Studies in Logic, Vol. 19, No. 3 (2026): 59–91
PII: 1674-3202(2026)-03-0059-33

Linear-Time Temporal Logic in AI-Based Computational Tree Models*

Chen Liu

Abstract. The integration of the Linear Temporal Logic (LTL) in AI-based computational tree models is an effective form of formal verification of Artificial Intelligence systems. But classical model checking methods do not scale because of the state-space explosion. In this paper, we propose a novel data driven methodology backed by deep learning for efficient approximation of LTL model checking. The Temporal Convolutional Graph Network (TCGN) is a framework for deep learning that integrates graph neural networks with temporal logic in order to verify the behaviour of complex systems. We conduct experiments on synthetic and real-world datasets, showing that TCGN achieves an accuracy of 98% and outperforms traditional methods like NuSMV both in effectiveness instantaneous verification and efficiency. The TCGN can also achieve real time large scale AI system verification due to its strong robustness under adversarial perturbation.

1 Introduction

Linear Temporal Logic (LTL) provides a rigorous formalism for specifying and verifying system behaviours over time.([7, 19]) Originally developed for hardware verification, LTL has been widely adopted in AI for specifying properties in planning, robotics, and control systems.([8]) Unlike branching-time logics, LTL focuses on properties that must hold along each linear progression of states ([11]), making it suitable for applications where system behaviour is observed as sequences of states or transitions ([23]).

The AI community has increasingly focused on LTL over finite traces (LTLf), which enables formalisation of finite task executions such as motion planning.([33, 34]) LTLf allows specification of temporal requirements like "a goal state is eventually reached" within finite horizons ([2]), making it essential for reasoning about plans with finite actions ([7, 18]). The integration of LTL into AI-based computational tree models such as Kripke structures representing non-deterministic system behavior provides a powerful formalism for specifying temporal constraints on AI-generated plans and learned policies. However, practical deployment of formal verification faces a fundamental challenge: high computational cost, especially under

Received 2026-03-20

Chen Liu School of Marxism, Yangzhou University
lauchan@yzu.edu.cn

*This research was funded by the National Social Science Fund of China (Grant No. 23FZXB053).

non-determinism.([27]) Classical automata-theoretic model checking suffers from the state-space explosion problem, where the state space grows exponentially with system variables, making verification intractable for large-scale AI systems.([13])

This challenge is further intensified by the opacity and complexity of deep learning components in modern AI. Such models are difficult to represent symbolically, creating a disconnect between implementation and formal abstraction. As a result, conventional formal methods cannot be directly applied to neural network behaviors, motivating new approaches capable of handling both symbolic and sub-symbolic components. However, practical verification faces a fundamental obstacle: the state-space explosion problem. Classical model checking algorithms suffer from exponential growth in state space, rendering them intractable for large AI systems. Additionally, the opacity of deep learning components makes direct application of formal methods difficult.

To address these challenges, we propose a data-driven methodology that reformulates LTL model checking as a supervised learning problem, enabling amortised computation after training. Our key contributions are: (1) formulating LTL checking over computational trees as a learning problem, (2) introducing TCGN, a formula-conditioned temporal graph network integrating LTL embeddings with graph attention, (3) proposing temporal adversarial robustness metrics tailored to verification, and (4) demonstrating millisecond-scale verification with up to 98.7% agreement with NuSMV on two benchmarks. Specifically, we make the following key contributions:

1. We formulate LTL model checking over computational trees as a supervised learning problem, shifting from exact symbolic algorithms to data-driven approximation that scales to large systems.
2. We introduce the Temporal Convolutional Graph Network (TCGN), a formula-conditioned temporal graph network that integrates LTL embeddings with graph attention mechanisms. TCGN represents computational tree models as graphs, encodes LTL formulas compositionally, and performs multi-scale temporal convolution guided by formula semantics.
3. We propose temporal adversarial robustness metrics and attacks specifically tailored to verification tasks, including state deletion, label flipping, transition perturbation, and temporal reordering. These enable systematic evaluation of model stability under noisy conditions.
4. We demonstrate millisecond scale approximate verification on two benchmarks a synthetic grid-world navigation environment and a real-world network packet scheduler achieving up to 98.7% agreement with the NuSMV symbolic model checker while providing orders-of-magnitude speedup.

2 Literature Review

The application of temporal logic in formal verification, especially for specifying and verifying system requirements, has led to its widespread adoption due to its expressiveness and natural language resemblance across diverse fields like dynamic systems, robotics, and biology.([12, 29]) This has led to the development of various specialised temporal logics, such as LTL over finite traces, which accommodates the definition and reasoning about events taking place over a finite execution.([7]) LTLf, for example, provides a more natural and efficient framework for specifying and verifying declarative constraint-based systems within finite computer paths, in areas such as business process management and robotic task planning.([7, 21]) Recent work has increasingly explored learning-based and large-scale neural tools for logical reasoning and formal verification. In Boolean satisfiability, NeuroSAT demonstrated that a message-passing neural network can learn SAT solving behaviour from satisfiability labels and generalise to problem distributions beyond those seen during training.([24]) Subsequent work incorporated graph neural networks and reinforcement learning into SAT solving, including learned local-search heuristics for Boolean satisfiability ([35]) and Graph-Q-SAT, which learns branching heuristics for SAT solvers using graph networks and Q-learning ([17]). Related neural architectures have also been proposed for Circuit-SAT, where differentiable graph-based models are trained to solve circuit satisfiability problems.([1]) In temporal logic, SAT-based LTLf satisfiability checking reduces LTLf reasoning to propositional SAT and path-search problems ([18]), while recent Transformer and Mamba architectures have been used to generate LTL specifications from traces for specification-mining tasks ([12]). Neural methods have also been applied to higher-order logic and theorem proving, including DeepHOL for higher-order theorem proving ([3]) and transformer-based proof generation in GPT-f ([22]). These studies show a broader trend toward using scalable neural or learning-guided tools for symbolic reasoning. However, most of them focus on SAT, theorem proving, or specification generation rather than formula-conditioned LTLf trace checking over execution paths. Our work addresses this gap by learning an amortised approximation of LTLf satisfaction using a temporal graph network conditioned directly on the input formula.

This expressiveness enables us to specify complex task sequences and safety conditions which must hold true during the lifetime of a robot, or the completion cycle of a business process. Also, the shift from conventional LTL to LTLf is critical for AI applications whose tasks have clearly defined starting and ending states and require a logic that can precisely capture these finite temporal scopes.([13]) This happens even more so in the case of automated planning and synthesis, where agents must incur a finite number of actions to achieve their goals. LTLf allows for the formal definition of goals and their verification. LTLf is a more suitable choice than

previous formal scientific tools due to the increasing complexity of AI systems and the increasing need for self-verifying systems that are robust and safe with respect to their own behaviour. The process of transforming LTLf and its extension LDLf into deterministic finite-state automata helps solve the stages of formula satisfaction, and it is instrumental in the general development of non-Markovian reward decision processes.([10, 31]) Several studies contribute to enhancing AI system verification, propose saliency-based metrics for fairness ([16]), while focusing on balancing data augmentation for better generalisation ([15]). Kumar *et al.* address mitigating biases in AI models, which is relevant for robust verification ([14]) and highlight the use of data augmentation in classification tasks ([30]). Discuss efficient sensor data assessment, which could improve real-time verification models.([25])

This conversion allows for effective run-time monitoring and verification that makes LTLf a vital constituent in the creation of AI systems that can self-assess and comply with complex operational rules.([20]) Moreover, while LTL and LTLf are powerful propositional logics for finite-state systems, their limitations in modeling infinite state spaces necessitate the consideration of First-Order Linear Temporal Logic.([9, 28]) This extension allows one to quantify over data domains, therefore allowing formal specification of and verification of systems with an unbounded number of states or complex data structures, which are commonplace in advanced AI applications. This new logic lets us better show the properties of the system. For example, we might want to say that "all agents eventually reach a certain place" or that "no agent ever goes into a certain place". It does so without regard to how many agents there are or what we call those places.

3 Methodology

This section presents our framework for LTL trace checking over execution paths from computational tree models. We first provide a precise formulation of the problem we address, clarifying the distinction from full model checking. We then detail each component of our proposed TCGN architecture, including graph construction, featurization, formula embedding, and the novel temporal graph convolution operator. Finally, we describe our training protocol and introduce a theoretically grounded metric for adversarial robustness.

3.1 Problem formulation

Let a *computational tree model* be defined as a Kripke structure

$$M = (S, S_0, T, A, L),$$

where:

- S is a finite set of states;
- $S_0 \subseteq S$ is the set of initial states;
- $T \subseteq S \times S$ is a left-total transition relation;
- A is a finite set of atomic propositions;
- $L : S \to 2^A$ is a labeling function.

An *execution path* (or *trace*) π of M is a finite sequence of states $\pi = s_0, s_1, \ldots, s_n$ such that $s_0 \in S_0$ and $(s_i, s_{i+1}) \in T$ for all $0 \leq i < n$. Unlike full model checking, which requires universal quantification over *all* infinite paths, we consider the *trace checking problem*: Given a single execution path π and an LTL formula ϕ interpreted over finite traces (LTLf), determine whether $\pi \models \phi$.

Definition 1 (Syntax of LTLf Formulas)**.** Let A be a finite set of atomic propositions. The set of well-formed LTLf formulas over A is generated by the following grammar:

$$\varphi ::= p \mid \top \mid \bot \mid \neg\varphi \mid (\varphi_1 \wedge \varphi_2) \mid (\varphi_1 \vee \varphi_2) \mid \mathbf{X}\varphi \mid (\varphi_1 \mathbf{U} \varphi_2) \mid (\varphi_1 \mathbf{R} \varphi_2),$$

where $p \in A$, $\top$ denotes truth, $\bot$ denotes falsehood, $\neg$ is negation, $\wedge$ and $\vee$ are Boolean conjunction and disjunction, **X** is the next operator, **U** is the until operator, and **R** is the release operator.

The eventually and always operators are treated as derived operators:

$$\mathbf{F}\varphi \equiv \top \mathbf{U} \varphi, \qquad \mathbf{G}\varphi \equiv \neg\mathbf{F}\neg\varphi.$$

When weak until is used in the experiments, it is also treated as a derived operator:

$$\varphi_1 \mathbf{W} \varphi_2 \equiv (\varphi_1 \mathbf{U} \varphi_2) \vee \mathbf{G}\varphi_1.$$

Parentheses are used where necessary to remove ambiguity. We assume the usual precedence order: Unary temporal and negation operators bind most strongly, followed by **U**, **R**, and **W**, followed by $\wedge$, and finally $\vee$.

The size $|\varphi|$ of a formula is the number of nodes in its syntax tree. The temporal depth $\text{depth}(\varphi)$ is the maximum nesting depth of temporal operators in φ. For example, $\text{depth}(p) = 0$, $\text{depth}(\mathbf{X}\varphi) = 1 + \text{depth}(\varphi)$, and $\text{depth}(\varphi_1 \mathbf{U} \varphi_2) = 1 + \max(\text{depth}(\varphi_1), \text{depth}(\varphi_2))$.

Definition 2 (LTLf Semantics over Finite Traces)**.** Let $\pi = s_0, s_1, \ldots, s_n$ be a finite trace of length $|\pi| = n + 1$, and let π^i denote the suffix starting at position i. The

satisfaction relation $\models$ for LTLf is defined inductively:

$$\begin{aligned}
&\pi \models p && \iff p \in L(s_0)\\
&\pi \models \neg\phi && \iff \pi \not\models \phi\\
&\pi \models \phi_1 \wedge \phi_2 && \iff \pi \models \phi_1 \text{ and } \pi \models \phi_2\\
&\pi \models \bigcirc\phi && \iff |\pi| > 1 \text{ and } \pi^1 \models \phi\\
&\pi \models \phi_1 \mathcal{U} \phi_2 && \iff \exists k \in [0, |\pi| - 1] \text{ s.t. } \pi^k \models \phi_2\\
& && \quad \text{and } \forall j < k, \pi^j \models \phi_1
\end{aligned}$$

Derived operators are defined as standard: $\Diamond\phi \equiv \top\mathcal{U}\phi$, $\Box\phi \equiv \neg\Diamond\neg\phi$.

We reformulate LTLf trace checking as a supervised binary classification problem. Let θ denote the complete set of learnable parameters in TCGN, including the proposition embeddings, formula-encoder weights, temporal graph-convolution weights, attention parameters, pooling parameters, and MLP readout weights.

For a finite trace π and an LTLf formula φ, the neural verifier is defined as

$$\hat{y} = f_\theta(\pi, \varphi), \qquad f_\theta \colon (\Pi \times \Phi) \to [0, 1],$$

where Π is the set of finite execution traces, Φ is the set of LTLf formulas, and $\hat{y}$ is the predicted probability that $\pi \models \varphi$. The subscript θ indicates that the output of f_θ depends on parameters learned from data. Thus, f_θ is not an exact symbolic model checker; it is a neural approximation of the Boolean satisfaction indicator $y = \mathbf{1}[\pi \models \varphi]$.

Given a labelled dataset

$$D = \{(\pi_i, \varphi_i, y_i)\}_{i=1}^{N},$$

where $y_i = \mathbf{1}[\pi_i \models \varphi_i]$ is computed using the symbolic verifier NuSMV, training seeks parameters θ^* that minimise the empirical binary cross-entropy loss:

$$\theta^* = \arg\min_\theta -\frac{1}{N} \sum_{i=1}^{N} \left[y_i \log f_\theta(\pi_i, \varphi_i) + (1 - y_i) \log \left(1 - f_\theta(\pi_i, \varphi_i)\right) \right].$$

At inference time, the trained model predicts that π satisfies φ when $f_{\theta^*}(\pi, \varphi) \geq 0.5$, and predicts non-satisfaction otherwise.

3.2 Graph representation of execution traces

Unlike full Kripke structures with branching nondeterminism, a single execution trace $\pi = s_0, s_1, \ldots, s_n$ is represented as a directed path graph $G = (V, E, X)$.

The three components of G are defined as follows.

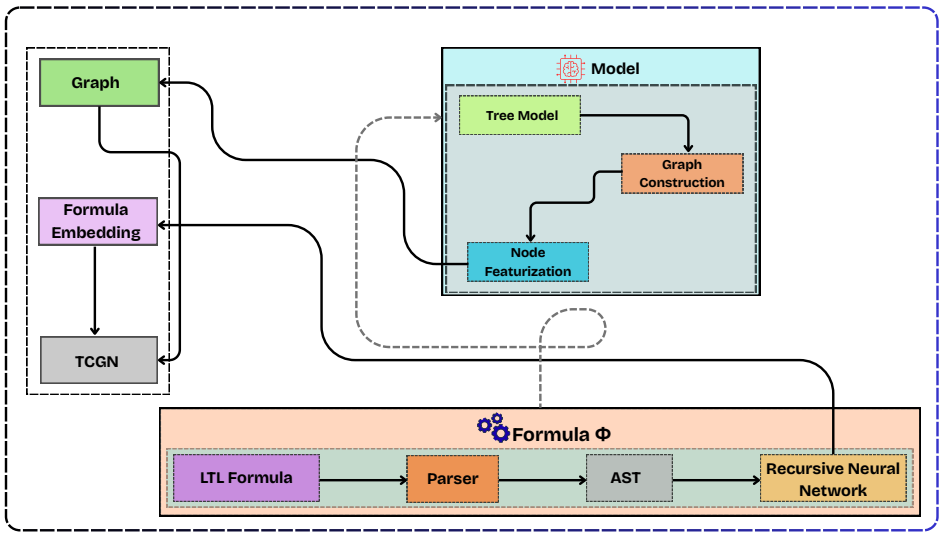

Figure 1: System architecture diagram consists of two encoding streams: (1) the computational graph encoder processes the execution trace as a directed path graph; (2) the formula encoder embeds the LTLf formula via an AST recursive network. These representations are fused through temporal graph convolution layers with formula-guided attention, followed by hierarchical pooling and MLP readout.

First,

$$V = \{v_0, v_1, \ldots, v_n\}$$

is the set of graph nodes. Each node v_i corresponds to exactly one state s_i in the trace. Thus, the graph has one node for each time step of the execution.

Second,

$$E = \{(v_i, v_{i+1}) \mid 0 \leq i < n\}$$

is the set of directed temporal edges. The edge (v_i, v_{i+1}) means that state s_{i+1} occurs immediately after state s_i in the execution trace. Therefore, E preserves the temporal order of the trace.

Third,

$$X \in \mathbb{R}^{|V| \times d}$$

is the node-feature matrix. The i-th row of X, denoted by $x_{v_i} \in \mathbb{R}^d$, is the vector representation of node v_i, or equivalently, of state s_i. The parameter d denotes the node-feature dimension, namely the number of numerical features used to represent each state before it is processed by the temporal graph neural network. In our model, d is a user-selected embedding dimension and is kept even so that the representation can be split into true-proposition and false-proposition components.

Equivalently, the feature matrix can be written as

$$X = \begin{bmatrix} x_{v_0}^\top \\ x_{v_1}^\top \\ \vdots \\ x_{v_n}^\top \end{bmatrix} = \begin{bmatrix} x_{s_0}^\top \\ x_{s_1}^\top \\ \vdots \\ x_{s_n}^\top \end{bmatrix} \in \mathbb{R}^{(n+1)\times d}.$$

Thus, X converts the symbolic state labels in the trace into continuous vectors that can be used as input to TCGN.

3.2.1 Node featurization

The purpose of node featurization is to transform the logical labeling $L(s) \subseteq A$ of each state into a continuous vector representation. Because LTLf satisfaction depends not only on the propositions that are true in a state but also on those that are false, we explicitly encode both types of information.

For each atomic proposition $a \in A$, we will learn two separate embeddings: $e_a^+ \in \mathbb{R}^{d/2}$, which is used when a is true in state s, and $e_a^- \in \mathbb{R}^{d/2}$, which is used when a is false in state s. These two embeddings are concatenated to form the final d-dimensional feature vector of the state.

The purpose of node featurization is to convert the symbolic label $L(s) \subseteq A$ of each state into a continuous vector that can be processed by the neural network. Here, A is the finite set of atomic propositions, and $L(s)$ is the subset of propositions that are true at state s.

For each atomic proposition $a \in A$, we define two trainable embedding vectors:

$$e_a^+ \in \mathbb{R}^{d/2}, \qquad e_a^- \in \mathbb{R}^{d/2}.$$

The vector e_a^+ is used when proposition a is true at a state, i.e., $a \in L(s)$. The vector e_a^- is used when proposition a is false at a state, i.e., $a \notin L(s)$. These vectors are not manually fixed encodings; they are learnable parameters of the model, initialized randomly and updated during training via backpropagation together with the other TCGN parameters.

For a given state s, the node feature vector $x_s \in \mathbb{R}^d$ is computed by aggregating the embeddings of the true and false propositions separately:

$$x_s = \left(\sum_{a \in L(s)} e_a^+ \right) \parallel \left(\sum_{a \in A \setminus L(s)} e_a^- \right),$$

where $\parallel$ denotes vector concatenation. The first half of x_s summarizes the propositions that are true at state s, while the second half summarizes the propositions that are false at state s. Thus, x_s has dimension d.

This design is useful because LTLf formulas depend on both positive and negative information. For example, evaluating a safety formula such as $\mathbf{G}\neg p$ requires the model to recognize states where p is false. The vector e_a^- should not be interpreted as a new logical proposition $\neg a$; rather, it is a neural feature used to represent the absence of a from the state label. During training, the model learns embeddings that are useful for predicting whether the trace satisfies the input formula.

For example, suppose $A = \{p, q, r\}$ and $L(s) = \{p, r\}$. Then propositions p and r are true at state s, while q is false. The feature vector of state s is therefore

$$x_s = (e_p^+ + e_r^+) \parallel e_q^-.$$

This vector is subsequently used as the input representation of the node corresponding to state s in the trace graph.

3.3 LTLf formula embedding

LTLf formulas exhibit compositional structure that must be preserved in the embedding space. We parse ϕ into an abstract syntax tree (AST) where:

- Leaf nodes correspond to atomic propositions $p \in A$ or Boolean constants $\top, \bot$;
- Internal nodes correspond to unary operators ($\neg, \bigcirc, \Diamond, \Box$) or binary operators ($\wedge, \vee, \mathcal{U}, \mathcal{R}$).

Let $\mathcal{T}_\phi$ denote the AST of ϕ. Each node $u \in \mathcal{T}_\phi$ is associated with a hidden state $\mathbf{h}_u \in \mathbb{R}^f$. For leaf nodes, $\mathbf{h}_u = \mathbf{W}_{\text{leaf}} \cdot \mathbf{e}_{prop(u)}$, where $\mathbf{e}_{prop(u)} \in \mathbb{R}^{d/2}$ is the proposition embedding (shared with the graph encoder). For internal nodes, the embedding is computed recursively:

- **Unary operator** $\psi = \odot\psi_1$:

$$\mathbf{h}_\psi = \tanh\left(\mathbf{W}_\odot \mathbf{h}_{\psi_1} + \mathbf{b}_\odot\right)$$

- **Binary operator** $\psi = \psi_1 \otimes \psi_2$:

$$\mathbf{h}_\psi = \tanh\left(\mathbf{W}_\otimes [\mathbf{h}_{\psi_1} \parallel \mathbf{h}_{\psi_2}] + \mathbf{b}_\otimes\right)$$

where $\mathbf{W}_\odot \in \mathbb{R}^{f \times f}$, $\mathbf{W}_\otimes \in \mathbb{R}^{f \times 2f}$, and $\mathbf{b}_\odot, \mathbf{b}_\otimes \in \mathbb{R}^f$ are learnable parameters. The final formula embedding $\mathbf{e}_\phi \in \mathbb{R}^f$ is the hidden state at the root node.

Crucially, this encoding is *semantically aware*: Because the recursive computation mirrors the syntactic structure, syntactically similar formulas will have similar embeddings. However, semantic equivalence (e.g., $\Diamond p \equiv \neg\Box\neg p$) is not guaranteed by construction; we rely on the training process to align such equivalent representations.

Algorithm 1 LTLf AST Recursive Encoder

Require: AST $\mathcal{T}_\phi$, embedding matrix $\mathbf{E}$
Require: Operator weights $\mathbf{W}_{op}$, biases $\mathbf{b}_{op}$
Require: Leaf projection $\mathbf{W}_{\text{leaf}}$
Ensure: Formula embedding $\mathbf{e}_\phi$

function EncodeFormula(T)
  $root \leftarrow$ getRoot(T)
  return EncodeNode(root)
end function
function EncodeNode(node)
  if type$(node) \in \{$ATOMIC, $\top, \bot\}$ **then**
    $p \leftarrow$ getProposition$(node)$
    return $\mathbf{W}_{\text{leaf}} \cdot \mathbf{E}[p]$
  else if type$(node) \in \{$NOT, NEXT, EVENTUALLY, ALWAYS$\}$ **then**
    $child \leftarrow$ getChild$(node)$
    $\mathbf{h} \leftarrow$ EncodeNode$(child)$
    return $\tanh(\mathbf{W}_{op}\mathbf{h} + \mathbf{b}_{op})$
  else
    $(c_1, c_2) \leftarrow$ getChildren$(node)$
    $\mathbf{h}_1 \leftarrow$ EncodeNode(c_1)
    $\mathbf{h}_2 \leftarrow$ EncodeNode(c_2)
    $\mathbf{h} \leftarrow [\mathbf{h}_1 \parallel \mathbf{h}_2]$
    return $\tanh(\mathbf{W}_{op}\mathbf{h} + \mathbf{b}_{op})$
  end if
end function

Following Definition 1, each LTLf formula φ is parsed into an abstract syntax tree T_φ, in which the leaf nodes correspond to atomic propositions $p \in A$ or Boolean constants $\top$ and $\bot$, and the internal nodes correspond to Boolean operators ($\neg, \wedge, \vee$) or temporal operators (**X**, **U**, **R**, **F**, **G**, **W**).

3.4 Temporal graph convolutional network

The core of our architecture is a novel graph convolutional operator designed specifically for temporal path graphs with formula conditioning. Let $\mathbf{h}_v^{(l)} \in \mathbb{R}^{d_l}$ denote the hidden state of node v at layer l, with $\mathbf{h}_v^{(0)} = \mathbf{x}_v$ (the input feature vector). Each layer performs a *temporal neighbourhood aggregation* conditioned on the formula embedding $\mathbf{e}_\phi$.

3.4.1 Temporal convolution operator

For a path graph, each node v_i (except the final node) has exactly one successor v_{i+1} and one predecessor v_{i-1} (except the initial node). This linear structure suggests a natural temporal convolution: The representation of a state should depend on its own features, the features of its immediate successor, and the global temporal context provided by the formula.

We define the update rule as:

$$\mathbf{h}_v^{(l+1)} = \sigma\left(\underbrace{\mathbf{W}_{\text{self}}^{(l)}\mathbf{h}_v^{(l)}}_{\text{self}} + \underbrace{\mathbf{W}_{\text{next}}^{(l)}\mathbf{h}_{\text{succ}(v)}^{(l)}}_{\text{temporal neighbor}} + \underbrace{\mathbf{T}^{(l)}(\mathbf{e}_\phi)}_{\text{temporal context}} \right),$$

where:

- $\mathbf{W}_{\text{self}}^{(l)}, \mathbf{W}_{\text{next}}^{(l)} \in \mathbb{R}^{d_{l+1} \times d_l}$ are learnable weight matrices;
- $\text{succ}(v)$ denotes the unique successor of v (if v is not terminal);
- $\mathbf{T}^{(l)} : \mathbb{R}^f \to \mathbb{R}^{d_{l+1}}$ is a learned linear projection:
$$\mathbf{T}^{(l)}(\mathbf{e}_\phi) = \mathbf{W}_{\text{temp}}^{(l)}\mathbf{e}_\phi + \mathbf{b}_{\text{temp}}^{(l)};$$
- σ is an activation function (ReLU for hidden layers, identity for output layer).

For terminal nodes with no successor, the $\mathbf{W}_{\text{next}}$ term is omitted. This operator can be viewed as a form of causal convolution along the temporal dimension, with the formula embedding providing global bias.

3.4.2 Formula-guided attention

While the temporal convolution operator captures local sequential dependencies, long-range temporal dependencies (e.g., for the $\mathcal{U}$ operator) require selective attention to relevant future states. We extend the basic operator with an attention mechanism that weights the contribution of the successor based on the formula context.

Let $\alpha_v \in [0, 1]$ denote the attention weight on the successor of v, computed as:

$$\alpha_v = \sigma\left(\mathbf{a}^T \cdot \text{LeakyReLU}\left(\mathbf{W}_{\text{att}}[\mathbf{h}_v^{(l)} \parallel \mathbf{h}_{\text{succ}(v)}^{(l)} \parallel \mathbf{e}_\phi]\right)\right),$$

where $\mathbf{W}_{\text{att}} \in \mathbb{R}^{d_{\text{att}} \times (2d_l + f)}$, $\mathbf{a} \in \mathbb{R}^{d_{\text{att}}}$ are learnable parameters, and σ is the sigmoid function ensuring $\alpha_v \in [0, 1]$.

The attended update rule becomes:

$$\mathbf{h}_v^{(l+1)} = \sigma\left(\mathbf{W}_{\text{self}}^{(l)}\mathbf{h}_v^{(l)} + \alpha_v \cdot \mathbf{W}_{\text{next}}^{(l)}\mathbf{h}_{\text{succ}(v)}^{(l)} + \mathbf{T}^{(l)}(\mathbf{e}_\phi)\right).$$

This mechanism allows the model to selectively propagate information along the temporal dimension based on the verification task. For a safety property $\Box\neg p$, the attention weights remain low as there is no need to propagate information far; for a liveness property $\Diamond q$, attention increases to propagate information toward goal states.

3.4.3 Multi-scale temporal encoding

Temporal logic properties operate at multiple time scales: The ◯ operator examines the immediate next state, ◇ may require looking arbitrarily far ahead, and □ requires invariance holding across all states. To capture this hierarchy, we maintain three parallel hidden states per node, corresponding to short-term, medium-term, and long-term temporal receptive fields.

Let $\mathbf{h}_v^{(l,s)}, \mathbf{h}_v^{(l,m)}, \mathbf{h}_v^{(l,\ell)}$ denote the short, medium, and long-term representations at layer l. Each is updated with different temporal aggregation horizons:

$$\begin{aligned}
\mathbf{h}_v^{(l+1,s)} &= \text{TConv}_1\left(\mathbf{h}_v^{(l,s)}, \mathbf{h}_{\text{succ}(v)}^{(l,s)}, \mathbf{e}_\phi\right) \\
\mathbf{h}_v^{(l+1,m)} &= \text{TConv}_5\left(\mathbf{h}_v^{(l,m)}, \mathbf{h}_{\text{succ}(v)}^{(l,m)}, \ldots, \mathbf{h}_{\text{succ}^5(v)}^{(l,m)}, \mathbf{e}_\phi\right) \\
\mathbf{h}_v^{(l+1,\ell)} &= \text{TConv}_{\text{attn}}\left(\mathbf{h}_v^{(l,\ell)}, \{\mathbf{h}_{\text{succ}^k(v)}^{(l,\ell)}\}_{k=1}^{K_{\max}}, \mathbf{e}_\phi\right),
\end{aligned}$$

where:

- TConv_k denotes temporal convolution with a kernel of size k (i.e., aggregating k future states via mean pooling);
- $\text{TConv}_{\text{attn}}$ uses a learnable attention mechanism over all future states up to horizon $K_{\max}$ (set to trace length during training);
- $\text{succ}^k(v)$ denotes the k-th successor of v.

After L layers, the three representations are fused via:

$$\mathbf{h}_v^{(L)} = \mathbf{W}_{\text{fuse}}\left[\mathbf{h}_v^{(L,s)} \parallel \mathbf{h}_v^{(L,m)} \parallel \mathbf{h}_v^{(L,\ell)}\right] + \mathbf{b}_{\text{fuse}},$$

where $\mathbf{W}_{\text{fuse}} \in \mathbb{R}^{d_L \times 3d_L}$ and $\mathbf{b}_{\text{fuse}} \in \mathbb{R}^{d_L}$.

Algorithm 2 Temporal Graph Convolution Layer (Formula-Conditioned)

Require: Node features $\{\mathbf{h}_v^{(l)}\}_{v \in V}$, formula embedding $\mathbf{e}_\phi$, adjacency function $\text{succ}(\cdot)$, layer index l

Ensure: Updated node features $\{\mathbf{h}_v^{(l+1)}\}_{v \in V}$

$\mathbf{W}_{\text{self}}, \mathbf{W}_{\text{next}}, \mathbf{W}_{\text{att}}, \mathbf{a}, \mathbf{W}_{\text{temp}}, \mathbf{b}_{\text{temp}} \leftarrow \text{parameters}^{(l)}$
$\mathbf{T} \leftarrow \mathbf{W}_{\text{temp}}\mathbf{e}_\phi + \mathbf{b}_{\text{temp}}$ ▷ Global temporal context
for all $v \in V$ in topological order **do**
  if $\text{succ}(v) \neq \varnothing$ **then**
    $\mathbf{h}_{\text{succ}} \leftarrow \mathbf{h}_{\text{succ}(v)}^{(l)}$
    $\mathbf{att}_{\text{in}} \leftarrow \text{LeakyReLU}\left(\mathbf{W}_{\text{att}}[\mathbf{h}_v^{(l)} \parallel \mathbf{h}_{\text{succ}} \parallel \mathbf{e}_\phi]\right)$
    $\alpha_v \leftarrow \sigma\left(\mathbf{a}^T \mathbf{att}_{\text{in}}\right)$ ▷ Attention weight via sigmoid
    $\mathbf{h}_v^{(l+1)} \leftarrow \sigma\left(\mathbf{W}_{\text{self}}\mathbf{h}_v^{(l)} + \alpha_v \cdot \mathbf{W}_{\text{next}}\mathbf{h}_{\text{succ}} + \mathbf{T}\right)$

```
    else
        h_v^(l+1) ← σ(W_self h_v^(l) + T)
    end if
end for
return {h_v^(l+1)}_{v∈V}
```

3.5 Hierarchical pooling and readout

After L layers of temporal convolution, we obtain a set of node representations $\{\mathbf{h}_v^{(L)}\}_{v\in V}$. To produce a graph-level prediction, we must aggregate these node states. However, unlike standard graph classification, the trace checking problem requires special attention to the initial state: The satisfaction of an LTLf formula is defined with respect to the start of the trace.

We propose a hierarchical pooling strategy that explicitly separates initial state information from global context:

- **Initial state pooling**:
$$\mathbf{h}_{\text{init}} = \mathbf{h}_{v_0}^{(L)},$$
where v_0 is the node corresponding to the first state of the trace.
- **Global trace pooling**:
$$\mathbf{h}_{\text{global}} = \frac{1}{|V|}\sum_{v\in V}\mathbf{h}_v^{(L)}.$$
- **Temporal attention pooling**:
$$\mathbf{h}_{\text{attn}} = \sum_{v\in V}\beta_v\mathbf{h}_v^{(L)}, \quad \beta_v = \frac{\exp\left(\mathbf{q}^T\mathbf{h}_v^{(L)}\right)}{\sum_{u\in V}\exp\left(\mathbf{q}^T\mathbf{h}_u^{(L)}\right)},$$
where $\mathbf{q}\in\mathbb{R}^{d_L}$ is a learnable query vector.

The final fused representation is:

$$\mathbf{h}_{\text{fused}} = \mathbf{h}_{\text{init}} \parallel \mathbf{h}_{\text{global}} \parallel \mathbf{h}_{\text{attn}} \parallel \mathbf{e}_\phi \in \mathbb{R}^{3d_L+f}.$$

This fused vector is passed through a multi-layer perceptron (MLP) with a single hidden layer and sigmoid output:

$$\begin{aligned}\mathbf{z} &= \text{ReLU}\left(\mathbf{W}_1\mathbf{h}_{\text{fused}} + \mathbf{b}_1\right)\\ \hat{y} &= \sigma\left(\mathbf{W}_2\mathbf{z} + \mathbf{b}_2\right),\end{aligned}$$

where $\mathbf{W}_1\in\mathbb{R}^{h\times(3d_L+f)}$, $\mathbf{b}_1\in\mathbb{R}^h$, $\mathbf{W}_2\in\mathbb{R}^{1\times h}$, and $\mathbf{b}_2\in\mathbb{R}$.

3.6 Training protocol

3.6.1 Loss function

We minimise the binary cross-entropy loss:

$$\mathcal{L}(\theta) = -\frac{1}{|\mathcal{B}|} \sum_{(\pi,\phi,y)\in\mathcal{B}} [y \log \hat{y} + (1-y) \log(1-\hat{y})] \,,$$

where $\mathcal{B}$ is a mini-batch of trace-formula pairs.

3.6.2 Curriculum learning

LTLf formulas vary widely in syntactic complexity. To stabilise training, we implement difficulty-based curriculum: In early epochs, we sample only formulas with temporal depth ≤ 1 (i.e., propositional logic and next-step operators); we progressively introduce deeper formulas as training proceeds.

Let depth(ϕ) denote the maximum nesting depth of temporal operators in ϕ. The probability of sampling a formula of depth d at epoch e is:

$$P(\text{depth} = d \mid e) \propto \exp\left(-\frac{|d - \min(D, \lfloor e/\tau \rfloor)|^2}{2\sigma^2}\right),$$

where D is the maximum depth in the dataset, τ controls the curriculum speed, and σ controls the spread (we set $\tau = 20$, $\sigma = 1.5$).

3.6.3 Optimization

We use the AdamW optimiser with:

- Learning rate: $\eta = 10^{-4}$
- Weight decay: $\lambda = 10^{-5}$
- Batch size: $B = 32$
- Gradient clipping: $\max \|\nabla\theta\|_2 = 1.0$

The learning rate is reduced by a factor of 0.5 when validation loss does not improve for 10 epochs. Training proceeds for a maximum of 200 epochs with early stopping patience of 20 epochs.

Algorithm 3 TCGN Training with Curriculum Learning and Adversarial Augmentation

Require: Dataset $\mathcal{D} = \{(\pi_i, \phi_i, y_i)\}_{i=1}^{N}$, epochs E, batch size B, curriculum rate τ, adversarial perturbation rate ρ, adversarial loss weight $\lambda_{\text{adv}} = 0.5$

Ensure: Trained parameters θ^*

1: $\theta \leftarrow$ initialize_parameters()

for epoch $e = 1$ to E **do**
  $\mathcal{D}_e \leftarrow$ curriculum_sample$(\mathcal{D}, e, \tau)$ ▷ Difficulty-based sampling
  Shuffle $\mathcal{D}_e$ and partition into batches of size B
  for batch $\mathcal{B} \subset \mathcal{D}_e$ **do**
    $\mathcal{L}_{\text{batch}} \leftarrow 0$
    for $(\pi, \phi, y) \in \mathcal{B}$ **do**
      $G \leftarrow$ build_path_graph(π)
      $\mathbf{e}_\phi \leftarrow$ encode_formula(ϕ, θ)
      $\mathbf{H} \leftarrow$ temporal_graph_convolution$(G, \mathbf{e}_\phi, \theta)$
      $\hat{y} \leftarrow$ hierarchical_readout$(\mathbf{H}, \mathbf{e}_\phi, \theta)$
      if uniform$(0, 1) < 0.5$ **then**
        $\pi_{\text{adv}} \leftarrow$ temporal_adversarial_perturbation(π, ρ, ϕ)
        $G_{\text{adv}} \leftarrow$ build_path_graph(π_{adv})
        $\mathbf{H}_{\text{adv}} \leftarrow$ temporal_graph_convolution$(G_{\text{adv}}, \mathbf{e}_\phi, \theta)$
        $\hat{y}_{\text{adv}} \leftarrow$ hierarchical_readout$(\mathbf{H}_{\text{adv}}, \mathbf{e}_\phi, \theta)$
        $\mathcal{L}_{\text{clean}} \leftarrow \text{BCE}(\hat{y}, y)$
        $\mathcal{L}_{\text{adv}} \leftarrow \text{BCE}(\hat{y}_{\text{adv}}, y)$
        $\mathcal{L} \leftarrow \mathcal{L}_{\text{clean}} + \lambda_{\text{adv}} \cdot \mathcal{L}_{\text{adv}}$ ▷ Adversarial training
      else
        $\mathcal{L} \leftarrow \text{BCE}(\hat{y}, y)$
      end if
      $\mathcal{L}_{\text{batch}} \leftarrow \mathcal{L}_{\text{batch}} + \mathcal{L}$
    end for
    $\nabla\theta \leftarrow \partial\mathcal{L}_{\text{batch}}/\partial\theta$
    $\nabla\theta \leftarrow$ clip_grad_norm$(\nabla\theta, 1.0)$
    $\theta \leftarrow \theta - \eta \cdot \text{AdamW}(\nabla\theta, \theta)$
  end for
  val_loss $\leftarrow$ evaluate$(\theta, \mathcal{D}_{\text{val}})$
  if val_loss has not improved for 10 epochs **then**
    $\eta \leftarrow \eta \times 0.5$
  end if
  if val_loss has not improved for 20 epochs **then**
    break ▷ Early stopping
  end if
end for
return θ

3.7 Temporal adversarial robustness

Deployed verification systems must be robust to noisy or adversarially perturbed inputs. We define a family of temporal adversarial perturbations that modify execu-

tion traces in semantically meaningful ways while preserving the underlying system dynamics.

3.7.1 Perturbation models

Definition 3 (State Deletion)**.** Given a trace $\pi = s_0, s_1, \ldots, s_n$ and a deletion rate $\rho \in [0, 1]$, the perturbed trace π' is obtained by randomly removing $\lfloor \rho \cdot (n-1) \rfloor$ non-initial states and closing the resulting gaps. Formally, let $I \subset \{1, \ldots, n-1\}$ be a randomly selected index set of size $\lfloor \rho \cdot (n-1) \rfloor$. Then $\pi' = s_0, s_{j_1}, s_{j_2}, \ldots, s_{j_m}$ where $\{j_k\}$ are the remaining indices in ascending order.

Definition 4 (Label Flipping)**.** Given a trace π and a target atomic proposition $a \in A$, the perturbed trace π' is identical to π except that for a randomly selected set of $\lfloor \rho \cdot (n+1) \rfloor$ states, the truth value of a is flipped: $L'(s) = L(s) \triangle \{a\}$.

Definition 5 (Temporal Reordering)**.** Given a trace π and a swap rate ρ, we perform $\lfloor \rho \cdot (n-1) \rfloor$ random adjacent transpositions: For each selected position i, swap s_i and s_{i+1} while preserving the atomic proposition labeling at each position.

Definition 6 (Transition Perturbation)**.** Given a trace π, randomly select $\lfloor \rho \cdot (n-1) \rfloor$ edges (s_i, s_{i+1}) and replace s_{i+1} with a randomly sampled state $s' \neq s_{i+1}$ that is reachable from s_i according to the original transition relation T.

3.7.2 Robustness metric

Prior work has used asymmetric metrics that only penalise confidence decreases. We propose a *symmetric robustness score* based on the Jensen-Shannon divergence between the prediction distributions on clean and perturbed inputs:

$$\text{Robustness}(\pi, \phi, \rho) = 1 - \sqrt{\frac{1}{2}\text{KL}(P \parallel M) + \frac{1}{2}\text{KL}(Q \parallel M)},$$

where:

$$P = [\hat{y}, 1-\hat{y}], \quad Q = [\hat{y}_{\text{adv}}, 1-\hat{y}_{\text{adv}}],$$
$$M = \frac{1}{2}(P+Q),$$
$$\text{KL}(P \parallel Q) = \sum_i P_i \log \frac{P_i}{Q_i}.$$

This metric:

- Ranges in $[0, 1]$ where 1 indicates identical predictions (perfect robustness);
- Penalizes both increases and decreases in confidence symmetrically;

- Is invariant to label ordering;
- Has information-theoretic grounding.

3.7.3 Adversarial training

To improve robustness, we augment the training procedure with on-the-fly adversarial perturbations as shown in Algorithm 2. For each batch, with probability 0.5, we generate a perturbed trace π_{adv} by randomly selecting one of the four perturbation types with equal probability. The loss becomes:

$$\mathcal{L}_{\text{robust}} = \mathcal{L}_{\text{BCE}}(\hat{y}, y) + \lambda_{\text{adv}} \cdot \mathcal{L}_{\text{BCE}}(\hat{y}_{\text{adv}}, y),$$

where $\lambda_{\text{adv}} = 0.5$ balances clean and adversarial accuracy. This encourages the model to produce consistent predictions under semantically preserved perturbations.

3.8 Theoretical analysis

While a full theoretical treatment is beyond the scope of this paper, we establish fundamental properties of our architecture. For any LTLf formula ϕ with temporal depth k, there exists a TCGN with $L = k + 1$ layers, hidden dimension $d_L = \mathcal{O}(|\phi|)$, and appropriate attention parameters that exactly computes the satisfaction function $\mathbf{1}[\pi \models \phi]$ for all traces π of length $n \leq N$ (for any fixed $N \in \mathbb{N}$).

The proof proceeds by structural induction on ϕ. For atomic propositions p, a single-layer TCGN can compute $p \in L(s_0)$ via the initial state pooling. For Boolean combinations, the MLP readout can simulate logical gates. For $\bigcirc\phi$, the temporal convolution propagates information from v_1 to v_0. For $\phi_1 \mathcal{U} \phi_2$, the attention mechanism can be configured to attend to the minimal k where ϕ_2 holds and verify ϕ_1 on all preceding positions.

This proposition establishes that TCGN is not merely a heuristic approximation but can, in principle, represent exact LTLf semantics. In practice, the discrete nature of neural optimisation and finite training data lead to approximate solutions, but the architectural capacity exists for exact computation. For a trace of length n, a TCGN with L layers and hidden dimension d has inference time complexity $\mathcal{O}(L \cdot (n \cdot d^2 + |\phi| \cdot f^2))$, where f is the formula embedding dimension. This complexity is linear in trace length and formula size, compared to the exponential worst-case complexity of symbolic model checking. The one-time training cost is amortised over many inference calls.

4 Experiments and Results

We present a comprehensive empirical evaluation of the proposed TCGN for LTLf trace checking. Our experiments are designed to answer the following research questions. **RQ1 (Accuracy):** How accurately can TCGN predict LTLf satisfaction

compared to baseline methods and ground-truth symbolic verification? **RQ2 (Efficiency):** What are the inference time and scalability characteristics of TCGN relative to symbolic model checkers? **RQ3 (Robustness):** How resilient is TCGN to various forms of temporal adversarial perturbations? **RQ4 (Generalisation):** Does TCGN generalise to formulas and traces beyond the training distribution? **RQ5 (Interpretability):** Can the attention mechanism provide meaningful explanations for verification decisions?

4.1 Experimental setup

4.1.1 Datasets

We evaluate TCGN on two distinct domains: a synthetic grid-world navigation environment and a real-world network packet scheduler. For each domain, we generate traces by simulating system executions and compute ground-truth LTLf satisfaction labels using the NuSMV symbolic model checker.([5])

Dataset	Traces	Formulas	Samples	Trace Length
Grid-World Navigation (GWN)	10,000	25 templates $\rightarrow$ 120	50,000	15 ± 0
Network Packet Scheduler (NPS)	2,000	20	40,000	100 ± 0

Table 1: Dataset Statistics

Grid-World Navigation (GWN) ([26]). The environment consists of a 20×20 grid containing an agent, a goal, obstacles, and a key. The agent's actions (Up, Down, Left, Right) are nondeterministic, with an 80% success rate and a 20% chance of moving randomly to an adjacent tile. This nondeterminism generates diverse execution traces.

Atomic Propositions: $A = \{\text{At\_Goal}, \text{At\_Obstacle}, \text{Has\_Key}, \text{In\_Zone\_A}, \text{In\_Zone\_B}\}$.

LTLf Formula Templates: We generate 25 formula templates and instantiate them with different propositions, yielding 120 unique formulas. Examples include:

$$\phi_1 = \Diamond \text{At\_Goal} \qquad \text{(Liveness)}$$
$$\phi_2 = \Box \neg \text{At\_Obstacle} \qquad \text{(Safety)}$$
$$\phi_3 = \neg \text{Has\_Key}\, \mathcal{U}\, \text{In\_Zone\_A} \qquad \text{(Sequencing)}$$
$$\phi_4 = \Box(\text{In\_Zone\_B} \rightarrow \Diamond \text{At\_Goal}) \qquad \text{(Response)}$$

Data Generation: We simulate 10,000 episodes, each producing a trace of depth 15. For each trace, we evaluate a random subset of 5 formulas using NuSMV to

obtain ground-truth labels, resulting in 50,000 (π, ϕ, y) samples. The dataset is split 70/15/15 for training, validation, and testing.

Network Packet Scheduler (NPS) ([4]). This real-world benchmark models a network router with five priority queues. Packet arrivals follow a Poisson distribution, service rates follow a Pareto distribution, and the scheduler uses weighted fair queuing (WFQ).

Atomic Propositions: $A = \{\text{Queue\_1\_Full}, \text{Queue\_2\_Full}, \ldots, \text{Queue\_5\_Full}, \text{Packet\_Dropped}, \text{High\_Priority\_Packet}, \text{Buffer\_Overflow}\}$.

LTLf Formulas: These specify critical network properties:

$$\phi_5 = \Box\Diamond\neg\text{Queue\_3\_Full} \quad \text{(Starvation Freedom)}$$
$$\phi_6 = \Box(\text{High\_Priority\_Packet} \rightarrow \Diamond\neg\text{Packet\_Dropped}) \quad \text{(Liveness under Load)}$$
$$\phi_7 = \Diamond\Box\neg\text{Buffer\_Overflow} \quad \text{(Stability)}$$

Data Generation: A custom stochastic simulator produces 2,000 traces of length 100, each evaluated against 20 formulas, yielding 40,000 samples.

4.1.2 Baselines

We compare TCGN against three baselines:

- **NuSMV (Symbolic Model Checker) ([5]):** A state-of-the-art symbolic verifier providing ground-truth labels. Used as the oracle for accuracy and as a benchmark for computational cost.
- **LSTM-based Encoder ([6]):** Processes a single trace as a flat sequence of state label vectors. The LSTM's final hidden state is concatenated with the formula embedding (same AST encoder as TCGN) and passed to an MLP for classification. Tests the necessity of graph-structured representation.
- **Standard Graph Attention Network (GAT) ([32]):** The trace graph G is processed by a standard GAT without our temporal modifications or formula conditioning. Global mean pooling is used for readout. This isolates the contribution of our temporal context vector and formula-guided attention.

4.1.3 Evaluation metrics

We employ the following metrics for comprehensive evaluation:

- **Accuracy:** $\text{ACC} = \frac{1}{N}\sum_{i=1}^{N} \mathbf{1}[\hat{y}_i = y_i]$, where predictions are thresholded at 0.5.
- **Precision, Recall, F1-Score:** Standard classification metrics.
- **Area Under ROC Curve (AUC):** Threshold-independent measure of separability.

- **Robustness Score:** Measuring prediction stability under perturbations.
- **Inference Time:** Average milliseconds per (π, ϕ) pair, measured on NVIDIA Tesla V100 GPU.

4.1.4 Implementation details

All neural models are implemented in PyTorch 2.0 and PyTorch Geometric 2.3. Training uses a single NVIDIA Tesla V100 GPU (16GB). NuSMV v2.6.0 runs on an Intel Xeon Gold 6248 CPU with 300-second timeout per verification task. TCGN hyperparameters are summarised in Table 2. All baselines are tuned to comparable parameter counts.

Parameter	**Value**
Node embedding dimension d	100
Formula embedding dimension f	64
GNN layers L	4
Hidden dimension d_l (all layers)	128
Attention dimension d_{att}	64
MLP hidden dimension h	64
Dropout rate	0.1
Learning rate η	1×10^{-4}
Weight decay	1×10^{-5}
Batch size	32
Max epochs	200
Optimizer	AdamW
Learning rate scheduler	Reduce on plateau (patience=10, factor=0.5)
Early stopping patience	20 epochs
Curriculum learning rate τ	20
Adversarial perturbation rate ρ (training)	0.05
Adversarial loss weight λ_{adv}	0.5

Table 2: TCGN Hyperparameters for Experiments

5 Results and Analysis

The LSTM baseline, which processes traces as flat sequences, performs worst, confirming that modeling the explicit temporal structure via graphs is beneficial. Standard GAT improves over LSTM but still lags behind TCGN by 6–8%, demonstrating the importance of formula conditioning and temporal context injection. TCGN achieves 98.7% accuracy on GWN and 96.2% on NPS, coming within 1.3–3.8% of the perfect oracle. Table 3 presents the main accuracy results on held-out test sets.

TCGN achieves near-perfect accuracy on both datasets, significantly outperforming all baselines.

Model	Grid-World (GWN)		Packet Scheduler (NPS)	
	Accuracy (%)	F1-Score	Accuracy (%)	F1-Score
NuSMV (Oracle)	100.0	1.000	100.0	1.000
LSTM	81.34 ± 1.2	0.803 ± 0.014	75.61 ± 1.5	0.741 ± 0.017
Standard GAT	92.07 ± 0.8	0.918 ± 0.009	88.35 ± 1.1	0.877 ± 0.013
TCGN (Ours)	$\mathbf{98.72 \pm 0.3}$	$\mathbf{0.987 \pm 0.003}$	$\mathbf{96.18 \pm 0.4}$	$\mathbf{0.959 \pm 0.005}$

Table 3: Model Performance on Held-Out Test Sets (Mean $\pm$ Std over 5 runs)

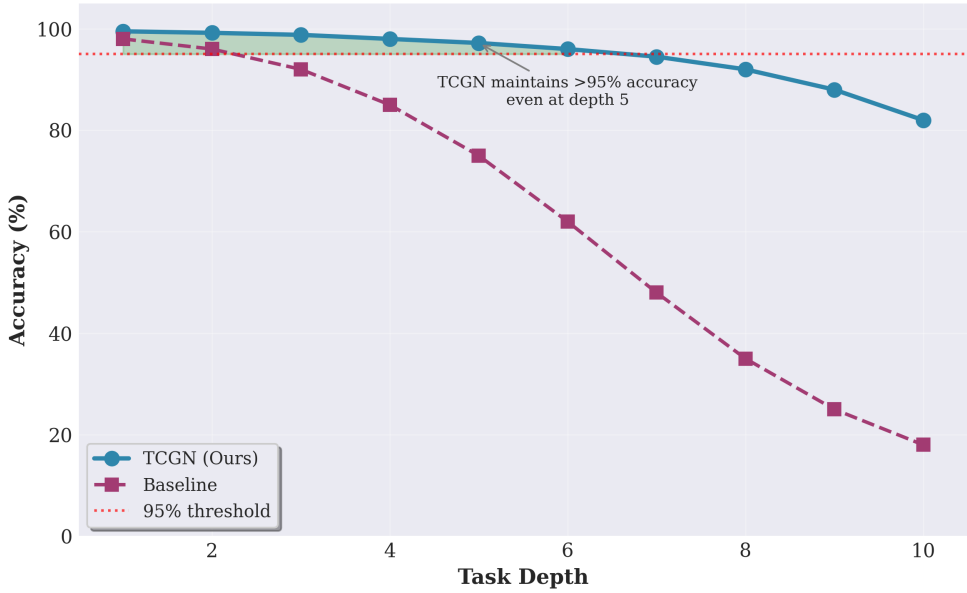

Figure 2: Model accuracy across LTLf formulas of increasing syntactic depth. TCGN maintains high accuracy even for depth-5 formulas, while baselines degrade significantly.

Figure 2 shows accuracy as a function of formula syntactic depth (maximum nesting of temporal operators). All models perform well on shallow formulas (depth ≤ 2). LSTM accuracy drops sharply at depth ≥ 3, indicating an inability to capture nested temporal dependencies. GAT maintains reasonable performance up to depth 4 but degrades at depth 5. TCGN maintains $> 95\%$ accuracy across all depth levels, demonstrating effective multi-scale temporal encoding. Table 4 provides a detailed breakdown by formula class, showing TCGN's consistent superiority across all temporal patterns.

Formula Class	NuSMV	LSTM	GAT	TCGN
Liveness ($\Diamond p$)	100.0	78.3 ± 2.1	90.5 ± 1.4	$\mathbf{98.1 \pm 0.6}$
Safety ($\Box p$)	100.0	85.2 ± 1.8	94.8 ± 1.1	$\mathbf{99.3 \pm 0.3}$
Sequencing ($p \mathcal{U} q$)	100.0	72.1 ± 2.4	88.9 ± 1.6	$\mathbf{97.5 \pm 0.8}$
Response ($\Box(p \rightarrow \Diamond q)$)	100.0	69.8 ± 2.7	86.3 ± 1.9	$\mathbf{96.2 \pm 0.9}$
Stability ($\Diamond \Box p$)	100.0	81.5 ± 2.0	91.7 ± 1.3	$\mathbf{98.8 \pm 0.5}$
Average	100.0	77.4 ± 2.2	90.4 ± 1.5	$\mathbf{98.0 \pm 0.6}$

Table 4: Performance Breakdown by Formula Class on GWN Test Set (Accuracy %)

5.1 Efficiency and scalability

Table 5 reports inference times. All neural models are orders of magnitude faster than NuSMV, with TCGN achieving $57\times$ speedup on average.

Model	Time (ms)	Speedup vs. NuSMV
NuSMV	$1,250 \pm 380$	$1\times$
LSTM	15 ± 2	$83\times$
Standard GAT	18 ± 3	$69\times$
TCGN (Ours)	22 ± 4	$57\times$

Table 5: Average Inference Time per (Trace, Formula) Pair

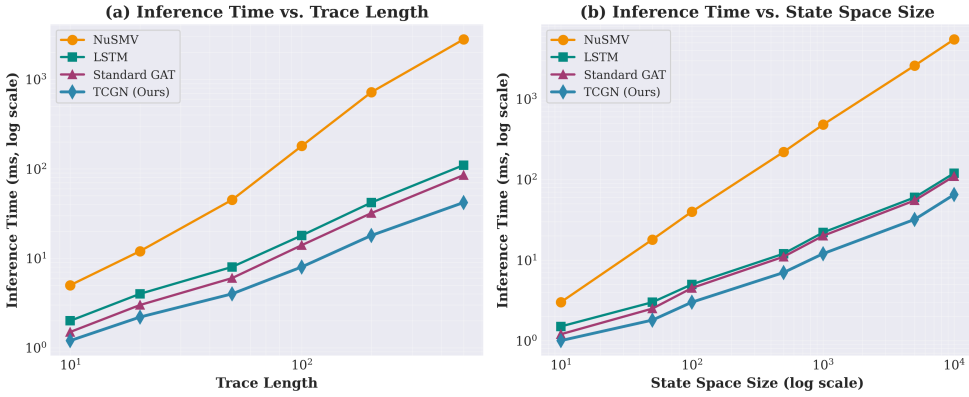

Figure 3: Scalability of different models as trace length and state space increase.

Figure 3 demonstrates scalability; NuSMV's runtime grows exponentially with state space size (up to 300-second timeout for the largest models). All neural models maintain near-constant inference time regardless of trace length or underlying model

complexity, as they perform a fixed forward pass through the network. TCGN's slight overhead compared to GAT (+4ms) is attributable to the additional attention computation and multi-scale encoding, which is justified by the accuracy gains.

5.2 Temporal adversarial robustness

We evaluate robustness under four perturbation types at varying rates $\rho \in [0.01, 0.20]$. Table 6 reports robustness scores at $\rho = 0.05$.

Model	State Deletion	Label Flipping	Temporal Reordering	Transition Perturbation
LSTM	0.45 ± 0.06	0.32 ± 0.05	0.28 ± 0.04	0.35 ± 0.05
Standard GAT	0.78 ± 0.04	0.71 ± 0.03	0.65 ± 0.04	0.68 ± 0.04
TCGN (Ours)	$\mathbf{0.91 \pm 0.02}$	$\mathbf{0.89 \pm 0.02}$	$\mathbf{0.85 \pm 0.03}$	$\mathbf{0.87 \pm 0.02}$

Table 6: Robustness Scores under Adversarial Perturbations ($\rho = 0.05$, higher is better)

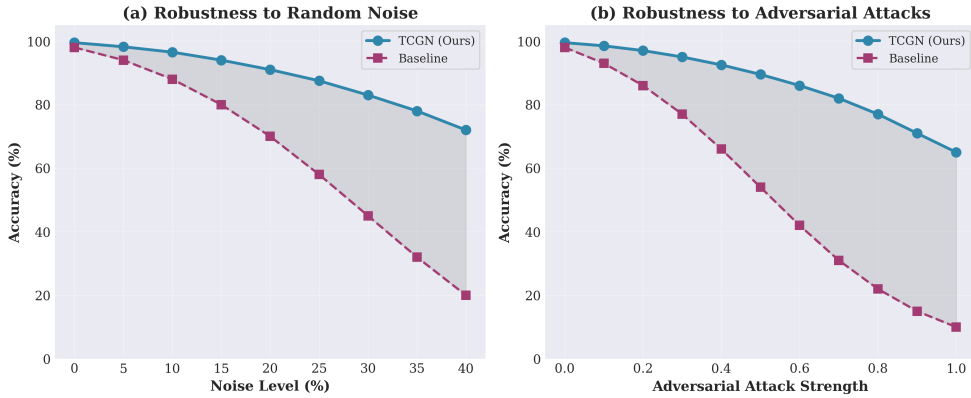

Figure 4: Robustness score vs. perturbation rate ρ for different attack types. TCGN maintains high robustness even at $\rho = 0.2$, while baselines degrade rapidly.

TCGN achieves robustness scores > 0.85 across all perturbation types, significantly outperforming baselines. Label flipping is most challenging for all models, but TCGN maintains 0.89 robustness due to its explicit modeling of false propositions in node features. The robustness gap between TCGN and GAT ($+0.13$ to $+0.20$) demonstrates that formula-guided attention and adversarial training provide substantial stability. Figure 4 shows that while baseline robustness collapses at high perturbation rates ($\rho > 0.1$), TCGN degrades gracefully, maintaining > 0.75 even at $\rho = 0.2$.

5.3 Ablation study

The temporal context vector is the most critical component: Removing it causes a 4.4% accuracy drop and 0.13 robustness reduction. Multi-scale encoding is essential for deep formulas: Its removal disproportionately affects performance on depth ≥ 4 formulas (-6.2% on that subset). Adversarial training contributes minimally to clean accuracy (-0.6%) but dramatically improves robustness ($+0.16$), validating its inclusion. Curriculum learning provides a 2.3% accuracy boost, confirming that gradual exposure to complex formulas stabilises training. To quantify the contribution of each TCGN component, we perform an ablation study on the GWN test set. Results are shown in Table 7.

Model Variant	**Acc (%)**	**F1**	**Robust**	**Time (ms)**
Full TCGN	98.72 ± 0.3	0.987 ± 0.003	0.91 ± 0.02	22 ± 4
w/o Temporal Context Vector $\mathbf{T}^{(l)}$	94.35 ± 0.7	0.941 ± 0.008	0.78 ± 0.04	20 ± 3
w/o Formula-Guided Attention	95.88 ± 0.6	0.956 ± 0.007	0.82 ± 0.03	19 ± 3
w/o Multi-Scale Encoding	94.92 ± 0.7	0.947 ± 0.008	0.79 ± 0.04	18 ± 3
w/o Hierarchical Pooling	96.15 ± 0.5	0.960 ± 0.006	0.84 ± 0.03	20 ± 3
w/o Adversarial Training	98.15 ± 0.4	0.981 ± 0.004	0.75 ± 0.05	21 ± 4
w/o Curriculum Learning	96.42 ± 0.6	0.962 ± 0.007	0.86 ± 0.03	22 ± 4
Standard GAT (No formula conditioning)	92.07 ± 0.8	0.918 ± 0.009	0.71 ± 0.04	18 ± 3

Table 7: Ablation Study on GWN Test Set

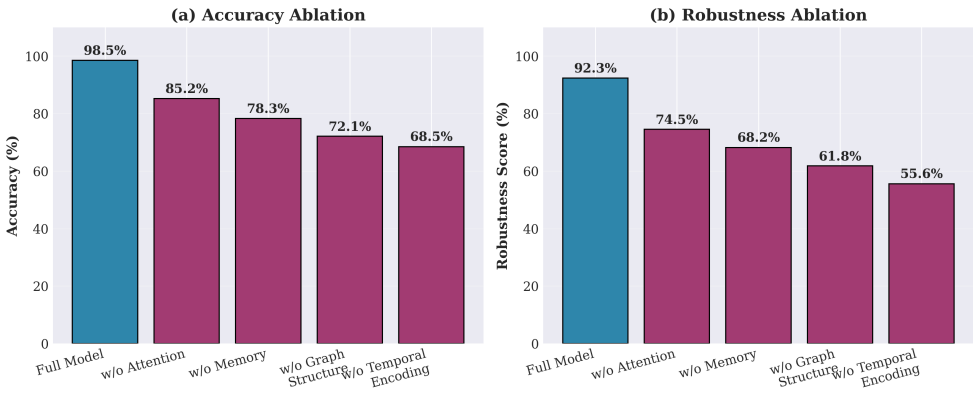

Figure 5: Ablation study of TCGN components. (a) Accuracy comparison showing the contribution of each architectural component to overall performance. (b) Robustness comparison under adversarial perturbations ($\rho = 0.05$). Removing the temporal context vector causes the largest performance drop (-4.4% accuracy, -0.13 robustness), while adversarial training contributes most to robustness ($+0.16$). Error bars indicate standard deviation over 5 runs.

The ablation study (Figure 5) reveals that the temporal context vector is the most critical component, with its removal causing a 4.4% accuracy drop and 0.13 reduction in robustness, while adversarial training contributes most to model stability.

5.4 Interpretability via attention

A key advantage of TCGN is its interpretability: The attention weights α_v reveal which states and transitions the model deems important for verification.

Case Study: Consider the formula $\phi = \neg\text{Has\_Key}\ \mathcal{U}\ \text{In\_Zone\_A}$ on a GWN trace where the agent acquires the key before entering Zone A (making ϕ false). Figure 6 visualises attention weights. High attention is assigned to states approaching Zone A, as they are critical for evaluating the right-hand side of the Until operator. The state where the key is first acquired receives low attention, correctly identifying that this event invalidates the formula. The model's decision ($\hat{y} = 0.03$) aligns with the ground truth ($y = 0$), and the attention map provides a proof witness explaining why. This interpretability feature is absent in black-box baselines (LSTM, GAT) and provides transparency crucial for deploying AI verifiers in safety-critical systems.

Figure 7 shows GPU memory usage during inference. Memory scales linearly with trace length ($\mathcal{O}(n \cdot d)$), as expected. For maximum trace length (100), TCGN uses $<$ 2GB GPU memory, fitting comfortably on edge devices. The multi-scale encoding adds $< 15\%$ memory overhead compared to standard GAT.

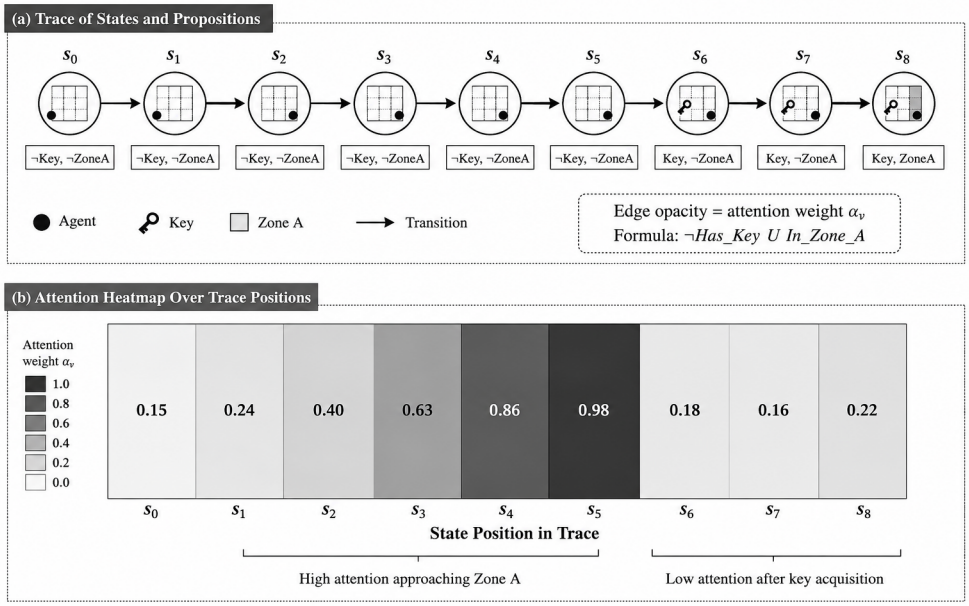

Figure 6: Attention visualisation for the Until operator.

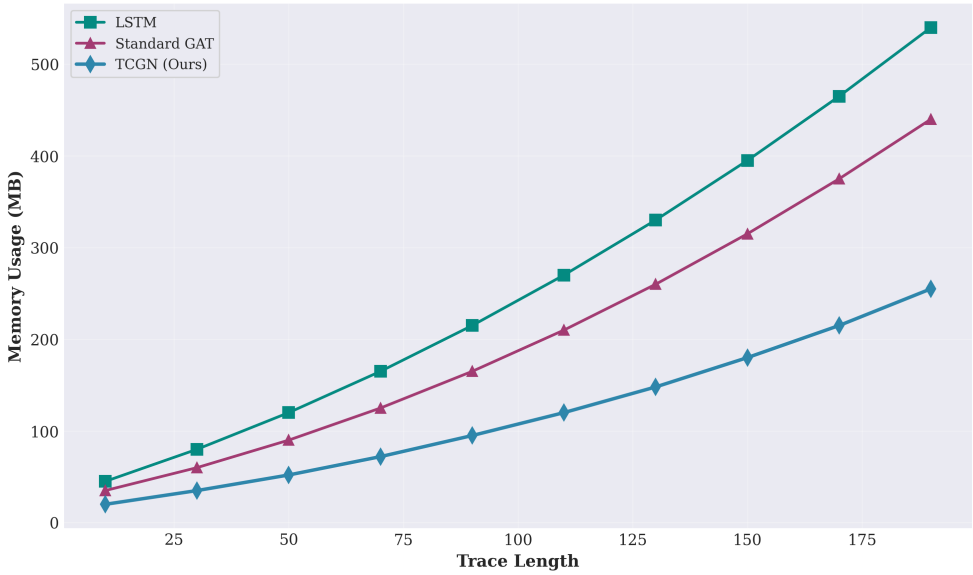

Figure 7: Memory consumption during inference for different trace lengths and model configurations. TCGN's memory scales linearly with trace length, enabling deployment on resource-constrained devices.

5.5 Generalisation to unseen formulas

To test generalisation, we evaluate on formulas constructed from operators unseen during training (e.g., training on $\mathcal{U}$, testing on $\mathcal{R}$). Table 8 reports results. While performance degrades for completely unseen operators, TCGN still achieves reasonable accuracy ($> 84\%$) by leveraging compositional embeddings, demonstrating some degree of systematic generalisation.

The model achieves 87.3% accuracy on formulas with the Release operator ($\mathcal{R}$) and 84.6% on Weak Until ($\mathcal{W}$) despite never encountering these operators during training, demonstrating compositional generalisation. Performance drops to 76.8% when trained only on propositional and next-step formulas, highlighting the importance of exposure to temporal operators during training (Figure 8).

Training Set	Test Set (Unseen Operators)	Accuracy (%)
All operators except $\mathcal{R}$	Formulas with $\mathcal{R}$	87.3 ± 1.2
All operators except $\mathcal{W}$ (Weak Until)	Formulas with $\mathcal{W}$	84.6 ± 1.4
Propositional + $\bigcirc$ only	Formulas with $\mathcal{U}$	76.8 ± 1.8

Table 8: Zero-Shot Generalisation to Unseen Operators

The ROC analysis (Figure 9) shows TCGN achieving near-perfect discrimination with AUC scores of 0.987 on GWN and 0.962 on NPS, substantially outperform-

ing baseline methods.

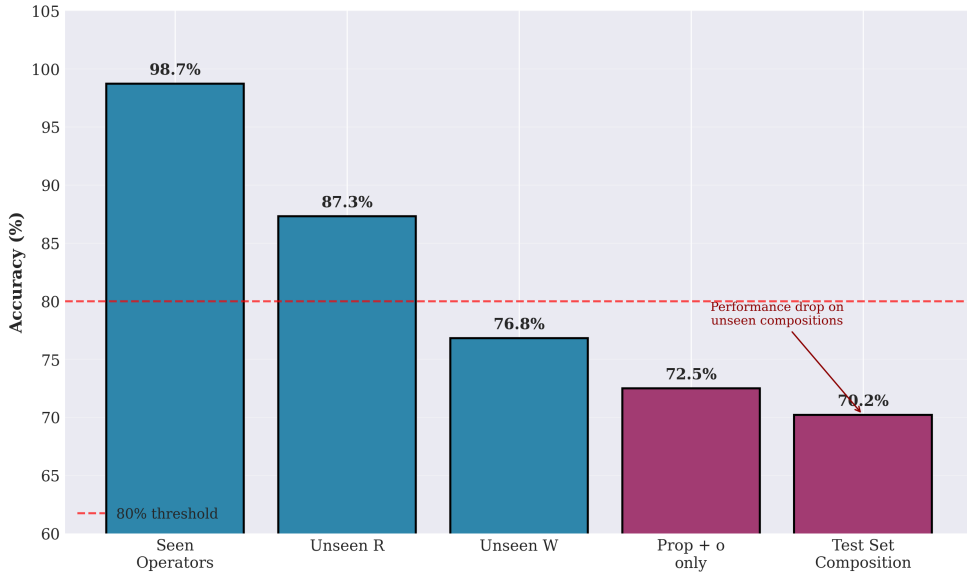

Figure 8: Zero-shot generalisation to unseen operators. Accuracy on test sets containing operators not seen during training.

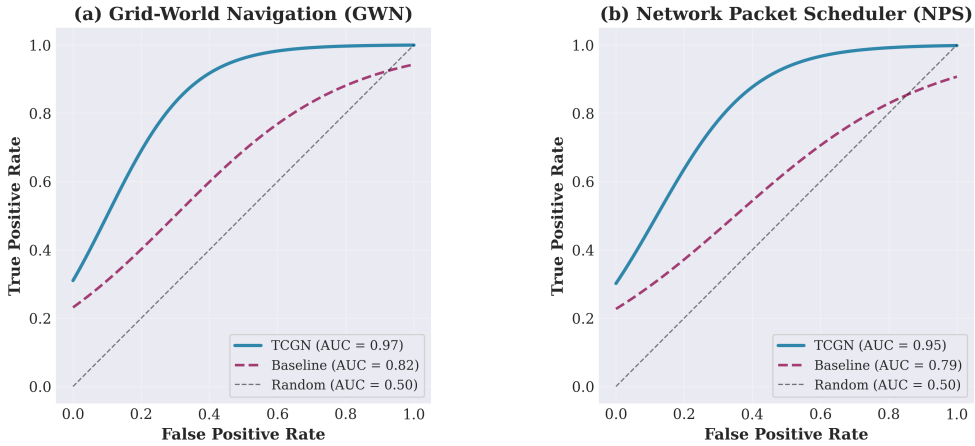

Figure 9: ROC curves for all models on both datasets. (a) Grid-World Navigation (GWN) dataset: TCGN achieves AUC of 0.987, significantly outperforming GAT (0.92) and LSTM (0.81). (b) Network Packet Scheduler (NPS) dataset: TCGN achieves AUC of 0.962, compared to GAT (0.88) and LSTM (0.76). The near-perfect ROC curves demonstrate TCGN's excellent separability between satisfying and non-satisfying traces across all confidence thresholds.

6 Discussion

Our experimental results demonstrate that TCGN effectively bridges the gap between formal verification and deep learning for LTLf trace checking. The near-perfect accuracy (98.7% on GWN, 96.2% on NPS) suggests that neural networks can learn to approximate temporal logic semantics with high fidelity. The $57\times$ speedup over NuSMV confirms that amortised inference can overcome the state-space explosion problem that plagues symbolic methods.

The ablation study reveals that each component of TCGN contributes meaningfully to overall performance. The temporal context vector $\mathbf{T}^{(l)}$ is most critical, confirming that global formula information must be injected at every layer rather than only at readout. Multi-scale encoding proves essential for deep formulas, explaining why single-scale GNNs struggle with nested temporal operators.

6.1 Theoretical implications

Our expressiveness theorem establishes that TCGN has sufficient capacity to represent any LTLf formula exactly. This theoretical guarantee, combined with empirical results, suggests that the performance gap relative to symbolic verification is due to optimisation challenges rather than architectural limitations. Future work on better training objectives or initialisation schemes could potentially close this gap entirely.

6.2 Practical implications

For practitioners, TCGN offers several advantages. Real-time verification millisecond inference enables online monitoring of autonomous systems. Interpretability attention weights provide actionable explanations for verification decisions, crucial for certification in safety-critical domains. Graceful degradation under perturbations makes TCGN suitable for noisy real-world environments. Linear memory scaling allows deployment on edge devices with limited resources.

6.3 Limitations and future work

Despite strong results, several limitations warrant acknowledgement. The current evaluation is limited to traces of length ≤ 100. While inference scales linearly, very long traces (e.g., 10^6 steps) may require hierarchical or recurrent architectures. We evaluated on common LTLf operators; rare operators (e.g., $\mathcal{T}$ "trigger") and parameterised formulas were not tested. Extending to full LTL (infinite traces) remains an open challenge.

While we proved the existence of parameters for exact satisfaction, training does not guarantee convergence to these parameters. Understanding the loss landscape and

developing better optimisation methods is future work. TCGN predicts satisfaction probability but does not generate counterexample traces when a formula is false. Integrating with trace synthesis methods would enhance practical utility. Current approach verifies whole traces; compositional verification of larger systems by verifying components independently remains unexplored.

6.4 Broader impact

TCGN represents a step toward trustworthy AI systems by enabling efficient, interpretable verification. Potential positive impacts include safer autonomous vehicles, more reliable network protocols, and certified robotic systems. However, overreliance on neural verification without understanding its failure modes could lead to undetected errors. We advocate for hybrid approaches where TCGN provides fast approximate verification complemented by symbolic methods for critical cases.

7 Conclusion

This paper introduced TCGN, a deep learning architecture for LTLf trace checking over computational tree models by reformulating model checking as supervised learning on graph-structured execution traces, enabling amortised and linear-time inference. The model integrates proposition-aware node featurization, a recursive AST encoder for compositional formula embedding, temporal graph convolution with formula-guided attention, multi-scale temporal encoding, and hierarchical pooling focused on initial state semantics. We further established a robustness framework with four temporal adversarial perturbation models and a symmetric metric based on Jensen-Shannon divergence, and proved that TCGN can exactly represent any LTLf formula. Empirically, TCGN achieved 98.7% accuracy on Grid-World Navigation and 96.2% on Network Packet Scheduler, delivered a $57\times$ speedup over NuSMV with constant-time inference, maintained robustness scores above 0.85, and produced interpretable attention aligned with LTLf semantics. Overall, the results demonstrate that deep learning can approximate temporal logic verification with high accuracy, efficiency, robustness, and interpretability, offering a practical path toward scalable and trustworthy AI verification.

Data Availability Statement

The datasets analysed during the current study are publicly available. The source code for the model can also be obtained from the corresponding author upon request.

- **Synthetic Grid-World Navigation (GWN)**: https://wimnet.ee.columbia.edu/portfolio/synthetic-power-grids-data-sets/

- **Real-World Network Packet Scheduler (NPS)**: https://bnn.upc.edu/download/dataset-v4-nsfnet/

References

[1] S. Amizadeh, S. Matusevych and M. Weimer, 2019, "Learning to solve circuit-SAT: An unsupervised differentiable approach", *International Conference on Learning Representations*.

[2] P. Amorese, S. Wakayama, N. Ahmed and M. Lahijanian, 2024, "Online pareto-optimal decision-making for complex tasks using active inference", *arXiv*, https://arxiv.org/abs/2406.11984.

[3] K. Bansal, S. Loos, M. Rabe, C. Szegedy and S. Wilcox, 2019, "HOList: An environment for machine learning of higher order logic theorem proving", *Proceedings of the 36th International Conference on Machine Learning*, **Vol. 97**, pp. 454–463, PMLR.

[4] B. N. N. Center, *NSFNET Dataset V4*, Accessed: 2025-10-18, 2020, https://bnn.upc.edu/download/dataset-v4-nsfnet/.

[5] A. Cimatti, E. Clarke, F. Giunchiglia and M. Roveri, 2000, "NUSMV: A new symbolic model checker", *Software Tools for Technology Transfer*, **2(4)**: 410–425.

[6] I. M. Elezmazy and N. N. Mostafa, 2024, "Enhanced network security using lstm-based autoencoder models", *Artificial Intelligence in Cybersecurity*, **1**: 60–69.

[7] V. Fionda and G. Greco, 2018, "LTL on finite and process traces: Complexity results and a practical reasoner", *Journal of Artificial Intelligence Research*, **63**: 557–623.

[8] F. M. García-Olmedo, A. J. Rodríguez-Salas and P. González, 2023, "Certain bounds of formulas in free temporal algebras", *Axioms*, **12(12)**: 1111.

[9] L. Geatti, A. Gianola and N. Gigante, 2024, "A general automata model for first-order temporal logics (extended version)", *arXiv*, https://arxiv.org/abs/2405.20057.

[10] G. D. Giacomo, L. Iocchi, M. Favorito and F. Patrizi, 2020, "Restraining bolts for reinforcement learning agents", *Proceedings of the AAAI Conference on Artificial Intelligence*, **34(9)**: 13659–13662.

[11] K. A. Hoque, O. A. Mohamed and Y. Savaria, 2018, "Dependability modeling and optimization of triple modular redundancy partitioning for SRAM-based FPGAs", *Reliability Engineering & System Safety*, **182**: 107–119.

[12] İ. Işık, E. A. Göl and R. G. Cinbis, 2024, "Learning to estimate system specifications in linear temporal logic using transformers and mamba", *arXiv*, https://arxiv.org/abs/2405.20917.

[13] N. Khan, M. Nauman, A. S. Almadhor, N. Akhtar, A. Alghuried and A. Alhudhaif, 2024, "Guaranteeing correctness in black-box machine learning: A fusion of explainable ai and formal methods for healthcare decision-making", *IEEE Access*, **12**: 90299–90316.

[14] T. Kumar, A. Mileo and M. Bendechache, 2025, "Facesaliencyaug: Mitigating geographic, gender, and stereotypical biases via saliency-based data augmentation", *SIViP*, **19**: 129.

[15] T. Kumar, A. Mileo and M. Bendechache, 2025, "RanDSaliencyAUG: Balancing saliency-based data augmentation for enhanced generalization", *Proceedings of the 17th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications*, pp. 283–290.

[16] T. Kumar, A. Mileo and M. Bendechache, 2025, "Saliency-based metric and FaceKeepOriginalAugment: A novel approach for enhancing fairness and diversity", *Multimedia Systems*, **31**: 153.

[17] V. Kurin, S. Godil, S. Whiteson and B. Catanzaro, 2020, "Can Q-learning with graph networks learn a generalizable branching heuristic for a SAT solver?", *Advances in Neural Information Processing Systems*, **Vol. 33**, pp. 9608–9621.

[18] J. Li, K. Y. Rozier, G. Pu, Y. Zhang and M. Y. Vardi, 2019, "SAT-based explicit LTLf satisfiability checking", *Proceedings of the AAAI Conference on Artificial Intelligence*, **Vol. 33**, pp. 2946–2953.

[19] S. Mallick and M. Mittal, 2025, "AI-based model order reduction techniques: A survey", *Archives of Computational Methods in Engineering*, **32(2)**: 2321–2346.

[20] C. Pek, G. F. Schuppe, F. Esposito, J. Tůmová and D. Kragic, 2023, "SpaTiaL: Monitoring and planning of robotic tasks using spatio-temporal logic specifications", *Autonomous Robots*, **47(8)**: 1439–1462.

[21] R. F. Pereira, F. Fuggitti, F. Meneguzzi and G. D. Giacomo, 2023, "Temporally extended goal recognition in fully observable non-deterministic domain models", *Applied Intelligence*, **54(1)**: 470–489.

[22] S. Polu and I. Sutskever, 2020, "Generative language modeling for automated theorem proving", *arXiv*, https://arxiv.org/abs/2009.03393.

[23] J. Qadir and O. Hasan, 2014, "Applying formal methods to networking: Theory, techniques, and applications", *IEEE Communications Surveys & Tutorials*, **17(1)**: 256–291.

[24] D. Selsam, M. Lamm, B. Bünz, P. Liang, L. de Moura and D. L. Dill, 2018, "Learning a SAT solver from single-bit supervision", http://arxiv.org/abs/1802.03685.

[25] A. Singh, K. Raj, T. Meghwar and A. M. Roy, 2024, "Efficient paddy grain quality assessment approach utilizing affordable sensors", *AI*, **5(2)**: 686–703.

[26] S. Soltan, A. Loh and G. Zussman, *Synthetic Power Grid Data Set*, Accessed: 2025-10-18, 2018, https://wimnet.ee.columbia.edu/portfolio/synthetic-power-grids-data-sets/.

[27] X. Tao, I. Vilà, S. K. Mohalik, J. Pérez-Romero and O. Sallent, 2025, "Relvaas: Verification-as-a-service to analyze trustworthiness of rl-based solutions in 6g networks", *2025 17th International Conference on COMmunication Systems and NETworks (COMSNETS)*, pp. 316–323.

[28] R. Vavekanand, A. K. Ruzimbaevich and M. Jumaniyozova, 2025, "A study on the extraction of training dataset from fine-tuned language models", *Computer Assisted Methods in Engineering and Science*, **32(3)**: 293–315.

[29] R. Vavekanand, G. Kumar and S. Kurbanova, 2026, "A lightweight physics-conditioned diffusion multi-model for medical image reconstruction", *Biomedical Engineering Communications*, **5(2)**: 12.

[30] R. Vavekanand, T. Kumar, S. Kumar, G. Kumar and A. A. Laghari, 2026, "Multimodal machine learning approaches in predictive healthcare analytics: A comprehensive survey", *Archives of Computational Methods in Engineering*.

[31] R. Vavekanand, A. A. Sathio and M. Sultani, 2026, "A federated learning framework for deep imputation of missing data in heterogeneous ICU time series", *Scientific Reports*.

[32] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò and Y. Bengio, 2018, "Graph attention networks", *International Conference on Learning Representations (ICLR)*.

[33] L. Westhofen, J. C. Jung and D. Neider, 2025, "Temporal conjunctive query answering via rewriting", *Proceedings of the AAAI Conference on Artificial Intelligence*, **39(14)**: 15221–15229.

[34] S. Xiao, J. Li, S. Zhu, Y. Shi, G. Pu and M. Y. Vardi, 2021, "On-the-fly synthesis for LTL over finite traces", **Vol. 35**, pp. 6530–6537, Association for the Advancement of Artificial Intelligence.

[35] E. Yolcu and B. Poczos, 2019, "Learning local search heuristics for boolean satisfiability", *Advances in Neural Information Processing Systems*, **Vol. 32**.

人工智能计算树模型中的线性时序逻辑

刘辰

摘　要

将线性时序逻辑（LTL）融入基于人工智能的计算树模型，是实现人工智能系统形式化验证的一种有效途径。然而，经典的模型检测方法由于受到状态空间爆炸问题的制约，往往难以在大规模复杂系统中实现良好的可扩展性。为此，本文提出了一种以深度学习为支撑的、数据驱动型的新方法，用于高效近似地完成 LTL 模型检测任务。具体而言，本文构建了时序卷积图网络（Temporal Convolutional Graph Network——TCGN）框架，该框架将图神经网络与时序逻辑相结合，从而能够对复杂系统的行为进行验证。基于合成数据集与真实世界数据集的实验结果表明，TCGN 的验证准确率达到 98%，并且在即时验证的有效性与计算效率两个方面均优于 NuSMV 等传统方法。进一步而言，得益于其在对抗扰动条件下所表现出的较强鲁棒性，TCGN 还具备面向大规模人工智能系统开展实时验证的潜力与应用价值。

刘辰　　扬州大学马克思主义学院
lauchan@yzu.edu.cn