MATADOR: Automated System-on-Chip Tsetlin Machine Design Generation for Edge Applications

Tousif Rahman*†, Gang Mao*†, Sidharth Maheshwari◇, Rishad Shafik*, Alex Yakovlev*
*Microsystems Research Group, Newcastle University, Newcastle upon Tyne, UK. ◇ IIT Jammu, Jammu, India
{s.rahman, g.mao2, rishad.shafik, alex.yakovlev}@newcastle.ac.uk, sidharth.maheshwari@iitjammu.ac.in
†Indicates equal contribution.

***Abstract*—System-on-Chip Field-Programmable Gate Arrays (SoC-FPGAs) offer significant throughput gains for machine learning (ML) edge inference applications via the design of co-processor accelerator systems. However, the design effort for training and translating ML models into SoC-FPGA solutions can be substantial and requires specialist knowledge aware trade-offs between model performance, power consumption, latency and resource utilization. Contrary to other ML algorithms, Tsetlin Machine (TM) performs classification by forming logic proposition between boolean actions from the Tsetlin Automata (the learning elements) and boolean input features. A trained TM model, usually, exhibits high sparsity and considerable overlapping of these logic propositions both within and among the classes. The model, thus, can be translated to RTL-level design using a miniscule number of AND and NOT gates. This paper presents MATADOR, an automated boolean-to-silicon tool with GUI interface capable of implementing optimized accelerator design of the TM model onto SoC-FPGA for inference at the edge. It offers automation of the full development pipeline: model training, system level design generation, design verification and deployment. It makes use of the logic sharing that ensues from propositional overlap and creates a compact design by effectively utilizing the TM model's sparsity. MATADOR accelerator designs are shown to be up to 13.4x faster, up to 7x more resource frugal and up to 2x more power efficient when compared to the state-of-the-art Quantized and Binary Deep Neural Network implementations.**

***Index Terms*—Tsetlin Machine, System-on-Chip, FPGA, Inference Accelerator, Machine Learning, Edge inference**

I. Introduction

Inference at the edge for Machine Learning (ML) classification tasks presents notable advantages of increased data security, reduced network-bandwidth and lower energy requirements in data movement when compared to cloud off-loading based approaches [1], [2]. The bottleneck, however, stems from the restrictive resources available at the edge [3], [4]. To deliver high throughput within the limited compute, memory and power budget, the ML algorithms employ techniques such as compression and quantization [3], [5], [6]. These software savings can then be further amplified into hardware accelerator solutions that leverage compute pipelining and parallelism for greater throughput and energy efficiency [3], [4].

Recent approaches with quantization of Deep Neural Network (DNN) models down to 1 or 2-bit weight and activation representation have demonstrated the possibility of using DNNs at the edge [1], [7]. These quantized models permit reduced memory usage enabling network parameters to remain on-chip, thus, avoiding data movement latency. With quantized weights and activation functions, the computation can be performed

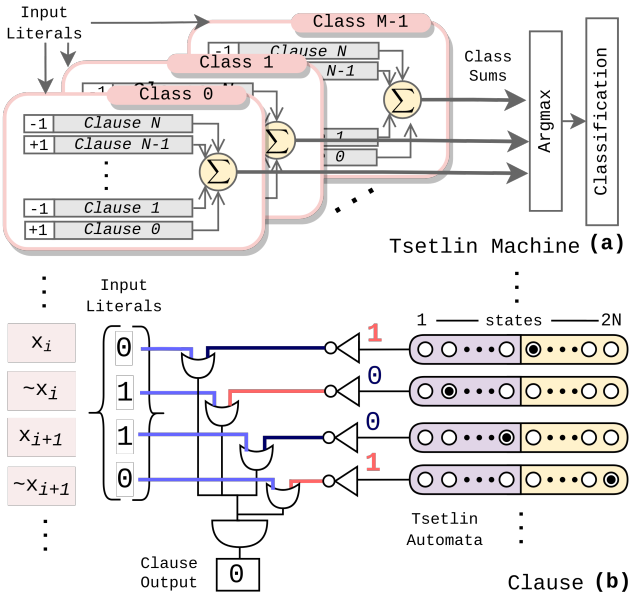

Fig. 1. Visualization of the Tsetlin Machine and its main components. (a) shows the clause voting mechanism that results in class sums and classification. (b) internals of the clause with the learning element Tsetlin Automata and propositional logic that knits it with input literals (equivalent of features).

using integer operations reducing the resource, computation time, and power requirements [1], [3]. There are, however, some drawbacks and limitations to accelerating and fully automating the offline training, design generation and edge deployment for DNN based designs. Firstly, there is an accuracy degradation as floating-point parameters are quantized to fixed-bit(s) representation. Secondly, even with quantization, considerable design space explorations of DNN architectures are required to find an optimum trade-off between performance, network parameters and energy efficiency [6]. This remains a key challenge as it requires substantial automation efforts in hardware-software co-design space [4] making automated ML hardware generation and integration into a system level solutions complex [4], [8].

This paper looks beyond DNNs to address the aforementioned challenges through exploring end-to-end automated hardware accelerator design of a recently proposed ML algorithm called the Tsetlin Machine (TM), as shown in Fig. 1. The TM classifies through voting mechanism of the clause outputs that are summed together in accordance with their polarity as shown in Fig 1(a), where the polarity changes alternatively between [+1,-1]. The clause is the ML compute engine of TM that consists of two crucial components: Tsetlin

979-8-3503-4859-0/DATE24/© 2024 EDAA

automata and propositional logic. Tsetlin automata is the main learning element that interprets the input data through changes in the state value while training, as shown in Fig 1(b). The final states obtained upon training are translated into boolean actions of `0` or `1` resulting in a long boolean sequence termed as the TM model. The propositional logic is formed through interlinking the input literals with their respective boolean actions to generate a clause output as shown in Fig 1(b). Here, each feature x_i will produce two literals: x_i and its inverse $\sim x_i$. For further details on the TM training and inference, the interested readers are encouraged to read [5], [9]. The intrinsic logic arithmetic and a limited design space exploration, with few hyperparameters [5], involved in TM qualifies it for edge translation using a System-on-Chip Field-Programmable Gate Array (SoC-FPGA).

This paper presents MATADOR, a boolean-to-silicon automation tool with a GUI interface that is capable of training Tsetlin Machines with user-guided hyperparameters, generating RTL-level design for the model and mapping it to SoC-FPGA architectures. MATADOR realizes the TM inference with Look-up-Table friendly `AND` and `NOT` operations emanating from the boolean actions of the automata used to create logic propositions, as shown in Fig 1(b). MATADOR can, essentially, translate the entire TM model to a single combinational circuit. However, due to the bandwidth constraint between the processor and the programmable logic, and to avoid timing violations, it automatically splits it into smaller units, depending on the number of input literals, each calculating a partial clause output.

MATADOR is the first to present a full system design automation approach, exploiting the algorithm-centric sparsity of Tsetlin Machine models. The fundamental design concept for the MATADOR accelerator takes inspiration from the streaming architectures presented in [3], [7]. They build custom architectures for Binary Neural Network (BNN) and Quantized Neural Network (QNN) based topologies, respectively, with dedicated compute engines for each layer (or group of layers as in [7]). In this paper, we demonstrate that MATADOR accelerator designs can be better in resource frugality and power efficiency in comparison to these state-of-the-art alternatives.

II. Boolean-to-Silicon

The Tsetlin machine [9] represents a significant departure from the floating-point arithmetic-based machine learning to a logic-based approach. At its core, it employs clauses containing learning automata, as finite-state machines, to acquire logical patterns in relation to the input dataset. These logical clauses form comprehensive descriptions of the learning problem. Consequently, the Tsetlin machine introduces the concept of logical interpretable learning, wherein both the learned model and the learning process are easily comprehensible and explainable [10]. As a result, it reduces the level of expertise required to efficiently apply machine learning techniques across various domains, in tandem, the parallelism in logic based inference can be further exploited in the hardware.

Recent developments with TM have explored applications such as Internet-of-Things [5], [11], [12] and estimating battery parameters in electric vehicles [13]. This has lead to a

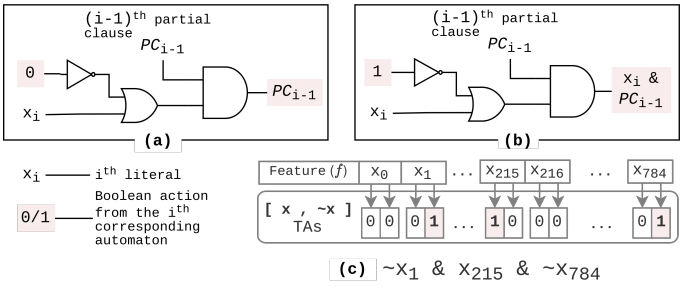

Fig. 2. Following Fig. 1, here we visualize the outcomes of partial clauses (`PC` via the propositional logic for different boolean actions of the automaton. (a,b) if boolean action is `0` then the corresponding literal can be excluded from clause computation on the other hand it needs to be included if the action is `1`. (c) the include-excludes result in boolean expression for the incoming input.

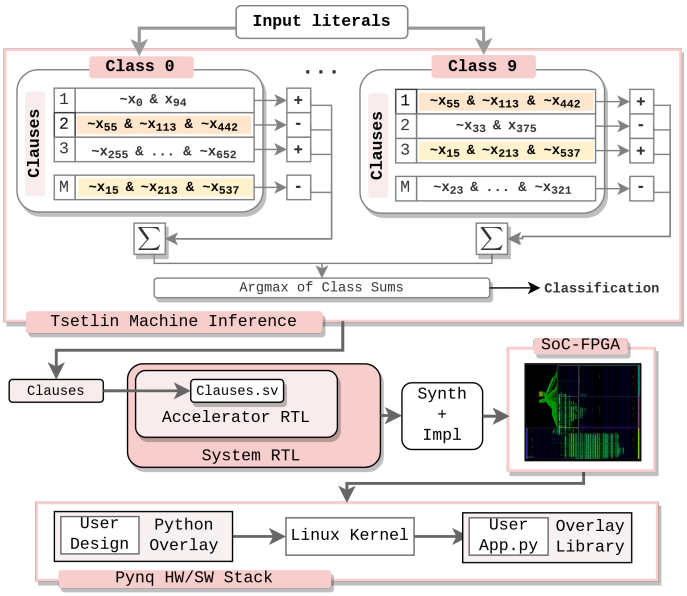

Fig. 3. Boolean-to-Silicon Overview: The boolean expression from the include-exclude decisions of the automata, are often found to be similar within various clauses, and classes, as highlighted in the Tsetlin Machine Inference box. Logic sharing can be used to shrink the combination circuit arising from the TM model. The clauses are the basis from which the SoC-FPGA system is built.

democratic development of novel TM architectures and training methods such as convolutional TM [14] with superior classification results, regression convolutional TM [15], and Coalesed TM [16] with small memory footprint while training. TM architecture and hyperparameter search paradigms are proposed in [5], [17], [18]. On-chip TM training needs to consider area, power, frequency and high-throughput of pseudo random numbers, being a stochastic process, as discussed in [19]–[21]. Energy-efficient and fast FPGA TM inference is proposed in [22], [23], however, they have used small datasets viz. 2D Noisy XOR and IRIS [24]. MATADOR extends inference with TMs to larger edge application datasets presented in Section V.

Fig. 2 showcases how a TM model can be translated into a combinational circuit. If the boolean action of the automaton is `0` then the corresponding literal will never affect the clause output and, thus, the related `NOT` and `AND` gates, shown in Fig. 2(a), can be excluded from the combinational circuit. Similarly, the boolean action `1` mandates that the literal needs to included in the circuit as it will determine the clause output,

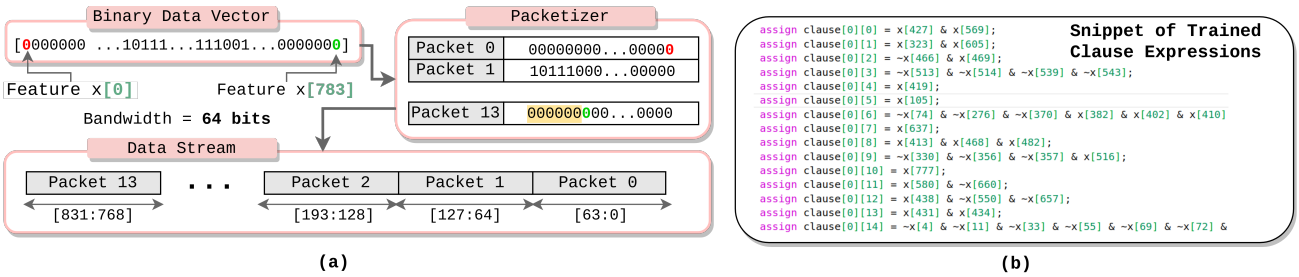

Fig. 4. (a): Packetization of a MNIST datapoint in the processor of the SoC-FPGA. (b): A snippet of the full clause expressions generated by MATADOR after training the Tsetlin Machine on MNIST. The clauses are a 2D array of dimensions `[number of classes][number of clauses]`.

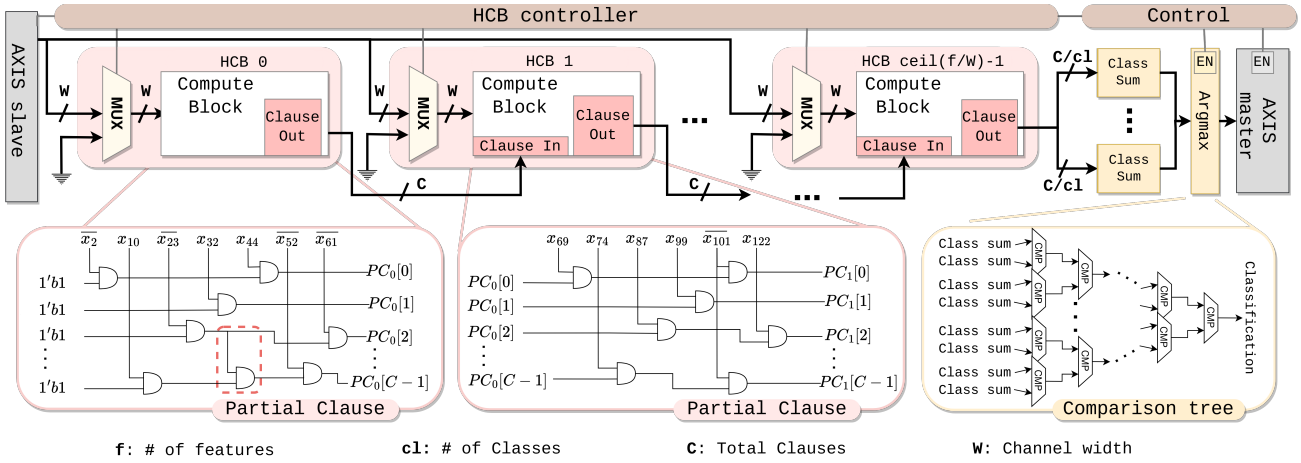

Fig. 5. Block diagram of the generated MATADOR inference accelerator architecture.

shown in Fig. 2(b). The clause output is an accumulation of partial clause outputs. Fig. 2(c) shows the translation of include(`1`)-exclude(`0`) decision into a boolean expression. Fig. 3 presents a crucial aspect of Tsetlin Machines that enables a resource frugal, performance and power efficient boolean-to-silicon transition of TM models. From extensive experimentation, we have empirically found that TM models exhibit extremely high sparsity in the occurrence of includes, and significant sharing of boolean expressions among the clauses within the class as well as among the classes. This observation is pivotal, leading to condensed and compact designs via sparsity and logic sharing. This along with an automated RTL to SoC-FPGA implementation, making use of the open-source *Pynq* Hardware/Software stack, makes MATADOR a true boolean-to-silicon framework.

III. Accelerator Architecture

The MATADOR design methodology takes its namesake from achieving the fastest possible inference time for a given model with respect to bandwidth - MATADOR: autoMated dATa bAndwidth Driven lOgic based infeRence. To that end, all the major compute units are designed to complete operations within one clock cycle. This section presents the design decisions that efficiently use the bandwidth constraints, the sparsity of includes in the trained TM and how this can all be incorporated into the compute units of accelerator.

Bandwidth Driven Data Partitioning: The generated accelerator architecture works with streamed input data where the system latency depends on the effective use of channel bandwidth. The input is, therefore, sent to the accelerator from the processor in packets. Fig. 4 shows the packetization and AXI4-stream protocol based streaming of the data.

Fig. 4(a) shows the packetization of an MNIST example with binary input vector of 784 bits. Assuming the channel bandwidth between the processor and FPGA chip to be 64 bits, 13 packets are needed to send each datapoint. On the processor side the *Packetizer* orders the data from the least significant bit (seen in red) and adds the appropriate zero padding to the last packet (see the yellow highlight after the most significant bit seen in green in Packet 13). For reference, a snippet of clause expressions of MNIST trained TM model is shown in Fig. 4(b).

Hard Coded Clause Blocks (HCBs): The staggered arrival of the input data posits a crucial design challenge - the organization of the clause compute unit in order to process the arriving time-multiplexed data-packets. Attempting to buffer all the packets of one datapoint before computing the full clause output in a single clock cycle leads to extensive sequential chains. As verified through experimentation, synthesizing these chains leads to many critical paths, timing violations and prevents MATADOR running at higher frequencies.

Instead, the MATADOR inference architecture splits the clauses into partial clauses encoded as Hard Coded Clause Blocks (HCB)s, as shown in Fig. 5. The clause expressions are divided into the respective indexes of the data packets, i.e. the first data packet of 64 bits (`[63:0]`) will interact with all the clauses (indicated as `C`) in all the classes containing the first 64 corresponding include decisions, as shown in Fig. 5. The next packet will be routed to clauses that contain the corresponding

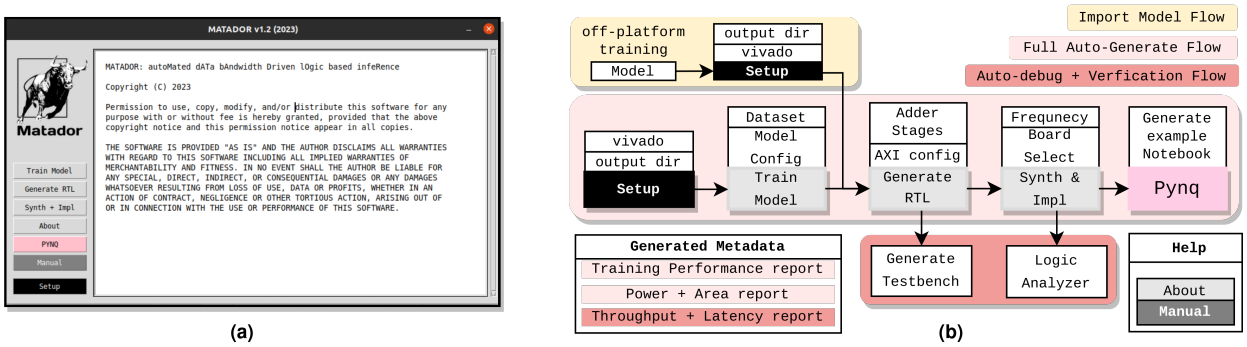

Fig. 6. Matador Design Flow. (a) shows the GUI used to guide the user through design space exploration and accelerator implementation. (b) shows the flow for how the design is generated - the main flow is shown in the pink. Each block is stacked with its respective dependencies. The yellow flow shows how TM models trained outside the MATADOR tool can be imported in. The dark pink flow shows the automated design verification.

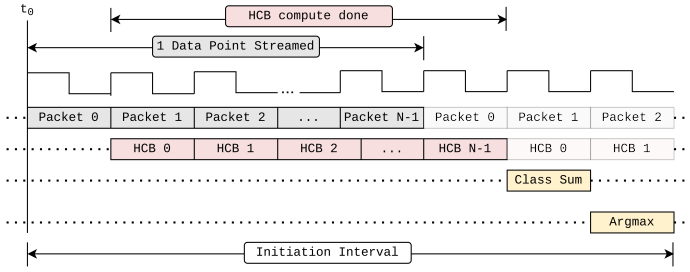

Fig. 7. Timing diagram depicting the routing of each data packet to its respective HCB and the pipelining opportunities in the subsequent Class Sum and Argmax stages.

next 64 ([`127:64`]) include decisions in relation to the packet. The multiplexer in HCB determines when to accept data from the stream. Fig. 5 shows two such partial clause units `HCB 0` and `HCB 1` where `HCB 0` initializes all the clauses in the system to `1'b1`, as it starts with the assumption that all clause outputs are 1.

The gate-level description of the partial clause, shown in Fig. 5, visualizes logic sharing using the highlighted dashed box in partial clause cut-away of `HCB 0`. Logic sharing occurs in both intra- and inter-units. Unlike the streaming architectures used in [3], where each compute block is seen as its own core, MATADOR combines the full design into one core to exploit logic sharing across all the HCB blocks, that is optimized during synthesis and routing.

The partial clause outputs (PC_i) stored in the `Clause Out` part of HCB is passed to next HCB, as `Clause In`, upon the arrival of the next packet. Therefore, the clause expressions continue to get evaluated until the final expressions are formed by the last HCB (the total number of HCBs is equal to the total number of packets needed for one datapoint). The control of the HCB blocks is performed by a dedicated HCB controller.

Class Sum and Argmax Computation: The class sum blocks tally the final clause votes, for each class, in according to the polarity of the clauses. Positive and negative polarity clause votes are accumulated separately and summed in the end to obtain final class sum for each class. Therefore, each class sum block has `2×cl` adders, where `cl` is number of classes. The MATADOR tool allows users to pipeline these adders.

MATADOR employs binary tree comparison, illustrated in Fig 5, to compute Argmax to find the largest class sum and reports the associated index as the classification. This design is parameterized, adopting a comparison tree with $2^{\lceil \log(cl) \rceil}$ inputs. Any classes beyond the actual count are assigned the minimum value at the input stage.

System Overview and Latency: The inference architecture is orchestrated from a dedicated control unit. This unit is used to handle the AXI-stream transactions and offer reset, stall, compute and idle functionalities to the architecture. At the core of the MATADOR design methodology is the concept of being bandwidth driven, the throughput of the MATADOR inference engines are close to the rate at which datapoints are transmitted between the processor and the FPGA fabric. The best model size and performance for the given application can then be determined from this property.

Fig. 7 examines the latency more thoroughly. It shows the initiation interval for the first datapoint that is streamed in. Each packet is routed to its respective HCB until the final full clause is computed, only then the class sum can be computed. This part of the design prevents the pipelining, but the class sum and argmax stages may be pipelined. Subsequent datapoints can be inferred at rate equal to the number of packets required to send the full datapoint. The simplicity of the design means TM models can be translated to RTL without high-level synthesis (HLS) abstractions.

IV. MATADOR Automation Tool

This section presents the automation blocks that create the MATADOR framework. From an end-user perspective MATADOR is a GUI interface that can guide them through the accelerator design flow from training the model, generating the RTL design, synthesizing and implementing on a specified SoC-FPGA and creating an example test code to examine throughput metrics with no coding effort as seen in Fig 6.

In this respect it offers greater ease of use than its contemporaries like [1], [3] which are packaged as hardware libraries that require developer level familiarity with training frameworks such as *Brevitas* or *Theano* as well as other third party library dependencies beyond using Xilinx's *Vits HLS* and *Vivado* for the synthesis and implementation flow. MATADOR also requires the same Xilinx tools but no other third party libraries. MATADOR is an open source tool available here: `https://github.com/nclaes/matador`.

TABLE I

TABLE SHOWING HOW MATADOR INFERENCE ACCELERATORS COMPARE VS THE CLOSEST COMPARABLE STATE-OF-THE-ART BNNS AND QNNS.

Models	Resource Usage								Test Acc (%)	Total Pwr (W)	Dyn Pwr (W)	Latency (us)	Throughput (inf/s)
	LUTs	Slice Registers	F7 Mux	F8 Mux	Slice	LUT as logic	LUT as mem	BRAM					
MNIST													
BNN-r-ref	5636	-	-	-	-	-	-	16	95.83	0.4	-	240	12200
BNN-f-ref	91131	-	-	-	-	-	-	4.5	95.83	7.3	-	0.31	12361000
FINN	11622	17990	172	16	6207	10425	1197	14.5	93.17	1.599	1.458	1.047	954457
MATADOR	8709	17440	5	0	4186	8516	193	3	95.48	1.427	1.292	0.32	3846153
KWS-6													
FINN	42757	45473	206	8	13160	41639	1118	126.5	84.6	3.002	2.796	1.33	750188
MATADOR	6063	10658	5	0	3609	5878	185	3	87.1	1.422	1.287	0.18	8333333
CIFAR-2													
FINN	23247	25654	162	0	9824	22221	1026	66	81.91	2.206	2.042	0.74	1369879
MATADOR	3867	33212	5		6283	3682	185	3	84.8	1.501	1.364	0.38	3125000
FMNIST													
FINN	40002	48901	214	0	13126	38894	1108	131	85.2	2.82	2.622	4.3	232114
MATADOR	13388	40280	5	0	8289	13195	193	3	87.67	1.501	1.364	0.32	3846153
KMNIST													
FINN	40206	49069	186	28	13093	38894	1108	131	89.31	2.695	2.503	3.9	255127
MATADOR	13911	48539	5	0	10499	13718	193	3	88.6	1.483	1.347	0.32	3846153

The "-" indicates this result is not reported. Dyn Pwr = Dynamic Power - taken from *Vivado* implementation reports.

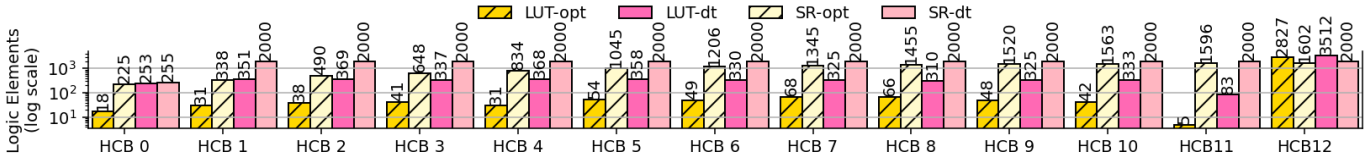

Fig. 8. Demonstration of the optimization opportunities from logic sharing using a MNIST model. LUT-opt and SR-opt are the Look-up-Table and Slice Register counts, respectively, post implementation in *Vivado*. LUT-dt and SR-dt are the counts when the `DON'T TOUCH` pragma is added to prevent optimizations.

TABLE II

TABLE SHOWING THE MODELS USED FOR EVALUATION

Dataset	Model	
	FINN	MATADOR
MNIST	BNN-r/f-ref: 256-256-256 1 bit Weight, Input and Activation Qunatization FINN:784-64-64-64-10 1 bit Weight, Input and Activation Qunatization	200 Clauses per Class
KWS6	377-512-256-6 1 bit Input, 2 bit Weight and Activation Qunatization	300 Clauses per Class
CIFAR 2	1024-256-128-2 1 bit input 1 bit Weight 2 bit Activation Quantization	1000 Clauses per Class
FMNIST	784-256-256-10 1 bit input 2 bit Weight 2 bit Activation Quantization	500 Claues per Class
KMNIST	784-256-256-10 1 bit input 2 bit Weight 2 bit Activation Quantization	500 Clauses per Class

MATADOR configuration states the number of clauses per class.

Given that MATADOR makes use of open-source Xilinx IPs, for validating the throughput MATADOR also supports auto-debug functionality through auto-generated testbench and the use of integrated-logic-analyzer (ILA) debug cores into the design so that AXI-stream transactions can be polled. MATADOR also provides a manual used to guide users with no experience with Tsetlin Machines through the automation. MATADOR also offers a sample jupyter notebook where the generated model can be validated for test accuracy and the throughput and latency can be measured. It follows the same throughput measuring procedure as the FINN flow. Throughput and initial latency can be further validated via the auto-debug flow. Given that MATADOR does not require any FPGA BRAM elements auto-debug functionality does not reduce the resource pool for the actual accelerator.

V. EVALUATION

This section evaluates auto-generated MATADOR implementations against the latest FINN flow as described in [1] with the closest comparable models chosen to examine resource usage, accuracy, power consumption, latency of one datapoint and throughput. The evaluation presents both the best models reported in [3] and also implements the FINN flow on the same platform as MATADOR for fair like-for-like comparison.

The generated designs are evaluated with five datasets:

MNIST [25]: This has been used as the main benchmarking dataset for the FINN and FINN-R frameworks with well defined models readily available in the FINN codebase repository. **KMNIST** [26] is also tested to examine character recognition. **CIFAR-2**: A 2 class variant derived from the CIFAR-10 [27]. CIFAR-2 groups all images into two classes of either animals or vehicles. Used in conjunction with **FMNIST** [28] to show possibilities of this flow in classification applications. **Google Speech Commands (KWS):** A modified version of the dataset is used with 6 chosen keywords: yes, no, up, down, left, right. This is referred to as KWS6, it is used to demonstrate audio classification possibilities.

Table I shows the comparison of the auto-generated MATADOR implementations against its closest comparable QNN and BNN implementations via the FINN flow. The BNNs highlighted in pink were taken from [3], they were implemented on the Zynq ZC706 SoC running at 200MHz. The remaining FINN and MATADOR accelerators were implemented on Zynq XC7Z020 (Pynq Z1), chosen due to it's more constrained logic resources. These FINN models were ran at 100MHz while all MATADOR designs were ran at optimum frequencies per design between 50MHz-65MHz. The model configurations for all accelerators for each dataset are given in Table II.

The results indicate that MATADOR is capable of offering a middle ground to the FINN BNN models where it can perform inferences at a faster rate than the resource efficient *BNN-r-ref*

but has a lower logic footprint compared to *BNN-f-ref*. The main advantages from MATADOR over the remaining models come from the bandwidth driven inference enabling high throughput and the logic sharing between the HCB blocks. To understand the extent of the optimizations that are made, the MNIST HCBs were passed through the synthesis and implementation flow with `DON'T TOUCH` pragmas as seen in Fig 8.

VI. Conclusion

This paper presents the first insights into automated edge inference accelerators for Tsetlin Machines via MATADOR, a boolean-to-silicon tool for training of Tsetlin Machines and their translation into tailored SoC-FPGA accelerators. The benefits of MATADOR based accelerators arise from three interlinked points, firstly, the intrinsic logic proposition forming nature of the TM opens the same opportunities in hardware as binary and quantized DNNs. This allows MATADOR to apply the same heterogeneous streaming architecture design principles found in its closest s.o.t.a. alternatives. Secondly the sparsity in the clause expressions learnt from the TM allows for models to be transitioned to logic constrained SoCs, keeping all the trained model parameters on the FPGA fabric and finally the shared literals or groups of literals between clauses can be exploited by the synthesis tool's logic absorption algorithms to create more resource frugal designs. Further work with MATADOR is concerned with accelerating other TM models for scalability to larger datasets.

References

[1] M. Blott, T. B. Preußer, N. J. Fraser, G. Gambardella, K. O'brien, Y. Umuroglu, M. Leeser, and K. Vissers, "FINN-R: An End-to-End Deep-Learning Framework for Fast Exploration of Quantized Neural Networks," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 11, no. 3, dec 2018.

[2] G. Gobieski, B. Lucia, and N. Beckmann, "Intelligence Beyond the Edge: Inference on Intermittent Embedded Systems," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '19. Association for Computing Machinery, 2019, p. 199–213.

[3] Y. Umuroglu, N. J. Fraser, G. Gambardella, M. Blott, P. Leong, M. Jahre, and K. Vissers, "FINN: A Framework for Fast, Scalable Binarized Neural Network Inference," in *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 65–74.

[4] Y. Wang, J. Xu, Y. Han, H. Li, and X. Li, "DeepBurning: Automatic Generation of FPGA-Based Learning Accelerators for the Neural Network Family," in *Proceedings of the 53rd Annual Design Automation Conference*, ser. DAC '16. New York, NY, USA: Association for Computing Machinery, 2016.

[5] S. Maheshwari, T. Rahman, R. Shafik, A. Yakovlev, A. Rafiev, L. Jiao, and O.-C. Granmo, "REDRESS: Generating Compressed Models for Edge Inference Using Tsetlin Machines," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 1–16, 2023.

[6] L. Deng, G. Li, S. Han, L. Shi, and Y. Xie, "Model Compression and Hardware Acceleration for Neural Networks: A Comprehensive Survey," *Proceedings of the IEEE*, vol. 108, no. 4, pp. 485–532, 2020.

[7] S. I. Venieris and C.-S. Bouganis, "fpgaConvNet: A Framework for Mapping Convolutional Neural Networks on FPGAs," in *2016 IEEE 24th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2016, pp. 40–47.

[8] J. J. Zhang, N. Bohm Agostini, S. Song, C. Tan, A. Limaye, V. Amatya, J. Manzano, M. Minutoli, V. G. Castellana, A. Tumeo, G.-Y. Wei, and D. Brooks, "Towards Automatic and Agile AI/ML Accelerator Design with End-to-End Synthesis," in *2021 IEEE 32nd International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, 2021, pp. 218–225.

[9] O. Granmo, "The Tsetlin Machine - A Game Theoretic Bandit Driven Approach to Optimal Pattern Recognition with Propositional Logic," *CoRR*, vol. abs/1804.01508, 2018.

[10] J. Sharma, R. K. Yadav, O. Granmo, and L. Jiao, "Drop Clause: Enhancing Performance, Interpretability and Robustness of the Tsetlin Machine," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 37, no. 11, 2023.

[11] A. Bakar, T. Rahman, A. Montanari, J. Lei, R. Shafik, and F. Kawsar, "Logic-Based Intelligence for Batteryless Sensors," in *Proceedings of the 23rd Annual International Workshop on Mobile Computing Systems and Applications*, ser. HotMobile '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 22–28.

[12] A. Bakar, T. Rahman, R. Shafik, F. Kawsar, and A. Montanari, "Adaptive Intelligence for Batteryless Sensors Using Software-Accelerated Tsetlin Machines," in *Proceedings of the 20th ACM Conference on Embedded Networked Sensor Systems*, ser. SenSys '22, 2023, p. 236–249.

[13] S. J. Ojukwu, S. Maheshwari, R. Shafik, A. Yakovlev, and M. Mamlouk, "AI-Driven Battery State-of-Charge Estimation using Electrochemical Impedance Spectroscopy," in *Second International Symposium on the Tsetlin Machine*, 2023.

[14] O. Granmo, S. Glimsdal, L. Jiao, M. Goodwin, C. W. Omlin, and G. T. Berge, "The Convolutional Tsetlin Machine," *CoRR*, vol. abs/1905.09688, 2019. [Online]. Available: http://arxiv.org/abs/1905.09688

[15] D. Abeyrathna, O.-C. Granmo, and M. Goodwin, "Convolutional regression tsetlin machine: An interpretable approach to convolutional regression," *2021 6th International Conference on Machine Learning Technologies*, 2021.

[16] S. Glimsdal and O. Granmo, "Coalesced multi-output tsetlin machines with clause sharing," *CoRR*, vol. abs/2108.07594, 2021. [Online]. Available: https://arxiv.org/abs/2108.07594

[17] T. Rahman, S. Maheshwari, R. Shafik, A. Yakovlev, and S. Das, "MILEAGE: An automated optimal clause search paradigm for Tsetlin Machines," in *First International Symposium on the Tsetlin Machine*, 2022.

[18] O. Tarasyuk, A. Gorbenko, T. Rahman, R. Shafik, A. Yakovlev, O.-C. Granmo, and L. Jiao, "Systematic Search for Optimal Hyper-parameters of the Tsetlin Machine on MNIST Dataset," in *Second International Symposium on the Tsetlin Machine*, 2023.

[19] S. Maheshwari, T. Rahman, A. Wheeldon, R. Shafik, A. Yakovlev, and F. Xia, "Introducing TRIM Automata for Tsetlin Machines," in *Second International Symposium on the Tsetlin Machine*, 2023.

[20] S. A. Tunheim, R. K. Yadav, L. Jiao, R. Shafik, and O.-C. Granmo, "Cyclostationary Random Number Sequences for the Tsetlin Machine," in *Advances and Trends in Artificial Intelligence. Theory and Practices in Artificial Intelligence*, H. Fujita, P. Fournier-Viger, M. Ali, and Y. Wang, Eds. Springer International Publishing, 2022, pp. 844–856.

[21] T. Rahman, G. Mao, S. Maheshwari, K. Krishnamurthy, R. Shafik, and A. Yakovlev, "Parallel Symbiotic Random Number Generator for Training Tsetlin Machines on FPGA," in *Second International Symposium on the Tsetlin Machine*, 2023.

[22] A. Wheeldon, R. Shafik, T. Rahman, J. Lei, A. Yakovlev, and O.-C. Granmo, "Learning automata based energy-efficient AI hardware design for IoT applications," *Philos. Trans. R. Soc. A Math. Phys. Eng. Sci.*, vol. 378, no. 2182, 2020.

[23] S. A. Tunheim, L. Jiao, R. Shafik, A. Yakovlev, and O.-C. Granmo, "A Convolutional Tsetlin Machine-based Field Programmable Gate Array Accelerator for Image Classification," in *2022 International Symposium on the Tsetlin Machine (ISTM)*, 2022, pp. 21–28.

[24] R. A. Fisher, "Iris," UCI Machine Learning Repository, 1988, DOI: https://doi.org/10.24432/C56C76.

[25] L. Deng, "The MNIST database of handwritten digit images for machine learning research," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 141–142, 2012.

[26] T. Clanuwat, M. Bober-Irizar, A. Kitamoto, A. Lamb, K. Yamamoto, and D. Ha. (2018) Deep Learning for Classical Japanese Literature.

[27] A. Krizhevsky, "Learning multiple layers of features from tiny images," University of Toronto, Tech. Rep., 2009.

[28] H. Xiao, K. Rasul, and R. Vollgraf. (2017) Fashion-MNIST: a novel image dataset for benchmarking machine learning algorithms.