TseHex

Jon Andreas Bull Larssen
Jon Ingvar Jonassen Skånøy
Isak Killingrød

Contents

List of Figures

List of Tables

TseHex

***Abstract*—The Graph Tsetlin Machine (GTM) extends the Tsetlin machine (TM), a logical and interpretable learning algorithm, to handle graph-structured data. This paper investigates the application of GTM to predict outcomes in the game of Hex, a combinatorial two-player board game known for its absence of ties and high strategic complexity. The study encompasses dataset generation through random play, graph construction methodologies, and feature encoding schemes for representing Hex board states as directed graphs.**

Experimental evaluations demonstrate the GTM's ability to capture game dynamics, leveraging hyperdimensional vectors and message-passing mechanisms to model hierarchical relationships within game states. A comprehensive hyperparameter tuning analysis is conducted to explore the operational space and performance characteristics of GTM across varying board sizes, rather than focusing solely on optimization. The interpretability of GTM is examined by analyzing its learned logical clauses, revealing correlations between feature importance and board configurations.

Performance analysis reveals that GTM achieves accuracies in the range of 99% on 5x5 boards, with a gradual decline to approximately 81% on 15x15 boards in final states, *without* optimized configurations. *With* targeted optimizations, the GTM achieves 99.46% accuracy on 11x11 boards in final states, 93.06% accuracy two moves before completion, and 90.55% accuracy five moves before completion, demonstrating robust performance with advanced feature engineering. However, scalability challenges arise with increasing board complexity, highlighting the need for further optimization.

This study provides a foundational perspective for applying GTM to strategic games, showcasing its potential as an interpretable artificial intelligence (AI) solution for combinatorial domains. Future directions include optimizing graph representations, enhancing GTM scalability for complex real-world applications, and exploring self-play mechanisms for dataset generation and enhancement.

I. Introduction

The GTM extends the TM, a logical and interpretable learning algorithm, to handle graph-structured data. This paper investigates the application of GTM to the game of Hex, a combinatorial board game with no ties and high strategic complexity. The primary contributions of this study are:

- Application of graph-based methodologies for representing Hex game states.
- Performance evaluation of GTM across varying board sizes.
- Identification of scalability challenges and potential avenues for optimization.
- Insights into interpretability through logical clause analysis.

[0]Source code is available at https://github.com/Jon-Ingvar-Skanoy/hex/tree/delivery

The game of Hex is an abstract strategy game that has been widely studied in both game theory and AI research due to its compelling nature. It combines very simple rules with a high level of strategic complexity. One of the fascinating aspects of Hex it that it can never end in a tie, the only way to prevent one's opponent from winning is to win the game[1].

The TM[2] is a novel approach to pattern recognition, capable of handling complex patterns in an interpretable manner. It employs teams of Tsetlin automata (TAs) to learn patterns as propositional logic. This approach offers a transparent and interpretable alternative to black-box methods, such as neural networkss (NNs). Compared to NNs, the TM is computationally efficient and competitive on a range of benchmark datasets[3].

The GTM is an extended version of the TM which can handle data as directed graphs. It leverages hyperdimensional representation and nested logical clauses to achieve high interpretability and performance. This extension makes GTM particularly suited for problems where data can naturally be represented as graph structures, such as the Hex board.

This report aims to explore the application of the GTM on the game of Hex, as this is a game which can be represented as a directed graph. The application of TM[4] has been tested earlier and achieved promising results. This work will explore the GTM in the context of the game of Hex.

Topics covered include game board generation, representation, preprocessing, graph encoding methods, and the hyperparameter space of the GTM. It provide extensive overviews of GTM's performance in the context with varying degree of optimizations. The experiments attempt to illuminate how GTM respond to adjustments for better understanding of it's inner workings. This work will also explore some methods of interpretability analysis of the GTM.

II. Background

A. The Game of Hex

Hex is an perfect information abstract strategy board game with two players[1]. The players try to connect their opposite sides of a board of hexagonal cells. The game is always won or lost as there is no possibility of a draw.

History: The game was created by a Danish mathematician called Piet Hein. As the game is known to always have a winner it has been the target of some game theory and computer science research[1].

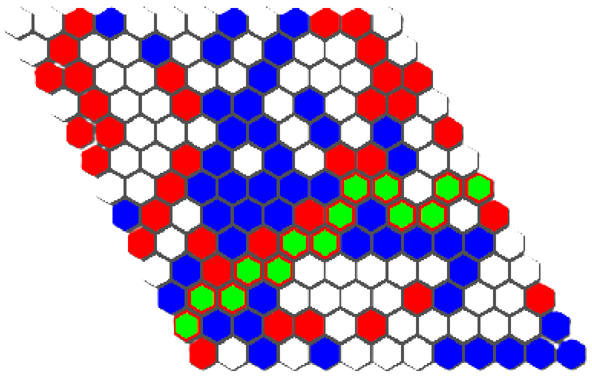


Fig. 1. Illustration of a game of Hex where red player win by connecting left and right side

B. The Tsetlin Machine (TM)

The TM is a novel method of machine learning which solves complex pattern recognition tasks with proportional logical formulas[2]. This is done with the novel TA. The TM operates on binary data.

Tsetlin Automaton: The TA is a way to find the optimal action in a unknown stochastic environment. The TA has a low computational complexity and achieves a rapid and accurate convergence. The TA is a solution to the multi-armed bandit problem[2].

The TA is trained through a combination of rewards an penalties which modify the internal state of the automaton. The TA is deterministic it decide its action based on its state. Figure 2 shows a two action automaton with 2N states. If the state is N or below action 1 is taken or if the state is N+1 or above action 2 is taken. When the Automaton is punished it moves towards the middle, when it rewarded, it moves away from the middle. In this manner The Automaton learns the best action over time even with a low signal to noise ratio. Multiple Automaton can also cooperate as demonstrated by its ability to solve the Goore game.

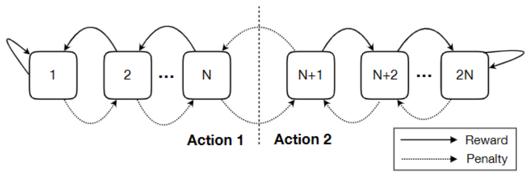


Fig. 2. A Two action Tsetlin automaton [2]

Clauses: A clause consists of a team of automatons[2]. Each automaton is responsible for whether a literal is included or excluded from a clause. There are multiple types of feedback for the TM to learn to classify data. Recognize feedback is for when the literal is true and the class is true. This moves the state of every automaton to maximally memorized. For for when the literal is false and the class is true there are erase feedback. For every automaton there is a $\frac{1}{s}$ chance of moving the literal 1 state towards maximally forgotten. s is the Hyperparameter specific which can be modified to control how specific the clauses are. When the literal is false and the class is false. every class in forgotten is moved one step towards minimally memorized[5].

With these 3 types of feedback one can train a clause. To classify you need multiple clauses. The predicted class is the class with the plurality of the positive clauses. To coordinate the clauses to achieve the best result, a vote margin is used. This means the TM only train clauses so that the margin is T, This results in clauses being able to cooperate and achieve much better results combined.

C. Graph Tsetlin Machine (GTM)

Graph Encoding: The GTM is a version of the TM modified to work on directed graphs. There is a number of problems which can be represented by graphs. This can for instance be money transfers, social networks, webpages on the internet or in this case, the game of Hex.

Hypervector: A HV is a high dimensional binary vectors which can represent complex features in a rich structure of data. In this manner it preserves the uniqueness of the data while improving accuracy and efficiency in decision making [6].

Messages: Message passing is an essential part for the GTM. A pool of clauses evaluates every node, when a clause corresponds to the properties of a node it sends a message to the nodes in the outgoing edges[7]. When a node receives a message it ads it to its properties. If this results in a match a new message is sends out in the next message round.

In this way the GTM uses nested clauses. As a class can use a the outcomes of other clauses to create hieratically clause structures, centered around various nodes[7].

D. Hyperparameters

The most influential parameters of a TM are the number of $clauses$, the specificity parameter s, and the voting threshold T. Their values and mutual balance significantly affect the TM's learning capacity and performance.

Number of Clauses: This parameter decides the size of the GTM. Generally a larger number of clauses increase the models capacity to understand complex patterns. This may result in a higher accuracy at an increased computational cost.

Specificity: Specificity or s determines how specific the clauses are. This is done partly by modifying the forget rate which is $\frac{1}{s}$, where s can be from 1 and higher. With more specific clauses more clauses are needed to represent the entire dataset.

Vote margin: Vote margin or T is the Tsetlin Machines method of coordinating clauses. Vote margin is the desired difference between the votes of the correct class and the wrong class. This is achieved when training the clauses.

Depth: In addition to the existing parameters for TM, GTM introduce a parameter for $depth$. Depth decides the number of message rounds in a model[7]. This decides how deep the nested clauses can be.

The hyperparameters affect each other. particularly T and s needs to work together to achieve a good result. A detailed parameter study is done on the MNIST dataset[8] with a TM which shows the importance of considering the combination of T and s[9].

E. Related Work

There have been numerous advancements in using AI algorithms to analyze or play strategic games, including Hex, chess, and Go. Training machine learning (ML) models to play games serves as a benchmark for developing algorithms that can learn complex decision-making, pattern recognition, and strategic planning. Games like Hex offer deterministic environments where success relies on mastering combinatorial reasoning, making them ideal for testing AI capabilities.

The development of game-playing AI has seen significant progress across different domains. AlphaGo's success in mastering Go demonstrated the power of combining deep learning with Monte Carlo tree search (MCTS) to achieve superhuman performance[10]. Similarly, AI systems for chess, such as Deep Blue, utilized search trees and heuristics to defeat human grandmasters. In the case of Hex, programs like Hexy[11] and MoHex[12] have dominated early competitions, leveraging virtual connections, MCTS, and H-search techniques. Later, MoHex 2.0[13] introduced further improvements through enhanced pattern recognition using the MiniMax algorithm.

In more recent developments, neural network-based approaches like NeuroHex[14] have applied deep reinforcement learning, while MoHexNet[15] combined depth-1 search trees with MCTS to win the 2016 Computer Olympiad. Despite these advances, one significant limitation of neural network-based systems is their lack of interpretability, which poses challenges for human-AI collaboration. This has motivated the exploration of interpretable AI methods for games like Hex.

Granmo et al.[2] explored the application of TMs in classical datasets like Binary Iris[16] and Binary Digits[17], achieving test accuracies of 95% and 95.7%, respectively, using relatively low numbers of clauses and computational resources. For more complex tasks, such as predicting outcomes in the board game Axis & Allies, the TM exhibited strong interpretability with an accuracy of 87.7% on test data, employing 10 000 clauses [2]. The TM's also excel in noisy environments, as demonstrated by their performance on the Noisy XOR task, where a binary classification accuracy of 99.3% was achieved even with 40% output noise.

Applications of the TM have extended natural language processing (NLP). Halenka et al.[6] applied TMs with HV's for sentiment classification on the IMDB dataset[18], achieving 88.1% accuracy with an HV size of 8192 and 4 active bits. For datasets like TREC[19], HV configurations demonstrated stability improvements with increased message bits, though accuracy remained consistent across configurations. Similarly, TMs have been effective in federated learning, as shown by Gohari et al.[20], achieving near state-of-the-art accuracy on MNIST[8] (98.94%) and Fashion-MNIST[21] (98.52%) datasets with personalized setups.

Researchers has also investigated the impact of hyperparameter choices on TM performance. Tarasyuk et al. [9] systematically analyzed the relationship between key hyperparameters, such as the voting threshold T and specificity s, demonstrating their significant influence on accuracy and training time. The study concluded that T and s must be carefully tuned to optimize the trade-off between learning capacity and computational efficiency, with optimal values scaling logarithmically and square-root-like with the number of clauses, respectively.

In the domain of board games, Giri et al.[4] applied TMs, not as a model predicting the next move, but as a model determining which player would win in a given game state. Using self-play data against MoHex 2.0, they achieved 92.1% accuracy on a dataset of over 280 000 board configurations. This study highlighted the potential of TMs in handling combinatorial game tasks while maintaining interpretability through clause analysis. Their approach visualize many clauses in a good understandable manner. Since this uses significantly different data the results are not directly comparable to the results in this report.

III. Methods

A. Experimental Design

There is little written about the optimal parameters for the GTM due to its novelty. In order to better understand the strengths and weaknesses of GTM this paper follow a very experimental approach within the context of predicting winner of games of Hex.

B. Hardware Setup

The experiments are performed on a combination of local machines and on a shared server. The experiments are primarily performed on either Nvidia RTX 3080 10GB or Nvidia Tesla V100 SXM3 32 GB. More hardware specifications are listed in appendix A.

C. Dataset Generation

The datasets are generated with a modified version of the given file Hex.c, available at[22][1]. Since the game of Hex is a turn-based game it is very simple to predict the winner of a game state close to the end of the game. In order to make sure that the model learn patterns in the games, not count number of

[1]Note that the approach for data generation is highly inefficient, but served it's purpose for this project

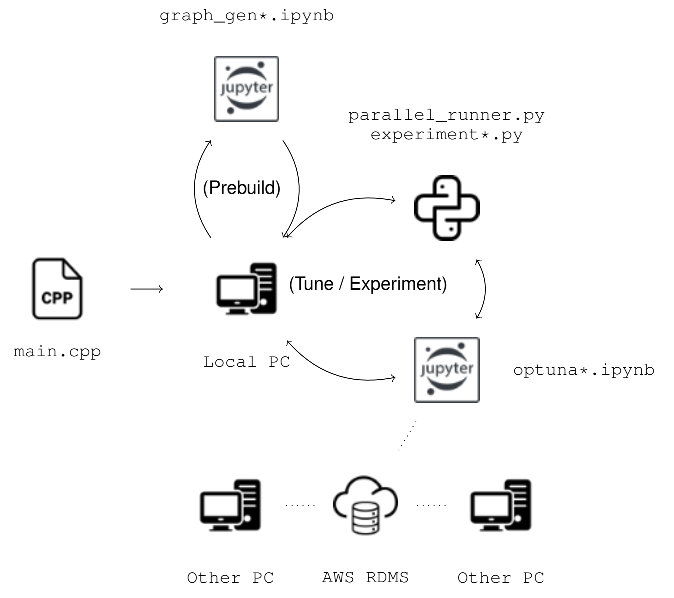


Fig. 3. Project Workflow.
('*' respresent that several files are used with similar structure and naming)

moves played, the program chooses a random starting player. Then each player alternately makes a random move. When one of the players win, the game is ended. The number of empty tiles are compared to a threshold if its above the threshold, the game is accepted. 3 versions of the game is saved: the final states, two moves before, 5 moves before. This is done until a certain number of games is created and saved.

There are two main parameters for this program, the game board size and the empty tiles threshold. The impact of these decisions are discussed in the experiments IV-C and IV-B. However be aware that for this program if the empty tiles threshold is set too high (above 40% of board size) the process of generating boards will be a lot slower than it already is.

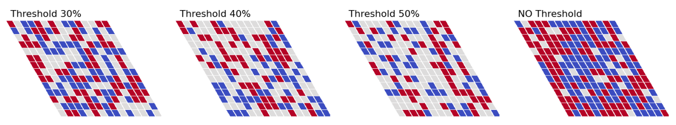


Fig. 4. Datasets with different percentage of open positions

Random Start Player: A traditional Hex game has a set starting player. However we discovered that when our datasets have a set starting player it has a much higher performance. This is highly likely due to a counting of the number of pieces which indicates who won. This does not give the challenge and understanding of the board we wanted, Therefore all boards are generated with a random starting player making counting pieces irrelevant.

Data Representation: The games in the dataset are stored in a `.csv` file where each row corresponds to a single game. The columns in the dataset include headers representing the state of every cell on the board, as well as metadata about the starting player and the winner of the game. Each cell is denoted using a naming convention based on its coordinates, such as `cell0_0`, `cell0_1`, ..., `cell3_3`, for a 4x4 board. The dataset format is illustrated as follows:

```
cell0_0,cell0_1,cell0_2,cell0_3,
cell1_0,cell1_1,cell1_2,cell1_3,
cell2_0,cell2_1,cell2_2,cell2_3,
cell3_0,cell3_1,cell3_2,cell3_3,
starting_player,winner,
-1,1,0,0,0,0,0,0,0,0,-1,0,1,0,0,0,1,1
```

Each cell's value represents the state of the game at that position: `-1` indicates that the cell is occupied by Player 1, `1` indicates Player 2, and `0` represents an empty cell. Additionally, the `starting_player` column specifies the player who began the game (`-1` for Player 1 and `1` for Player 2), and the `winner` column indicates the outcome of the game using the same player identifiers.

Data Preprocessing: Fig. 3 roughly illustrates the workflow of the project. When the datasets are generated with `main.cpp`, useful information is encoded in the filenames. Table I explains that filename like `8x8_20000_30_2.csv` has a board size of 8x8, 200 000 samples, 30 open postitions and the games are represented 2 moves before game ends. The `graph_gen*.ipynb` files then process the datasets (`.csv`), build graphs, and store them as pickle objects (`.pkl`). The table also explain the naming convention for the prebuilt graphs with filenames like `8x8_10000_30_2_8192_1_dh1.pkl` where information of Hypervector size (HVS) (8192), Hypervector bits (HVB) (1) and double-hashing (dh1) is encoded.

The earlier versions of code for generating datasets didn't ensure unique games, that is why dataset files had twice as many games generated than was actually built in the graph objects.

Attribute	**Dataset**	**Graphs**
File Type	.csv	.pkl
Board Size	8x8	8x8
Number of Samples	20000	10000
Open Positions (%)	30	30
Moves Before End	2	2
HVS	-	8192
HVB	-	1
Double Hashing[2]	-	dh1

TABLE I. Naming Conventions for Dataset and Graph Files.

The graphs objects vary in size, and therefore also construction time, which is why it makes sense to prebuild the graph objects for tasks like hyperparameter tuning or for experiments that don't require different types of graph configurations.

The `graph_gen*.ipynb` files that prebuild graphs use

[2]Double Hashing: dh2=double_hashing, dh1=not double_hashing

datahandler.py[3] for loading, balancing and splitting[4] the datasets. datahandler.py control that there is no data-leakage between train and test and ensures that the graph_gen*.ipynb[5] files build graphs with 50/50 balance between winning players in both train and test data.

D. Graph Construction

The graphs utilized by a GTM are composed of nodes, each associated with specific features and connections (edges) to other nodes. These nodes can be encoded in various ways, with directions representing the relationships between nodes. This directional encoding captures relational information that can aid the GTM in recognizing patterns and accurately classifying the game board.

For the game of Hex, there are multiple approaches to representing the board as a graph, and the choice of encoding method can significantly impact the model's ability to identify relevant patterns. Additionally, the way nodes and edges are encoded also contributes to the interpretability of the model, making it wise to address this aspect during graph construction. Comparing different encoding methods is beneficial for identifying the most effective approach for constructing the board representation.

Although the starting player for each game is included as an attribute in the dataset, this information is not incorporated into the generated graphs and is therefore not provided to the model in any of the experiments.

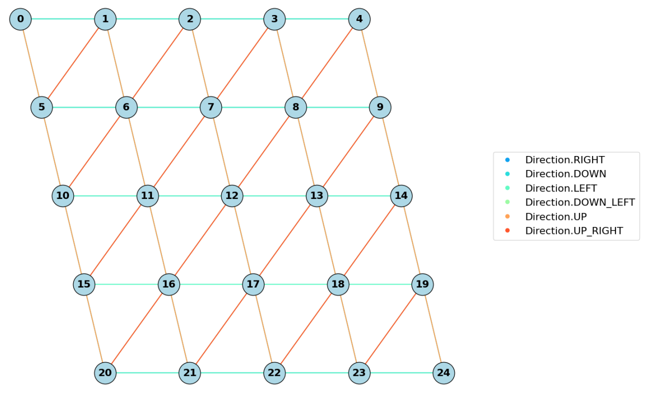


Fig. 5. Basic Graph Encoding with adapted version of [23]

To describe Hex, the board representations outlined in this report all represent each cell on the board as a node in the graph. Each node is encoded with a feature describing whether the cell has a red piece played on it, and another feature for whether or not the cell has a blue piece. Unless specified otherwise, each experiment makes use of a directional encoding in which each cell is connected to its "hex-adjacent" cells (see Fig. 6), with a different encoding for each combination of direction and color of the adjacent cell (such as "down_right_red", "left_blue" or "up_left_empty). In total, this system of encoding directions results in 18 different directions between nodes. By encoding color and relative position into the direction of the connections, the hope is that this more granular information makes it easier for the GTM to pick up on the importance of the color of adjacent nodes in relation to each node.

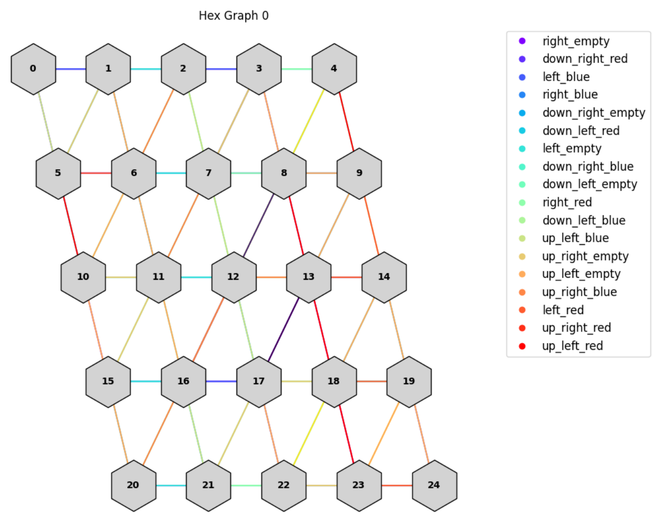


Fig. 6. Extended Graph Encoding with adapted version of [23]

Feature Encoding: When describing the features of an individual node, there is an important balance to be aware of. If one encodes some knowledge that is already known to be important through human understanding from before, it is possible to aid the GTM in looking through what is relevant and picking up on important patterns. However, if one passes through too much of what is known about the game state, to the point where an introduced feature easily causes the model to reach 100% accuracy, the problem might become trivial, and the value of the experiment in terms of evaluating the GTM is lost.

The most important features encoded into each node is whether a blue or red piece is placed within the corresponding cell, which is represented as two boolean values. Each node also has its position on the board, represented as coordinates along columns and rows, where both column number and row number are thermometer encoded separately.

In addition, red and blue nodes are each encoded with a feature called "CONNECTED_BLUE" and "CONNECTED_RED" which is set to True if two adjacent nodes are of the same color as the node, or if the node is on a relevant edge of the board and there is at least one adjacent node. For instance, if a red cell on the board is on the top or bottom row

[3]datahandler.py also contain functionality for data inspection as well as loading and saving graph objects

[4]Default split is 80 train / 20 test

[5]Some of graph_gen*.ipynb use graphhandler.py

and has an adjacent red cell, or a red cell has two adjacent red cells, ”CONNECTED_RED” is set to True. As the results of several experiments will demonstrate, including these features makes it easier for the GTM to recognize that the connections between tiles and leads to improved performance. However, the inclusion of these features does not entirely trivialize the problem, as the model still misclassify boards at a range of board sizes.

Graph And Feature Variants: Throughout this report, the graph and feature encoding described above has been used for all experiments if not stated otherwise, and will therefore be referred to as the baseline representation. However, to understand the impact the board representation has on the performance of the GTM, other experiments involving alternate board representations have been performed.

One alternate board experiment involves using four nodes to represent the edges of the board, and connecting these to the nodes along each corresponding edge using a unique set of directions (”Left”, ”Right”, ”Up”, ”Down”), and a separate boolean feature for each node indicating its value (”LEFT”, ”RIGHT”, ”UP”, ”DOWN”). The reason for this feature, referred to as ”anchor edge nodes” in the rest of the report, is to put additional trait on the outermost nodes in the hopes that it will make it clearer for the model that the outermost nodes have an important attribute in the context of the game.

The second alternate board experiment uses the baseline encoding, but removes the features ”CONNECTED_BLUE” and ”CONNECTED_RED” in order to estimate the performance impact of the presence of these features.

The third and fourth alternate board experiment tests a different way of encoding directions and connections between nodes in the graph. This variant, which is referred to as Only-Colors for the rest of the report, removes all connections going to empty nodes, and instead only has outgoing connections to red or blue nodes. This experiment tests two variants, one in which empty nodes can still have outgoing connections to red and blue nodes (OnlyColor + Empty), and one in which empty nodes are completely isolated on the graph (OnlyColor - Empty).

The fifth and sixth alternate board experiments test another way of encoding graph directions, which is by not encoding connections between blue and red nodes. This method, referred to as InterColor for the rest of the report, builds the graph in such a way that blue nodes can only have connections to adjacent blue nodes, and red nodes can only have connections to adjacent red nodes. This graph representation also has two variants, one which allows empty nodes to have outgoing connections to red and blue nodes (InterColor + Empty), and one which isolates empty nodes entirely (InterColor - Empty).

The seventh alternate board experiment removes the thermometer encoding from the baseline representation, instead using only a single boolean for each row and column, in order to estimate the impact thermometer encoding has.

The eighth alternate board experiment introduces a new set of features to the baseline representation, called ”GAP_RED” and ”GAP_BLUE”. These features are true for any empty node with two adjacent nodes which have the same color, or one color and a related edge. An empty node would for instance have ”GAP_BLUE” set to True if it is along the left edge and has an adjacent blue node. The goal of testing this feature is to nudge the model in the right direction when classifying unfinished games without giving the answer away by itself.

E. Hyperparameter Tuning

Hyperparameter tuning is crucial to maximizing the efficiency and accuracy of TMs, particularly in complex tasks like Hex game predictions. Inspired by prior research, our tuning strategy focused on exploring the interaction between the most influential parameters: the number of $clauses$, specificity (s), and voting threshold (T)[9]. To avoid data leakage Optuna studies where run on different datasets than used in this report.

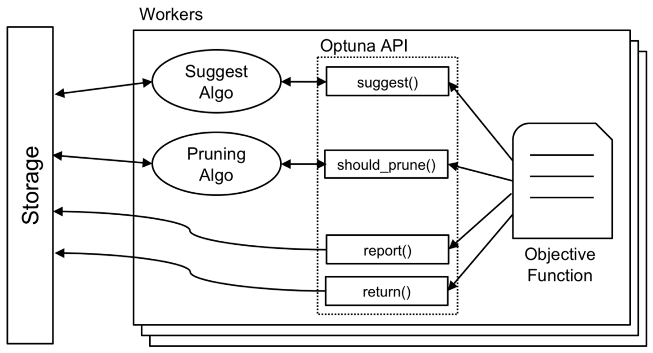


Fig. 7. Illustration of Optuna Hyperparameter Setup[24].

In order to share the progress of the tuning process across different workers, this project utilize AWS[6] db.t3.micro as storage for Optuna studies.

IV. Results

In this study, approximately 1 000 datasets were generated across various configurations, resulting in a cumulative total of approximately 37 million game boards. These configurations included variations in board sizes (ranging from 4x4 to 18x18), sample sizes (1k, 10k, 100k), the number of open positions (from 0 to 50 in steps of 5), and moves before the game’s conclusion (0, 2, 5). To gain a comprehensive understanding of the application of the GTM in this problem domain, several thousand tuning trials were conducted across numerous Optuna studies. This extensive effort reflects the significant allocation of time and resources aimed at exploring the model’s capabilities and behavior within the given context.

[6]Relevant code is in `dbhandler.py`

Overview of experiments

- IV-A **Initial Performance Overview**
- IV-C **Number of Open Board Positions**
- IV-B **Board Size and Moves before end**
- IV-D **Graph Encoding**
- IV-E **Number of Clauses**
- IV-F **Number of Literals**
- IV-G **Graph Depth**
- IV-H **Hypervector Size and Message Bits**
- IV-J **Interpretability**

A. Initial Performance Overview

The first step was to get an overview of how a GTM behaves in this context, without any major optimizations. The literature demonstrate that increased number of samples and number of clauses usually result in increased accuracy. The experiment[7] is therefore performed as a grid search on prebuilt graph objects from datasets with samples of 1 000 and 10 000 and board sizes from 4x4 to 15x15.

Small initial screening runs[8] were performed beforehand and indicated that increased number of open positions in the game board also increase accuracy. The graphs were encoded as described in "*Graph Encoding*" with basic connections as illustrated in Fig. 5, with 40% open positions[9].

Since "*Related Work*" doesn't clearly state which parameters that are suitable for the problem domain the initial approach is wide with mostly default values[11] of GTM are used. The most influential parameters of a TM is the number of $clauses$, values for s and T, as well as the correlation between them. For a starting point parameters were set to $s = 1.0$[25], $T = clauses \times 0.8$[4], $q = 1.0$ and $depth = 1$.

Observations: Due to the novelty of GTM and the lack of literature on the topic, the complete overview of initial performance is provided in Table II. Several observations can be made from this data:

Performance across board sizes: Test accuracy shows a clear downward trend as board sizes increase. This pattern is consistent with the expectation that larger board sizes is a more complex problem for the GTM which require more advanced strategies or parameter tuning to achieve high accuracy. For instance, accuracy drops from 99% to 62% as board size increases from 4x4 to 15x15.

Sample Sizes: Across all board sizes, 10 000 samples outperform 1 000 samples, as expected. The gap in performance is more significant as the board size increase as illustrated om Fig. 9

[7]Exp. "Initial Performance". Experiment Runner Script., Experiment Script.

[8]Screening runs are not documented in the paper

[9]Exp. "Initial Performance". Graph generation notebook.

[10]Exp. "Initial Performance". Server, 1 process per parameter combination, 12 parallel processes, 5 runs x 150 epochs

[11]Exp. "Initial Performance". Parameters: Appendix B, Table XII

	Initial Performance[10]				
Size	Samples	Clauses	Train Acc	Test Acc	F1
4	1000	1000	99.75 (0.09)	97.10 (1.52)	0.97 (0.02)
		10000	100.00 (0.00)	98.70 (0.27)	0.98 (0.01)
		100000	100.00 (0.00)	99.00 (0.00)	0.99 (0.00)
	10000	10000	99.77 (0.02)	99.39 (0.02)	0.99 (0.00)
		100000	99.81 (0.01)	99.35 (0.04)	0.99 (0.00)
5	1000	1000	94.02 (2.22)	89.30 (2.41)	0.89 (0.03)
		10000	98.20 (0.67)	93.60 (1.14)	0.94 (0.01)
		100000	98.00 (0.08)	93.20 (0.27)	0.93 (0.01)
	10000	10000	99.63 (0.05)	99.11 (0.16)	0.99 (0.00)
		100000	99.63 (0.05)	99.07 (0.03)	0.99 (0.00)
6	1000	1000	97.18 (1.01)	83.80 (1.89)	0.84 (0.02)
		10000	99.78 (0.06)	89.00 (0.35)	0.89 (0.00)
		100000	99.85 (0.06)	89.30 (0.45)	0.89 (0.01)
	10000	10000	96.90 (0.50)	95.10 (0.57)	0.95 (0.01)
		100000	96.76 (0.21)	95.03 (0.09)	0.95 (0.00)
7	1000	1000	93.80 (1.65)	82.80 (2.08)	0.83 (0.02)
		10000	99.83 (0.21)	91.20 (1.15)	0.91 (0.01)
		100000	99.95 (0.07)	93.10 (0.82)	0.93 (0.01)
	10000	10000	93.80 (0.73)	91.96 (0.57)	0.92 (0.00)
		100000	94.57 (0.28)	92.76 (0.34)	0.93 (0.01)
8	1000	1000	91.57 (2.02)	75.20 (1.57)	0.76 (0.02)
		10000	99.02 (0.70)	81.80 (1.92)	0.80 (0.03)
		100000	99.37 (0.26)	83.00 (0.61)	0.81 (0.01)
	10000	10000	96.02 (0.38)	92.55 (0.73)	0.93 (0.01)
		100000	97.27 (0.08)	93.40 (0.20)	0.94 (0.01)
9	1000	1000	88.90 (2.03)	74.60 (2.68)	0.73 (0.03)
		10000	99.38 (0.34)	79.60 (2.38)	0.76 (0.02)
		100000	99.83 (0.07)	80.40 (1.24)	0.77 (0.02)
	10000	10000	94.82 (0.32)	89.51 (0.47)	0.90 (0.01)
		100000	96.43 (0.29)	91.16 (0.26)	0.91 (0.00)
10	1000	1000	90.75 (1.21)	71.20 (1.52)	0.73 (0.03)
		10000	99.75 (0.25)	73.40 (1.56)	0.74 (0.02)
		100000	99.65 (0.11)	74.10 (0.96)	0.74 (0.01)
	10000	10000	90.60 (2.43)	85.45 (1.70)	0.87 (0.01)
		100000	93.20 (1.00)	87.02 (0.60)	0.88 (0.00)
11	1000	1000	86.45 (4.31)	71.50 (2.85)	0.67 (0.05)
		10000	96.18 (1.64)	75.40 (1.29)	0.72 (0.02)
		100000	97.75 (0.35)	74.50 (1.17)	0.70 (0.02)
	10000	10000	89.12 (0.65)	83.50 (0.57)	0.85 (0.00)
		100000	94.44 (0.69)	87.42 (0.81)	0.88 (0.01)
12	1000	1000	93.35 (3.05)	64.50 (2.40)	0.66 (0.04)
		10000	99.98 (0.05)	69.70 (1.44)	0.71 (0.01)
		100000	100.00 (0.00)	70.80 (0.91)	0.72 (0.01)
	10000	10000	92.04 (0.19)	83.48 (0.39)	0.83 (0.00)
		100000	96.68 (0.03)	86.85 (0.57)	0.86 (0.01)
13	1000	1000	88.45 (4.61)	64.60 (1.82)	0.64 (0.02)
		10000	99.98 (0.05)	66.40 (1.47)	0.67 (0.01)
		100000	100.00 (0.00)	66.10 (0.42)	0.67 (0.00)
	10000	10000	89.96 (1.66)	79.35 (1.06)	0.78 (0.02)
		100000	85.98 (4.12)	76.95 (2.12)	0.72 (0.04)
14	1000	1000	92.80 (2.08)	59.00 (2.67)	0.57 (0.04)
		10000	99.80 (0.38)	59.60 (1.29)	0.58 (0.02)
		100000	100.00 (0.00)	61.20 (0.76)	0.59 (0.01)
	10000	10000	86.68 (0.28)	77.42 (1.17)	0.74 (0.01)
		100000	93.09 (0.82)	81.50 (0.92)	0.80 (0.02)
15	1000	1000	92.25 (3.57)	61.00 (1.84)	0.61 (0.02)
		10000	100.00 (0.00)	62.90 (2.79)	0.61 (0.02)
		100000	100.00 (0.00)	62.00 (1.54)	0.60 (0.02)
	10000	10000	89.32 (0.65)	77.62 (1.29)	0.78 (0.01)
		100000	95.08 (1.23)	81.73 (1.52)	0.82 (0.01)

TABLE II. MEAN (STD) OF METRICS FOR EXPERIMENT: INITIAL PERFORMANCE.

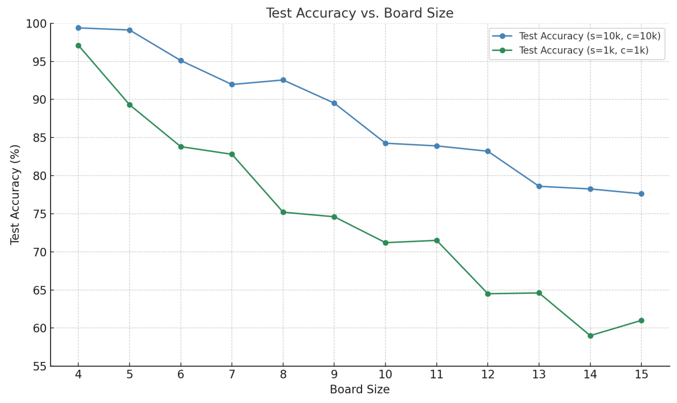


Fig. 8. Effect of Board Size on Test Accuracy

Scalability: The gradual decline in accuracy illustrate the challenge of scaling the (default) GTM to handle more complex and larger problem spaces effectively. While this trend is expected, it underscores the need for parameter tuning or adjustments in the graph construction to improve performance in these scenarios.

It is also observable that the increase from 10 000 to 100 000 clauses has little effect on both train and test accuracy, which confirm the findings from [9] for the GTM as well.

Future Directions: The initial performance results suggest that while the GTM exhibits strong baseline performance for smaller board sizes, its scalability to larger board sizes is limited under the current parameter configurations. Future work should focus on:

Data Representation: The initial experiment only evaluate boards with a fixed portion of open positions in their final states. Explorations of other configurations should be conducted to understand how varying the number and distribution of open positions impacts performance.
Graph Construction: Explore modifications to the GTM framework to improve its capacity for handling larger problem spaces.
Parameter Optimization: Conducting a more targeted exploration of key parameters such as number of $clauses$, s, T, and the number of clauses to identify configurations that better support larger board sizes.

In summary, the GTM demonstrate potential in solving combinatorial problems within this domain, but further optimizations and scalability enhancements are important for tackling larger, more complex board configurations effectively. This lays a promising foundation for subsequent experiments and refinements in the application of GTM to predictions on the game of Hex.

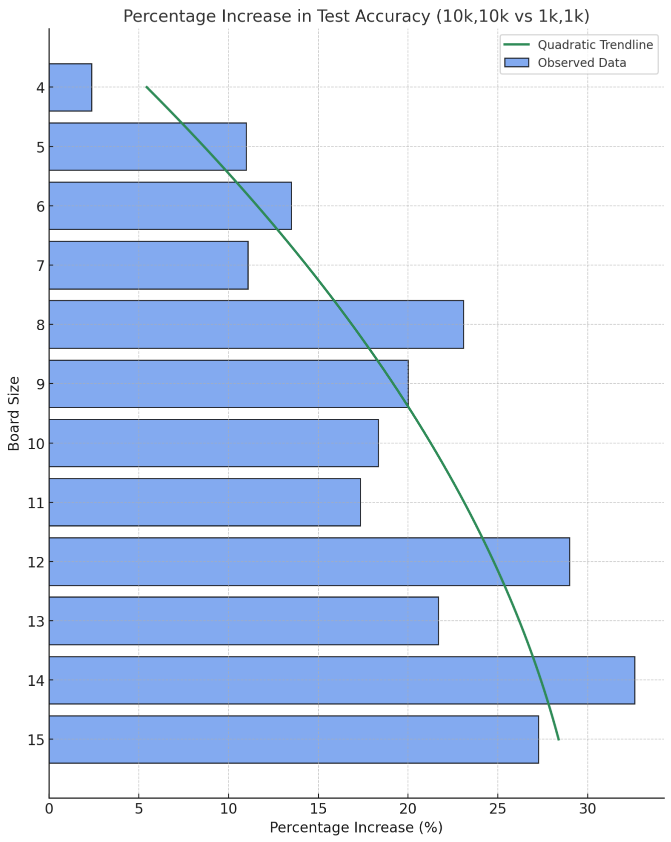


Fig. 9. Effect of Sample Size on Test Accuracy

B. Board Size and Game States

A larger board size is a more difficult task. This experiment seeks to demonstrate how this effects the 3 task this report handles. predicting the winner at end state, 2 moves before and 5 moves before.

For this experiment a number of datasets with varying sizes where generated with a minimum number of empty tiles at 40%. A challenge here is that for the datasets of 11x11 and 13x13 there were a high number of duplicates in the generated datasets and therefore a lower number of unique samples. The datasets were saved in it's final state, 2 and 5 moves before the final state. For each dataset there were constructed graphs with temperature encoding and without anchor edge nodes[12]. GTMs where trained on each graph with the hyperparameters in Table. XVII.

As shown in Table. III a larger board size is more difficult for both the final state and 2 moves before. For the board state with 5 moves before a size of 11x11 seems to be easiest. This is likely due to the fact that for a small board there are many cases where both players can win in the next 5 moves, but for

[12]"Anchor edge nodes" are auxiliary or boundary nodes added outside the board to represent connections to the board's edges, simplifying edge-specific logic.

larger board sizes this is less likely. The table also shows that it is easiest to classify the final state, then 2 moves before and then 5 moves before. It would be really interesting to determine to what degree this is because of the games where both players have the opportunity to win. The table also shows that with a larger dataset and optimized hyperparameters the GTM is able to scale to much larger board sizes than shown in the initial performance review.

Size	Moves before	Train Accuracy	Test Accuracy	Time (s)	Total Samples
5x5	0	99.99 (0.00)	99.99 (0.00)	457.53 (5.11)	199982
	2	98.44 (0.03)	98.28 (0.04)	455.66 (6.03)	199977
	5	78.51 (0.82)	78.34 (0.78)	547.11 (8.47)	199795
7x7	0	99.96 (0.00)	99.89 (0.01)	394.32 (1.31)	200000
	2	95.39 (0.14)	95.15 (0.13)	452.50 (1.11)	200000
	5	83.84 (0.60)	83.42 (0.72)	549.13 (3.99)	200000
9x9	0	99.86 (0.02)	99.81 (0.02)	420.99 (2.40)	200000
	2	95.02 (0.44)	94.62 (0.44)	520.03 (68.96)	200000
	5	86.46 (0.84)	85.96 (0.95)	604.26 (51.12)	200000
11x11	0	99.67 (0.03)	99.46 (0.03)	302.88 (2.25)	139606
	2	93.47 (0.88)	93.06 (0.80)	345.71 (0.43)	139606
	5	91.12 (0.12)	90.55 (0.09)	365.59 (6.78)	139606
13x13	0	99.59 (0.03)	99.10 (0.08)	117.21 (0.41)	44892
	2	94.26 (0.35)	93.13 (0.34)	127.96 (0.48)	44892
	5	90.13 (0.82)	88.66 (0.80)	134.98 (0.56)	44892

TABLE III. METRICS FOR DATASETS WITH VARYING SIZES AND AT END STATE, 2 MOVES BEFORE AND 5 MOVES BEFORE

C. Number of Open Board Positions

One of the most important factor in determining the difficulty of a board is the number of open positions. When generating a board we use a version of the HEX.c file[22] which uses random play. We believe this results in particularly difficult data and therefore we have put a limit unless the random players are able to find a winner with x empty tiles the game is discarded.

In this experiment we have generated a number of sets of 7x7 games, with a number of different limits ranging from 25 to 0 were 49 is the total number of tiles. For the experiment there are generated graphs with temperature encoding and without anchor edge nodes. Then there were trained GTM's with the hyperparameters in Table. XVII.

As shown in Table. IV the number of open positions is the most important factor in determining the difficulty of the dataset. A 7x7 dataset with a limit of 10 or below is more difficult then a 13x13 with 40% empty as shown in Table. III. Our result on 7x7, with 25% open positions, is what we submitted to the leader board. In this experiment 4 of 5 runs reached 100% with the last one at 99.995%. Since more open positions are unlikely to result in higher computation time, this table also shows the impact of performance on training time. The models at 99.999% accuracy trained faster then those as 93%.

Param	Train Accuracy	Test Accuracy	Time (s)
25	99.999 (0.000)	99.999 (0.002)	486.56 (5.02)
20	99.943 (0.011)	99.886 (0.007)	487.99 (7.50)
15	99.505 (0.038)	99.481 (0.032)	490.10 (0.91)
10	97.473 (0.142)	97.434 (0.128)	504.91 (6.17)
5	96.515 (0.113)	96.036 (0.088)	519.38 (6.64)
0	93.130 (0.356)	93.111 (0.254)	542.11 (14.16)

TABLE IV. METRICS FOR DIFFERENT NUMBER OF FORCED EMPTY TILES.

D. Graph Encoding

The graph representation of a Hex board has a significant impact on the performance of the GTM. To estimate the impact of the graph representation used in most other experiments, this section describes the results of experiments using the alternate graph representations described in section "*Graph And Feature Variants*". Each of the 9 variations of the graph representation has been used on the same dataset of 139 606 unique games on an 11x11 board, at the finished board as well as 2 and 5 moves prior to completion, giving a total of 27 datasets. For each dataset, a GTM is trained on the dataset. This is repeated 5 times for each dataset, and the average test and train accuracy of the GTM on each dataset is reported in the tables below. Notably, the GTM is trained using the same hyperparameters for every dataset, rather than trying to find the ideal hyperparameters for each dataset, due to time constraints. The hyperparameters used for these experiments can be found in appendix C, table XVII.

Experiment	Training Accuracy	Test Accuracy
Baseline	99.62	99.43(0.04)
Anchor Edge Nodes	99.35	99.19(0.10)
No CONNECTED	92.12	91.95(0.24)
OnlyColor - Empty	99.47	99.39(0.05)
OnlyColor + Empty	99.63	99.44(0.06)
InterColor - Empty	99.61	99.47(0.03)
InterColor + Empty	99.65	99.47(0.06)
No Thermometer	99.59	99.42(0.01)
Gap Features	99.54	99.37(0.05)

TABLE V. PERFORMANCE OF ALTERNATE GRAPH REPRESENTATONS ON COMPLETED BOARDS

As shown by Table. V, when classifying completed games of Hex, most representation variants perform similarly, with two exceptions. The addition of anchor edge nodes might achieve the opposite of its intended effect, and add more information which does not contribute to accurate classification, which might explain the loss in performance observed when using anchor edge nodes.

When removing the "CONNECTED_RED" and "CONNECTED_BLUE" features, accuracy drops by several percentage points compared to baseline, showing that the features makes it easier for the model to accurately classify the board. However, the GTM still manages to achieve an accuracy greater than 90% on the relatively simplistic board representation that remains when the "CONNECTED" features are removed, which indicates that the "CONNECTED" features are not the sole linchpin on which classification depends.

Experiment	Training Accuracy	Test Accuracy
Baseline	94.72	94.43(0.14)
Anchor Edge Nodes	93.12	92.64(0.33)
No CONNECTED	87.86	87.56(0.56)
OnlyColor - Empty	95.82	95.45(0.09)
OnlyColor + Empty	95.83	95.45(0.26)
InterColor - Empty	95.62	95.20(0.21)
InterColor + Empty	95.49	95.0(0.22)
No Thermometer	95.55	94.98(0.14)
Gap Features	95.93	95.46(0.25)

TABLE VI. PERFORMANCE OF ALTERNATE GRAPH REPRESENTATIONS ON BOARDS 2 MOVES BEFORE COMPLETION

Experiment	Training Accuracy	Test Accuracy
Baseline	91.09	90.59(0.22)
Anchor Edge Nodes	89.93	89.42 (0.40)
No CONNECTED	84.13	83.56(0.30)
OnlyColor - Empty	90.62	90.05(0.19)
OnlyColor + Empty	90.77	90.11(0.14)
InterColor - Empty	89.24	88.52(0.16)
InterColor + Empty	90.36	89.84(0.17)
No Thermometer	90.60	89.89(0.17)
Gap Features	90.33	89.38(0.23)

TABLE VII. PERFORMANCE OF ALTERNATE GRAPH REPRESENTATIONS ON BOARDS 5 MOVES BEFORE COMPLETION

As shown by table VI, when classifying games two moves before a player wins, the addition of anchor edge nodes has an even more pronounced negative impact on accuracy when compared to the baseline, further supporting the findings that anchor edge nodes are not useful. Removing the ”CONNECTED” features have a similar effect on accuracy as it does for classifying completed boards, further reinforcing the same findings.

An accuracy increase can be noted when using any of the four alternate ways of representing connections between nodes, with the GTM particularly benefiting from the encoding which has outgoing connections which reach another node which is either red or blue. Allowing empty cells to have outgoing connections to red or blue cells has little discernible impact on performance compared to having empty cells have no connections at all.

As with the test on completed boards, thermometer encoding of distance shows no significant impact on accuracy compared to baseline, which implies that it does not give significantly better understanding of the distance between nodes to the GTM. This could be due to the GTM already having a good understanding of the relative positions of nodes, as this is also represented by the connections between nodes on the graph.

On this dataset, the ”GAP_RED” and ”GAP_BLUE” features seem to bring an increase in accuracy, while having little impact when classifying completed games. This is likely due to empty nodes with adjacent red or blue nodes being significantly less relevant on a completed board than those of a nearly completed board. For the board two moves before completion, these features seem to make the GTM somewhat better at picking up on the relevance of available critical cells each player would like to place pieces on, but does not see to have as significant of an impact on accuracy as the ”CONNECTED” features.

As shown by table VII, the GTM still performs with a lower accuracy when anchor edge nodes are introduced, once again reinforcing that this attribute is not beneficial for the model. The same findings are repeated for the ”CONNECTED” features as well.

Unlike with two moves prior to completion, the GTM here seems to perform worse when trained on any variant of the representation compared to the baseline. One possible explanation for this is that the board five moves before completion is more sparse, and the features and connection representations tested are all to do with nodes that are directly adjacent. With sparser boards, the number of connections could be significantly reduced if outgoing connections to empty nodes are not encoded, and the ”GAP” features would also be set to False more often, reducing their significance. One observation which might support this is that InterColor encoding results in a lower accuracy than OnlyColor, with InterColor being the representation which forms fewer connections between nodes.

E. Number of Clauses

The number of clauses determines the size of the GTM. This determines the complexity of the solution. It also determines the computational requirements of the model. This experiment test if a larger model is necessarily better, and reports the increased training time with a larger model. This experiment used the same prebuilt graph with connected attributes, anchor edge nodes and thermometer encoding. It used a dataset of 11x11 where a 40% limit on empty tiles where set. Of the 200 000 games in the dataset 139 606 where unique and used with a 80 20 test split. the GTM where trained with the hyperparameters in table XVI. This experiment was performed on a local computer and therefore computation time is not directly comparable between tables.

as shown in table VIII a certain number of clauses are necessary to achieve the best results, however going much past this level is not necessarily beneficial. a large number of clauses above 10 000 seems to result in a higher standard deviation of the results, and therefore this test is uncertain of the average performance of 50 000 and 100 000, but it does not seam to be a significant improvement. the test is done with the same s and a T as a multiple of the number of clauses. It is possible that the area around 10 000 is best because of the hyperparameters.

The computation time shows that for 10 000 clauses and below the number of clauses seems to not be the most significant determining factor when it comes to computation time.

F. Number of Literals

The Max number of literals is a Hyperparameter which decides the maximum number of literals a clause can have. Limiting the number of literals leads to more general clauses while increasing results in more specific clauses.

Number of clauses	Number of Clauses		
	Train Accuracy	Test Accuracy	Time (s)
500	98.5198 (0.2663)	98.4263 (0.2775)	371.0697 (27.1194)
1000	99.1965 (0.0828)	99.0961 (0.0791)	330.0546 (21.8981)
3500	99.6641 (0.0635)	99.5079 (0.0573)	333.3941 (21.0863)
10000	99.7604 (0.0541)	99.5616 (0.0458)	342.5992 (8.7871)
50000	99.6390 (0.1050)	99.5008 (0.0754)	492.4520 (9.4255)
100000	99.6284 (0.1510)	99.5108 (0.1117)	673.2220 (16.3556)

TABLE VIII. MEAN (STD) OF METRICS FOR VARYING NUMBER OF CLAUSES.

This experiment used the same prebuilt graph as the number of clauses experiments. the GTM where trained with the hyperparameters in table XV.

As shown in table IX a to low or too high Max_literals parameter hurts performance. an optimal value is likely around 10-20. This shows that the forced generalization of the clauses contribute to a higher performance, as long as its not overdone.

Max literals	Max Literals		
	Train Accuracy	Test Accuracy	Time (s)
5	88.5454 (1.7163)	88.3089 (1.7660)	372.6853 (1.5940)
10	99.5845 (0.0355)	99.4800 (0.0398)	327.0443 (21.8667)
20	99.6094 (0.0682)	99.4664 (0.0610)	299.9450 (0.7382)
59	99.5715 (0.0806)	99.4048 (0.0495)	301.5932 (5.1441)
100	99.6426 (0.0492)	99.4800 (0.0599)	300.1179 (1.4598)
200	98.1399 (0.1152)	98.0181 (0.1124)	317.2385 (1.6453)
None	98.0069 (0.0902)	97.8748 (0.1443)	315.7649 (4.5410)

TABLE IX. MEAN (STD) OF METRICS FOR VARYING MAX LITERALS.

G. Depth

The depth determines the number of message rounds in a model[7]. This decides how deep the nested clauses can be.

This experiment used the same prebuilt graph as the number of clauses experiments. The GTM was trained with the hyperparameters in table XIII.

Depth has a large impact on the training time of the GTM as shown in Table.X. As seen in the table it does not have a large impact on the results of the model. It only seems like a depth of 2 and 5 harms performance while a depth of 1 and 5 are best and within each other Standard deviation range. But the results are to similar to determine anything conclusively.

H. Hypervector Size and Message Bits

Inspired by the work of Halenka et al. [6], we aim to reduce the number of $clauses$ by exploring different configurations of HVS and number of non-empty bits in each HV, HVB. The goal is to reduce the number of $clauses$ needed while maintaining or improving performance.

It is performed as a spaced sampling approach, although time and space constraints limit the scope of the search

Depth	Performance Metrics		
	Train AC	Test AC	Time(s)
1	99.6630 (0.0257)	99.5201 (0.0279)	391.8801 (30.7633)
2	99.5811 (0.0988)	99.4477 (0.0829)	769.9150 (42.0917)
3	99.6050 (0.1061)	99.4929 (0.0906)	1140.6267 (5.5083)
5	99.6175 (0.0355)	99.4793 (0.0349)	1974.5095 (13.3393)
10	99.6429 (0.0387)	99.5086 (0.0291)	4037.6033 (14.4059)

TABLE X. MEAN (STANDARD DEVIATION) OF TRAIN/TEST ACCURACY AND COMPUTATION TIME FOR EACH DEPTH.

space. The constraints also promote the use of lower board sizes and fewer sample sizes. This experiment make use of 1 000 samples for each of the board sizes from 5x5-8x8, with 40% open postitions. The data is generated, represented, split and graphs are encoded similar to the experiment ”Initial Performance” and illustrated in Fig. 5.

The choice of $clauses = board\_size^2$ as a hypothesis-free exploration, guided more by curiosity than by justification, with the aim of observing whether any interesting or unexpected patterns might arise. In an attempt to isolate the impact of varying HVS and HVB, other parameter were kept constant or at a constant ratio (T). To clarify[13]: $s = 1.0$, $T = clauses \times 0.8$, $q = 1.0$ and $depth = 1$. Notice: The parameters MS and MB for the model is set equal to HVS and HVB used in generation of graphs since the effect of varying these are of the authors knowledge undocumented. In this section, HVS and MS, as well as HVB and MB, are used interchangeably to refer to hypervector size and message bits, respectively.

The selection of MS $= [128, 1024, 8192]$ and MB $= [2, 8, 16]$ was because they represent a subsample of the ones demonstrated by Halenka et al. [6]. The combination $128, 2$ might be considered redundant, but is included to verify similarity between the approach in this experiment and ”Initial Performance” as well as a reference point for the $clauses$. The remainder was selected to obtain outcomes approximating both the ”mid-range” and the ”upper-range”.

Each combination of parameters run 3 with 100 epochs each time. The metrics is calculated as the mean of each metric for 3 runs. See Appendix D for more details.

Observations

The experiment demonstrates how variations in MS and MB impact the GTM’s performance across different board sizes. Several key observations emerge from the results:

Scalability across board sizes: Performance across all configurations decreases as the board size increases, consistent with the trends observed in ”Initial Performance”. This decline highlights the complexity of larger board sizes and the need for more robust configurations to handle these scenarios effectively.

[13]”Hypervectors”. Parameters: Appendix C, Table XIV

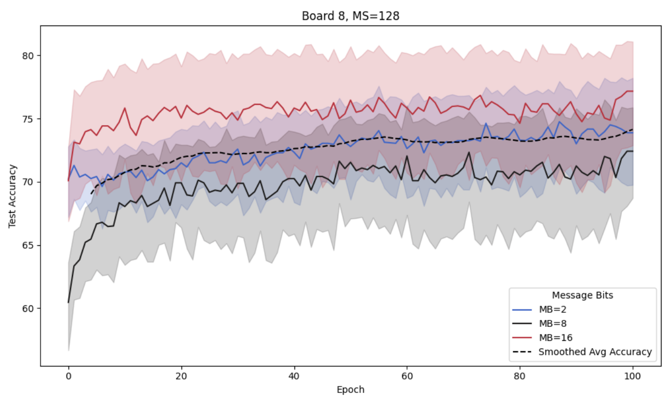


Fig. 10. Effect of MB on GTM on 8x8 board with MS=128

Effect of MS: The effect of hypervector size on performance varies with the board size and configuration. For MS values of 128, the test accuracy trends are more fluctuating across MB values on lower number of clauses, but more stable when number of clauses reach $board\_size^3$.

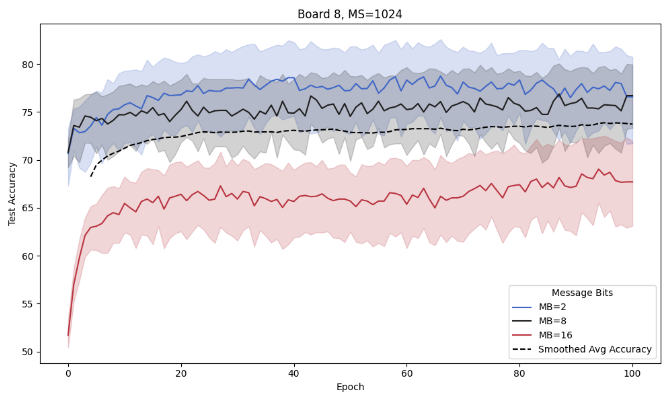


Fig. 11. Effect of MB on GTM on 8x8 board with MS=1024

Effect of MB: The number of MB has no clear impact on performance, however the effect is not uniform across MS. For MS=128, MB=16 performs better than other configurations, as seen in the test accuracy trends in Fig 10. For MS=1024, the accuracy trends show that MB=2 has slightly better performance than the others (Fig. 11). This suggests that the optimal MB depends on the chosen MS, with smaller MB values being more effective as MS increases. But again, the tables in Appendix D demonstrate too much variation to make definitive conclusions.

Interaction Between MS and MB: The interaction between MS and MB is evident in the smoothed summary plots (Appendix D, Fig. 18). The heatmap in Fig. 12 highlights an interaction effect between MS and MB. The combination of MS = 128 and MB = 16 achieves the best performance on average across configurations. However, for higher hypervector size like MS = 8192, MB = 2 provides more consistent and higher accuracy. This result align with the work of Halenka et al.[6].

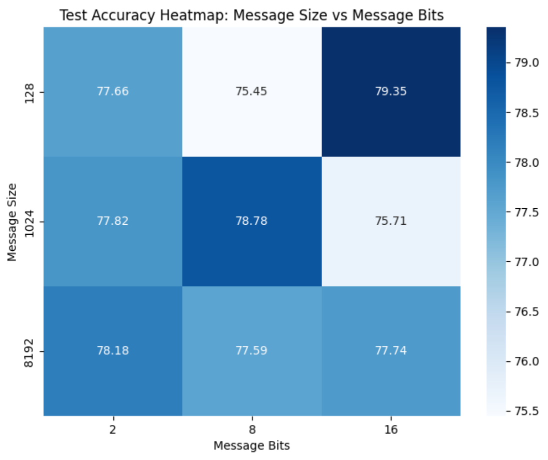


Fig. 12. Heatmap showing the interaction between MS and MB in experiment setup.

Future Directions

Effect with tuned parameters: This experiment should be repeated with other configurations for s, T, $depth$, $state\_bits$, and with $double\_hashing = True$ to find a better understanding for the space of which MS and MB can efficiently influence model performance.

In summary, the current setup it don't show that larger MS and MB significantly compensate for the reduction in number of clauses with the unoptimized setup, but the experiment highlights that the choice of MS and MB influence the GTM's performance and thereby setting the stage for further investigastions into how these parameters interact with other hyperparameters to enhance GTM scalability and accuracy in more complex scenarios.

I. *Standard Result*

This section shows some plots of the performance of the 11x11 with 40% empty cells and the hyperparameters in XVII. The graphs were build with temperature encoding and without anchor edge nodes. In figure 13 it is clearly displayed that the model learns over the epochs and that the train accuracy outperforms the test set. In the confusion matrix in figure 14 one can see that there is an even label distribution and that a there are significantly more of false positives than false negatives.

J. *Interpretability*

The Tsetlin machine is known to be interpretable. This however does not mean one can simply look at a GTM and determine how it works. We have done some experiments to determine how it operates.

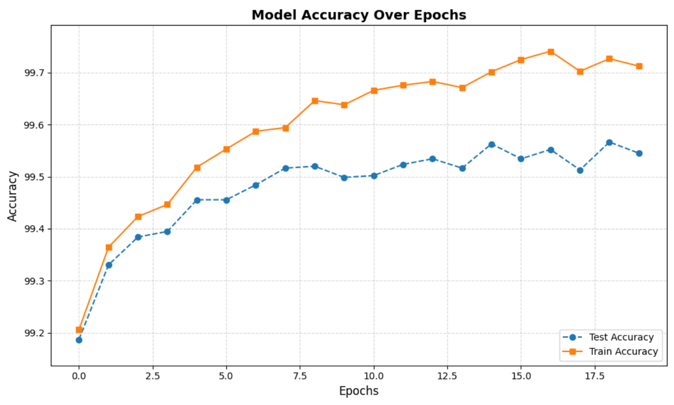


Fig. 13. Accuracy over epoch on a standard implementation on a 11x11 board

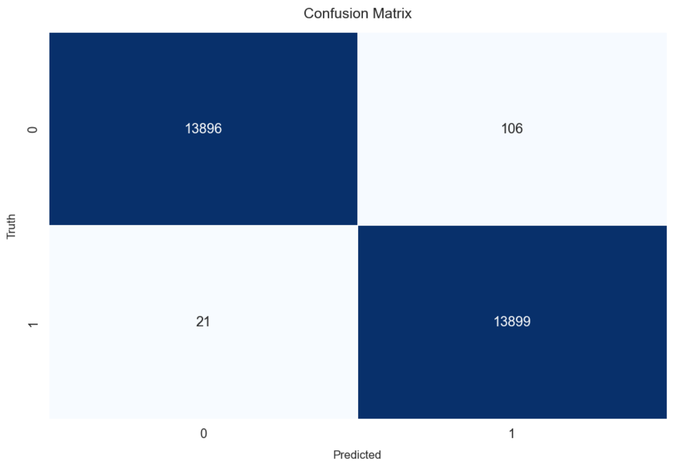


Fig. 14. Confusion matrix of a standard implementation on a 11x11 board

Our experiments focuses on global interpretability, meaning understanding how the model works in general. We have used the code from the example NoisyXOR_print_clauses.py[26] to print the clauses of a model we trained. Reusing the example code requires us to use 2 as the depth parameter, and we have used 3500 clauses to achieve good results with minimal clauses. Our code print_clauses.py writes all clauses and messages to a .txt file. Examples of clauses can be seen in Fig. 19. We've performed the interpretability tests on the same dataset of games on an 11x11 board as used in testing graph representation variants and number of clauses.

This file explains how the model operates, but a 4.8 gigabyte txt file is difficult for a human to understand by itself. We have done some further analysis of the file with a focus on the clauses. Our analysis counts literals in clauses and considers that a proxy for how important the literal is. This counting method does not incorporate the weight of the clauses in its approximation of literal importance.

Counting the literals in the clauses gives some insight in what literals the model deems most important. The 10 most important literals according to this experiment listed in order of importance: COL_4, RED, BLUE, COL_7, ¬ CONNECTED_BLUE, COL_3, ROW_1, COL_6, ¬ROW_9, ¬ COL_9 and ¬ CONNECTED_RED. This shows that the column 4 is particularly important to the GTM. RED and BLUE are the second and third most common literals, showing that whether there is a piece played in the cells is also important. The GTM also performs reasoning by negation on the connected attributes, which means that it has learned the absence of this attribute is significant.

Combinations of rows and columns indicates what locations are most important. In this experiment we have plotted combinations of positive position literals and combinations of negative literals. It is important to note that since the games are played completely randomly, this does not give insight in to what are the most important cells on the board, but which the model deems most important. Combinations of positive location literals indicates clauses which may only be true in that location, as show in Fig. 15. Combinations of negative literals indicate clauses that can not be true in that row or column, as shown in Fig. 16.

V. Conclusion

Board Size: The results show that a higher board size affects each of the 3 tasks differently. Predicting the winner at end state and 2 moves before achieves the highest accuracy on smaller board sizes, whereas for the task of predicting 5 moves before, a board size of 11x11 achieves the highest accuracy, this is likely due to the effects of random play, in that when playing random moves on a larger board, it's less likely that both players have winnable positions 5 moves prior to the end.

Number of Clauses: The results show that the number of clauses is an important hyperparameter, and that a higher parameter count does not necessarily mean better results. The results also shows that for models below the size of 10 000 clauses, training time is more impacted by model accuracy than the model size, as feedback when the model misclassifies datapoints takes more time than feedback when the model is correct.

Open Board Positions: The results demonstrate that the number of open positions has a significant impact on the performance of the GTM, with more open positions being easier for the model to classify. Having fewer open positions on the board effectively adds more nodes which could be relevant, but are likely not, which further makes it more difficult for the model to interpret the board.

Graph Representation: The results from the experiments testing various graph representations indicate that the changes to representation which improves accuracy when two moves are missing can have an adverse impact on accuracy when applied to classification of boards five moves before the end, which demonstrates that different stages of the board have different optimal representations. When doing classification for only one type of board state, one could therefore try to search for a single ideal board representation, and one might

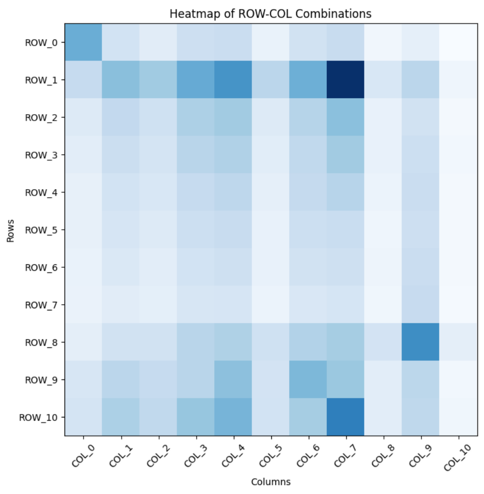


Fig. 15. The locations most rules affect

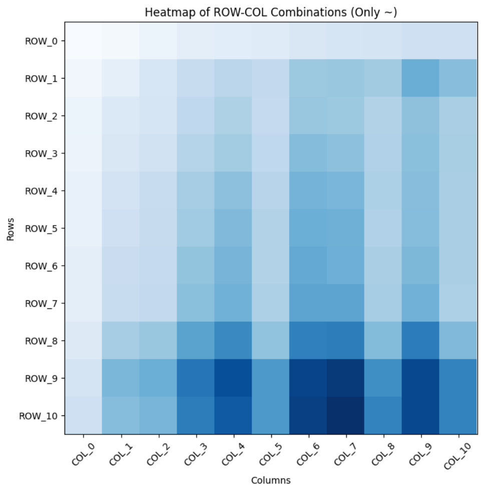


Fig. 16. The locations most rules do not affect

see improvements when deviating from what worked best for other board states. However, if one were to train a GTM to have a general understanding of Hex at all stages of the game, one might instead have to try to find the representation which is the best performing compromise, even if the accuracy for each board state is lower than that of a GTM trained on a singular type of board state.

Number of Literals: The results suggest the maximum number of literals is an important hyperparameter, which cannot be too small or too large without significantly hurting performance.

Hypervector Size: The results suggests the optimal hypervector size is dependent on the hypervector bits and vice versa. This suggests that for each problem space, the GTM faces a number of combinations of HVS and HVB should be tested.

Potential of GTM beyond the game of Hex: The potential of the GTM extends beyond the game of Hex. The introduction of depth significantly broadens the applicability of the TA approach across diverse domains. By leveraging message passing, the GTM can now explore multidimensional structures, complex connections, and intricate compositions that were previously less accessible or inaccessible. This enhancement positions GTM to compete more effectively with black-box architectures in a wider range of tasks, while retaining its key advantage: interpretability. With these improvements, GTM offers a promising avenue for developing interpretable AI solutions in domains requiring logical reasoning and structured data analysis.

Limitations

Time and computational resources: The experiments point to improved performance with combinations of larger HV's, more samples, deeper message passing, more epochs. In order to reach the deadline of this project, computational time consumption has affected the choice of parameters (e.g lower sample sizes, depth, etc.) which limits the exploration of areas that the experiments indicate will yield higher performance.

The GTM is a very recent algorithm. There are no design principles for the GTM known to the authors. This has made the design of the graphs have been very experimental and there are likely undiscovered methods to achieve better results.

A limitation of this work is that the used games which were created with random play, especially for the challenge of predicting the winner 2 or 5 moves before completion, This method is limited as one ideally would assume optimal play when predicting which player has the best board.

For the experiments done in this report, the average results and the standard deviation of 5 runs of each experiment are reported, This is done to provide a more accurate measure of the approach performance and reduce uncertainty. However there are some simplifications done to save time. The graphs are encoded one time for all the runs, there may be some performance variation between the encoding. This means also that for each experiment there are only tested one test split variation, and an average of multiple test splits may give a more representative result.

Further work

The introduction of GTM provides a wide array of opportunities for further exploration. Future research can delve into various aspects and applications to increase the potency and expand the application of this method.

A more extensive exploration of the findings from the work of Tarasyuk et al. [9], with GTM for games of Hex, might provide a deeper insight into parameter spaces for various contexts.

Regarding the specific game of Hex, work with intelligently created datasets as done in [4], is needed to further investigate games which is not at its end state. As it is necessary to determine which player has the best position given optimal play.

There is also room for further refining the methods for interpreting the model. In terms of global interpretability, this report does not investigate the creation of interpretable explanations of the nested clauses used by the model, and in terms of local interpretability this report does not investigate the creation of an understandable explanation as to why a specific board was given its classfication by the model.

To generate more realistic data, self-play for the TM, where multiple TM's compete against each other in the game of Hex, was briefly explored. This initial implementation employed Unity for simulation and basic automaton operations. Even with this simplistic setup, both "players" demonstrated improved, albeit basic, strategies. Such an environment holds potential for addressing dataset generation, strategy exploration, learning, and reasoning, providing a strong foundation for advancing in-game strategy development and reasoning capabilities in future work.

Aknowledgements

A. Code and Dataset Availability

The datasets, code and Optuna studies used in this project are available upon reasonable request from the corresponding authors, but requests must be made within 2-3 months after the delivery of this work. The code is available at https://github.com/Jon-Ingvar-Skanoy/hex/blob/delivery.

B. Use of AI tools

In this assignment ChatGPT 4o[27] and GitHub Copilot[28] was used to write and debug code, in particular code for visualizations like plots and LaTeX tables. For the writing of this report ChatGPT was used to write BibTeX, format latex tables, proofreading and help write the abstract.

To understand how to program with the GTM we have used the examples provided on GitHub[7] as information and as a template.

References

[1] Cameron Browne. *Hex Strategy: Making the right connections*. AK Peters/CRC Press, 2009.

[2] Ole-Christoffer Granmo. *The Tsetlin Machine – A Game Theoretic Bandit Driven Approach to Optimal Pattern Recognition with Propositional Logic*. 2021. arXiv: 1804.01508 [cs.AI]. URL: https://arxiv.org/abs/1804.01508.

[3] Kuruge Darshana Abeyrathna et al. "Massively parallel and asynchronous tsetlin machine architecture supporting almost constant-time scaling". In: *International Conference on Machine Learning*. PMLR. 2021, pp. 10–20.

[4] Charul Giri et al. *Logic-based AI for Interpretable Board Game Winner Prediction with Tsetlin Machine*. 2022. arXiv: 2203.04378 [cs.AI]. URL: https://arxiv.org/abs/2203.04378.

[5] Ole-Christoffer Granmo. "The Tsetlin Machine: A New Paradigm for Pattern Recognition". In: *The Tsetlin Machine: Fundamentals and Applications*. AI Publisher, 2022. Chap. 2. URL: https://tsetlinmachine.org/wp-content/uploads/2022/11/Tsetlin_Machine_Book_Chapter_2.pdf.

[6] Vojtech Halenka et al. *Exploring Effects of Hyperdimensional Vectors for Tsetlin Machines*. 2024. arXiv: 2406.02648 [cs.LG]. URL: https://arxiv.org/abs/2406.02648.

[7] Centre for AI Research (CAIR). *GraphTsetlinMachine*. Accessed: 2024-12-14. URL: https://github.com/cair/GraphTsetlinMachine.

[8] Yann LeCun, Corinna Cortes, and Christopher Burges. *MNIST Database of Handwritten Digits*. http://yann.lecun.com/exdb/mnist/. 1998. URL: http://yann.lecun.com/exdb/mnist/.

[9] Olga Tarasyuk et al. "Systematic Search for Optimal Hyper-parameters of the Tsetlin Machine on MNIST Dataset". In: *2023 International Symposium on the Tsetlin Machine (ISTM)* (2023), pp. 1–8. URL: https://api.semanticscholar.org/CorpusID:268245542.

[10] David Silver et al. "Mastering the game of Go with deep neural networks and tree search". In: *Nature* 529.7587 (2016), pp. 484–489.

[11] Vladimir Anshelevich. "The game of Hex: An automatic theorem proving approach to game programming". In: *AAAI/IAAI*. 2000, pp. 189–194.

[12] Broderick Arneson, Ryan B Hayward, and Philip Henderson. "MoHex: A Monte Carlo Hex program". In: *IJCAI*. 2009, pp. 803–808.

[13] Simin Huang, Broderick Arneson, and Ryan B Hayward. "MoHex 2.0: A pattern-based MCTS Hex player". In: *ICGA Journal* 35.1 (2013), pp. 3–11.

[14] Chris Young and Martin Müller. "NeuroHex: A deep Q-learning Hex agent". In: *arXiv preprint arXiv:1604.07009* (2016).

[15] Ryan Hayward. "Mohex wins hex tournament". In: *ICGA journal* 35 (June 2012), pp. 124–127. DOI: 10.3233/ICG-2012-35212.

[16] Dheeru Dua and Casey Graff. *Iris Dataset*. UCI Machine Learning Repository. 2017. URL: https://archive.ics.uci.edu/ml/datasets/Iris.

[17] Dheeru Dua and Casey Graff. *Optical Recognition of Handwritten Digits Dataset*. UCI Machine Learning Repository. 2017. URL: https://archive.ics.uci.edu/ml/datasets/Optical+Recognition+of+Handwritten+Digits.

[18] Andrew L. Maas et al. *Large Movie Review Dataset*. Stanford AI Lab. 2011. URL: https://ai.stanford.edu/~amaas/data/sentiment/.

[19] Ellen Voorhees and Dawn Tice. *TREC Question Classification Dataset*. Text Retrieval Conference. 1999. URL: https://cogcomp.seas.upenn.edu/Data/QA/QC/.

[20] Mona Gohari et al. "TPFL: Tsetlin Personalized Federated Learning". In: *arXiv preprint arXiv:2407.09162* (2024).

[21] Han Xiao, Kashif Rasul, and Roland Vollgraf. *Fashion-MNIST: A Novel Image Dataset for Benchmark-*

ing Machine Learning Algorithms. arXiv preprint arXiv:1708.07747. 2017. URL: https://github.com/zalandoresearch/fashion-mnist.

[22] Jon Andreas Bull Larssen, Jon Ingvar Jonassen Skånøy, and Isak Killingrød. *simhex*. https://github.com/Jon-Bull/simhex/tree/main. Accessed: 2024-12-14.

[23] Vojtech Halenka. *Helper functions for graph visualization*. Retrieved from Discord on December 17, 2024. 2024.

[24] Takuya Akiba et al. *Optuna: A Next-generation Hyperparameter Optimization Framework*. 2019. arXiv: 1907.10902 [cs.LG]. URL: https://arxiv.org/abs/1907.10902.

[25] Ahmed K. Kadhim et al. *Exploring State Space and Reasoning by Elimination in Tsetlin Machines*. 2024. arXiv: 2407.09162 [cs.LG]. URL: https://arxiv.org/abs/2407.09162.

[26] Centre for Artificial Intelligence Research (CAIR). *NoisyXOR_print_clauses.py - GraphTsetlinMachine Examples*. Accessed: 2024-12-18. 2024. URL: https://github.com/cair/GraphTsetlinMachine/blob/master/examples/NoisyXOR_print_clauses.py.

[27] OpenAI. *ChatGPT 4.0*. Accessed: 2024-12-06. 2024. URL: https://openai.com/chatgpt.

[28] GitHub. *GitHub Copilot*. https://github.com/features/copilot. Accessed: 2024-12-06. 2024.

[29] Centre for AI Research $CAIR$. *tm.py from GraphTsetlinMachine*. Commit f023cb0, Accessed: 2024-12-14. URL: https://github.com/cair/GraphTsetlinMachine/blob/f023cb0d890d74463947c65fd0422c6c53c52e24/GraphTsetlinMachine/tm.py.

Appendix A
Hardware Setup

Component	Details
Shared Server	
OS	Ubuntu 22.04.4 LTS
CPU	Intel Xeon Platinum 8168 @ 2.70GHz
RAM	1.48TB
Disk	1TB
GPU	Tesla V100-SXM3-32GB
Local Machine 1	
OS	Windows 11, WSL2 (Ubuntu 22.04.4)
CPU	Intel Core i5-9600K @ 3.70GHz
RAM	32GB
Disk	480GB SSD + 22TB HDD
GPU0	NVIDIA GeForce RTX 3080 10GB
GPU1	NVIDIA GeForce RTX 3070 8GB
Local Machine 2	
OS	Windows 11, WSL2 (Ubuntu 22.04.4)
CPU	Intel Core i7-10700 @ 2.90GHz
RAM	16GB
Disk	1TB SSD + 7TB HDD
GPU0	NVIDIA GeForce RTX 3070 8GB

TABLE XI. Hardware Setup

Appendix B
Default parameters

Parameter	Value
Graphs	
number_of_graphs	800[14]
hypervector_size	128
hypervector_bits	2
double_hashing	False
symbols	['RED', 'BLUE', 'UP', 'DOWN', 'RIGHT','LEFT']
MultiClassGTM	
number_of_clauses	1000
T	800[15]
s	1
q	1
max_included_literals	None[16]
boost_true_positive_feedback	1
state_bits	8
depth	1
message_size	128[17]
message_bits	2 [18]
double_hashing	False
grid	(16*13,1,1)
block	(128,1,1)

TABLE XII. Default parameters GTM (Initial Performance Experiment)

[14]number_of_graphs is set to 1 graph per training sample
[15]T = number_of_clauses $\times$ 0.8 [4]
[16]"None" means max_included_literals = hypervector_size $\times$ 2 [29]
[17]message_size is set to the same as hypervector_size
[18]message_bits is set to the same as hypervector_bits

Appendix C Hyperparameters

Parameter	Value
number_of_clauses	3500
T	5590.0
s	1
message_size	64
message_bits	3
epochs	20
state_bits	8
max_literals	20
hypervector_bits	2
hypervector_size	128

TABLE XIII. Parameters and their values (depth experiment).

Parameter	Value
number_of_clauses	$board\_size^2$
T	$clauses \times 0.8$
s	1
q	1
depth	1
message_size	x
message_bits	x
epochs	100
state_bits	8
max_literals	None
hypervector_bits	x
hypervector_size	x

TABLE XIV. Parameters and their values (Hypervector experiment).

Parameter	Value
number_of_clauses	3500
T	5590.0
s	1
message_size	64
message_bits	3
Depth	1
epochs	20
state_bits	8
max_literals	20
hypervector_bits	2
hypervector_size	128

TABLE XV. Parameters and their values (Max literals experiment).

Parameter	Value
number_of_clauses	x
T	number_of_clauses*1.597
s	1
message_size	64
message_bits	3
Depth	1
epochs	20
state_bits	8
max_literals	20
hypervector_bits	2
hypervector_size	128

TABLE XVI. Parameters and their values (number of clauses experiment).

Parameter	Value
number_of_clauses	10 000
T	15970
s	1
message_size	64
message_bits	3
Depth	1
epochs	20
state_bits	8
max_literals	20
hypervector_bits	2
hypervector_size	128

TABLE XVII. Parameters and their values.

APPENDIX D HYPERVECTORS

	Hypervectors 5x5 [19]					
Size	Clauses	HV Size	HV Bits	Train Acc	Test Acc	F1
5	25	128	2	60.88 (9.10)	60.67 (9.41)	0.69 (0.04)
			8	56.21 (0.81)	57.83 (2.93)	0.26 (0.11)
			16	63.83 (4.39)	64.83 (7.02)	0.59 (0.16)
		1024	2	65.04 (2.63)	62.83 (5.35)	0.59 (0.11)
			8	69.75 (3.60)	66.83 (2.75)	0.70 (0.03)
			16	67.29 (2.30)	64.00 (4.77)	0.55 (0.11)
		8192	2	62.88 (4.82)	63.67 (5.25)	0.61 (0.08)
			8	57.37 (7.57)	56.33 (5.53)	0.68 (0.02)
			16	65.29 (8.69)	60.50 (10.50)	0.47 (0.26)
	125	128	2	83.79 (2.20)	82.50 (1.80)	0.83 (0.01)
			8	69.50 (9.54)	66.17 (8.25)	0.47 (0.21)
			16	84.04 (1.46)	83.67 (2.52)	0.85 (0.02)
		1024	2	82.33 (1.13)	78.83 (2.84)	0.79 (0.02)
			8	82.67 (4.20)	80.00 (5.50)	0.76 (0.08)
			16	72.71 (3.00)	68.33 (3.21)	0.58 (0.07)
		8192	2	82.92 (2.13)	81.33 (3.33)	0.82 (0.03)
			8	80.63 (7.43)	78.17 (7.08)	0.81 (0.04)
			16	81.50 (1.98)	80.67 (2.47)	0.80 (0.03)
	625	128	2	96.46 (0.56)	92.33 (1.89)	0.92 (0.02)
			8	95.92 (1.27)	92.33 (1.76)	0.92 (0.02)
			16	96.79 (1.37)	94.33 (2.08)	0.94 (0.02)
		1024	2	98.00 (1.02)	95.33 (0.29)	0.96 (0.01)
			8	95.42 (1.66)	90.67 (1.89)	0.90 (0.02)
			16	97.12 (0.13)	89.67 (0.29)	0.89 (0.01)
		8192	2	97.42 (1.37)	92.50 (3.12)	0.93 (0.03)
			8	95.71 (3.64)	93.33 (1.04)	0.93 (0.01)
			16	97.96 (0.74)	94.50 (1.00)	0.95 (0.01)
	3125	128	2	99.46 (0.14)	96.67 (1.04)	0.96 (0.01)
			8	97.79 (1.02)	92.50 (0.87)	0.92 (0.01)
			16	99.88 (0.13)	99.00 (0.87)	0.99 (0.01)
		1024	2	99.84 (0.08)	98.33 (0.29)	0.99 (0.01)
			8	99.58 (0.07)	96.00 (0.00)	0.96 (0.00)
			16	99.62 (0.00)	93.83 (0.29)	0.94 (0.00)
		8192	2	99.58 (0.19)	96.17 (0.29)	0.96 (0.01)
			8	99.84 (0.08)	97.33 (0.58)	0.97 (0.01)
			16	99.79 (0.15)	97.00 (0.00)	0.97 (0.00)

TABLE XVIII. MEAN (STD) OF METRICS FOR EXPERIMENT: HYPERVECTORS ON 5X5 BOARD.

	Hypervectors 6x6 [20]					
Size	Clauses	HV Size	HV Bits	Train Acc	Test Acc	F1
6	36	128	2	65.83 (2.31)	65.17 (2.47)	0.69 (0.05)
			8	60.62 (4.06)	54.33 (3.06)	0.36 (0.15)
			16	66.67 (2.60)	66.17 (4.01)	0.66 (0.06)
		1024	2	62.00 (5.86)	58.17 (5.80)	0.65 (0.02)
			8	55.17 (1.50)	53.50 (2.78)	0.25 (0.08)
			16	62.13 (6.09)	65.67 (3.79)	0.50 (0.15)
		8192	2	64.38 (2.19)	56.50 (2.65)	0.51 (0.04)
			8	65.96 (3.00)	63.83 (4.62)	0.55 (0.08)
			16	64.00 (3.40)	58.50 (1.80)	0.42 (0.07)
	216	128	2	79.46 (7.32)	73.50 (3.91)	0.75 (0.02)
			8	82.67 (0.40)	75.67 (2.75)	0.76 (0.02)
			16	83.75 (0.87)	77.00 (3.04)	0.75 (0.04)
		1024	2	81.96 (2.14)	75.83 (5.69)	0.76 (0.03)
			8	75.67 (8.91)	74.17 (9.09)	0.68 (0.17)
			16	80.87 (4.39)	76.17 (4.54)	0.69 (0.10)
		8192	2	85.17 (2.04)	81.00 (1.80)	0.81 (0.02)
			8	79.75 (4.80)	75.50 (3.50)	0.69 (0.07)
			16	83.58 (5.06)	76.50 (4.44)	0.74 (0.07)
	1296	128	2	95.33 (1.18)	86.83 (1.89)	0.87 (0.03)
			8	95.96 (1.32)	87.33 (3.51)	0.88 (0.04)
			16	95.75 (0.76)	89.33 (2.25)	0.88 (0.03)
		1024	2	96.46 (1.22)	88.00 (0.87)	0.87 (0.01)
			8	95.88 (1.13)	89.00 (1.50)	0.89 (0.02)
			16	97.50 (0.69)	87.83 (2.08)	0.86 (0.02)
		8192	2	96.08 (0.63)	90.50 (0.87)	0.91 (0.01)
			8	92.17 (3.24)	86.17 (3.06)	0.84 (0.05)
			16	95.04 (0.19)	85.17 (2.93)	0.85 (0.03)
	7776	128	2	98.75 (0.22)	90.17 (2.02)	0.90 (0.02)
			8	99.50 (0.45)	89.83 (0.58)	0.90 (0.01)
			16	99.13 (0.55)	92.67 (1.15)	0.92 (0.02)
		1024	2	98.88 (0.57)	92.67 (0.29)	0.92 (0.01)
			8	98.58 (0.47)	91.17 (0.29)	0.91 (0.00)
			16	99.54 (0.26)	90.83 (0.76)	0.90 (0.01)
		8192	2	97.17 (0.68)	93.00 (1.32)	0.93 (0.01)
			8	96.46 (0.96)	87.33 (0.76)	0.85 (0.01)
			16	98.63 (0.33)	91.17 (1.15)	0.92 (0.01)

TABLE XIX. MEAN (STD) OF METRICS FOR EXPERIMENT: HYPERVECTORS ON 6X6 BOARD.

[19]Exp. ”Hypervectors”: Machine 1, sequential, 3 runs x 100 epochs

[20]Exp. ”Hypervectors”: Machine 1, sequential, 3 runs x 100 epochs
[21]Exp. ”Hypervectors”: Machine 1, sequential, 3 runs x 100 epochs
[22]Exp. ”Hypervectors”: Machine 1, sequential, 3 runs x 100 epochs

	Hypervectors 7x7 [21]					
Size	Clauses	HV Size	HV Bits	Train Acc	Test Acc	F1
7	49	128	2	65.50 (1.89)	61.67 (1.53)	0.56 (0.05)
			8	66.33 (2.02)	58.50 (2.60)	0.63 (0.04)
			16	64.29 (2.36)	60.67 (5.25)	0.68 (0.02)
		1024	2	56.88 (2.08)	55.67 (1.76)	0.70 (0.02)
			8	63.38 (5.05)	63.17 (1.61)	0.72 (0.01)
			16	57.37 (4.44)	60.67 (2.02)	0.39 (0.11)
		8192	2	65.08 (2.52)	67.00 (2.00)	0.70 (0.01)
			8	67.46 (1.61)	64.00 (2.00)	0.65 (0.06)
			16	67.25 (2.38)	63.50 (2.78)	0.66 (0.03)
	343	128	2	81.25 (2.13)	72.83 (4.16)	0.72 (0.07)
			8	84.37 (2.83)	71.33 (0.76)	0.75 (0.02)
			16	80.54 (1.92)	68.33 (2.08)	0.72 (0.02)
		1024	2	79.17 (2.00)	71.33 (5.01)	0.76 (0.04)
			8	83.12 (1.35)	76.33 (4.48)	0.79 (0.02)
			16	81.42 (3.17)	73.00 (1.32)	0.69 (0.03)
		8192	2	81.17 (4.14)	77.50 (6.14)	0.79 (0.03)
			8	82.54 (2.19)	75.83 (1.26)	0.77 (0.02)
			16	84.83 (2.45)	74.67 (3.33)	0.74 (0.05)
	2401	128	2	95.25 (1.93)	80.67 (1.15)	0.81 (0.01)
			8	96.46 (1.35)	84.83 (1.61)	0.87 (0.02)
			16	96.67 (1.06)	78.00 (3.12)	0.79 (0.02)
		1024	2	96.13 (1.52)	78.50 (2.65)	0.81 (0.02)
			8	96.29 (0.71)	85.17 (2.93)	0.86 (0.03)
			16	97.00 (0.45)	84.00 (0.87)	0.83 (0.01)
		8192	2	96.25 (2.22)	85.33 (2.36)	0.86 (0.02)
			8	97.34 (0.51)	81.50 (2.00)	0.82 (0.02)
			16	97.17 (0.84)	85.17 (2.25)	0.85 (0.02)
	16807	128	2	98.04 (0.81)	84.00 (2.18)	0.84 (0.03)
			8	98.79 (0.61)	86.83 (0.29)	0.89 (0.01)
			16	99.88 (0.00)	87.00 (1.00)	0.87 (0.01)
		1024	2	99.46 (0.19)	83.33 (0.29)	0.85 (0.00)
			8	99.00 (0.12)	87.67 (2.25)	0.89 (0.02)
			16	98.92 (0.58)	86.50 (1.32)	0.85 (0.02)
		8192	2	99.25 (0.25)	89.50 (0.50)	0.90 (0.01)
			8	99.66 (0.08)	85.00 (1.80)	0.84 (0.02)
			16	98.75 (0.37)	87.17 (0.76)	0.86 (0.01)

TABLE XX. MEAN (STD) OF METRICS FOR EXPERIMENT: HYPERVECTORS ON 7X7 BOARD.

	Hypervectors 8x8 [22]					
Size	Clauses	HV Size	HV Bits	Train Acc	Test Acc	F1
8	64	128	2	64.83 (5.08)	62.17 (1.15)	0.51 (0.13)
			8	64.67 (2.02)	63.33 (5.58)	0.60 (0.13)
			16	69.67 (4.73)	67.17 (1.89)	0.69 (0.03)
		1024	2	67.21 (1.87)	65.17 (8.02)	0.69 (0.07)
			8	66.50 (2.84)	67.33 (1.53)	0.73 (0.02)
			16	60.17 (4.39)	55.83 (4.65)	0.33 (0.13)
		8192	2	56.37 (2.76)	55.33 (1.76)	0.69 (0.01)
			8	67.42 (2.44)	65.17 (5.30)	0.70 (0.05)
			16	64.96 (1.65)	59.17 (2.02)	0.50 (0.09)
	512	128	2	83.50 (6.21)	73.67 (1.53)	0.68 (0.03)
			8	85.25 (1.56)	70.17 (0.58)	0.70 (0.00)
			16	82.95 (3.55)	74.00 (4.58)	0.71 (0.07)
		1024	2	80.54 (0.73)	74.33 (1.61)	0.78 (0.01)
			8	88.21 (1.09)	76.33 (2.75)	0.78 (0.02)
			16	75.00 (8.04)	66.67 (3.25)	0.58 (0.07)
		8192	2	80.42 (8.18)	70.00 (3.91)	0.75 (0.03)
			8	86.54 (1.77)	72.50 (1.32)	0.73 (0.03)
			16	83.54 (2.82)	71.17 (1.89)	0.68 (0.04)
	4096	128	2	97.04 (0.19)	78.83 (2.36)	0.76 (0.03)
			8	96.83 (1.14)	78.17 (2.31)	0.79 (0.03)
			16	97.87 (0.54)	82.67 (1.44)	0.83 (0.01)
		1024	2	97.17 (0.52)	82.50 (2.18)	0.84 (0.02)
			8	97.54 (0.40)	80.17 (0.29)	0.82 (0.01)
			16	89.79 (0.83)	71.83 (0.76)	0.66 (0.01)
		8192	2	94.67 (3.08)	76.83 (0.29)	0.80 (0.01)
			8	97.04 (0.94)	81.83 (2.47)	0.81 (0.02)
			16	95.96 (2.29)	81.33 (1.89)	0.81 (0.02)
	32768	128	2	98.96 (0.19)	80.83 (1.89)	0.78 (0.02)
			8	98.63 (0.69)	78.00 (0.87)	0.78 (0.01)
			16	99.66 (0.08)	84.83 (0.29)	0.85 (0.00)
		1024	2	98.71 (0.40)	84.33 (0.76)	0.86 (0.01)
			8	99.08 (0.07)	83.00 (0.00)	0.84 (0.01)
			16	93.29 (0.38)	76.50 (1.32)	0.72 (0.02)
		8192	2	94.79 (1.12)	75.33 (1.26)	0.79 (0.01)
			8	98.29 (0.52)	83.83 (1.26)	0.83 (0.02)
			16	99.25 (0.13)	82.83 (1.44)	0.83 (0.02)

TABLE XXI. MEAN (STD) OF METRICS FOR EXPERIMENT: HYPERVECTOR ON 8X8 BOARD.

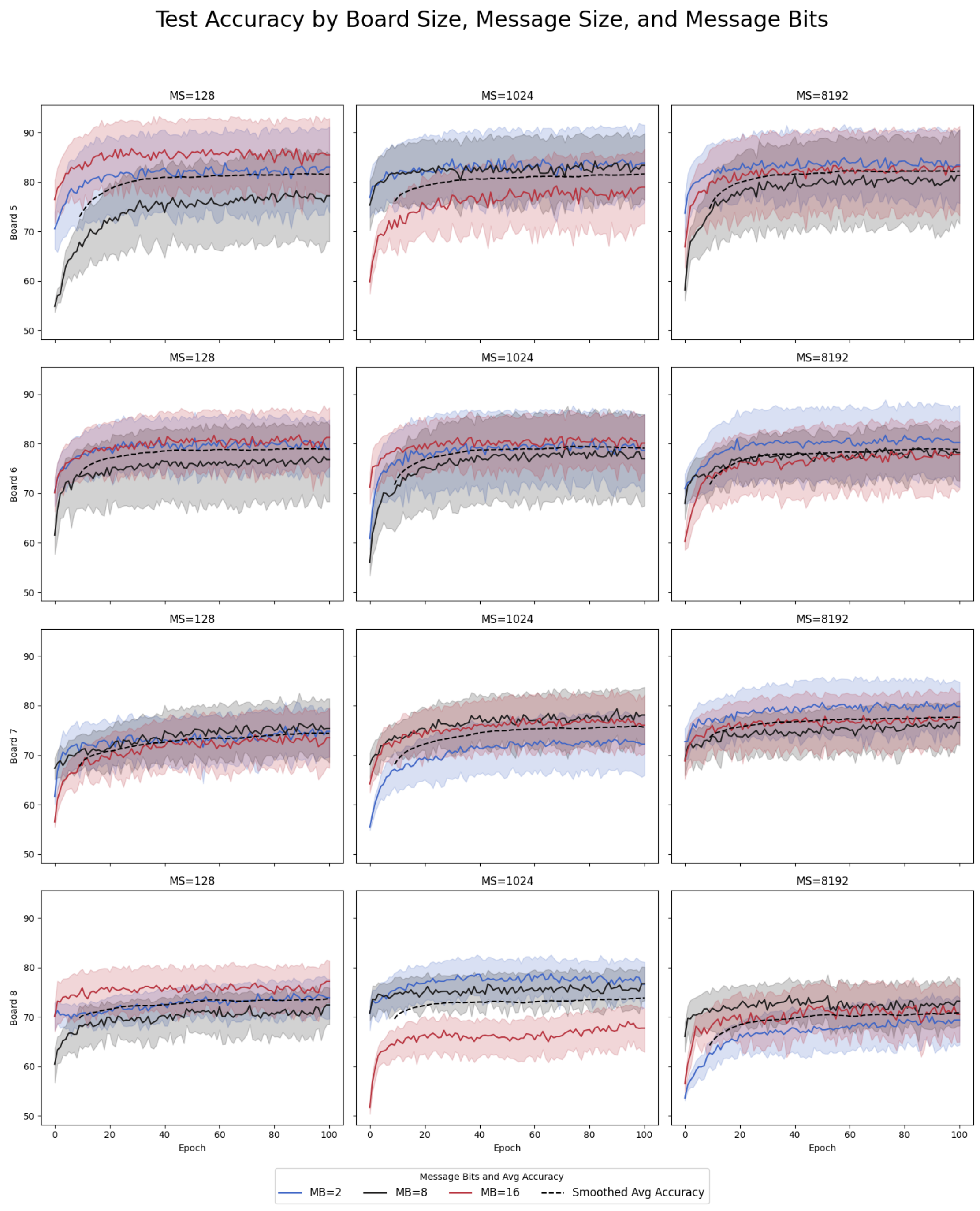


Fig. 17. Test Accuracy by board size, MS, MB: The performance of GTM across different board sizes (5x5 to 8x8) is analyzed by varying message sizes (128, 1024, and 8192) and message bits (2, 8, and 16). Each subplot illustrates the accuracy trends over 100 epochs, with shaded regions representing variance across trials.

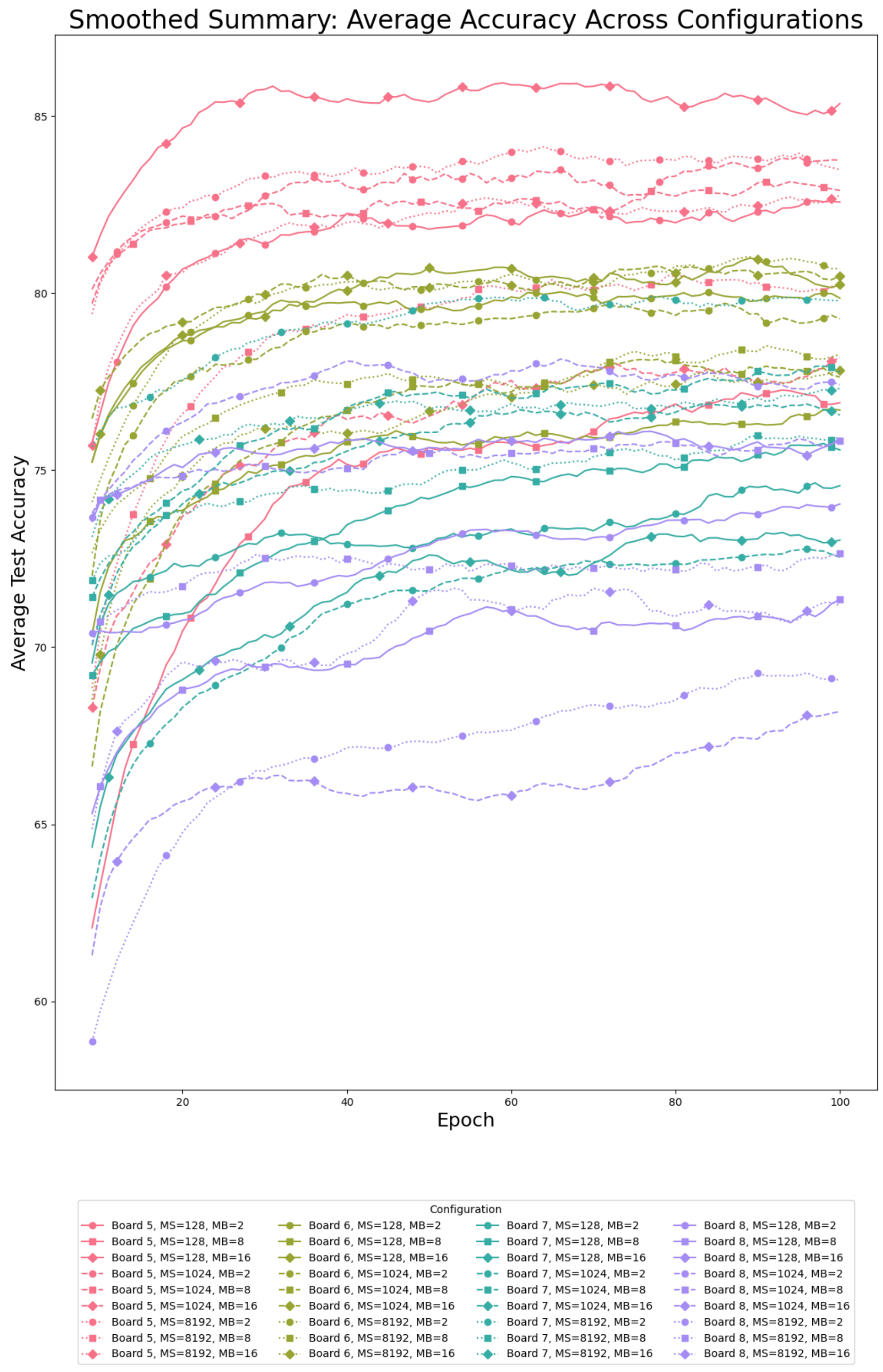
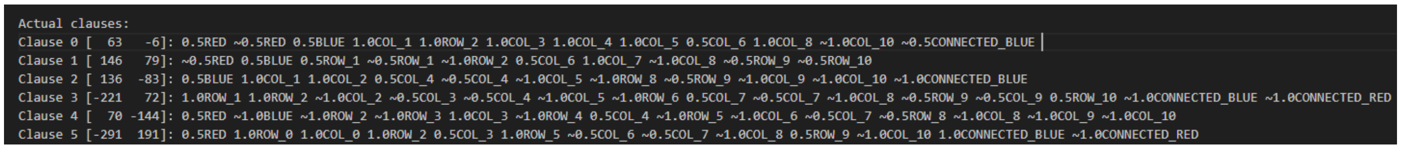


Fig. 18. Smoothed summary of average test accuracy across configurations: A consolidated view of test accuracy for varying configurations of board size, MS, and MB. The plot highlights the interaction effects of these parameters on performance, with each line corresponding to a specific configuration and smoothed to enhance clarity.

Appendix E
Clauses



```
Actual clauses:
Clause 0 [  63   -6]: 0.5RED ~0.5RED 0.5BLUE 1.0COL_1 1.0ROW_2 1.0COL_3 1.0COL_4 1.0COL_5 0.5COL_6 1.0COL_8 ~1.0COL_10 ~0.5CONNECTED_BLUE
Clause 1 [ 146   79]: ~0.5RED 0.5BLUE 0.5ROW_1 ~0.5ROW_1 ~1.0ROW_2 0.5COL_6 1.0COL_7 ~1.0COL_8 ~0.5ROW_9 ~0.5ROW_10
Clause 2 [ 136  -83]: 0.5BLUE 1.0COL_1 1.0COL_2 0.5COL_4 ~0.5COL_4 ~1.0COL_5 ~1.0ROW_8 ~0.5ROW_9 ~1.0COL_9 ~1.0COL_10 ~1.0CONNECTED_BLUE
Clause 3 [-221   72]: 1.0ROW_1 1.0ROW_2 ~1.0COL_2 ~0.5COL_3 ~0.5COL_4 ~1.0COL_5 ~1.0ROW_6 0.5COL_7 ~0.5COL_7 ~1.0COL_8 ~0.5ROW_9 ~0.5COL_9 0.5ROW_10 ~1.0CONNECTED_BLUE ~1.0CONNECTED_RED
Clause 4 [  70 -144]: 0.5RED ~1.0BLUE ~1.0ROW_2 ~1.0ROW_3 1.0COL_3 ~1.0ROW_4 0.5COL_4 ~1.0ROW_5 ~1.0COL_6 ~0.5COL_7 ~0.5ROW_8 ~1.0COL_8 ~1.0COL_9 ~1.0COL_10
Clause 5 [-291  191]: 0.5RED 1.0ROW_0 1.0COL_0 1.0ROW_2 0.5COL_3 1.0ROW_5 ~0.5COL_6 ~0.5COL_7 ~1.0COL_8 0.5ROW_9 ~1.0COL_10 1.0CONNECTED_BLUE ~1.0CONNECTED_RED
```

Fig. 19. Example of Clauses