (A0 → A1) → A1 (embedding_index=103)
    ├── [Conjunction] A0 ∧ (A0 → A1) (embedding_index=100)
    │   ├── [Letter] A0 (embedding_index=1)
    │   └── [Implication] A0 → A1 (embedding_index=103)
    │       ├── [Letter] A0 (embedding_index=1)
    │       └── [Letter] A1 (embedding_index=2)
    └── [Letter] A1 (embedding_index=2)
```

Figure 1: Embedding-annotated tree of the formula $A_0 \wedge (A_0 \to A_1) \to A_1$. Each node contains the root subformula f, the syntactic type of f, and the assigned token $id(n)$ symbolically encoding its structure.

For model training, each data example consists of a binary syntax tree whose nodes are annotated with embedding indices, encoding the logical structure of the corresponding formula. Each tree is paired with a Boolean label indicating whether the formula is a tautology or not.

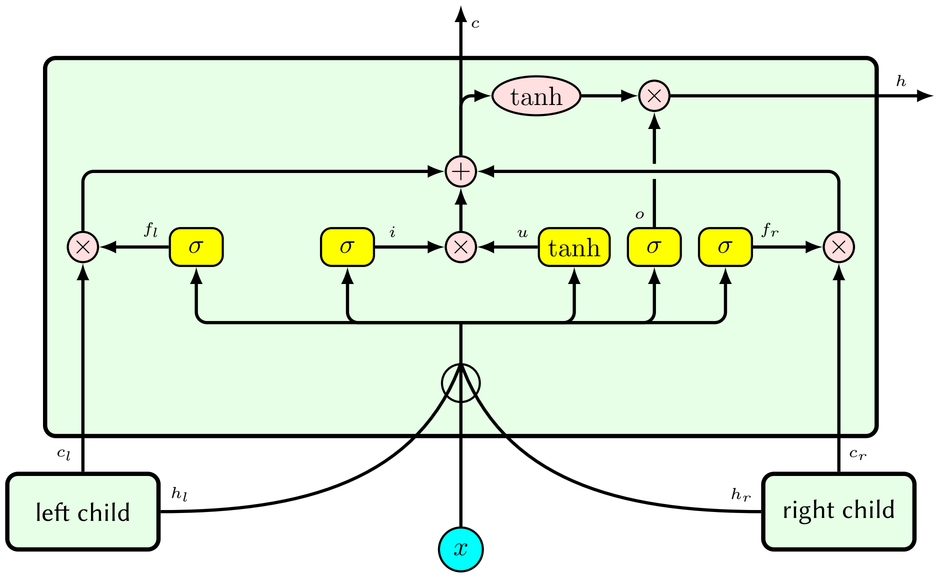

Figure 2: Diagram of a Tree-LTSM unit. Each unit receives a pair of c and h vectors as input and produces new c and h vectors. These outputs are then passed to the Tree-LSTM unit corresponding to a higher-level (outer) node in the tree structure. Elliptic pink nodes represent simple pointwise operations, while rectangular yellow nodes represent perceptron layers.

3.3. Tree-LSTM Architecture

Tree-LSTM networks generalize standard recurrent architectures to hierarchical structures such as syntax tree. In our setting, each formula is represented as a binary tree, and the model computes hidden representation for each node by recursively combining information from its children. This enables the model to capture long-range dependencies and structural relationships that arise in propositional logic formulas.

In the binary Tree-LSTM variant, each internal node receives an input vector $x \in \mathbb{R}^E$ and two pairs of hidden and cell states, (h_l, c_l) and (h_r, c_r), from its left and right children, respectively. At each node in the formula tree, the Tree-LSTM performs a local computation — referred to as a Tree-LSTM unit — which outputs a hidden state vector $h \in \mathbb{R}^d$ and a memory cell vector $c \in \mathbb{R}^d$, where d is the hidden dimension. These two vectors encode the entire subtree rooted at that node and are recursively propagated upward to the parent node.

To compute these vectors, the Tree-LSTM unit maintains five internal components: an *input gate* $i \in \mathbb{R}^d$, two *forget gates* $f_l, f_r \in \mathbb{R}^d$ for the left and right children, an *output gate* $o \in \mathbb{R}^d$, and an *update vector* $u \in \mathbb{R}^d$.

The Tree-LSTM transition equations, depicted in Figure 2, are the following:

$$\begin{aligned}
h_{\text{cat}} &= [h_l, h_r] \\
i &= \sigma(W^{(i)}x + U^{(i)}h_{\text{cat}} + b^{(i)}) \\
f_l &= \sigma(W^{(f_l)}x + U^{(f_l)}h_{\text{cat}} + b^{(f_r)}) \\
f_r &= \sigma(W^{(f_r)}x + U^{(f_r)}h_{\text{cat}} + b^{(f_r)}) \\
o &= \sigma(W^{(o)}x + U^{(o)}h_{\text{cat}} + b^{(o)}) \\
u &= \tanh(W^{(u)}x + U^{(u)}h_{\text{cat}} + b^{(u)}) \\
c &= i \otimes u + f_l \otimes c_l + f_r \otimes c_r \\
h &= o \otimes \tanh(c)
\end{aligned}$$

Here, x is the input embedding of the current node, σ denotes the logistic sigmoid activation function, and $\otimes$ denotes elementwise multiplication. The hidden state vector h can be interpreted as a distributed representation of the subformula rooted at the node, integrating structural and semantic information from its subtree.

Conceptually, the input gate i controls the incorporation of new information into the memory cell; the forget gates f_l, f_r regulate the contribution of information from the left and right child children, respectively; and the output gate o determines which parts of the memory cell are exposed to the hidden state. As in standard LSTMs, the gating mechanism enables the model to learn multiscale representations and retain relevant information over long-range structural contexts.

In our implementation, leaf nodes (e.g., propositional letters or constants) are initialized with zero-valued hidden and cell states. Unary nodes (e.g., negations) replicate the same child's state as both left and right inputs, in order to maintain a consistent binary interface.

The output of the Tree-LSTM — that is, the hidden state h at the root of the tree — is passed through a small multi-layer perceptron (MLP) consisting of a fully connected layer with ReLU activation, and a final single-neuron output layer for binary classification.

3.4. Loss Function

To address class imbalance in the dataset, we adopt a variant of Focal Loss known as *Asymmetric Focal Loss* [7]. This loss function assigns greater penalty to false negatives in the minority class (tautologies) by introducing independent weighting and focusing parameters for each class. The loss is defined as:

$$\mathcal{L}(y, \hat{y}) = -\alpha_{\text{pos}} y (1-\hat{y})^{\gamma_{\text{pos}}} \log(\hat{y}) - \alpha_{\text{neg}} (1-y) \hat{y}^{\gamma_{\text{neg}}} \log(1-\hat{y})$$

where $\alpha_{\text{pos}}, \gamma_{\text{pos}}$ are the weight and focal parameter for the tautology class, respectively; and $\alpha_{\text{neg}}, \gamma_{\text{neg}}$ are the corresponding parameters for the non-tautology class, respectively. The criteria used to select the values of α and γ for both classes are detailed in the experimental Section 5.

This formulation represents one possible instantiation within a broader class of methods designed to address class imbalance during training. By dynamically modulating the contribution of individual training examples based on their difficulty, it mitigates the dominance of the majority class and places increased emphasis on correctly classifying underrepresented tautologies. In doing so, it aims to enhance recall on the minority class without significantly compromising overall performance.

4. RNN in an MCTS Agent

The Tree-LSTM classifier, described in the previous section, provides a value estimate for propositional formulas in the form of a predicted probability of being a tautology. We integrate this trained model into a Monte Carlo Tree Search (MCTS) agent for automated proof search. The goal is to replace pure pattern-based classification with an agent capable of actively constructing proofs within the sequent calculus framework, guided by a learned probability distribution over inference rules and an estimated value of provability for each sequent.

4.1. Proof Search as Guided (Hyper-)Tree Exploration

Proof construction is framed as search over a dynamically growing tree. Each node represents a sequent to be proved, and inference rules decompose it into child sequents. Different inference rules may be applied to the same sequent. Therefore, we have two different kinds of branch, which have to be dealt with differently in checking provability and in cost aggregation: all sequents coming from the same inference rule must be proved in order to prove the original sequent, while the success of a single inference rule is enough. Therefore, what we will call *proof tree* is actually an hyper-tree in which every edge has a source node and a finite set of target nodes. Alternatively, we can view proof-trees as and-or trees.

4.2. MCTS Architecture

We implement a variant of Monte Carlo Tree Search that combines the symbolic structure of sequent calculus with a learned neural policy and value. The MCTS agent performs a series of simulations, each involving four phases:

- *Selection*: Starting from the root sequent, the agent recursively selects child nodes based on a polynomial Upper Confidence Bound for Tree (PUCT) [8] formula. At each step, it selects the move whose subgoals maximize a logic-sensitive utility, computed from visit counts, value estimates, and neural prior probabilities. The traversal stops when a leaf node is reached.
- *Evaluation*: Upon reaching a leaf, the agent evaluates whether the current sequent is an axiom. If so, the corresponding node is marked as proved and assigned maximum value. Otherwise, the neural network is queried to produce a value estimate for provability and a distribution over possible inference rules. These outputs guide expansion and backpropagation.
- *Expansion*: If the sequent is not an axiom, all applicable inference rules are applied to generate subgoals. These are inserted into both the search tree. Each move is associated with the neural prior probability produced in the evaluation step.
- *Backpropagation*: The values estimate is then propagated recursively back through the path taken during selection. Multiple children of the same move (hyper-edge) are aggregated by taking the minimum value, while different moves are aggregated by taking their maximum value.

This architecture implements a hybrid approach in which symbolic rule applications are guided by probabilities. The neural model steers the exploration toward plausible proof paths, while the MCTS framework ensures that all constructed proofs adhere to the formal semantics of the Sequent Salculus. The next section presents the results of our experiments with this architecture.

5. Experiments

In this section we present the result of our experiments on both the Tree-LTSM model and the MCTS agent. In particular, Sections 5.1 to 5.6 present the tuning process and the final results of the Tree-LTSM model, while Section 5.7 deal with the MCTS agent, with either a random policy or a policy derived from the Tree-LTSM model.

All experiments were conducted on the synthetic dataset described in Section 3, generated using a controlled recursive procedure with a maximum syntax tree depth of 5 and up to 7 distinct propositional variables. This limit was chosen as a trade-off between the expressive richness of the generated formulas and their interpretability, both for training analysis and for proof traceability. As discussed, the final dataset includes 13,000 formulas labeled as either tautologies or non-tautologies, with the proportion of tautologies equal to 26.29%. The dataset was randomly split into 80% for training and 20% for testing.

All models were trained in an environment running on a virtualized Ubuntu 22.04.5 LTS server equipped with an 8-core AMD EPYC processor, 31GB of RAM, and NVIDIA Tesla T4 GPU with 15GB of memory. The system uses CUDA 12.6 and NVIDIA driver version 560.35.05.

Training time for each Tree-LSTM variant was approximately 3 hours, while the GRU baseline — trained with half the epochs — completed in approximately 1.5 hours.

5.1. Baseline Model: Bidirectional GRU

Before introducing the tree-based architecture, we first established a baseline for experiments, using a sequential model based on stacked bidirectional GRU [9] layers. The model processes tokenized formulas as sequences and aggregates bidirectional hidden states to perform binary classification.

The architecture consists of an embedding layer of dimension 32, followed by two bidirectional GRU layers: the first with 128 hidden units, and the second with 64 hidden units. The final hidden states of the second GRU layer are concatenated and passed through a fully connected layer of size 32 with ReLU activation, followed by an output layer with a single neuron for binary classification .

The model was trained using the Adam optimizer [10] with a learning rate of 5×10^{-4} and the same Asymmetric Focal Loss employed in subsequent experiments. Specifically, we used $\alpha_{\text{pos}} = 0.3$, $\alpha_{\text{neg}} = 0.7$, $\gamma_{\text{pos}} = 3.0$, and $\gamma_{\text{neg}} = 1.5$, based on a heuristic analysis of the class imbalance in the training set. These values emphasize the penalty for false negatives in the minority class while avoiding excessive gradient suppression in the majority class. These initial parameters were later refined through a systematic grid search, as detailed in the following section.

Although it lacks explicit access to the compositional tree structure of formulas, the GRU model achieved promising results. Table 1 reports the metrics over 10 training epochs.

Epoch	Train Loss	Train Acc.	Test Loss	Test Acc.
1	0.0347	87.84%	0.0198	95.94%
2	0.0165	95.48%	0.0136	96.59%
3	0.0134	95.96%	0.0146	96.01%
4	0.0107	96.18%	0.0097	97.12%
5	0.0088	96.80%	0.0091	95.86%
6	0.0084	96.64%	0.0076	96.89%
7	0.0078	96.78%	0.0071	97.24%
8	0.0072	97.02%	0.0075	97.01%
9	0.0065	97.17%	0.0066	97.47%
10	0.0059	97.38%	0.0086	97.28%

Table 1
Performance of the bidirectional GRU model over 10 training epochs.

These results indicate that even a sequential model can learn relevant regularities in propositional formulas. However, we hypothesized that a model explicitly designed to exploit the tree structure of formulas would be more effective at capturing logical composition and semantic abstractions. This motivated the transition to the Tree-LSTM architecture.

In the next section, we introduce a Tree-LSTM model that leverages the hierarchical syntax of logical formulas to improve classification accuracy and generalization.

5.2. Tree-LSTM: Hyperparameters Search

To determine the best configuration for the Tree-LSTM classifier, we conducted a structured three-stage grid search over a reduced subset of 1,000 formulas sampled from the full dataset, preserving the original class distribution to ensure representativeness.

5.3. Hidden Size and Fully Connected Layer Dimensions

In the first stage, we varied the hidden state size of the Tree-LSTM unit and the dimension of the fully connected layer. The embedding dimension was fixed at 32, and each trial was run for 5 epochs to control computational cost.

The parameters of the Asymmetric Focal Loss were initially set based on the same heuristic used for the bidirectional GRU baseline: $\alpha_{\text{pos}} = 0.3$, $\alpha_{\text{neg}} = 0.7$, $\gamma_{\text{pos}} = 3.0$, and $\gamma_{\text{neg}} = 1.5$. We used the Adam optimizer with a learning rate of 5×10^{-4}.

The Tree-LSTM configuration with 128 hidden units and a fully connected layer of size 32 achieved the best performance in this phase, reaching 92% test accuracy and a test loss of 0.0248.

5.4. Asymmetric Focal Loss Parameters

In the second stage, we fixed the architecture based on the results from the previous phase and varied the class weights ($\alpha_{\text{pos}}, \alpha_{\text{neg}}$) and focusing parameters ($\gamma_{\text{pos}}, \gamma_{\text{neg}}$). The goal was to improve the model's ability to detect tautologies by refining the sensitivity of the loss function to class imbalance. The best results were obtained with $\alpha_{\text{pos}} = 0.25, \alpha_{\text{neg}} = 0.7, \gamma_{\text{pos}} = 3.0, \gamma_{\text{neg}} = 2.5$. This configuration achieved a test accuracy of 93% and a test loss of 0.0154.

5.5. Learning Rate Optimization

In the final stage, we optimized the learning rate, testing values from 1×10^{-4} to 1×10^{-3} using the best model and loss parameters from the previous steps. The best performance was achieved with a learning rate of 9×10^{-4}, which yielded a test accuracy of 93% and a test loss of 0.0112.

This final configuration — combining architectural tuning, improved loss weighting, and learning rate optimization — was adopted as the default setup for all experiments on the full dataset, discussed in the next section.

Sweep	Test Accuracy	Train Accuracy	Test Loss	Train Loss
Hidden Size & FC Layer (128, 32)	0.920	0.921	0.0248	0.0226
Focal Loss Params ($\alpha_{pos} = 0.25$, $\gamma_{pos} = 3.0$)	0.930	0.930	0.0154	0.0120
Learning Rate (9×10^{-4})	0.930	0.943	0.0112	0.0098

Table 2
Results of the three-stage grid search on the reduced dataset.

5.6. Final Training on the Full Dataset

After selecting the optimal hyperparameters through grid search, we trained two Tree-LSTM variants on the full dataset of 13,000 examples.

5.6.1. Model without Dropout

The first model corresponds to the final architecture determined in the previous sections and does not include any dropout regularization. It was trained for 20 epochs and exhibited a stable and consistent learning trajectory. The highest test accuracy was obtained at epoch 17, reaching 99.31%, with a corresponding test loss of 0.0026. While training accuracy converged to 100% in the later epochs, test accuracy remained consistently high across the training process. The test loss also stayed low and stable, indicating that the model was able to generalize well despite the absence of explicit regularization.

These results suggest that overfitting had minimal impact on overall performance.

5.6.2. Model with Dropout

To assess the impact of regularization, we trained a second variant that includes a dropout layer, with dropout rate of 30%, after the first fully connected layer. The training configuration and number of epochs were kept identical. This model also achieved high performance, with a peak test accuracy of 99.08% at epoch 17. In the final epochs, test accuracy fluctuated between 98.35% and 99.08%, and test loss varied in a narrow range between 0.0017 and 0.0066. Although these variations are not large in absolute terms, they indicate a slightly less stable generalization behavior compared to the model without dropout.

Overall, both models performed strongly, but the version without dropout slightly outperformed the regularized variant in terms of both peak accuracy and consistency. Based on these results, the model without dropout was selected as the final version for evaluation and integration into the MCTS agent.

5.6.3. Evaluation Metrics

To better assess the classification performances of the final model, we computed a confusion matrix and standard evaluation metrics focused on the tautology class. The confusion matrix obtained in the test set is the following:

Model	Epoch	Train Acc.	Test Acc.	Test Loss
No Dropout	17	99.90%	99.31%	0.0026
Dropout = 0.3	17	99.94%	99.08%	0.0035

Table 3
Comparison between model variants trained on the full dataset.

	Predicted: Non-Taut	Predicted: Taut
Actual: Non-Taut	1916	4
Actual: Taut	14	666

Table 4
Confusion matrix on the test set for the final model.

Based on this matrix, we derived the following metrics for the tautology class:

$$\begin{aligned}
\text{Precision} &= \frac{\text{TP}}{\text{TP} + \text{FP}} = \frac{666}{666 + 4} = 0.9941 \\
\text{Recall} &= \frac{\text{TP}}{\text{TP} + \text{FN}} = \frac{666}{666 + 14} = 0.9795 \\
\text{F1-score} &= \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} = \frac{2 \cdot 0.9941 \cdot 0.9795}{0.9941 + 0.9795} = 0.9867
\end{aligned}$$

These results confirm that the model achieves strong performance not only in overall accuracy, but also in terms of recall and precision on the tautology class — the more challenging and less frequent class in the dataset.

5.6.4. Qualitative Error Analysis

Despite the model's strong overall performance, a detailed inspection of the misclassifications reveals certain limitations in generalization and abstraction capabilities. Table 5 reports the list of misclassified formulas, grouped by type — false positives and false negatives — and accompanied by their predicted confidence scores.

These misclassifications reveal that the model struggles with recognizing tautologies that lack syntactic regularity or that require a deeper level of semantic abstraction. Even structurally trivial tautologies involving contradictions or logical identities may be missed if expressed in unusual forms.

These residual errors illustrate the limitations of purely statistical generalization. While the model performs well on structurally familiar inputs, it remains sensitive to syntactic variations — confirming the absence of true deductive reasoning and the reliance on learned syntactic heuristics.

5.7. Experiments with MCTS Agent

The previous evaluation revealed that the Tree-LSTM classifier exhibits limitations in abstraction and generalization — particularly when confronted with tautologies expressed in uncommon syntactic forms.

To test whether the learned value predictions could support structured inference, we embedded the Tree-LSTM classifier within a symbolic agent based on Monte Carlo Tree Search (MCTS). This agent navigates the proof space of the sequent calculus by applying inference rules to decompose sequents into subgoals. The aim was to evaluate the model not in isolation, but as a guide within a search process governed by logical constrains.

Experiments were conducted on the subset of 680 tautological formulas extracted from the test set, each encoded as a sequent with an empty set of premises and the formula as the conclusion. The agent's task was to find a proof for each tautology using a limited number of inference steps.

Two different policy-value functions were tested:

Formula	Confidence
False Positives	
$(((A_0 \to A_1) \lor ((\neg A_2) \to A_3)) \lor ((((A_4 \lor A_5) \land (A_6 \to A_7)) \to (A_0 \to (A_1 \lor A_2))) \lor A_3))$	0.8290
$((A_0 \lor A_1) \lor ((((A_2 \to A_3) \land A_4) \land ((\neg A_5) \land (A_6 \to A_7))) \to (A_0 \lor (A_1 \land (A_2 \land A_3)))))$	0.7575
$(((A_0 \to A_1) \to (A_2 \to A_3)) \lor (((A_4 \land (\neg \bot)) \to (\neg (A_5 \land A_6))) \lor (\neg A_0)))$	0.6575
$((((A_0 \to (A_1 \lor A_2)) \lor (A_3 \lor (A_4 \land A_5))) \lor A_6) \lor (((\neg (\neg A_7)) \land (A_0 \land (A_1 \to A_2))) \lor A_3))$	0.5837
False Negatives	
Formula	**Confidence**
$(((A_0 \to ((\neg A_1) \to (A_2 \land A_3))) \lor (A_4 \land ((A_5 \land A_6) \land A_7))) \lor A_0)$	0.1864
$(((A_0 \to (A_1 \land (A_2 \to A_3))) \land (((\neg A_4) \lor (A_5 \to A_6)) \lor (\neg A_7))) \to (A_0 \to A_1))$	0.0726
$((A_0 \land (A_1 \land ((\neg A_2) \to (A_3 \to A_4)))) \to ((\neg ((A_5 \lor A_6) \lor A_7)) \to A_0))$	0.4809
$((\bot \lor (((A_0 \land A_1) \land A_2) \land (A_3 \lor (A_4 \lor A_5)))) \to (\bot \lor (A_6 \to A_0)))$	0.1971
$(((\neg A_0) \lor (((A_1 \to A_2) \land (A_3 \lor A_4)) \land A_5)) \lor (A_6 \to ((A_7 \lor A_0) \lor A_1)))$	0.1938
$(((\bot \land (A_0 \to A_1)) \lor A_2) \to ((\bot \land (A_0 \to A_1)) \lor A_2))$	0.4884
$(((A_0 \to (A_1 \land A_2)) \lor (((\neg A_3) \land (A_4 \to A_5)) \land (A_6 \to A_7))) \lor A_0)$	0.1470
$(A_0 \to ((((A_1 \lor A_2) \to (A_3 \lor \bot)) \lor ((A_4 \lor A_5) \to A_6)) \lor A_0))$	0.1161
$((A_0 \to A_1) \lor (((A_2 \lor (A_3 \land A_4)) \to (A_5 \to (A_6 \lor A_7))) \lor A_0))$	0.2859
$((A_0 \to (((A_1 \lor A_2) \lor A_3) \to ((A_4 \lor A_5) \land (A_6 \land A_7)))) \lor A_0)$	0.0552
$(A_0 \lor ((((\neg A_1) \land (A_2 \lor A_3)) \lor (A_4 \lor A_5)) \lor (((A_6 \land \bot) \lor (A_0 \land A_1)) \to (\neg (A_2 \land A_3)))))$	0.0463
$((\neg ((A_0 \lor A_1) \land ((\neg A_2) \to (\neg A_3)))) \lor ((((A_4 \land A_5) \land A_6) \lor (A_7 \lor A_0)) \lor A_1))$	0.0096
$(((\neg (\bot \lor \bot)) \to ((\neg \bot) \land (\neg \bot))) \land (((\neg \bot) \land (\neg \bot)) \to (\neg (\bot \lor \bot))))$	0.4514
$(((A_0 \lor (A_1 \to (\bot \to A_2))) \lor A_3) \lor A_4)$	0.4908

Table 5
Misclassified formulas by the Tree-LSTM model: above, false positives (non-tautologies predicted as tautologies); below, false negatives (tautologies predicted as non-tautologies), along with the model's confidence scores. Notably, the sixth and the penultimate formulas among the false negatives were generated as instances of tautological schemata introduced during data augmentation.

- *Uniform policy* assigns equal prior to all applicable moves ad uses a constant value estimate of 0.5.
- *Tree-LSTM model policy* uses the Tree-LSTM classifier to estimate the probability of success for each subgoal generated by legal inference moves, and assigns move priors accordingly.

5.7.1. Uniform Policy Baseline

We conducted a parameter sweep over two key settings: the number of MCTS simulations per step $n_{\text{playout}} \in \{2, 3, 4, 5, 10\}$ and the maximum number of proof steps allowed max_steps $\in \{3, 4, 5\}$. The best performance achieved with the uniform policy was 63.24% accuracy at $n_{\text{playout}} = 10$ and max_steps $= 5$, as shown in Table 6.

5.7.2. Tree-LSTM Policy

We then evaluated the MCTS agent guided by the Tree-LSTM classifier using the same configuration values explored for the uniform policy. In this setup, the model assigns prior probabilities to inference rules based on the predicted provability of the resulting subgoals.

As shown in Table 6, the Tree-LSTM guided agent consistently outperformed the uniform baseline across all tested configurations. With $n_{\text{playout}} = 5$ and max_steps $= 5$, the model-guided agent reached a peak accuracy of 91.42%, compared to 56.76% for the uniform policy under the same conditions. Strong performance was achieved even with minimal simulations per step, suggesting that the classifier provides meaningful guidance that helps focus the search toward promising inference paths.

6. Related Work

Numerous studies have explored using machine-learning techniques to partially or fully assist in constructing proofs for logical systems. A survey can be found in [2]. Here, we review the work most closely related to ours in either aims or methodology:

n_playout	max_steps	Uniform Policy	Tree-LSTM Policy
10	5	63.24%	89.12%
5	5	56.76%	91.42%
3	5	48.97%	82.15%
10	4	47.50%	85.38%

Table 6
Comparison between Uniform and Tree-LSTM policies across selected MCTS configurations on 680 tautologies.

- Kusumoto et al. [11] is the work most similar to our. While both approaches are essentially logic-agnostic, we experiment with classical logic, while they focus on intuitionistic logic. Moreover, they employ Graph Neural Networks instead of Tree-LSTMs and rely on a simple greedy search rather than MCTS.
- Lample et al. [12] is, to our knowledge, the most advanced effort pursuing goals comparable to those of the *DeepSAT* project. They, too, use MCTS adapted to hypertrees, but target richer logical systems such as MetaMATH, Lean, and Equation. Their proof-space exploration is guided by natural-language models built on the `seq2seq` paradigm [13]. Conversely, we want to avoid natural-language processing to keep our models as lightweight as possible.

7. Conclusion and Future Work

In this paper, we have shown that it is possible to train a recurrent neural network to predict the validity of propositional formulas. Furthermore, we have shown that a network trained in this manner can effectively guide the exploration of a Monte Carlo Tree Search (MCTS) agent in the task of finding proofs for sequents in propositional logic. This represents an initial step in the *DeepSAT* project, which ultimately aims to develop a system capable of automatically proving theorems in type theory.

One might wonder why formulas of the form $\phi_1 \to \phi_2 \to \ldots \to \phi_n$ which frequently arise when translating sequents into single formulas, are not well represented in the training data. This is a consequence of the fact that, in this initial phase, the dataset was constructed independently of the proof tasks. However, this limitation is expected to be naturally overcome in the next phase of the project: by switching to reinforcement learning, the network will be trained directly on formulas encountered during proof search episodes, which include precisely those implicative structures typical of sequents. In this way, the training distribution will become progressively aligned with the actual inference tasks the model is meant to solve.

In the next phase of our project, we aim to address this and other limitations by transitioning from supervised to reinforcement learning. The dataset of formulas currently used to train the neural network in a supervised fashion will instead serve as a foundation for training, via reinforcement learning, a neural network that computes the external *policy* of the MCTS agent. In parallel, we will continue to train the network responsible for the external *value* estimate using supervised learning, with the training data drawn from the formulas that appear in the intermediate nodes of the proof trees.

Additionally, we plan to explore alternative neural network architectures, potentially incorporating attention mechanisms as in [14] to improve the model's ability to focus on relevant substructures within complex formulas, or non-shared biases to improve generalization, as in [15, 16]. Also, the use of variants of MCTS for and-or trees, such as [17, 18], may be explored. Finally, we plan to use static analysis techniques [19] to formally verify the correctness of the code.

Acknowledgments

This work was partially supported by the PNRR project FAIR – Future AI Research (PE00000013), Spoke 9 - Green-aware AI, under the NRRP MUR program funded by the NextGenerationEU; Istituto Nazionale di Alta Matematica – INdAM groups GNCS and GNSAGA.

Declaration on Generative AI

During the preparation of this work, the authors used X-GPT-4 to: Drafting content, Paraphrase and reword, Improve writing style. After using these tools, the authors reviewed and edited the content as needed and take full responsibility for the publication's content.

References

[1] J. K. Fichte, D. L. Berre, M. Hecher, S. Szeider, The silent (r)evolution of SAT, Commun. ACM 66 (2023) 64–72. doi:10.1145/3560469.

[2] Z. Li, J. Sun, L. Murphy, Q. Su, Z. Li, X. Zhang, K. Yang, X. Si, A Survey on Deep Learning for Theorem Proving, 2024. URL: https://openreview.net/forum?id=zlw6AHwukB#discussion.

[3] M. Crouse, I. Abdelaziz, B. Makni, S. Whitehead, C. Cornelio, P. Kapanipathi, K. Srinivas, V. Thost, M. Witbrock, A. Fokoue, A Deep Reinforcement Learning Approach to First-Order Logic Theorem Proving, Proceedings of the AAAI Conference on Artificial Intelligence 35 (2021) 6279–6287. doi:10.1609/aaai.v35i7.16780.

[4] K. S. Tai, R. Socher, C. D. Manning, Improved Semantic Representations From Tree-Structured Long Short-Term Memory Networks, in: C. Zong, M. Strube (Eds.), Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers), Association for Computational Linguistics, Beijing, China, 2015, pp. 1556–1566. doi:10.3115/v1/P15-1150.

[5] L. Kocsis, C. Szepesvári, Bandit based Monte-Carlo planning, in: Proceedings of the 17th European Conference on Machine Learning, ECML'06, Springer-Verlag, Berlin, Heidelberg, 2006, p. 282–293. doi:10.1007/11871842_29.

[6] S. Hochreiter, J. Schmidhuber, Long short-term memory, Neural Computation 9 (1997) 1735–1780. doi:10.1162/neco.1997.9.8.1735.

[7] T. Ridnik, E. Ben-Baruch, A. Noy, L. Zelnik-Manor, Asymmetric loss for multi-label classification, 2021. doi:10.48550/arXiv.2009.14119.

[8] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, et al., Mastering the game of Go without human knowledge, Nature 550 (2017) 354–359. doi:10.1038/nature24270.

[9] M. Schuster, K. K. Paliwal, Bidirectional recurrent neural networks, IEEE Transactions on Signal Processing 45 (1997) 2673–2681. doi:10.1109/78.650093.

[10] D. P. Kingma, J. Ba, Adam: A method for stochastic optimization, International Conference on Learning Representations (ICLR) (2015). URL: https://arxiv.org/abs/1412.6980. arXiv:1412.6980.

[11] M. Kusumoto, K. Yahata, M. Sakai, Automated Theorem Proving in Intuitionistic Propositional Logic by Deep Reinforcement Learning, 2018. doi:10.48550/arXiv.1811.00796.

[12] G. Lample, M.-A. Lachaux, T. Lavril, X. Martinet, A. Hayat, G. Ebner, A. Rodriguez, T. Lacroix, Hypertree proof search for neural theorem proving, in: Proceedings of the 36th International Conference on Neural Information Processing Systems, NIPS '22, Curran Associates Inc., Red Hook, NY, USA, 2022, pp. 26337–26349. doi:10.5555/3600270.3602180.

[13] I. Sutskever, O. Vinyals, Q. V. Le, Sequence to sequence learning with neural networks, in: Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2, NIPS'14, MIT Press, Cambridge, MA, USA, 2014, p. 3104–3112. doi:10.5555/2969033.2969173.

[14] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, I. Polosukhin, Attention is all you need, in: Advances in Neural Information Processing Systems, 2017, pp. 5998–6008. doi:10.5555/3295222.3295349.

[15] C. Metta, M. Fantozzi, A. Papini, G. Amato, M. Bergamaschi, S. G. Galfre, A. Marchetti, M. Veglio, M. Parton, F. Morandin, Increasing biases can be more efficient than increasing weights, in:

IEEE Winter Conference on Applications of Computer Vision, 2024, p. 2798 – 2807. doi:10.1109/WACV57701.2024.00279.

[16] C. Metta, M. Fantozzi, A. Papini, G. Amato, M. Bergamaschi, A. Fois, S. G. Galfrè, A. Marchetti, M. Vegliò, M. Parton, F. Morandin, Increasing biases can be more efficient than increasing weights, Advances in Data Analysis and Classification (2025). doi:10.1007/s11634-025-00649-2.

[17] T. M. Moerland, J. Broekens, A. Plaat, C. M. Jonker, Monte carlo tree search for asymmetric trees, 2018. doi:10.48550/arXiv.1805.09218.

[18] E. Doe, M. H. M. Winands, D. J. N. J. Soemers, C. Browne, Combining Monte-Carlo tree search with proof-number search, in: 2022 IEEE Conference on Games (CoG), IEEE Press, 2022, p. 206–212. doi:10.1109/CoG51982.2022.9893635.

[19] G. Amato, M. Rubino, F. Scozzari, Inferring linear invariants with parallelotopes, Sci. Comput. Program. 148 (2017) 161–188. doi:10.1016/j.scico.2017.05.011.