Hex Winner Prediction: Using the Graph Tsetlin Machine to predict winner in a Hex Game

Thomas Joon Hetland
Student
University of Agder
Grimstad, Norway
thomasjh@uia.no

Synne Rønnekleiv
Student
University of Agder
Grimstad, Norway
synnero@uia.no

***Abstract*—With the rise of machine learning algorithms, a popular application area has emerged: machine learning in game theory. A well-studied example is the Hex game. The Hex game is a simple, deterministic game where two players compete to form a connected path across the board, in which exactly one player must win. In 2025, Granmo et al. introduced the Graph Tsetlin Machine (GraphTM), a logic-based model designed to learn from graph-structured data. This project applies the GraphTM to the Hex game to predict the winner of completed games. We model the Hex board as a graph, representing each cell as a node and introducing four virtual nodes, one for each board edge. We evaluate the proposed solution across a range of board sizes and include an interpretability analysis to better understand the model's learned patterns.**

***Index Terms*—Tsetlin Machine (TM), Graph Tsetlin Machine (GraphTM), Graphs, Hex-game, Game Theory**

I. Introduction

Hex is a connection-based strategy game deeply explored in game theory, where players aim to create an unbroken path across the board. Game theory provide the theoretical bases for understanding optimal play, winning strategies and advantages in board states, which is needed information for structural reasoning. From a computational perspective are Hex games challenging. Even small board sizes yield a combinatorial explosion of possible configurations, and the optimal move often depends on long-range structural relationships rather than local patterns alone.

The Tsetlin Machine (TM) is a rule-based learning algorithm that perform classification through interpretable propositional logic clauses. The Graph Tsetlin Machine (GraphTM) extend this algorithm by introducing graph structures, enabling reasoning over relational data by treating inputs as graphs. Nodes and edges within the graphs encodes the properties and connectivity-patterns that a feature vector alone would not be able to capture.

This project explore how the GraphTM can be used to predict the winner of a Hex game. Each Hex board game is represented as a graph, allowing the model to learn logical clause rules that reflect structural winning patterns.

II. Background and related work

A. Tsetlin Machine

The TM is an interpretable, rule-based machine learning algorithm, introduced by Granmo [1]. Given a set of input, the TM constructs human-readable logic statements to capture patterns through propositional logic and rule-based learning.

The TM uses conjunctive clauses, constructed of a collection of literals represented by TAs. Each TA consist of $2N$ states, with states 1 to N representing action α_1, and states $N+1$ to $2N$ representing action α_2 [2]. The TA interacts with an environment that follows a unknown probability distribution, to receive either reward or penalty to update the states. Rewards reinforces the current action, increasing the likelihood of repeating, while penalty discourages, making the TA explore alternative states.

Given an input vector of binary feature representation $x = [x_1, x_2, \ldots, x_o] \in \{0,1\}^o$, the TM derives a set of $2o$ literals represented as $L = (l_1, \ldots l_o, \neg l_1, \ldots, \neg l_o)$. Each literal in a clause corresponds to a TA that determines whether a feature is included or excluded. The inclusion of negated features enhance the capacity to form discriminative patterns and identify informative correlations [1].

The pattern recognition is achieved by constructing m conjunctive clauses formed as logical AND over literals. The total number of clauses governs the size and complexity of the TM [1]. To ensure balanced voting, the clauses are divided evenly into positive polarity clauses c_j^+ and negative polarity clauses c_j^-, where j ranges from 1 to $\frac{m}{2}$ [3].

$$C_j^{\xi}(x) = \bigwedge_{l_k \in L_j^{\xi}} l_k, \quad where\ \xi \in \{+,-\} \tag{1}$$

Equation 1 indicates that for a clause to evaluate to 1, each included literal $L_j^{\xi} \subseteq L$ must be matched [4].

The prediction decision is determined through majority voting. Each clause that matches the input pattern contributes a vote based on its assigned polarity. The total vote count v is calculated as the sum of contributions from all active clauses.

$$\hat{y} = u\left(\sum_{i=1}^{m/2} C_i^+(x) - \sum_{j=1}^{m/2} C_j^-(x)\right) \tag{2}$$

The final classification is performed by applying a threshold unit step to produce the predicted class label $\hat{y}$.

The voting mechanism create a balanced decision-making process, where positive clauses support for prediction $y = 1$ and negative clauses support for prediction $y = 0$. The TM uses two feedback to guide the TA toward optimal behavior; Type I and Type II feedback. Type I reinforces clauses that correctly identify the target class to suppress false negatives. Type II penalizes clauses that mistakenly identify the incorrect class to suppress false positives. Through this process the TA learn to capture meaningful and discriminative patterns in the input data.

B. Graph Tsetlin Machine

In 2025, Granmo et al. introduced the Graph Tsetlin Machine (GraphTM) [5], extending the standard TM framework. Although the TM provides interpretable and efficient learning with competitive performance, its use of purely Boolean input representations limits its ability to handle many real-world applications. The GraphTM addresses this limitation by supporting multimodal input through hypervectors, building upon the Hypervector Tsetlin Machine [6]. While the Convolutional Tsetlin Machine (CTM) [7] captures context by applying convolutional windows across spatially organized data, the GraphTM captures context through graph topology, allowing clauses to propagate information across nodes, edges, and neighborhoods. This provides a structured and scalable way to model relationships within graph-based data.

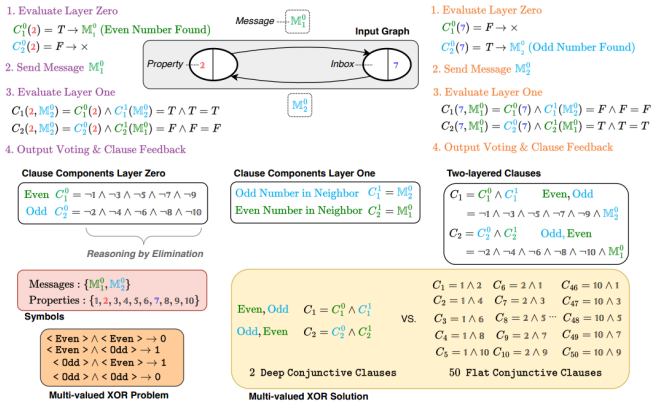

Fig. 1: The GraphTM processes graph-structured input and uses topological information to construct deep conjunctive clauses. By applying De Morgan's laws and reasoning by elimination, it achieves an exponential reduction in the number of required clauses compared to flat conjunctive formulations. Reference: [5]

The GraphTM evaluates each node's properties using the layer-zero component C_j^0 (1.), where if C_j^0 matches the properties of another node, the clause send a message to neighboring nodes $\mathbb{M}_j^0$ (2.). When the neighbors receives the message in their inboxes, they process the corresponding layer-one clause component C_j^1, which collectively determines the output of each full clause C_j in the two-layer architecture (3.). The model performs classification and clause updates using the standard TM learning (4.).

The GraphTM operates on graph input by evaluating clause components across multiple layers of computation. The input is a directed and typed multigraph

$$G = (V, E, P, T) \tag{3}$$

where each node $v_q \in V$ carries a set of properties $P_q \subseteq P$, and each edge $(v_q, v_r, t) \in E$ has a type $t \in T$. Node properties, edge types, and messages are represented as a sparse hypervector, giving the model a symbolic high-dimensional feature space.

A clause is a conjunction of layer-wise components

$$C_j = C_j^0 \wedge C_j^1 \wedge \cdots \wedge C_j^D \tag{4}$$

where layer-zero components C_j^0 evaluates node properties and high-layer components $C_j^0(v_q) \in \{0, 1\}$

If $C_j^0(v_q) = 1$, the clause emits a message symbol M_j^0 which is transmitted along all outgoing edges of v_q. Each node maintains an inbox I_q^i at every layer $i > 0$, and the clause components in these layers evaluate conjunctive conditions over the messages stored in the inbox:

$$C_j^i(v_q) = f\left(I_q^i\right), \tag{5}$$

where f is a conjunction over message literals such as M_j^{i-1} or $\neg M_j^{i-1}$.

Messages are encoded using vector symbolic computation. When a message M_j^i is sent along an edge of type t, the GraphTM binds the message hypervector to the edge-type hypervector using

$$H = M_j^i \otimes t, \tag{6}$$

ensuring that both the content of the message and the relational context of its transmission are preserved. Incoming messages at a node are aggregated through the bundling operation,

$$I_q^i = \bigoplus_{(v_r, v_q, t) \in E} \left(M_j^{i-1} \otimes t\right), \tag{7}$$

forming a high-dimensional representation of the local multi-hop neighborhood.

Clause evaluation proceeds recursively across layers. For node v_q, the cumulative truth value of clause C_j up to depth i is defined as

$$M_{j,q}^i = \bigwedge_{d=0}^{i} C_j^d(v_q). \tag{8}$$

A clause therefore activates at a node only if all of its components across all layers are satisfied. The truth value of clause C_j on the entire graph is then given by

$$C_j(G) = \bigvee_{v_q \in V} M_{j,q}^D, \tag{9}$$

which is true if the clause matches at any node in the deepest layer.

Learning follows the standard TM mechanism, where every literal in every clause component is controlled by a TA that learns to include or exclude the literal based on feedback. Through message passing across graph edges and layer-wise clause construction, the GraphTM produces nested logical patterns capable of capturing complex relational structures with significantly fewer clauses than a flat TM. This depth-enabled message propagation is illustrated in Figure 1, where clause components successively incorporate node features, neighbor messages, and multi-hop context.

C. Machine Learning and Game Theory

Machine learning and game theory provide powerful frameworks for analyzing complex decision-making processes, although they approach these challenges from different perspectives. Machine learning focuses on discovering patterns in data and building models that can make accurate predictions or adaptive decisions, while game theory offers a mathematical foundation for identifying suitable strategies for interacting players, enabling applications across several domains [8].

Real-world settings such as cybersecurity can be viewed as decision-making problems under uncertain, multi-agent conditions. In this context, game theory provides a structured way to model strategic interactions between defenders, attackers, and automated systems [9].

Machine learning complements this by incorporating game-theoretic principles, which has been shown to improve performance in real-time and adversarial environments [8].

D. Hex game

Hex is an abstract strategy game played on an $n \times n$ array of hexagon cells. As shown in Figure 2, the game is a two player game, where each player is assigned a distinct pair of opposing borders. The objective is to create an unbroken path of one's own color connecting their assigned side of the board. In this case, white tries to connect from left to right, while black tries to connect from top to bottom. In addition, according to The Hex Theorem, a game can never end in a draw, one player must always win. This is because if you completely block of your opponent, you also have a fully connecting bridge between your borders [10], [11].

Fig. 2: A empty and fully completed hex game, where the white player has blocked their opponent (black) while also achieving a unbroken connection between their border [11].

Beyond its role as a strategy game, Hex has long been of interest in mathematics, topology, and theoretical artificial intelligence [11]. The Hex Theorem states that a game of Hex can never end in a draw: one player must inevitably complete a connecting path. Although this fact may appear intuitive, to formally prove it requires deep ideas in topology, as the Hex Theorem is logically equivalent to the Brouwer Fixed Point Theorem. David Gale later provided a more accessible graph-theoretic argument based on tracing interfaces between the two colors [12].

Hex is also notable for its "strategy-stealing" proof, which shows that the first player always has a winning strategy on any $n \times n$ board [11]. If the second player had a guaranteed winning response, then the first player could simply make an arbitrary opening move and thereafter imitate the second player's supposed strategy. Since extra stones of one's own color cannot be detrimental, this leads to a contradiction.

III. Method of Approach

In this Chapter, we outline the methodology and research approach used to accomplish the objectives of this project. This project follows the Constructive Research Method [13]:

1) Find a practically relevant problem that also has research potential
2) Obtain a general and comprehensive understanding of the topic
3) Innovate
4) Demonstrate that the solution works
5) Show the theoretical connections and the research contribution of the solution concept
6) Examine the scope of applicability of the solution.

The following section will detail each component to our research method approach.

A. Find a practically relevant problem that also has research potential

Machine learning has increasingly become a central tool in game theory, where it is used to predict outcomes and learn optimal policies. Games such as Go, Chess, and Hex provide structured environments in which learning algorithms can be evaluated under deterministic rules and defined win conditions.

With the introduction to the GraphTM by Granmo et al. in 2024 [5], a new opportunity has emerged to investigate how rule-based models can reason over graph-structured inputs.

In this project, we aim to find a optimal solution to predict the winner of completed hex games. This task not only explores the capabilities of the GraphTM in a non-trivial game-theoretic environment but also provides research potential by examining how symbolic and structural representations influence interpretable learning in competitive games.

B. Obtain a general and comprehensive understanding of the topic

To develop a comprehensive understanding of the topic, we completed the course IKT457: Learning Systems, which covers the TM and its main extensions. In addition, we conducted a literature study using Google Scholar to gather relevant information on game theory, the Hex game, graph theory, and the GraphTM.

C. Innovate

To innovate, we turned to graph theory. In many graph-structured problems, identifying whether two distant regions are connected can be reduced to maintaining disjoint-set (union–find) structures, where groups of adjacent nodes form *islands*. A bridge exists whenever two nodes that touch different regions of interest are found to belong to the same connected set.

To capture this idea symbolically, we can introduce *virtual nodes* that represent conceptual regions of the graph. As illustrated in Figure 3, nodes located along the top and bottom sides of the grid are respectively connected to TOP and BOTTOM virtual nodes. Whenever two nodes share an adjacency relationship, they form an *island*.

Under this representation, a bridge can be detected by simply checking whether two nodes that touch different virtual regions belong to the same island. If a node connected to the bottom virtual node and a node connected to the top virtual node appear in the same component, this component spans both regions and therefore constitutes a bridge.

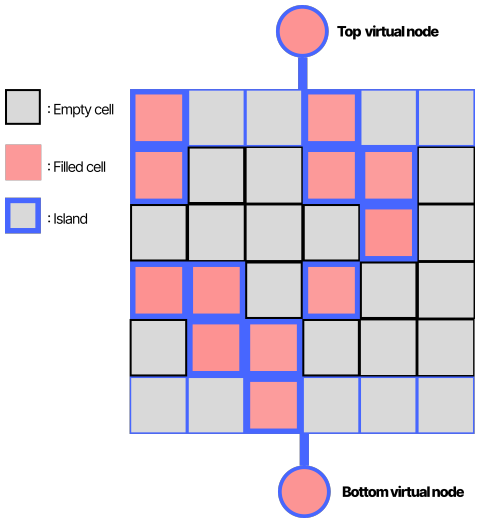

Fig. 3: Illustration of island formation using virtual nodes. Red cells represent occupied nodes, grey cells are empty, and blue outlines indicate connected islands. The top and bottom virtual nodes are connected to their respective rows. When occupied nodes form a continuous island that touches both virtual nodes, a bridge is established between the two regions.

D. Demonstrate that the solution works

After initial implementation, we evaluate the model using performance metrics to iteratively refine and optimize the model. These experiments are done in Chapter V and VI.

E. Show the theoretical connections and the research contributions of the solution concept

Our solution builds upon the theoretical framework of GraphTM and Graph Theory as discussed in Chapter II and Section III-C. This is further detailed in Chapter IV.

F. Examine the scope of applicability of the solution

To assess the applicability of our solution, we experiment and evaluate its performance and interpretability on various sized Hex games, which is explain in Chapter V and VI.

IV. HEX WINNER SOLUTION USING VIRTUAL NODES

In this section, we introduce our proposed solution to the Hex winner prediction task.

A. Hex game representation

Before creating the graph, we transform the Hex game structure into a 2D matrix. We consider an $n \times n$ Hex board and index the cells in a matrix–like fashion. Let

$$\mathcal{I} = \{0, 1, \ldots, n-1\}$$

and denote the cell in row $i \in \mathcal{I}$ and column $j \in \mathcal{I}$ by its coordinate (i, j). The state of a Hex position is encoded as a matrix

$$B \in \{0, 1, 2\}^{n \times n},$$

where

$$B_{i,j} = \begin{cases} 0, & \text{if cell } (i,j) \text{ is empty,} \\ 1, & \text{if cell } (i,j) \text{ is occupied by Player 1,} \\ 2, & \text{if cell } (i,j) \text{ is occupied by Player 2.} \end{cases}$$

For convenience, each cell (i, j) is also associated with a unique integer node index

$$v(i, j) = i \cdot n + j,$$

which allows us to flatten the board into a one–dimensional ordering when needed.

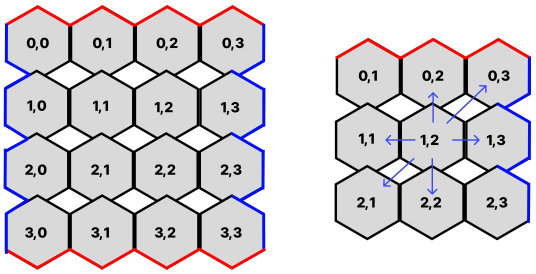

Fig. 4: A 2D matrix representation of a Hex board. Each node is labeled with its coordinate (i, j), and the colored borders indicate the sides controlled by each player: blue for Player 1 and red for Player 2. This coordinate system allows us to compute the six hexagonal neighbors of each node directly while retaining the underlying Hex structure. as shown in illustration (2).

Although the board is stored as a regular matrix, as shown in Figure 4, we preserve the hexagonal neighborhood structure by defining, for each cell (i, j), the set of neighboring cells

$$\mathcal{N}(i,j) = \big\{(i+1,j),\ (i+1,j-1),\ (i,j+1),\ (i-1,j),\ (i,j-1),\ (i-1,j+1)\big\} \cap (\mathcal{I} \times \mathcal{I}). \tag{10}$$

That is, each interior cell has up to six neighbors, and near the borders we simply discard those pairs that fall outside $\mathcal{I} \times \mathcal{I}$. These neighborhood relations are used to construct the graph edges between board cells in our Hex graph representation.

B. Symbols

To express the Hex board as a symbolic graph suitable for the GraphTM, we assign a set of propositional symbols to every node and edge in the representation.

For each board cell (i, j), we define a node $Node_{i,j}$ and associate the following symbols:

- `Empty`: Cell (i, j) is empty.
- `Player1`: Cell (i, j) is occupied by Player 1.
- `Player2`: Cell (i, j) is occupied by Player 2.
- `Placement_`i`_`j: Placement symbol encoding the cell's position.
- `Connected_`i`_`j: Cell (i, j) is connected to two or more nodes belonging to it's own Player.
- `r`i: A row-indexing symbol encoding structural information about the cell's row position.
- `c`j: A column-indexing symbol encoding structural information about the cell's column position.

In addition to node symbols, each edge in the graph is labeled with one of the following edge types:

- `Plain`: A regular adjacency edge.
- `Player 1`: An edge between two nodes of Player 1.
- `Player 2`: An edge between two nodes of Player 2.

C. Graph Architecture

Building on the 2D matrix representation of the Hex board from Figure 4, we construct a graph whose structure reflects both the spatial layout of the board and the winning conditions of the game. Each cell (i, j) is represented as a node, and edges are added according to the six possible hexagonal neighbors described in Section IV-A. This preserves the true geometry of the Hex board despite the underlying matrix layout.

To capture the connection-based win condition, we augment the graph with four virtual nodes: LEFT, RIGHT, TOP, and BOTTOM. Each virtual node represents one of the game's border. A cell located on a given border is connected to the corresponding virtual node, producing a graph structure in which paths through the player's stones correspond directly to graph connectivity. For example, Player 1's goal of connecting the left and right sides becomes equivalent to the existence of a path between the LEFT and RIGHT virtual nodes via Player 1-occupied nodes. Player 2 is handled analogously using the TOP and BOTTOM virtual nodes. The conceptual idea comes from Section III-C, where we can form islands to create a bridge from one border to another.

An overview of this architecture is shown in Figure 5. The resulting graph integrates the board's spatial relationships with the game's win condition, enabling the GraphTM to learn clause-based patterns over a meaningful structural representation of Hex positions.

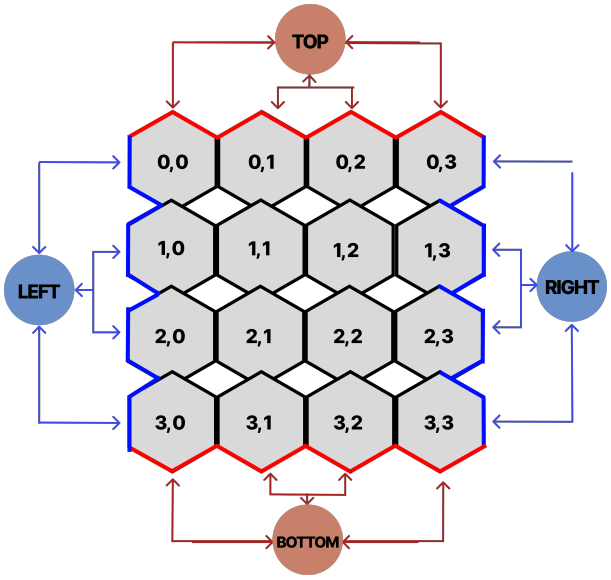

Fig. 5: Overview of the graph architecture. Each cell in the Hex board is represented as a node, and four additional virtual nodes (LEFT, RIGHT, TOP, BOTTOM) are attached along the corresponding borders.

D. Node Properties

After defining the symbol vocabulary, each graph is populated with node properties by iterating over all board cells in every game instance. For each node (i, j), we determine the corresponding cell state in the game dictionary and attach the appropriate symbol from the set described in Section IV-B. Concretely, every node is assigned one of `Empty`, `Player1`, or `Player2` depending on whether the cell is unoccupied, occupied by Player 1, or occupied by Player 2.

In addition to the cell-state symbol, three structural properties are added to each node. Firstly, we add the position of the node using `Placement_i_j`. Furthermore, the column-indexing symbol `c(j)` encodes the horizontal position of the node, while the row-indexing symbol `r(i)` encodes its vertical position. In theory, `Placement` should encode the same information as `r` and `c` does together. However, we observed a fall in performance when `Placement` was not set as a property.

Finally, each node is checked for local connectivity. By counting how many of its neighbors on the Hex graph share the same player label, a node is marked with the symbol `Connected_i_j` whenever two or more adjacent cells belong to the same player. This helps the GraphTM encode local connectivity and islands.

E. Edge Properties

Edges are assigned symbolic properties during graph construction, based on the relationship between the two incident nodes. For each board cell (i, j), we examine its six neighboring coordinates given by the Hex adjacency structure. If the neighbor lies within the board, we retrieve its cell value and assign the edge a label reflecting whether both endpoints

belong to the same player. Edges connecting two stones of Player 1 receive the label `Player 1`, edges connecting two stones of Player 2 receive `Player 2`, and all other edges are labeled `Plain`.

Edges involving virtual border nodes are handled in the same way. Depending on whether the neighboring border cell matches the player's designated winning direction, the border node is treated as belonging to Player 1 (left and right edges) or Player 2 (top and bottom edges). Thus, a board cell on Player 1's side connected to the LEFT or RIGHT virtual node may also receive a `Player 1` edge label when the cell belongs to Player 1. Otherwise, the connection is labeled `Plain`.

V. EXPERIMENTS

A. Datasets

For our experiments, we required 10 different Hex datasets with board sizes ranging from 3×3 to 12×12. Each dataset contains 100,000 completed games, generated using the C implementation provided by Granmo, which produces full games through random legal moves. A 80/20 train–test split were used to ensure reproducibility across all experiments.

Before proceeding, it is important to emphasize that the Hex games in our datasets are generated through random legal moves. As a result, the distribution of stone placements does not necessarily reflect realistic human gameplay or strategic patterns typically seen in a normal Hex game. Consequently, the outcomes and the performance of our model may differ if the datasets were replaced by human-played games or games produced by stronger AI agents.

From the moves, we can transform the data into a 2D matrix, as discussed in IV. We preserve the information of each game by calculating the neighbors of each stone. Table I illustrates a 10×10 Hex game, where 0 represents empty, 1 represents Player 1, and 2 represents Player 2.

1	2	1	2	2	1	2	1	1	0
2	1	1	2	1	2	2	2	1	2
1	2	2	1	1	2	1	1	2	2
2	1	2	2	1	2	2	1	1	2
1	1	2	1	2	1	2	2	1	1
2	2	1	1	2	2	1	2	2	0
1	0	2	1	1	2	0	2	1	2
1	1	2	1	2	2	2	1	2	1
2	1	1	2	1	1	2	1	2	1
1	1	2	0	1	2	2	1	2	0

TABLE I: Example of a 10×10 Hex board used in preprocessing. Values represent Empty (0), Player 1 (1), and Player 2 (2). In this game, Player 2 has won.

B. Experimental Setup

All experiments on completed games were trained on 300 epochs, with some adjustments to their parameters according to the board size. Performance was evaluated using accuracy.

Table 6 summarizes the complete hyperparameter settings across all board sizes.

All experiments were conducted on an NVIDIA DGX-2 system equipped with dual Intel Xeon Platinum 8168 CPUs (2.7 GHz) and 16 NVIDIA Tesla V100 GPUs (32 GB). The system runs Ubuntu 18.04 LTS (64-bit).

VI. RESULTS AND DISCUSSION

This section presents the experiments, evaluations, and empirical results conducted on our solution. We have performed a comprehensive set of assessments, including different board sizes, incomplete games, and clause analysis.

A. Empirical Results

1) Completed Hex games: The initial objective of this project was to predict the winner of completed Hex games using GraphTM.

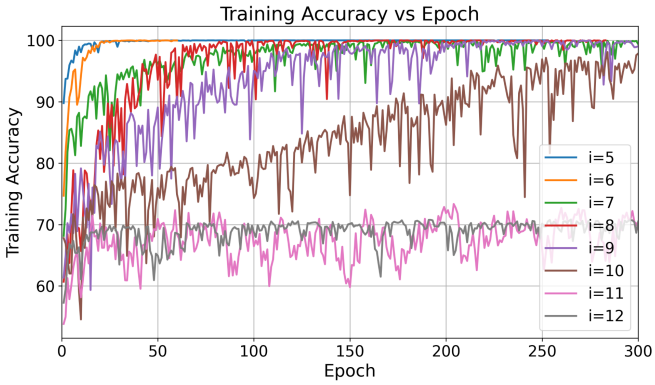

Fig. 7: Training accuracy over 300 epochs for board size ranging from 5-12

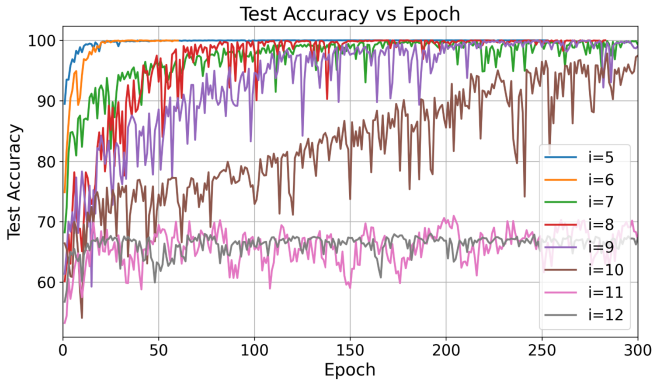

Fig. 8: Test accuracy over 300 epochs for board size ranging from 5-12

As shown in Figures 7 and 8, we achieved 100% accuracy on board sizes ranging from 5×5 to 9×9. For 10×10 boards, the model approached perfect accuracy but did not fully converge within 300 epochs. The brown curve, which represents the 10×10 performance, remains unconverged at epoch 300, indicating that additional training would likely be required to reach 100% accuracy. After 10×10 board size, the accuracy was kept between $60 - 70\%$, suggesting the

Board Size	Epochs	Clauses	T	s	Depth	HV Size	Hv bits	Message Size	Message bits
5x5	300	5,000	3,125	1.0	2	67	1	512	32
6x6	300	5,000	3,125	1.0	2	91	1	512	32
7x7	300	5,000	3,125	1.0	2	118	1	512	32
8x8	300	5,000	3,125	1.0	2	150	1	512	32
9x9	300	5,000	3,125	1.0	2	186	1	512	32
10x10	300	5,000	3,125	1.0	2	226	1	512	32
11x11	300	5,000	3,125	1.0	2	270	1	512	32
12x12	300	5,000	3,125	1.0	2	318	1	512	32

Fig. 6: Model hyperparameter configurations across the different board sizes. HV Size is the same as the amount of symbols.

complexity becomes too high and the current symbols does not generalize enough for the GraphTM to learn the patterns of larger board sizes.

2) Incomplete Hex games: In addition to completed game, we conducted experiments on incomplete Hex games state to evaluate the model's performance under uncertainty. Specifically, we truncated each game by removing a fixed number of final moves. In the first experiment, the last two moves were removed. In the second experiment, the last five moves were removed.

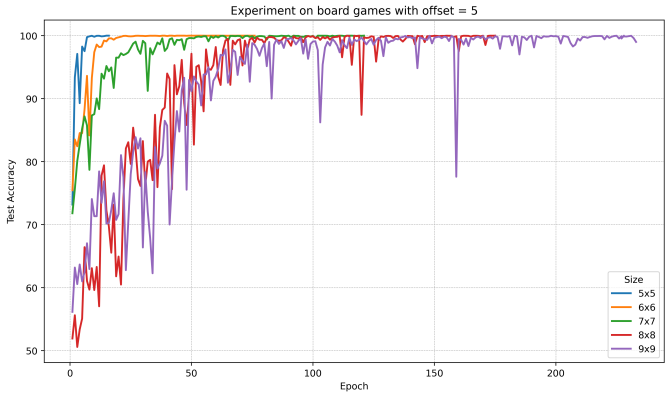

Fig. 9: Test accuracy across epochs for the offset-2 experiment, where the last two moves of each Hex game were removed.

As shown in Figure 9, the smaller board sizes converge rapidly towards 100% accuracy and the larger board sizes learn more slowly. The increase in board size make the prediction task more difficult where the last two moves are removed for each Hex game.

In Figure 10, the experiment demonstrates that stopping the game five moves early retains sufficient information for effective learning on smaller board sizes. Specifically, the 6×6 board achieves the highest and most stable performance, and the 7×7 board performs well by showing strong convergence with only minor fluctuations. In contrast, as the board size increases to 8×8 and 9×9, the models exhibit significant instability and fail to converge reliably. The 5×5 board peaks before the accuracy declines, likely because the model starts to overfit the limited state space and lose generalization while training.

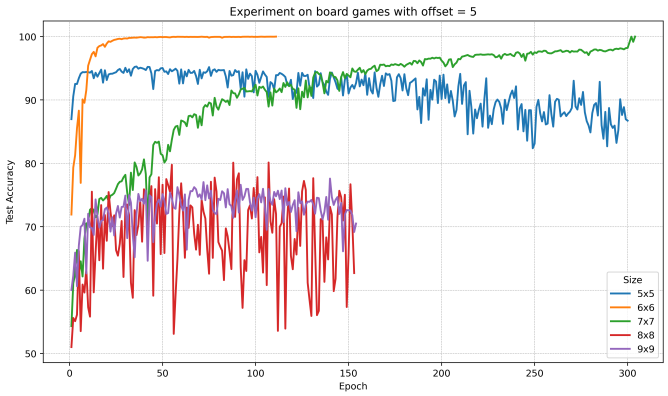

Fig. 10: Test accuracy across epochs for the offset-5 experiment, where the last five moves of the hex game were removed.

B. Message size experiment

According to [5], increasing the number of symbols and clauses leads to a denser message hypervector. Consequently, the message size must scale accordingly to preserve representational capacity. To investigate this, we conducted an experiment using 7000 *clauses* and a message size of 4096 on a 12×12 sized dataset. A shown in Figure 11, the training accuracy improves by more than 10%, while test accuracy increases approximately 2%. However, we also observe clear signs of overfitting. The training accuracy reaches 83.5%, whereas the test accuracy plateaus at only 72.2%. As discussed in Chapter IV-B, each node in the graph representation is assigned several symbolic features (such as `Connected_{i}_{j}`, `r{i}`, and `c{j}`). While these features enrich the representation, they may also cause the GraphTM to become overly specialized on patterns unique to the training data, thereby contributing to the observed overfitting.

As we increase the hypervector size, the hypervector may becomes more sparse. This can lead to overfitting and a reduction in the number of activated bits.

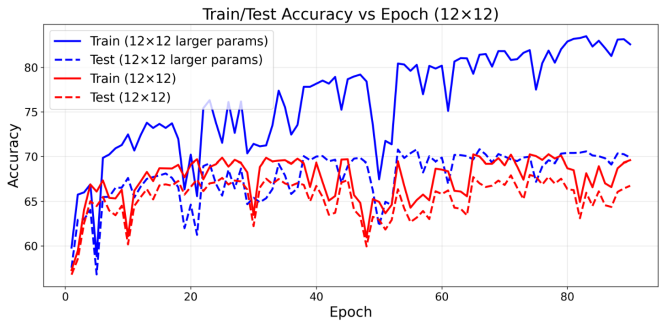

Fig. 11: Accuracy for 12×12 board size with 5000 clauses and 512 message size compared to 7000 clauses and 4096 message size. The blue line represents the accuracy of the larger configuration, while the red line represents the accuracy of the configuration detailed in Figure 6

C. Interpretability analysis

When applying machine learning to game theory, it is important to understand why and how the model makes decisions. In Hex, the winning condition is based on forming a complete connection between two opposite borders. To investigate what the GraphTM focuses on during inference, we performed an interpretability analysis using the active clauses and mapped their positive and negated literals onto heatmaps. As shown in Figure 12, the positive literals partly reflect the structure of the game state through the symbol *Placement_i_j*.

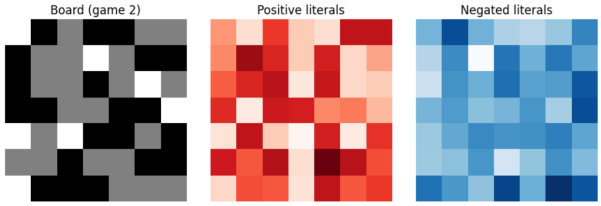

Fig. 12: Interpretability analysis of a 7×7 Hex game from the test dataset. Here, Player 2 has won by forming a left to right connection. White square represent empty cells, black squares represent Player 1, and grey square represent Player 2. The matrix on the left shows the game state. The red heatmap visualized the positive literals in the active clauses and the blue heat map on the right shows the negated literals. The positive literals show partial pattern that resembles the game state. The negated literals provide little to no meaningful interpretation.

Although the outline of active regions is somewhat visible, the current symbol configuration and feature extraction approach do not allow us to determine whether a highlighted cell corresponds to Player 1 or Player 2. This align with the observations discussed in [14], where the authors emphasize that machine learning models are only as interpretable as the features they rely on. This also applies to the Tsetlin Machine.

In addition, the negated literals are especially difficult to interpret in practice. The patterns they produce do not map cleanly to the underlying board state and therefore provide little insight into the reasoning process of the model.

D. Largest board contest

In addition to the primary solution, we conducted a second set of experiments in which the Hex game was transformed into a XOR learning task. This approach was developed specifically for the Largest Board Contest and serves as a unconventional, but conceptually innovative, demonstration of how the Graph Tsetlin Machine can solve the winner-prediction problem using only parity information. Because this method does not reflect meaningful strategic analysis of Hex, it was not used as our main solution.

For any Hex game, the total number of stones places by each player tells us who won, since Player 1 always moves first. Consequently, the parity of the stone counts encodes the same information. If Player 1 played an odd number of stones and Player 2 an even number, Player 1 must have made the final move and therefore must have won. On the other hand, if Player 1 and Player 2 both played a odd number, Player 2 must have won. This observation allows the entire Hex game state to be compressed into a XOR parity problem of the two players stone counts.

We replaced the full Hex game with a two-node graph consisting of one node representing Player 1 and one representing Player 2. Each node was assigned exactly one symbol indicating the parity of its stone count:

- Player 1 received either `p1_even` or `p1_odd`
- Player 2 received either `p2_even` or `p2_odd`

Thus, every game belongs to exactly one of four possible symbol configurations:

(even,even), (even,odd), (odd,even), (odd,odd).

The resulting model is structurally equivalent to training a GraphTM on binary XOR function, where the winner is determined by the parity relationship between the two nodes, and as expected, was able to get 100% accuracy on 30×30.

E. Future Work

Our solution enables the GraphTM to form connected islands and predict the winner through the use of virtual nodes. However, several directions for future work remain. First, our current configuration would benefit from a systematic hyperparameter search. As observed for 10×10 board, the model approaches but does not achieve 100% accuracy. We expect that increasing number of epochs, expanding the number of clauses, or adjusting the message size could further improve performance.

Furthermore, more general or expressive symbols may reduce computational requirements and lower the number of causes and message size needed to generalize effectively. With the current setup, a full hyperparameter search on larger boards would take several weeks, which motivates the search for more compact symbolic encoding.

Finally, we suggest conducting a more comprehensive interpretability analysis. Currently, we inspect the active clauses that vote for the predicted winner for a specific board and map the positive and negated position literals. This approach has

two limitations. First, with our current symbol configuration, we can not distinguish between Player 1 and Player 2 in the heatmaps. Even if a clause votes for Player 1, the interpretation can not reveal whether a *Placement_i_j* literal corresponds to Player 1 or Player 2. Second, the mapping of negated literals is difficult to interpret, which highlights the challenge of relying on negated features. A more interpretable heatmap could potentially be obtained by subtracting the negated literal map from the positive literal map. In theory, this should produce a clearer alignment between the board state and the resulting heatmap and explanation.

VII. Conclusion

The goal of this project was to explore how we can predict the winner of Hex games using the Graph Tsetlin Machine. Our solution represents the Hex board as a graph where each node corresponds to a cell and edges represent adjecency between cells. In addition, we introduced four virtual nodes, one for each border, which connect all cells on that border to the corresponding virtual node. Each node is assigned a set of symbols based n its properties:*Empty*,*Player 1*, *Player 2*, *Placement_i_j*, *Connected_i_j*, *r_i*, and *c_j*, where i denotes the row index, and j denotes the column index.

Our solution demonstrates good performance. It achieves 100% accuracy on 9×9 boards and approaches 100% on 10×10 boards over 300 epochs. We also conducted an interpretability analysis using heatmaps. The heat maps provide partial insight into the model's decision process, although the interpretability still need improvements. However, when the board size is increased over 10×10 and the complexity becomes greater, the model is not able to generalize enough to learn winning patterns. Therefore, the model may benefit from more generalized symbols.

Overall, the solution shows promising potential, but can benefit from symbol generalization, a systematic hyperparameter search, and improve feature extraction to enhance interpretability.

Acknowledgment

References

[1] O.-C. Granmo, "The tsetlin machine–a game theoretic bandit driven approach to optimal pattern recognition with propositional logic," *arXiv preprint arXiv:1804.01508*, 2018.

[2] G. T. Berge, O.-C. Granmo, T. O. Tveit, M. Goodwin, L. Jiao, and B. V. Matheussen, "Using the tsetlin machine to learn human-interpretable rules for high-accuracy text categorization with medical applications," *IEEE Access*, vol. 7, pp. 115 134–115 146, 2019.

[3] U. Anjum and J. Zhan, "A novel tsetlin machine with enhanced generalization," *Electronics*, vol. 13, no. 19, p. 3825, 2024.

[4] C. Kinateder, "A novel approach to implementing knowledge distillation in tsetlin machines," *arXiv preprint arXiv:2504.01798*, 2025.

[5] O.-C. Granmo, Y. Abdelwahab, P.-A. Andersen, P. F. A. Clarke, K. Dumbre, Y. Grønninsæter, V. Halenka, R. Helin, L. Jiao, A. Khalid, R. Omslandseter, R. Saha, M. Shende, and X. Zhang, "A tsetlin machine for logical learning and reasoning with graphs," *Fourthcomming*, 2025. [Online]. Available: https://github.com/cair/GraphTsetlinMachine

[6] V. Halenka, A. K. Kadhim, P. F. A. Clarke, B. Bhattarai, R. Saha, O.-C. Granmo, L. Jiao, and P.-A. Andersen, "Exploring effects of hyperdimensional vectors for tsetlin machines," 2024. [Online]. Available: https://arxiv.org/abs/2406.02648

[7] O.-C. Granmo, S. Glimsdal, L. Jiao, M. Goodwin, C. W. Omlin, and G. T. Berge, "The convolutional tsetlin machine," 2019. [Online]. Available: https://arxiv.org/abs/1905.09688

[8] T. Hazra and K. Anjaria, "Applications of game theory in deep learning: a survey," *Multimedia Tools and Applications*, vol. 81, no. 6, pp. 8963–8994, 2022.

[9] C. A. Kamhoua, C. D. Kiekintveld, F. Fang, and Q. Zhu, *Game theory and machine learning for cyber security*. John Wiley & Sons, 2021.

[10] C. Yang, "A brief review of fixed points, hex game and hex theorem," *arXiv preprint arXiv:2508.00865*, 2025.

[11] P. Henderson, "Playing and solving the game of hex," 2010.

[12] D. Gale, "The game of hex and the brouwer fixed-point theorem," *The American mathematical monthly*, vol. 86, no. 10, pp. 818–827, 1979.

[13] C. McGregor, "Using constructive research to structure the path to transdisciplinary innovation and its application for precision public health with big data analytics," *Technology Innovation Management Review*, 2018. [Online]. Available: https://timreview.ca/article/1174

[14] A. Zytek, I. Arnaldo, D. Liu, L. Berti-Equille, and K. Veeramachaneni, "The need for interpretable features: Motivation and taxonomy," *SIGKDD Explor. Newsl.*, vol. 24, no. 1, p. 1–13, Jun. 2022. [Online]. Available: https://doi.org/10.1145/3544903.3544905

Appendix

A. Performance tables

All results are available in the `results` folder of the GitHub repository:
https://github.com/thomshetland/graphTM-project