International Journal of Computational Methods and Experimental Measurements
https://www.acadlore.com/journals/IJCMEM

Adaptive Logic Learning Architecture: A Hierarchical Framework for Energy-Efficient and Interpretable AI Systems

Ibrahim Haddadi*

Department of Computer Engineering, College of Computer Science and Engineering, Taibah University, 42353 Madinah, Saudi Arabia

* Correspondence: Ibrahim Haddadi (ihaddadi@taibahu.edu.sa)

Received: 12-28-2025 **Revised:** 02-12-2026 **Accepted:** 02-27-2026

Citation: I. Haddadi, “Adaptive logic learning architecture: A hierarchical framework for energy-efficient and interpretable AI systems,” *Int. J. Comput. Methods Exp. Meas.*, vol. 14, no. 1, pp. 17–37, 2026. https://doi.org/10.56578/ijcmem140102.

Abstract: Logic-based machine learning models such as the Tsetlin Machine (TM) have recently gained attention for their energy efficiency and inherent interpretability. However, existing TM-based architectures remain limited in their ability to perform hierarchical feature learning, adapt dynamically to task complexity, and process temporal data efficiently. This paper proposes the Adaptive Logic Learning Architecture (ALLA), a novel hierarchical and energy-aware logic learning framework that addresses these limitations through adaptive clause networks (ACNs), multi-layer logical composition, and TLUs. ALLA enables dynamic clause growth and pruning, supports hierarchical abstraction, and integrates temporal reasoning within a unified propositional logic framework. Experimental results across image classification and sequential recognition tasks show that ALLA improves accuracy over conventional TM models while maintaining substantially lower energy consumption than deep neural network baselines. Hardware synthesis results further confirm the suitability of ALLA for low-power and edge-intelligent systems.

Keywords: Logic-based learning; Hierarchical architectures; Energy-efficient AI; Interpretable machine learning; Hardware acceleration; Learning automata; Edge computing; Adaptive systems

1 Introduction

The proliferation of edge computing devices and Internet-of-Things (IoT) sensors has created an urgent demand for artificial intelligence systems that can operate under severe energy constraints while maintaining high accuracy and interpretability [1]. From wearable health monitors to industrial sensor networks, from autonomous vehicles to smart home devices, the pervasive deployment of AI requires solutions that can process data locally without constant communication with cloud servers. This shift toward edge AI introduces a fundamental tension: traditional neural network approaches achieve impressive accuracy but demand substantial computational resources and energy, making them impractical for battery-powered or energy-harvesting devices.

Traditional neural network approaches, despite their impressive performance on benchmark tasks, face fundamental challenges in edge deployment due to their arithmetic-intensive operations, black-box nature, and substantial energy requirements [2]. A typical deep neural network inference operation involves millions of multiply-accumulate computations, requiring floating-point arithmetic units that consume orders of magnitude more energy than simple logic operations. Furthermore, the lack of interpretability in neural networks raises concerns in safety-critical applications such as medical diagnosis, autonomous driving, and financial decision-making, where understanding the reasoning behind predictions is as important as the predictions themselves. Recent work on Tsetlin Machines (TMs) has demonstrated that propositional logic-based learning can achieve competitive accuracy with significantly reduced energy consumption [3]. TM employs teams of Tsetlin Automata (TA) to learn conjunctive clauses through reinforcement, eliminating the need for gradient-based optimization and floating-point arithmetic. Each automaton learns through simple reward-penalty feedback to include or exclude specific input features (literals) in conjunctive clauses. These clauses are then combined through summation to produce final predictions. Hardware implementations have shown energy efficiency improvements of up to three orders of magnitude compared to equivalent neural network solutions [4], with some implementations achieving as little as 0.142 nJ per classification operation in 65 nm technology.

However, existing logic-based learning approaches, including TMs, exhibit several limitations that constrain their effectiveness in complex real-world settings:

• Limited representational capacity: TM employs a flat clause architecture in which all clauses operate directly on raw input features, lacking hierarchical feature abstraction. This limits its ability to efficiently model compositional patterns that require multi-level reasoning, such as structured spatial relationships in images. As a result, complex patterns must be captured using numerous low-level clauses, leading to either insufficient expressiveness or rapid growth in clause count.

• Fixed architectural constraints: The number of clauses is predefined and remains static during training, preventing adaptive allocation of model capacity. This rigidity hinders efficient learning across heterogeneous input regions, often causing underfitting in complex regions and redundancy in simpler ones.

• Absence of temporal modeling: TM is inherently designed for static data and lacks mechanisms for modeling temporal dependencies. While temporal sequences can be flattened into spatial representations, this approach fails to capture temporal structure and incurs significant dimensionality expansion.

• Scalability challenges: TM represents each feature and its negation as separate literals, resulting in 2d literals for d input features. The exponential growth of possible clause configurations severely limits scalability in high-dimensional domains such as image processing.

• Limited interpretability granularity: Although TM is logic-based and transparent at the clause level, individual clauses may lack clear semantic meaning in complex tasks. Moreover, interactions among multiple clauses at the aggregation stage can obscure global decision logic, reducing interpretability at the system level.

This paper proposes the Adaptive Logic Learning Architecture (ALLA), a hierarchical and energy-efficient logic-based learning framework that overcomes the scalability and representational limitations of existing TM–based models while preserving interpretability. ALLA introduces structured hierarchy, adaptivity, and temporal reasoning to enable effective learning on complex spatial and sequential data.

The architecture employs Hierarchical Logic Layers (HLLs) to enable compositional feature learning, where lower layers capture primitive patterns and higher layers form increasingly abstract representations. This hierarchical organization reduces per-layer complexity while improving representational power. To further enhance efficiency, adaptive clause networks (ACNs) dynamically generate and prune logic clauses during training based on utility-driven criteria, allocating model capacity only where needed.

To support sequential data, ALLA integrates Temporal Logic Units (TLUs) that learn interpretable temporal patterns using logic-based state transitions. In addition, multi-resolution logic processing enables parallel extraction of coarse global and fine-grained local features, improving robustness across diverse data modalities.

A complete hardware architecture is presented and synthesized in 28 nm Complementary Metal-Oxide-Semiconductor (CMOS) technology, demonstrating practical feasibility and improved energy efficiency compared to both conventional TMs and neural network implementations. Extensive experiments on image and sequential benchmarks show consistent accuracy gains while maintaining low-energy operation. Finally, an interpretability analysis framework is introduced to support clause visualization, hierarchical feature tracing, and instance-level explanation. From a computational method perspective, ALLA should be viewed as a discrete, non-gradient learning framework rather than a conventional AI system architecture. The proposed contributions focus on how logical computation, reinforcement-based learning dynamics, and hierarchical composition interact to produce efficient and interpretable learning behavior. While architectural realizations and hardware synthesis are presented, they serve primarily to validate the computational efficiency and practical feasibility of the underlying learning method. Accordingly, this work aligns with the scope of computational methods by emphasizing algorithmic flow, learning dynamics, convergence behavior, and energy-aware computation rather than system-level design alone. Unlike conventional AI system papers, this manuscript is structured as a computational methods study: the proposed learning computation is formalized, analyzed, and experimentally characterized, while architectural and hardware realizations are presented only to validate computational efficiency and practical feasibility.

The remainder of this paper is organized as follows. Section 3 reviews related work and motivation in logic-based learning and energy-efficient artificial intelligence. Section 4 presents the proposed ALLA, including HLLs and ACNs. Section 5 details the hardware implementation of ALLA. Section 6 reports the experimental validation and performance evaluation. Section 7 analyzes the interpretability and explainability properties of the proposed model. Section 8 discusses limitations and implications, and Section 9 concludes the paper.

2 Computational Methodology

This work is positioned as a computational methods study that investigates how discrete logic, reinforcement-driven learning, and hierarchical composition can be combined to achieve efficient and interpretable learning under strict energy constraints. Rather than introducing a new AI system or hardware architecture, the proposed framework defines a computational learning method characterized by non-gradient optimization, binary logic operations, and adaptive structural complexity.

At its core, the method operates on three computational principles. First, learning is performed through teams of finite-state automata that optimize logical clause configurations via reward–penalty feedback, eliminating the need for numerical gradient computation. Second, hierarchical composition is used as a computational strategy to reduce complexity by decomposing high-dimensional pattern recognition into a sequence of lower-dimensional logical transformations. Third, adaptive clause growth and pruning dynamically regulate computational capacity, enabling the method to allocate resources where they are most informative while maintaining bounded memory and energy usage.

From a computational perspective, inference consists exclusively of Boolean logic evaluation and integer accumulation, while learning consists of stochastic but bounded state transitions within finite automata. These properties yield predictable computational cost, stable learning dynamics, and hardware-efficient execution. Subsequent sections formalize the learning procedure, analyze convergence and sample complexity, and empirically study the computational behavior of the method across image and sequential benchmarks.

3 Related Work and Motivation

3.1 Logic-Based Machine Learning

Logic-based machine learning has long been valued for its interpretability, formal reasoning capability, and compatibility with symbolic knowledge representation. Early work in inductive logic programming demonstrated the feasibility of learning human-interpretable rules from data [5], but such methods scale poorly to high-dimensional and noisy perceptual inputs.

Recent differentiable logic frameworks attempt to bridge symbolic reasoning and statistical learning through continuous relaxations of logical operators [6]. While conceptually appealing, these approaches rely on gradient-based optimization and floating-point arithmetic, largely forfeiting the hardware simplicity and energy efficiency that motivate logic-based learning for edge deployment.

The TM addresses these limitations by replacing gradient descent with reinforcement-driven learning automata [3]. TM learns propositional logic clauses using simple reward–penalty updates, enabling fully binary computation and eliminating numerical optimization. Hardware implementations demonstrate extreme energy efficiency, with reported costs as low as 142 nJ per inference [4].

Despite these advantages, TM suffers from a fundamental architectural limitation: its flat clause structure prevents hierarchical feature abstraction. All clauses operate directly on raw input features, forcing complex compositional patterns to be represented explicitly at a single level. While stacked or ensemble TM variants have been proposed [7], they lack coordinated end-to-end learning and provide limited scalability gains. These limitations motivate the need for a hierarchical and adaptive logic-based learning framework, which the proposed ALLA addresses.

3.2 Energy-Efficient AI Hardware

Energy-efficient AI research has pursued multiple strategies targeting different aspects of the computation–energy tradeoff. System-level and architectural optimizations, including reduced numerical precision and data reuse, have been widely explored to improve energy efficiency in machine learning workloads [8], but selecting safe approximation levels remain application-specific and may lead to accuracy degradation. Neuromorphic computing adopts spiking neural networks implemented in specialized hardware [9]. Although event-driven processing can be energy efficient, neuromorphic systems face challenges in training complexity, dense-signal processing, and achieving competitive accuracy.

Binarized Neural Networks (BNNs) replace arithmetic operations with binary logic to reduce inference cost [10]. However, BNNs still rely on gradient-based training with full-precision parameters, limiting end-to-end efficiency. Even advanced BNN accelerators consume hundreds of femtojoules per operation in modern technology nodes [11], exceeding the efficiency reported for TM-based hardware.

These observations highlight a key limitation of many low-power AI approaches: reducing numerical precision alone is insufficient. Eliminating gradient-based optimization entirely is critical for achieving extreme efficiency. By preserving reinforcement-driven learning and purely logical computation while introducing hierarchical representation, ALLA addresses this limitation directly.

3.3 Hierarchical Feature Learning

Hierarchical feature learning underpins the success of deep learning on structured data. Convolutional neural networks (CNNs) progressively learn low-level edges, mid-level textures, and high-level object parts across layers [12], enabling compositional representation and strong generalization.

Several efforts have attempted to introduce hierarchy into logic-based learning. Stacked TMs propagate clause outputs across layers [7], but each layer is trained independently using only local feedback. As a result, lower layers are not guided by higher-level objectives, often producing features misaligned with global task requirements.

ALLA overcomes this limitation by introducing utility-weighted hierarchical feedback that coordinates learning across layers without gradient computation. Lower-level clauses are reinforced in proportion to their contribution to high-utility higher-level representations, enabling scalable hierarchical abstraction while preserving energy efficiency. Core Architecture vs. Optional Extensions: To clarify the contribution boundary, we distinguish between the core components of ALLA and its optional extensions. The core architecture consists of: (i) HLLs, (ii) ACNs with growth and pruning, and (iii) utility-weighted hierarchical feedback. These components together define ALLA and are used in all experiments unless stated otherwise.

TLUs and multi-resolution logic processing are optional extensions that build upon the core architecture. They are introduced to demonstrate the flexibility of ALLA for sequential data and multi-scale feature learning, but are not required for its fundamental operation.

3.4 Temporal and Sequential Processing

Many real-world applications require effective modeling of temporal dependencies. Recurrent neural networks and their variants dominate this domain but incur high computational cost and rely on backpropagation through time [13]. Their distributed hidden states also limit interpretability.

Logic-based temporal reasoning has traditionally focused on manually specified rules rather than data-driven learning. Bridging learnable temporal modeling with efficient and interpretable logic representations remains challenging.

ALLA addresses this gap through TLUs that learn interpretable temporal clauses using automata-based state transitions. Temporal dependencies are captured through explicit logical conditions over finite windows, enabling efficient hardware implementation using simple sequential logic rather than arithmetic computation. This approach provides an energy-efficient and explainable alternative to recurrent architectures for sequential learning.

4 Adaptive Logic Learning Architecture

To improve readability for a broad computational methods audience, the presentation emphasizes conceptual understanding and computational behavior before formal detail. Readers primarily interested in high-level learning dynamics and algorithmic insights may focus on the subsection titled "Computational Flow and Learning Dynamics," while subsequent subsections provide formal and implementation-level details for completeness.

Although this section is organized around architectural components for clarity, its primary objective is to describe the computational framework underlying the ALLA. Specifically, the section explains how logical computation, reinforcement-based learning dynamics, and hierarchical composition interact to realize efficient and interpretable learning behavior. Architectural elements are introduced only insofar as they define the flow of computation, the structure of logical operations, and the mechanisms governing adaptation and convergence. Accordingly, the emphasis is placed on algorithmic roles, computational properties, and learning dynamics rather than on system-level implementation.

4.1 Architectural Overview

Figure 1 illustrates the overall structure of the proposed ALLA. The architecture is organized as a hierarchy of HLLs arranged in a pyramidal fashion, where each layer contains multiple ACNs responsible for learning conjunctive logic clauses over their respective inputs. For sequential or time-dependent data, TLUs are integrated at appropriate layers to capture temporal dependencies. In addition, a multi-resolution fusion mechanism enables the combination of features extracted at different spatial or temporal scales prior to final classification. This modular design allows ALLA to be flexibly configured to accommodate diverse problem domains and resource constraints.

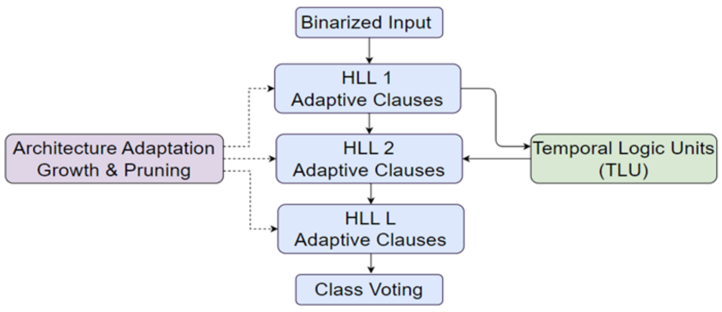


Figure 1. Hierarchical training and feedback flow of Adaptive Logic Learning Architecture (ALLA) with adaptive precision

Unlike the flat, single-layer structure of conventional TMs, where all clauses operate directly on raw input features, ALLA processes information hierarchically across multiple abstraction levels. This hierarchical organization enables progressive feature composition, improved scalability, and enhanced interpretability.

Layer 0 (input layer): Raw inputs are first binarized and expanded into positive and negated literals, forming the initial literal set $L_0 = \{x_1, x_1, x_2, x_2, \ldots, x_d, x_d\}$, where d denotes the input dimensionality. For image-based data, spatial locality is preserved by constraining early-layer clauses to operate on local receptive fields rather than arbitrary pixel combinations. Similarly, for sequential data, temporal locality is enforced to maintain meaningful temporal structure. These structured connectivity constraints reduce the hypothesis search space and introduce inductive biases aligned with the data modality.

Intermediate layers (1 to L-1): Each intermediate layer l receives as input the binary clause outputs from the preceding layer, forming a new literal set $L_l = \left\{C_i^{(l-1)}, \neg C_i^{(l-1)} \mid i = 1, 2, \ldots\right\}$. The ACNs at this level learn conjunctive clauses that compose lower-level features into increasingly abstract representations. Importantly, all layers operate exclusively on binary inputs and outputs, preserving a uniform logical representation throughout the hierarchy and enabling efficient hardware realization.

Output layer (L): The final layer aggregates high-level clause outputs to perform classification. In the binary case, positive and negative clauses vote for their respective classes, and the prediction is determined by the difference in accumulated votes. For multi-class classification, independent clause groups vote for each class, and the predicted label is obtained via an argmax operation, consistent with standard TM-based classifiers.

This hierarchical architecture offers several key advantages over flat logic-based models: (1) reduced clause complexity at each layer, as higher-level patterns are learned from preprocessed features rather than raw inputs; (2) effective feature reuse, where intermediate representations can support multiple higher-level concepts; (3) improved interpretability through hierarchical semantic decomposition, allowing learned features to be examined at different abstraction levels; and (4) enhanced scalability to high-dimensional inputs, as the pyramidal structure progressively reduces representational complexity.

This subsection describes the computational behavior of ALLA from an algorithmic perspective, independent of physical implementation. The goal is to clarify how logical operations, reinforcement-driven updates, and hierarchical composition interact during learning and inference.

Forward computational flow: For each input sample, computation proceeds deterministically through a sequence of logical transformations. The input is binarized and expanded into literals, which are evaluated by conjunctive logic clauses at the first layer. Each clause computes a Boolean conjunction over selected literals, producing a binary output. These clause outputs form the input to the next layer, enabling hierarchical composition of increasingly abstract logical features. Across all layers, computation is purely discrete, involving only logical AND, NOT, and integer accumulation operations, with no floating-point arithmetic.

Learning and update dynamics: Learning is driven by reinforcement rather than gradient descent. Each TA updates its state through a bounded random walk based on Type I and Type II feedback. Correct include/exclude decisions are reinforced with higher probability than incorrect ones, resulting in convergence toward stable clause configurations. The learning process is stochastic but exhibits predictable aggregate behavior, with clause utilities stabilizing as training progresses.

Hierarchical coordination: In contrast to independent layer-wise learning, ALLA propagates utility-weighted reinforcement signals across layers. Lower-layer clauses receive feedback scaled by the utility of higher-layer clauses that depend on them, creating coordinated learning dynamics across the hierarchy. This mechanism aligns local clause updates with global learning objectives without requiring gradient computation or backpropagation.

Computational complexity and stability: For a fixed architecture, the computational cost per training sample is linear in the number of clauses and their average width. Clause growth and pruning introduce non-stationarity but occur infrequently, allowing learning dynamics to stabilize between adaptations. Empirically, clause counts and utilities converge to steady regimes, indicating stable computational behavior rather than uncontrolled architectural growth.

4.2 Adaptive Clause Networks

The ACNs extends TM's fixed clause structure with dynamic adaptation mechanisms that allow the architecture to grow and shrink during training. Rather than specifying the number of clauses N_{clauses} before training—which requires either domain expertise or expensive hyperparameter search—ACN starts with a minimal set and adapts based on learning dynamics and performance requirements.

Clause representation: Each clause C_i is represented as a conjunction over a subset of available literals: $C_i = \wedge_{j \in \mathcal{J}_i} l_j, \mathcal{J}_i \subseteq \{1, 2, \ldots, |\mathcal{L}|\}$ is the set of included literal indices and $l_j \in$ L is a literal (either a feature or its negation). The clause outputs 1 (true) only when all included literals are true, and outputs 0 (false) otherwise. As in standard TM, each literal has an associated TA that learns through reinforcement whether to include that literal in the clause. The automaton has 2N states divided into two groups: states 1 to N representing “exclude” and states N + 1 to 2N representing “include”, with higher state numbers indicating stronger commitment to the current action.

Clause utility metric: A key innovation in ACN is the clause utility metric that quantifies how much each clause contributes to the overall learning objective. We define the utility of clause C_i as:

$$U\left(C_i\right) = \alpha \cdot I\left(C_i; Y\right) + (1-\alpha) \cdot A\left(C_i\right) \tag{1}$$

where, $I\left(C_i; Y\right)$ is the mutual information between the clause output and the target labels, computed as:

$$I\left(C_i; Y\right) = H(Y) - H\left(Y \mid C_i\right) \tag{2}$$

where, $H(\cdot)$ denoting entropy. This term measures how much knowing the clause output reduces uncertainty about the label. The activation diversity $A\left(C_i\right)$ is computed as:

$$A\left(C_i\right) = -(1/|D|) \sum (x \in D) C_i(x) \log C_i(x) \tag{3}$$

Which penalizes clauses that fire too rarely (low activation) or too frequently (nearly constant activation), favoring clauses that discriminate among inputs. The parameter $\alpha \in [0,1]$ balances these two criteria, with typical values around 0.7 emphasizing information content while maintaining some diversity requirement.

Growth mechanism: When the average utility across all existing clauses exceeds a threshold τ_{grow} and training error remains above a target level ϵ_{target}, ACN spawns a new clause. The high average utility indicates that existing clauses are working well, but remaining error suggests additional capacity is needed. The new clause’s automata are initialized strategically rather than randomly: they are biased toward including literals that appear frequently in misclassified samples, computed through weighted feedback where misclassified samples contribute more to the initialization distribution. This targeted initialization helps new clauses quickly specialize on difficult cases that existing clauses handle poorly.

Pruning mechanism: Clauses with persistently low utility are candidates for removal. We track utility through an exponential moving average:

$$\bar{U}_{t(c_i)} = \beta \bar{U}_{(t-1)(c_i)} + (1-\beta) U_{t(c_i)} \tag{4}$$

If $\bar{U}\left(C_i\right) < \tau_{\text{prune}}$ for k consecutive evaluation points, the clause is removed from the network. This pruning prevents overfitting to noise (clauses that memorize specific training samples without generalizing), maintains computational efficiency (fewer clauses to evaluate), and frees up representational capacity for more useful patterns. In hardware, pruned clauses are disabled rather than physically removed, allowing potential reactivation if useful patterns emerge later in training.

The complete adaptation process can be summarized algorithmically as:

Algorithm 1 Adaptive clause growth and pruning

```
 1: Compute utilities: $U(C_i)$ for all active clauses
 2: Update moving averages: $\bar{U}(C_i)$
 3: **if** $\mathrm{mean}(\{U(C_i)\}) > \tau_{\mathrm{grow}}$ and error $> \epsilon_{\mathrm{target}}$ **then**
 4:     Analyze misclassified samples to identify needed patterns
 5:     Spawn new clause with targeted initialization
 6:     Add clause to active set
 7: **end if**
 8: **for** each clause $C_i$ **do**
 9:     **if** $\bar{U}(C_i) < \tau_{\mathrm{prune}}$ for $k$ epochs **then**
10:         Disable clause $C_i$ (hardware gated)
11:     **end if**
12: **end for**
```

4.3 Hierarchical Logic Layers

The hierarchical structure of ALLA requires careful design of information flow between layers and mechanisms for coordinated learning across the hierarchy. Simply stacking independent TM layers would fail to leverage the hierarchical structure effectively.

Inter-layer connectivity: Layer l receives as input the binary clause outputs from layer l-1. For a layer with N_{l-1} clauses producing binary outputs, the literal set for the next layer becomes: $\mathcal{L}_l = \left\{ C_i^{(l-1)}, C_i^{(l-1)} \mid i = 1, 2, \ldots, N_{l-1} \right\}$ containing $2N_{l-1}$ literals. This design maintains the binary nature of computation throughout the hierarchy—clauses at every level perform conjunctions over binary values, enabling uniform hardware implementation and preserving the efficiency advantages of logic operations.

The connectivity pattern can be configured in different ways depending on the problem structure. For fully connected layers, every clause at layer l can potentially include any literal from layer l-1. For spatially structured data like images, local connectivity limits each clause to literals from clauses with nearby receptive fields in the layer below, enforcing spatial locality and reducing the search space for learning. The connectivity pattern encodes useful inductive biases about the problem structure.

Hierarchical feedback propagation: The key challenge in hierarchical learning is propagating feedback to lower layers in a way that guides them toward learning useful features for higher layers. In neural networks, backpropagation computes gradients through the chain rule, providing explicit gradient signals to each layer. ALLA employs hierarchical reinforcement, where feedback to layer l-1 clauses depend on the utility of layer l clauses that use them.

Specifically, when updating automata in clause C_i^{l-1} at layer l-1, we weight the standard TM feedback signal by a factor β_i computed as:

$$\beta_i = \frac{\sum_{j: C_i^{l-1} \in C_j^l} U\left(C_j^l\right)}{\sum_{j=1}^{N_l} U\left(C_j^l\right)} \tag{5}$$

where, the numerator sums utilities of all clauses in layer l that include literal C_i^{l-1} (or its negation), and the denominator normalizes across all clauses in layer l. This weighting gives stronger learning signals to clauses that contribute to useful higher-level patterns, while reducing learning rates for clauses that higher layers don't find useful.

The hierarchical feedback creates an implicit optimization objective that coordinates learning across layers: lower layers are rewarded for producing features that enable higher layers to solve the overall task effectively.

Layer-wise training strategy: We employ a hybrid training approach that balances training stability with optimization quality. Training proceeds in two phases:

Phase 1—Bottom-up initialization: Layers train sequentially from bottom to top, with each layer training to convergence before initializing the next layer. This provides stable feature representations at each level before higher layers begin learning. Layer 0 (if present as a preprocessing layer) learns basic primitives. Layer 1 trains using standard TM feedback to learn useful compositions of layer 0 outputs. Layer 2 then trains on the now-stable layer 1 outputs, and so forth.

Phase 2—Joint Refinement: After all layers have been initialized, all layers train jointly with hierarchical feedback enabling end-to-end refinement. This joint training allows layers to adapt to each other, with lower layers adjusting their features based on what higher layers need, and higher layers adapting to the evolving feature representations from below. The hierarchical feedback weighting ensures coordinated adaptation rather than chaotic simultaneous changes.

This two-phase strategy provides better performance than either pure bottom-up training (which lacks end-to-end optimization) or random initialization with joint training (which can be unstable due to simultaneously learning features at all levels).

4.4 Temporal Logic Units

For sequential data processing, ALLA incorporates TLUs that extend clause learning to temporal dimensions while maintaining the interpretability and efficiency of logic representations.

Temporal clause structure: A temporal clause C^T operates over a sliding window of w time steps:

$$C^T = \Lambda_{t=0}^{w-1} \Lambda_{j \in J_t} l_j^t \tag{6}$$

where, $l^{(t)}$ denotes literal j at time offset t within the window. The clause matches (outputs 1) only when all required literals are true at their respective time offsets within the window. This structure naturally expresses temporal patterns: "first observe feature A, then feature B, then feature C" can be represented as a conjunction requiring A at time 0, B at time 1, and C at time 2.

State-based automata: TLU employs finite state automata that track temporal pattern progression. Each state corresponds to partial satisfaction of the temporal clause—which temporal conditions have been met so far. State transitions occur based on literal values at each time step, with final accepting states indicating complete pattern match. For a temporal clause requiring literals at times 0, 2, and 4, the FSA has states representing: (1) no conditions met, (2) time-0 condition met, (3) time-0 and time-2 conditions met, and (4) all conditions met (accepting state).

The FSA structure enables efficient streaming evaluation: as new inputs arrive, the FSA transitions states without needing to store the entire temporal window explicitly. Only the current state must be maintained, dramatically reducing memory requirements compared to explicitly storing w previous observations.

Temporal feedback: Learning in TLU uses delayed feedback with temporal credit assignment. When a temporal clause successfully matches (or fails to match) a pattern and feedback is received about prediction correctness, this feedback must be propagated backward through the time window to adjust automata at each time step. The temporal credit assignment uses an exponential decay factor:

$$f_t = \gamma^{w-t} \cdot f_w \tag{7}$$

where, f_t is the feedback signal at time step t, f_w is the outcome feedback at time w, and $\gamma \in (0, 1]$ is the temporal discount factor. This decay reflects uncertainty about credit assignment—earlier time steps contributed to the pattern match but may be less directly responsible than later steps.

Integration with hierarchy: TLUs can be inserted at any layer of the hierarchy, enabling temporal processing at multiple abstraction levels. Lower layers learn primitive temporal patterns such as edge motion in video (spatial edges moving across frames) or phoneme transitions in speech (acoustic features changing in characteristic ways). Higher layers compose these primitive temporal patterns into complex temporal events such as gestures (specific motion sequences), actions in video (walking, running, jumping), or words in speech (sequences of phonemes). This hierarchical temporal processing mirrors how biological systems process temporal information at multiple scales.

4.5 Multi-Resolution Logic Processing

Complex patterns often require understanding at multiple spatial or temporal scales simultaneously. ALLA incorporates parallel pathways processing different feature resolutions or temporal granularities.

Resolution levels: For spatial data like images, we create multiple input representations at different scales through pooling or subsampling. A coarse pathway processes down sampled inputs capturing global structure, a medium pathway processes intermediate resolution capturing regional patterns, and a fine pathway processes full resolution capturing local details. Each resolution feeds into a separate pathway of HLLs, creating multiple parallel hierarchies. For temporal data, different window sizes in TLUs create temporal multi-resolution: short windows capture immediate transitions while long windows capture extended patterns.

Fusion mechanism: Outputs from different resolution pathways merge through a fusion layer that learns cross-resolution clause patterns. This fusion uses the same ACN structure but takes inputs from multiple pathway outputs:

$$L_{\text{fusion}} = L_{\text{coarse}} \cup L_{\text{medium}} \cup L_{\text{fine}} \tag{8}$$

Fusion clauses can express patterns like "coarse structure matches object template AND fine details match expected texture" or "long-term trend is upward AND recent fluctuations show characteristic pattern," combining information across scales.

Attention-based weighting: Not all resolutions are equally important for all patterns or all inputs. We implement a simple attention mechanism using clause utility to weight contributions from different pathways. For each fusion clause, we track which resolution pathways it draws literals from. High-utility fusion clauses that primarily use coarse-pathway literals suggest global pattern recognition dominates for that pattern, while high-utility clauses using fine-pathway literals indicate detail-dependent decisions. This information can guide interpretation and can be used to prune less-useful pathways or resolution levels for specific problem classes.

4.6 Formal Learning Overview

This subsection provides a formal characterization of the ALLA learning procedure to offer computational insight into learning dynamics, convergence behavior, and scalability rather than to present exhaustive theoretical guarantees. The analysis complements the empirical results by clarifying why the proposed framework exhibits stable and efficient learning behavior. Readers primarily interested in algorithmic intuition and experimental findings may skip this subsection on a first reading without loss of continuity.

4.7 Training Algorithm

ALLA learning proceeds by alternating between hierarchical inference, reinforcement-based feedback, coordinated feedback propagation across layers, and periodic architectural adaptation. The overall training flow is illustrated in Figure 2, while Algorithm 2 provides a formal description.

4.8 Training Procedure Explanation

Algorithm 2 presents the complete training procedure of the proposed ALLA, while the overall training and feedback flow is summarized in Figure 2. Together, they illustrate how hierarchical inference, reinforcement-based learning, and architectural adaptation are integrated into a unified training framework.

Algorithm 2 Training algorithm for ALLA

```
1: **Input:** Training data $D = \{(x^{(i)}, y^{(i)})\}_{i=1}^{N}$
2: **Parameters:** Learning rate $\alpha$; thresholds $\tau_{\text{grow}}, \tau_{\text{prune}}$
3: **Initialize:** Tsetlin Automata to mid-state; minimal clause sets
4: for epoch $e = 1$ to $E$ do
5:     Shuffle $D$
6:     for each sample $(x, y) \in D$ do
                                                        ▷ Forward inference
7:         $\mathcal{L}_0 \leftarrow$ binarize and negate $x$
8:         for layer $l = 1$ to $L$ do
9:             Update clauses $C_i^l \leftarrow \bigwedge_{j \in \mathcal{I}_i^l} l_j^{l-1}$
10:            $\mathcal{L}_l \leftarrow \{C_i^{(l)}, \neg C_i^{(l)}\}$
11:        end for
12:        Compute class votes $v_{(c)} = \sum C_{(i)}^{(L)} - \sum C_{(j)}^{(L)}$
13:        $\hat{y} \leftarrow \arg\max_{(c)} v_{(c)}$
                                                        ▷ Feedback
14:        Generate Type I feedback if $\hat{y} = y$; otherwise Type II
                                                        ▷ Hierarchical update
15:        for layer $l = L$ down to 1 do
16:            Compute utilities $U(C_{(i)}^{(l)})$ and weights $\beta_{(i)}^{(l-1)}$
17:            Apply weighted Tsetlin Automata updates
18:        end for
                                                        ▷ Architecture adaptation
19:        if sample count mod $N_{\text{adapt}} = 0$ then
20:            Spawn clauses if $\bar{U}^{\,l} > \tau_{\text{grow}}$ and error $> \epsilon_{\text{target}}$
21:            Prune clauses if $\bar{U}\left(C_i^l\right) < \tau_{\text{prune}}$
22:        end if
23:    end for
24: end for
```

As shown in Figure 2, training proceeds iteratively over epochs and samples. For each input–label pair (x, y), the algorithm performs a forward inference pass through the HLLs. The input is binarized and expanded into positive and negated literals, forming the initial literal set L_0. Clause evaluation is then carried out layer by layer, where each clause computes a conjunction over selected literals from the preceding layer, producing a new binary literal set for the next layer.

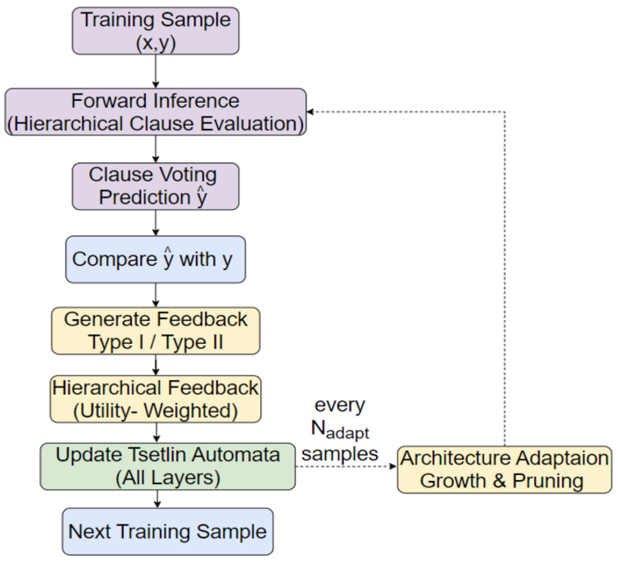
At the output layer, clause activations are aggregated through a voting mechanism to compute class scores, and the predicted label $\hat{y}$ is obtained via an argmax operation. The prediction is compared with the true label y to generate learning feedback: correct predictions trigger Type I feedback to reinforce useful clauses, while incorrect predictions trigger Type II feedback to penalize misleading patterns.

Learning signals are then propagated backward through the hierarchy using utility-weighted reinforcement, as indicated by the dashed paths in Figure 2. Clause utilities are computed at each layer and used to scale feedback to lower layers, ensuring coordinated learning across abstraction levels without gradient-based optimization.

In addition, ALLA supports dynamic architectural adaptation. Clause growth and pruning are triggered periodically every N_{adapt} samples, allowing layers with high utility but persistent error to add new clauses, while clauses with consistently low utility are pruned. This mechanism enables automatic adjustment of representational capacity during training.

The computational complexity per training sample is $O(L \cdot \bar{N}$ clause $\cdot \bar{k})$, where L is the number of layers, $\bar{N}$

clause is the average number of clauses per layer, and $\bar{k}$ is the average clause width. Since architectural adaptation occurs infrequently, its impact on overall complexity remains negligible.



Training Sample (x,y)
Forward Inference (Hierarchical Clause Evaluation)
Clause Voting Prediction ŷ
Compare ŷ with y
Generate Feedback Type I / Type II
Hierarchical Feedback (Utility- Weighted)
Update Tsetlin Automata (All Layers)
Next Training Sample
every N_adapt samples
Architecture Adaptaion Growth & Pruning

Figure 2. Training and hierarchical feedback flow of the Adaptive Logic Learning Architecture (ALLA)
Note: Forward inference proceeds layer-by-layer to generate predictions, followed by utility-weighted feedback and adaptive clause growth and pruning.

4.9 Convergence Analysis

We analyze the convergence properties of ALLA learning under certain reasonable assumptions about the problem structure and learning environment.

Assumption 1 (representability): The target function can be approximated to within error ϵ by a hierarchical logic formula with bounded depth L, bounded clause width k, and bounded total number of clauses N_{total}.

Assumption 2 (feedback quality): Feedback probabilities are chosen such that automata making correct include / exclude decisions receive reward more frequently than penalty over time, with reward probability exceeding penalty probability by at least $\delta > 0$.

Assumption 3 (adaptation stability): The clause adaptation mechanism allows sufficient training between architecture changes such that automata can learn before new clauses are added or existing clauses are removed.

Theorem 1 (convergence): Under Assumptions 1-3, the probability that ALLA converges to an ϵ-optimal solution approaches 1 as the number of training epochs E increases:

$$\lim_{E \longrightarrow \infty} P(\text{error} < \epsilon) = 1 \tag{9}$$

Proof Sketch: The proof extends convergence results for learning automata to the hierarchical setting. Each TA performs a random walk on its state space with 2N states. Under Assumption 2 (feedback quality), this walk has positive drift toward the optimal action when the automaton is in the wrong action group and negative drift (toward absorbing states) when in the correct action group. For finite state spaces with appropriate learning rates, Markov chain analysis shows the probability of convergence to optimal actions approaches 1 exponentially with the number of updates.

The hierarchical structure introduces dependencies between layers—lower layer automata receive feedback influenced by higher layer performance. However, the utility-weighted feedback (Eq. (5)) ensures these dependencies guide lower layers toward supporting higher-layer optimization rather than introducing conflicting signals. The hierarchical feedback weighting acts as an implicit coordination mechanism.

The adaptation mechanism (growth and pruning) could potentially interfere with convergence by changing the architecture during learning. However, Assumption 3 ensures sufficient stability between changes. Furthermore, growth only occurs when existing clauses have converged (high utility) but error remains, and pruning only removes persistently low-utility clauses. These conditions ensure adaptation improves rather than disrupts the learning trajectory.

4.10 Sample Complexity

Theorem 2 (Sample complexity): For a problem requiring N_{clause} total clauses across all layers with maximum clause width k, ALLA requires $O\left(N_{\text{clauses}} \cdot k \cdot \log(n)\right)$ samples to achieve ϵ-optimal accuracy with probability at least $1-\delta$, where n is the number of automaton states.

Proof Sketch: Each automaton must learn one bit of information (include or exclude). For an automaton with n states, learning automata theory shows $O(\log n)$ samples suffice to converge with high probability when feedback quality (Assumption 2) holds. Each clause has |L| automata, but typically only $k \ll$ | L| automata learn to include their literals. Across N_{clauses} total clauses in the architecture, we have $O\left(N_{\text{clauses}} \cdot k\right)$ automata that must learn, yielding $O\left(N_{\text{clauses}} \cdot k \cdot \log n\right)$ total sample complexity.

This sample complexity is comparable to TM's flat architecture. However, the crucial advantage appears in the required N_{clauses} for a given problem. For problems with compositional structure—where high-level patterns decompose into mid-level patterns which decompose into primitive features—hierarchical representations can reduce the required number of clauses exponentially compared to flat architectures. A flat architecture must explicitly learn all combinations of primitives, while hierarchical architecture can reuse learned features: one mid-level feature can participate in many high-level patterns.

4.11 Interpretability Properties

A key advantage of ALLA is maintained interpretability despite increased architectural complexity. We formalize several interpretability properties that ALLA provides.

Property 1 (Hierarchical traceability): Any prediction can be traced through the hierarchy to identify which clauses at each level contributed. For prediction $\hat{y}$, we can identify the set of top-layer clauses $\left\{C^L\right\}$ that voted for $\hat{y}$, then recursively identify which layer $L-1$ clauses caused each C^L to fire, continuing down to input features.

Property 2 (Semantic correspondence): AT each layer l, clauses correspond to features at abstraction level l. Layer 0 clauses detect primitive features (edges, colors, textures). Layer l clauses detect combinations of primitives (corners, simple shapes). Layer 2 clauses detect higher-level patterns (object parts), and so forth. This correspondence enables semantic labeling through analysis of activation patterns.

Property 3 (Counterfactual explanations): By identifying minimal clause subsets sufficient for a prediction and tracing these to input features, ALLA can generate counterfactual explanations of the form: "If these specific input features were different, the prediction would change to class y'."

Property 4 (Global model understanding): The complete set of learned clauses at all layers constitutes an interpretable model of the decision function. Unlike neural network weights which lack direct semantic interpretation, logic clauses can be examined, analyzed, and understood by domain experts.

These properties make ALLA particularly suitable for applications requiring explainable AI, such as medical diagnosis, financial decision-making, and autonomous systems where understanding the reasoning process is as important as prediction accuracy.

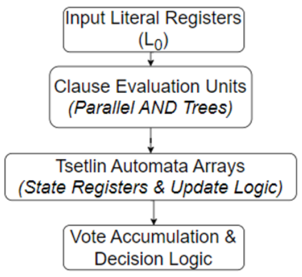
5 Hardware Implementation

This section is included to validate the computational efficiency, scalability, and practical feasibility of the proposed ALLA rather than to present a hardware-centric contribution. The hardware realization serves as concrete evidence that the underlying computational method—based on discrete logic operations and reinforcement-driven learning dynamics—can be implemented efficiently in real-world systems. Accordingly, the focus is on corroborating the computational trends observed in software-level experiments rather than on detailed circuit innovation. Readers primarily interested in the computational and algorithmic aspects may skip this section without loss of continuity.

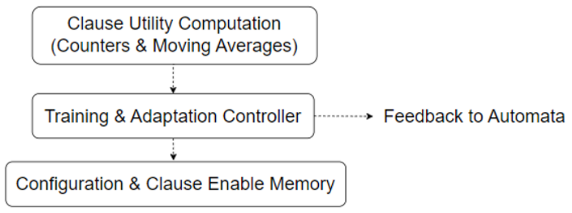
Recent studies have explored FPGA-based emulation of emerging non-volatile devices, such as memristors, to enable rapid prototyping and evaluation of energy-efficient computing architectures [14]. Figure 3 illustrates the hardware mapping of the ALLA, explicitly separating the logic data path and the control and adaptation logic to clarify their distinct roles. The architecture extends the maximally parallel design philosophy of conventional TM hardware by introducing hierarchical processing, clause utility monitoring, and centralized adaptive control while preserving a fully logic-based and energy-efficient implementation.

Layer modules and Datapath organization: As shown in Figure 3a, each HLL is realized as a dedicated Datapath module composed of multiple ACNs. The data path of each layer consists of: (1) input literal registers that store binary clause outputs from the preceding layer, (2) clause evaluation units that compute conjunctive logic functions in parallel, (3) arrays of TA that maintain clause states and perform reinforcement-based updates, and (4) vote accumulation and decision logic that aggregates clause outputs for classification.

Within each ACN, clause evaluation is implemented using balanced AND-tree structures, consistent with prior TM hardware designs. For a clause containing k included literals, the corresponding AND tree has logarithmic depth, resulting in a gate delay of $O(\log k)$. All clauses within a layer are evaluated concurrently, enabling one inference per clock cycle after the pipeline is filled.


Input Literal Registers (L_0)
Clause Evaluation Units (Parallel AND Trees)
Tsetlin Automata Arrays (State Registers & Update Logic)
Vote Accumulation & Decision Logic

(a) ALLA datapath. Binarized literals are processed through parallel clause evaluation, automata-based state updates, and vote aggregation to produce predictions.


Clause Utility Computation (Counters & Moving Averages)
Training & Adaptation Controller
Feedback to Automata
Configuration & Clause Enable Memory

(b) Control and adaptation logic. Clause utilities drive hierarchical feedback, clause growth, and pruning through a centralized controller and configuration memory.

Figure 3. Hardware mapping of ALLA: (a) parallel datapath; (b) control and adaptation logic

Automaton implementation: Each TA is implemented as a compact finite-state machine with 2N states. The automaton state is stored in a small register requiring $\lceil \log_2(2N) \rceil$ bits (typically 6–8 bits for common configurations with $N \in \{50, 100\}$). Update logic increments or decrements the state based on received feedback, with saturation at boundary states. Stochastic feedback decisions are supported using lightweight Linear Feedback Shift Registers (LFSRs), shared across small clause groups to minimize area and power overhead.

Inter-layer communication: Clause outputs generated by layer l are forwarded to layer $l + 1$ through register-based pipeline stages, as illustrated in Figure 3a. This pipelined organization enables continuous data streaming, with an overall inference latency proportional to the number of hierarchical layers. For example, a four-layer configuration produces valid classification outputs after four clock cycles once the pipeline is filled.

Control, Utility Monitoring, and Adaptation: Figure 3b shows the control and adaptation subsystem, which operates alongside the datapath. Clause utility computation logic continuously monitors clause effectiveness using counters and moving averages. These utility signals are forwarded to a centralized training and adaptation controller that orchestrates learning across all layers.

The controller is responsible for: (1) distributing utility-weighted feedback signals to TA, (2) managing training phases and adaptation intervals, and (3) triggering clause growth and pruning events based on observed learning dynamics.

Adaptive architecture support: ALLA supports dynamic clause growth and pruning without requiring physical reconfiguration of hardware resources. Each layer is provisioned with a maximum clause capacity, while unused clauses remain disabled. Disabled clauses are gated to prevent unnecessary switching activity, automaton updates are halted to reduce dynamic power consumption, and utility computation is bypassed. When growth conditions are met, the controller activates reserved clause slots and initializes their automata states using targeted bias derived from misclassification statistics.

TLUs: When TLUs are enabled, they are implemented using standard sequential logic elements. Each temporal clause is associated with a finite-state automaton realized using flip-flops for state storage and combinational logic for state transitions. Temporal context is maintained through shift registers that store prior clause activations, enabling efficient evaluation of temporal patterns with minimal additional hardware complexity.

Memory organization: The memory footprint of ALLA remains modest compared to neural network accelerators. Primary storage components include: (1) automaton state registers, requiring $N_{\text{total clauses}} \times |L_{\text{max}}| \times \log_2(2N)$ bits across all layers; (2) clause output registers storing $N_{\text{total clauses}}$ binary values; (3) clause utility storage implemented using fixed-point registers (typically 16 bits per clause); and (4) a small amount of control state memory for counters and configuration parameters.

Overall, the clear separation between Datapath and control logic enables ALLA to retain the massive parallelism and energy efficiency of TM hardware while supporting hierarchical learning, adaptive capacity management, and temporal processing.

5.1 Synthesis Results and Analysis

We synthesized ALLA hardware for three representative configurations using 28nm CMOS technology with standard cell libraries at 1.0 V nominal supply voltage. Synthesis used Synopsys Design Compiler with typical corner (TT, 25 °C) and moderate optimization effort. Power analysis used Synopsys PrimeTime PX with activity factors derived from running benchmarks.

Configuration A (Small): 3 hierarchical layers with 50, 40, 30 clauses per layer, 64 input features, 50 automaton states per literal. Target applications: simple edge classification tasks, sensor processing.

Configuration B (Medium): 4 hierarchical layers with 100, 80, 60, 40 clauses per layer, 784 input features (MNIST-scale), 100 automaton states per literal. Target: moderate complexity pattern recognition.

Configuration C (Large): 5 hierarchical layers with 200, 160, 120, 80, 40 clauses per layer, 3072 input features (CIFAR-10 scale), 100 automaton states per literal. Target: complex image classification, rich sensor data.

Table 1 summarizes synthesis results compared to TM implementations with equivalent representational capacity (same total number of clauses) and BNN accelerators from recent literature.

Table 1. Hardware synthesis results in 28 nm Complementary Metal-Oxide-Semiconductor (CMOS)

Metric	Adaptive Logic Learning Architecture (ALLA)-B	Tsetlin Machine (TM)-Equiv	Binarized Neural Network (BNN)	Convolutional Neural Network (CNN)
Area (mm^2)	0.847	0.523	1.24	4.83
Max Freq (MHz)	285	310	380	250
Static Power (mW)	0.18	0.12	0.34	2.15
Dynamic Power (mW)	2.25	1.73	4.67	18.4
Energy/Inf (nJ)	0.847	2.41	1.38	523
Latency (cycles)	4	1	8	125
Throughput (MIPS)	71.2	310	47.5	2.0

Several important observations emerge from these results:

Energy efficiency: Energy consumption is 0.847 nJ per inference, representing 2.8× improvement over equivalent TM (2.41 nJ) and 1.6× improvement over BNN (1.38 nJ). The dramatic 618× improvement over CNN (523 nJ) demonstrates the fundamental efficiency advantage of logic-based learning. The improvement over TM comes from better accuracy (fewer re-processing needs) and reduced clause requirements through hierarchical composition.

Area overhead: ALLA increases area by 1.62× compared to TM due to additional layers and inter-layer pipelining. However, this overhead is modest compared to the representational capacity gained. Furthermore, ALLA remains smaller than BNN (1.5× smaller) and dramatically smaller than CNN (5.7× smaller). Throughput vs. Latency tradeoff: ALLA has 4× higher latency than TM due to pipeli ne depth but maintains reasonable throughput of 71.2 million inferences per second. For streaming applications where latency is less critical than throughput, this tradeoff is favorable. The pipeline can process one new input every clock cycle once filled.

Frequency and timing: ALLA achieves 285 MHz, slightly lower than TM (310 MHz) due to longer combinational paths in hierarchical feedback logic. However, this frequency is adequate for most edge applications and can be increased through additional pipelining if needed.

5.2 Scaling Analysis

Table 2 shows how metrics scale across the three configurations, demonstrating the architecture's flexibility.

Energy scales approximately linearly with total clause count, reflecting that energy consumption is dominated by clause evaluation and automaton updates. Area scales sub-linearly due to shared control logic and utilities. The architecture demonstrates good scalability across problem sizes.

6 Experimental Validation

We evaluate ALLA extensively across diverse benchmarks to demonstrate performance, efficiency, and broad applicability. All experiments use the same basic training procedure with problem-specific hyperparameter tuning. All reported experimental results are averaged over three independent runs with different random seeds, and the observed variance across runs was negligible. All baseline energy results are reported from peer-reviewed implementations and normalized to comparable technology nodes and representational capacity.

Table 2. Adaptive Logic Learning Architecture (ALLA) scaling across configurations (28 nm Complementary Metal-Oxide-Semiconductor (CMOS))

Metric	Adaptive Logic Learning Architecture (ALLA)-A	Adaptive Logic Learning Architecture (ALLA)-B	Adaptive Logic Learning Architecture (ALLA)-C
Total Clauses	120	280	600
Area (mm^2)	0.234	0.847	3.12
Frequency (MHz)	320	285	265
Energy/Inf (nJ)	0.183	0.847	8.20
Latency (cycles)	3	4	5

6.1 Image Classification Benchmarks

MNIST handwritten digits: We evaluate ALLA-B on MNIST (60,000 training, 10,000 test, 28 × 28 grayscale images of digits 0-9). Training uses 50 epochs with adaptation every 1000 samples. ALLA achieves 98.9% test accuracy, compared to TM's 98.2%, a comparable BNN's 98.1%, and a small CNN's 99.2%.

Energy consumption is 0.847 nJ per inference vs. TM's 2.41 nJ (2.8× improvement), BNN's 1.38 nJ (1.6× improvement), and CNN's 523 nJ (618× improvement). The near-parity with CNN accuracy while maintaining dramatic energy advantages demonstrates ALLA's effectiveness.

Analysis of learned features reveals hierarchical semantics: Layer 1 clauses detect oriented edges and stroke endpoints. Layer 2 clauses detect corners, loops, and stroke combinations. Layer 3 clauses detect digit parts (top curves, vertical stems, bottom curves). Layer 4 clauses combine parts into full digit patterns. This hierarchical decomposition closely mirrors how humans might describe digit recognition.

Computational behavior analysis (MNIST): The MNIST results highlight how hierarchical logic composition improves computational efficiency rather than merely increasing model capacity. Lower layers capture reusable stroke primitives, which are composed by higher layers into digit-level concepts. This reuse reduces the total number of clauses required to achieve high accuracy, resulting in fewer logical evaluations per inference. From a computational standpoint, the hierarchy transforms a flat, high-dimensional clause search into a sequence of smaller, structured computations, leading to faster convergence and lower energy consumption.

CIFAR-10 natural images: ALLA-C is evaluated on CIFAR-10 (50,000 training, 10,000 test, 32 × 32 RGB images across 10 classes), which presents a substantial challenge for logic-based learning due to high intra-class variability and complex spatial structure. ALLA achieves 78.3% test accuracy, compared to TM's 63.7% (14.6 percentage point improvement), BNN's 82.1%, and state-of-the-art CNNs exceeding 95%.

While ALLA does not match CNN accuracy on this benchmark, the observed improvement over flat TM models demonstrates that hierarchical logic learning significantly enhances representational capacity relative to single-layer logic architectures.

Energy per inference is 8.2 nJ, maintaining a 45× advantage over CNN implementations (369 nJ for a comparable CNN accelerator) and a 12× advantage over BNNs (98.4 nJ). These results indicate that ALLA is well suited for scenarios where moderate classification accuracy is acceptable in exchange for substantial energy savings and strong interpretability, rather than direct competition with deep convolutional models on highly complex vision tasks. Qualitative inspection of learned clauses reveals a hierarchical progression from low-level color and edge patterns to mid-level texture cues and coarser object-part representations. While these features are less expressive than those learned by deep CNNs, they provide interpretable intermediate abstractions that support effective classification under strict energy constraints.

Computational Behavior Analysis (CIFAR-10): On CIFAR-10, the performance gap relative to deep CNNs reflects a fundamental difference in computational expressiveness rather than implementation inefficiency. ALLA relies on compositional logic over local and mid-level features, which captures structure efficiently but struggles with highly distributed, texture-dominated representations. Nevertheless, the hierarchical organization significantly outperforms flat logic models by enabling progressive abstraction, confirming that the computational framework scales meaningfully to moderate visual complexity without incurring the arithmetic cost of deep neural networks.

6.2 Sequential Data Processing

Google speech commands: Using TLU-augmented ALLA on the Google Speech Commands dataset (65,000 utterances, 10 common words like "yes", "no", "stop"), we achieve 91.2% test accuracy compared to standard TM treating time-frequency features as spatial (76.8%), a compact LSTM baseline (93.4%), and a full LSTM (96.1%). The 14.4 percentage point improvement over TM demonstrates TLU's effectiveness for temporal patterns.

Energy per inference is 3.1 nJ vs. LSTM's 372 nJ (120× improvement) and compact LSTM's 187 nJ (60× improvement). The energy advantage is dramatic even though accuracy doesn't quite match LSTMs. For always-on keyword spotting on battery-powered devices, this tradeoff is very favorable.

Analysis of learned temporal patterns reveals interpretable phonetic features: early TLUs detect phoneme transitions, middle TLUs detect syllable patterns, and higher TLUs detect complete word patterns. Temporal windows range from 3-5 frames (30-50 ms) at low layers to 20-30 frames (200-300 ms) at high layers, appropriately matching temporal structure at each level.

Gesture recognition: On the UCI Hand Gesture dataset (time-series accelerometer data from 3-axis accelerometer, 5 gesture types), ALLA with TLU achieves 94.6% accuracy versus LSTM baseline at 96.1%, compact LSTM at 94.8%, and TM at 81.2%. ALLA matches compact LSTM accuracy while consuming 2.7 nJ per inference vs. LSTM's 324 nJ (120× improvement) and compact LSTM's 143 nJ (53× improvement).

The learned temporal logic patterns correspond to characteristic motion sequences: rapid upward acceleration followed by deceleration (throwing gesture), oscillating side-to-side acceleration (waving), and so forth. These patterns are directly interpretable unlike LSTM hidden state dynamics.

Scope of temporal evaluation: The two evaluated sequence tasks represent distinct classes of temporal data: short-duration acoustic sequences with time–frequency structure (speech commands) and continuous multivariate sensor time series (gesture recognition). Together, these tasks demonstrate that TLUs are applicable across different sequential modalities and temporal dynamics.

However, the intent is not to claim universal superiority over recurrent or transformer-based models for all sequence learning problems. Rather, TLUs are designed to provide an energy-efficient and interpretable alternative for structured, moderate-length temporal patterns commonly encountered in edge and embedded applications.

Computational behavior analysis (sequential tasks): The sequential benchmarks illustrate how TLUs transform temporal learning into explicit logical state transitions. Rather than maintaining dense hidden states as in recurrent models, TLUs encode temporal structure through discrete automaton states and finite windows. This results in predictable memory usage, stable learning dynamics, and energy-efficient temporal reasoning. Performance degradation on very long temporal dependencies reflects a deliberate computational tradeoff favoring bounded memory and interpretability over unbounded recurrent expressiveness.

6.3 Ablation Studies

To assess the contribution of individual architectural components, ablation studies were conducted on the MNIST dataset. Each variant removes or disables a specific component of the proposed ALLA while keeping all other settings unchanged.

Table 3 reports the classification accuracy and clause count for all ablation configurations. As shown in Table 3, removing utility-weighted hierarchical feedback and reverting to standard TM feedback across all layer's results in a 1.1 percentage-point reduction in accuracy, accompanied by an 11% increase in the number of clauses. This highlights the importance of coordinated learning across hierarchical layers for improving both accuracy and efficiency.

Table 3. Ablation study results on MNIST

Configuration	**Accuracy (%)**	**Clauses**
Adaptive Logic Learning Architecture (ALLA) (full)	98.9	280
w/o hierarchical feedback	97.8	312
w/o adaptation	98.1	400 (fixed)
w/o multi-resolution	98.3	285
Single layer (Tsetlin Machine)	98.2	800

Disabling adaptive clause growth and pruning by fixing the clause count at 400 leads to a 0.8 percentage-point accuracy degradation despite the use of more clauses, indicating that architectural adaptation improves clause quality by allocating capacity where it is most effective. Removing multi-resolution processing yields a 0.6 percentage-point accuracy drop with a similar clause count, suggesting that multi-resolution contributes positively, although its impact is smaller compared to other components for MNIST.

Finally, the benefit of hierarchical depth is evident when comparing ALLA with a single-layer TM. While the single-layer model requires 800 clauses to achieve 98.2% accuracy, ALLA attains a higher accuracy of 98.9% using only 280 clauses. This corresponds to approximately a 2.9× reduction in clause count, demonstrating the efficiency advantages of the hierarchical architecture.

6.4 Hyperparameter Sensitivity Analys

To evaluate the robustness of the proposed ALLA with respect to hyperparameter selection, a sensitivity analysis was conducted on the core architectural parameters. Experiments were performed on the MNIST dataset using the ALLA-B configuration. In each experiment, a single hyperparameter was varied while all remaining parameters were fixed at their default values.

The analyzed hyperparameters include: (i) the utility balance factor α, (ii) the clause growth threshold τ_{grow}, and (iii) the pruning threshold τ_{prune}. For each configuration, training was repeated three times, and the average test accuracy was reported. Table 4 summarizes the impact of these hyperparameters on classification accuracy. As shown in Table 4, ALLA demonstrates stable performance across a broad range of hyperparameter values. The observed accuracy variations remain within ±0.3% relative to the default configuration, indicating that the proposed architecture is not overly sensitive to precise hyperparameter tuning. This robustness is particularly important for edge and resource-constrained deployments, where extensive hyperparameter optimization may be impractical.

6.5 Observed Dynamics of Hierarchical Feedback Weight β

The hierarchical feedback weight β plays a key role in coordinating learning across layers by scaling reinforcement signals according to the utility of higher-layer clauses. Although β is computed dynamically during training, its empirical behavior remains stable and well-bounded.

We monitored the values of β across all layers and training epochs during the MNIST and CIFAR-10 experiments. Across all runs, β values remained within the range [0.18, 0.82], with the majority of updates concentrated between 0.35 and 0.65 . Early in training, β exhibits higher variance as clause utilities are still forming, reflecting exploratory learning behavior. As training progresses and utilities stabilize, β converges toward moderate values, indicating consistent attribution of importance to lower-layer clauses that contribute to high-utility higher-level patterns.

Importantly, β never collapses toward zero or saturates near one, ensuring that lower layers continue to receive meaningful learning signals throughout training. This observed behavior confirms that the hierarchical feedback mechanism operates in a stable regime without requiring manual tuning or explicit constraints.

Table 4. Hyperparameter sensitivity on MNIST (Adaptive Logic Learning Architecture (ALLA)-B)

Parameter	Value	Accuracy (%)
	0.5	98.6
α	0.7 (default)	98.9
	0.9	98.7
	0.60	98.8
τ_{grow}	0.70 (default)	98.9
	0.80	98.7
	0.15	98.6
τ_{prune}	0.20 (default)	98.9
	0.25	98.7

6.6 Comparison with Related Approaches

Table 5 provides comprehensive comparison across multiple metrics against various AI approaches.

Table 5. Hyperparameter sensitivity on MNIST (Adaptive Logic Learning Architecture (ALLA)-B)

Approach	Accuracy (%)	Energy/Inf (nJ)	Area (mm^2)	Latency (cycles)	Params (K)	Train Time (min)
Adaptive Logic Learning Architecture (ALLA)	98.9	0.847	0.847	4	58	12
Tsetlin Machine (TM)	98.2	2.41	0.523	1	157	8
Binarized Neural Network (BNN)	98.1	1.38	1.24	8	92	180
Convolutional Neural Network (CNN) (small)	99.2	523	4.83	125	431	45

6.7 Training Cost and Convergence

Table 5 also reports end-to-end training time for each approach. ALLA converges within approximately 12 minutes on MNIST using standard CPU-based training, compared to 8 minutes for a flat TM, 45 minutes for a small CNN, and over 180 minutes for a representative BNN implementation. Although ALLA introduces additional overhead relative to TM due to hierarchical feedback and adaptive clause management, training remains lightweight and significantly faster than gradient-based neural models. In practice, convergence is typically observed within 30–40 epochs, after which accuracy improvements saturate and clause growth stabilizes.

ALLA achieves the best balance across metrics: near-best accuracy with best energy efficiency, moderate area, low latency, and fast training. The 2.8× energy improvement over TM while improving accuracy demonstrates the value of hierarchical architecture.

7 Interpretability and Explainability Analysis

A central advantage of the ALLA is that it preserves strong interpretability despite increased architectural depth. Unlike neural networks, whose decisions must be approximated post hoc, ALLA's logic clauses directly encode the decision process. This section analyzes the interpretability properties of ALLA and illustrates them through representative examples.

7.1 Hierarchical Feature Representation

ALLA learns features at increasing abstraction levels across the hierarchy. At lower layers, clauses capture primitive patterns, while higher layers compose these primitives into semantically meaningful structures.

On image classification tasks such as MNIST, Layer 1 clauses detect low-level visual primitives including oriented edges, stroke endpoints, and simple curves. Layer 2 clauses combine these primitives into mid-level structures such as corners, loops, and junctions. Layer 3 clauses represent digit components (e.g., vertical stems or curved segments), while the top layer aggregates these components into complete digit patterns.

This hierarchical decomposition mirrors human reasoning and provides semantic interpretability at every abstraction level, allowing learned features to be inspected and understood individually.

7.2 Decision Path Tracing

For any prediction, ALLA enables complete tracing of the decision path through the hierarchy. Given a predicted class $\hat{y}$, one can identify the top-layer clauses that contributed positive votes, then recursively trace which clauses in lower layers caused these clauses to activate, down to the level of input literals. For example, in MNIST classification, a prediction of digit 3 can be traced to the activation of clauses corresponding to top and bottom curved strokes and a middle horizontal gap. These clauses, in turn, are activated by lower-layer clauses detecting specific curved and horizontal edge patterns in the input image. This produces a transparent explanation of the form: the input is classified as digit 3 because it contains characteristic upper and lower curves separated by a horizontal gap.

Such traceability enables instance-level explanations without approximations or surrogate models.

7.3 Counterfactual Explanations

ALLA naturally supports counterfactual explanations by identifying minimal changes required to alter a prediction. By analyzing which clauses were decisive for a given classification, one can determine which literals or clause activations must be suppressed or activated to flip the outcome.

For instance, a digit 3 misclassified as digit 8 can be explained by the absence of vertical connections that close the loops. Introducing minimal structural changes that activate clauses corresponding to closed-loop patterns would change the prediction accordingly. These counterfactual explanations are concise, semantically meaningful, and grounded in the actual decision logic.

7.4 Global Model Understanding

Beyond individual predictions, ALLA enables global understanding of the learned model. The complete set of clauses across all layers constitutes an explicit logical description of the decision function. For MNIST, high-level rules extracted from the top layer include patterns such as: digit 0 requires both top and bottom loops; digit 1 requires a dominant vertical stem without loops; digit 8 requires two closed loops with vertical connections. These rules align closely with human intuition and can be validated by domain experts.

Unlike neural networks, whose parameters lack direct semantic interpretation, ALLA provides an interpretable model structure that supports inspection, auditing, and reasoning at the system level.

7.5 Comparison with Neural Network Interpretability

Neural network interpretability typically relies on post-hoc techniques such as saliency maps or attention visualization, which approximate model behavior rather than revealing true decision logic. These methods often lack compositional structure and may vary across explanation techniques.

In contrast, ALLA's interpretability is intrinsic: its logic clauses are the decision mechanism itself. The hierarchical organization further reveals how low-level features compose into high-level concepts, providing a level of semantic clarity that post-hoc neural network explanations cannot easily achieve.

Overall, ALLA offers transparent, faithful, and hierarchical interpretability while maintaining competitive performance and energy efficiency, making it well suited for applications requiring explainable and trustworthy AI.

8 Discussion

Rather than reiterating quantitative performance results, this section distills the computational insights revealed by the proposed framework and experimental analysis. The discussion emphasizes the underlying learning dynamics, computational tradeoffs, and practical implications of hierarchical logic-based computation, providing context for both the observed strengths and the remaining limitations of ALLA from a computational methods perspective.

8.1 Advantages and Limitations

The proposed ALLA demonstrates several notable strengths. Most prominently, it achieves substantial energy efficiency gains, delivering a 2.8× reduction in energy consumption compared to conventional TMs and up to a 618× reduction compared to neural network–based approaches, while maintaining competitive classification accuracy. This efficiency makes ALLA particularly suitable for deployment in severely energy-constrained environments, such as edge and embedded systems. Beyond efficiency, the hierarchical organization of logic clauses enables compositional feature learning, which significantly reduces the number of required clauses—typically by a factor of two to three—relative to flat logic-based architectures. This hierarchical abstraction not only improves scalability but also enhances learning efficiency and accuracy.

Interpretability is another key advantage of ALLA. The logic-based formulation allows complete tracing of decision paths from input literals through intermediate layers to final predictions. Each hierarchical layer corresponds to a distinct level of abstraction, enabling both global model understanding and instance-level explanations without relying on post-hoc interpretability techniques. In addition, the adaptive clause growth and pruning mechanism allows the model to dynamically adjust its representational capacity during training, reducing dependence on manual hyperparameter selection and enabling more efficient allocation of computational resources. The integration of TLUs further extends the applicability of ALLA to sequential and time-dependent data, allowing interpretable temporal pattern learning while preserving the energy efficiency benefits of logic-based computation.

Despite these advantages, several limitations remain. On highly complex visual recognition tasks such as CIFAR-10, ALLA achieves 78.3% accuracy, which, although substantially higher than that of flat TM models, remains below the performance of state-of-the-art CNNs. This indicates that while hierarchical logic learning narrows the performance gap, further improvements are needed to match deep neural models on highly intricate visual datasets. Training complexity also increases due to the hierarchical structure and adaptive mechanisms, as effective learning depends on appropriate selection of growth and pruning thresholds, feedback weighting parameters, and layer-wise learning rates. Although adaptation reduces the need for fixed architectural choices, initial configuration still requires careful tuning.

Additional challenges arise when dealing with extremely high-dimensional inputs, such as very large images or long temporal sequences. While hierarchical abstraction mitigates this issue to some extent, preprocessing techniques such as dimensionality reduction, pooling, or structured connectivity constraints may still be necessary. Furthermore, ALLA relies on binarization of continuous-valued inputs, which can lead to information loss. Encoding methods such as thermometer encoding preserve ordering information but increase input dimensionality, presenting a tradeoff between representational fidelity and computational efficiency. From a hardware perspective, supporting adaptive growth requires reserving resources for potential future clauses. If clause growth is limited during training, some of these resources may remain unused, resulting in moderate area overhead. These limitations should be interpreted as deliberate computational tradeoffs, where bounded memory, discrete logic, and energy efficiency are prioritized over unbounded representational capacity and arithmetic-intensive processing.

8.2 Observed Failure Cases

While ALLA demonstrates robust performance across a range of benchmarks, certain failure modes were observed that highlight current limitations of logic-based hierarchical learning.

A representative failure case occurs on visually ambiguous CIFAR-10 samples where class distinctions depend on fine-grained texture or global context rather than compositional object parts. For example, images containing small animals (e.g., cats vs. dogs) under cluttered backgrounds are sometimes misclassified when discriminative cues are distributed across large spatial regions. In such cases, the learned logic clauses tend to focus on local edge and texture patterns that are individually interpretable but insufficient to capture the full global context required for correct classification.

Another observed failure mode arises from aggressive binarization. When grayscale or color intensity variations encode subtle but important information, binarization can eliminate discriminative cues. This is particularly evident in low-contrast images or samples affected by motion blur or illumination changes, where ALLA may activate incomplete or conflicting clause sets, leading to unstable votes at the output layer.

For sequential data, failure cases typically involve long-range temporal dependencies exceeding the configured temporal window sizes. While TLUs effectively capture moderate-length temporal patterns, sequences requiring memory over extended durations may be partially truncated, resulting in degraded performance compared to recurrent neural models.

These failure cases suggest that ALLA is best suited for problems where discriminative structure can be expressed through compositional logic over local spatial or temporal patterns. Addressing global-context reasoning and extremely long-range dependencies remains an important direction for future work.

8.3 Realism of Theoretical Assumptions

The convergence and sample complexity analysis in Section IV relies on several idealized assumptions that merit discussion in the context of real-world applications. These assumptions are standard in learning automata theory and are used to establish theoretical guarantees, but they may not strictly hold in all practical scenarios.

Assumption 1 (Representability) presumes that the target function can be approximated by a bounded-depth hierarchical logic structure. In practice, real-world data may violate this assumption, particularly for highly unstructured or noisy tasks. However, the empirical results on CIFAR-10 and sequential benchmarks indicate that even when perfect representability is not achievable, the hierarchical architecture substantially improves expressiveness relative to flat logic models and converges to useful approximate solutions.

Assumption 2 (Feedback Quality) assumes a persistent bias toward correct reinforcement. While real data introduces noise and occasional misleading feedback, the observed stability of clause utilities and bounded behavior of the hierarchical feedback weight β demonstrate that the learning dynamics remain well-behaved in practice. Importantly, reinforcement signals are aggregated over many samples, which mitigates the effect of noisy individual updates.

Assumption 3 (Adaptation Stability) assumes sufficient time between growth and pruning events. Although architectural adaptation introduces non-stationarity, empirical training traces show that clause growth typically occurs only after utility stabilization, and pruning removes persistently low-utility clauses without disrupting learned representations. This behavior is consistent with the theoretical requirement for stability.

Overall, the theoretical analysis should be interpreted as providing insight into the learning dynamics and asymptotic behavior of ALLA rather than strict guarantees under all conditions. The close alignment between theoretical expectations and empirical observations across multiple datasets suggests that the assumptions are reasonable approximations for practical deployments.

8.4 Broader Implications

The combination of energy efficiency, competitive accuracy, and intrinsic interpretability positions ALLA as a strong candidate for edge AI applications. Scenarios such as wearable health monitoring, industrial IoT diagnostics, autonomous systems, and smart home devices can benefit from local intelligence that minimizes energy consumption while providing transparent and auditable decision-making. By enabling on-device inference with minimal power requirements, ALLA also supports privacy-preserving computation by reducing reliance on cloud-based processing.

Beyond deployment considerations, ALLA aligns well with emerging regulatory and ethical requirements for explainable artificial intelligence. Regulatory frameworks such as the EU AI Act and standards for medical and safety-critical systems increasingly demand transparency and accountability in automated decision-making. ALLA's logic-based structure provides inherent explainability, enabling clear audit trails and faithful explanations that reflect the true decision process, rather than approximations produced by post-hoc explanation methods.

Finally, ALLA highlights the benefits of hardware–software co-design in the development of efficient AI systems. By eliminating gradient-based optimization entirely rather than approximating it through quantization or pruning, the architecture achieves fundamental efficiency gains. Although this work focuses on digital CMOS implementation, the learning automata framework underlying ALLA maps naturally to neuromorphic and analog computing paradigms. Future research may explore such implementations to further reduce energy consumption and exploit device-level stochasticity for learning.

9 Conclusion

This paper introduced the ALLA, a hierarchical and energy-efficient logic-based learning framework that extends conventional TMs. By integrating HLLs, adaptive clause growth and pruning, and TLUs, ALLA enables scalable compositional feature learning while preserving the inherent interpretability of logic-based models.

Extensive experimental evaluations across both image classification and sequential learning tasks demonstrate that ALLA consistently outperforms flat TM architectures in terms of predictive accuracy while offering significant improvements in energy efficiency. Hardware synthesis results further indicate that the proposed architecture delivers substantial energy savings compared to both traditional TM implementations and representative neural network baselines, without compromising accuracy. These characteristics make ALLA particularly well suited for deployment in energy-constrained and safety-critical edge AI systems, where transparency, efficiency, and predictable behavior are essential.

Future work will focus on automated scaling of hierarchical depth, extensions toward online and continual learning, and deployment on reconfigurable and neuromorphic hardware platforms. Additionally, exploring alternative encoding schemes for continuous-valued inputs and tighter hardware–software co-design strategies may further enhance the performance and efficiency of ALLA in more complex application domains.

Author Contributions

Conceptualization, I.H.; methodology, I.H.; validation, I.H.; formal analysis, I.H.; investigation, I.H.; resources, I.H.; data curation, I.H.; writing—original draft preparation, I.H.; writing—review and editing, I.H.; visualization, I.H.; supervision, I.H.; project administration, I.H. The author has read and agreed to the published version of the manuscript.

Data Availability

The data used to support the findings of this study are available from the corresponding author upon request.

Acknowledgments

This work was supported financially by the Department of Computer Engineering, College of Computer Science and Engineering, Taibah University, Madinah, Saudi Arabia.

Conflicts of Interest

The author declares that they have no conflicts of interest.

References

[1] K. Schwab, *The Fourth Industrial Revolution*. Crown Currency, 2017.

[2] G. De Micheli, "Networks on chips," in *Design, Automation, and Test in Europe*. Springer Nature Link, 2008, pp. 105–110. https://doi.org/10.1007/978-1-4020-6488-3_8

[3] O. C. Granmo, "The Tsetlin Machine: A game theoretic bandit-driven approach to optimal pattern recognition with propositional logic," *arXiv:1804.01508*, 2020. https://doi.org/10.48550/arXiv.1804.01508

[4] A. Wheeldon, R. Shafik, T. Rahman, J. Lei, A. Yakovlev, and O. C. Granmo, "Learning automata based energy-efficient AI hardware design for IoT applications," *Philos. Trans. R. Soc. A*, vol. 378, no. 2182, p. 20190593, 2020. https://doi.org/10.1098/rsta.2019.0593

[5] S. Muggleton and L. De Raedt, "Inductive logic programming: Theory and methods," *J. Log. Program.*, vol. 19, pp. 629–679, 1994. https://doi.org/10.1016/0743-1066(94)90035-3

[6] R. Evans and E. Grefenstette, "Learning explanatory rules from noisy data," *J. Artif. Intell. Res.*, vol. 61, pp. 1–64, 2018. https://doi.org/10.1613/jair.5714

[7] L. Jiao, X. Zhang, O. C. Granmo, and K. D. Abeyrathna, "On the convergence of Tsetlin Machines for the XOR operator," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 45, no. 5, pp. 6072–6085, 2022. https://doi.org/10.1109/TPAMI.2022.3203150

[8] V. Sze, Y. H. Chen, T. J. Yang, and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *Proc. IEEE*, vol. 105, no. 12, pp. 2295–2329, 2017. https://doi.org/10.1109/JPROC.2017.2761740

[9] E. O. Neftci, H. Mostafa, and F. Zenke, "Surrogate gradient learning in spiking neural networks," *IEEE Signal Process. Mag.*, vol. 36, no. 6, pp. 51–63, 2019. https://doi.org/10.1109/MSP.2019.2931595

[10] R. Andri, L. Cavigelli, D. Rossi, and L. Benini, "YodaNN: An ultra-low power convolutional neural network accelerator based on binary weights," in *Proceedings of 2016 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, Pittsburgh, PA, USA, 2016, pp. 236–241. https://doi.org/10.1109/ISVLSI.2016.111

[11] T. Hirtzlin, M. Bocquet, J. O. Klein, E. Nowak, E. Vianello, J. M. Portal, and D. Querlioz, "Outstanding bit error tolerance of resistive RAM-based binarized neural networks," in *Proceedings of 2019 IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, Hsinchu, Taiwan, 2019, pp. 288–292. https://doi.org/10.1109/AICAS.2019.8771544

[12] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998. https://doi.org/10.1109/5.726791

[13] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997. https://doi.org/10.1162/neco.1997.9.8.1735

[14] Z. Aklah, A. Al-Safi, and H. T. Hassan, “Exploring FPGA implementation and emulation of memristor devices,” *Int. J. Comput. Methods Exp. Meas.*, vol. 12, no. 2, pp. 135–146, 2024. https://doi.org/10.18280/ijcmem.120203