2025 Design, Automation & Test in Europe Conference (DATE 2025)

TrackScorer: Skyrmion Logic-in-Memory Accelerator for Tree-based Ranking Models

Elijah Seth Cishugi*, Sebastian Buschjäger†, Martijn Noorlander*, Marco Ottavi*‡ and Kuan-Hsun Chen*
*University of Twente, The Netherlands
†The Lamarr Institute for Machine Learning and Artificial Intelligence, Germany
‡University of Rome Tor Vergata, Italy
*{e.s.cishugi, m.ottavi, k.h.chen}@utwente.nl, sebastian.buschjaeger@tu-dortmund.de

***Abstract*—Racetrack memories (RTMs) have been shown to have lower leakage power and higher density compared to traditional DRAM/SRAM technologies. However, their efficiency is often hindered by the need to shift the targeted data to access ports for read and write operations. Suitable mapping approaches are therefore essential to unleash their potential. In this work, we explore the mapping of the popular tree-based document ranking algorithm, Quickscorer, onto Skyrmion-based racetrack memories (SK-RTMs). Our approach leverages a Logic-in-Memory (LiM) accelerator, specifically designed to execute simple logic operations directly within SK-RTMs, enabling an efficient mapping of Quickscorer by exploiting its bitvector representation and interleaved traversal scheme of tree structures through bitwise logical operations. We present several mapping strategies, including one based on a quadratic assignment problem (QAP) optimization algorithm for optimal data placement of Quickscorer onto the racetracks. Our results demonstrate a significant reduction in read and write operations and, in certain cases, a decrease in the time spent shifting data during Quickscorer inference.**

***Index Terms*—Racetrack Memory, Logic-In-Memory, Quickscorer, Document Ranking**

I. Introduction

The rapidly increasing demand for data storage and computational efficiency in modern high-performance computing has led to significant research into high-density, low-power memory technologies [1]. Among the most promising candidates is Racetrack memory (RTM), a non-volatile memory technology that stores data within the domains of a ferromagnetic wire [2] or as skyrmions [3]. RTMs offer several advantages over conventional memory technologies such as DRAM and SRAM, including significantly lower leakage power and higher storage density [4]. However, because data cannot be randomly accessed in RTMs, these advantages come with the challenge of shifting the ferromagnetic domains or skyrmions to an access port before data can be read or written. These shift operations introduce additional latency. To mitigate this issue, efficient data placement strategies are essential, as minimizing the number of shifts is key to unlocking the potential of RTM technologies [5].

Beyond addressing storage efficiency, a significant limitation of conventional memory systems is the substantial time and bandwidth required to transfer large volumes of data between memory and processing units. To overcome the growing disparity between processor speed and memory bandwidth, often referred to as the ”memory wall issue” [6], the concept of Logic-in-Memory (LiM) has been introduced. LiM enables computations to occur directly within the memory, thereby reducing data movement overhead. Specifically, skyrmion-based LiM has demonstrated the capability to perform fundamental logical operations such as binary NOT, AND, and OR by exploiting the unique physical properties of skyrmions [7], [8].

Given that efficient data placement in RTMs is crucial to reduce the number of shifts and optimize performance, applications require careful design to ensure effective operation. In prior research, various applications, including decision trees [9], sorting algorithms [10], and AES encryption [11], have been mapped to RTMs, utilizing them primarily as scratchpads and main memories. However, despite these efforts, the mapping of document ranking algorithms, which are critical for information retrieval systems, has not yet been explored in this context. A prominent example is Quickscorer [12], a document ranking algorithm based on tree ensembles that represent nodes as bitvectors, enabling the use of simple binary AND operations to identify exit leaves. This characteristic makes Quickscorer an ideal candidate for implementation in skyrmion-based LiM on RTM, as it precisely leverages the capabilities advantageous for LiM operations on racetrack structures. Quickscorer and its derivatives, including V-Quickscorer [13] and Rapidscorer [14], have demonstrated significant performance improvements. Yet, the mapping of such document ranking algorithms onto RTMs remains unexplored.

Our Contributions: To address this gap, we make the following key contributions:

- We define the mapping problem and propose multiple strategies to map Quickscorer onto Skyrmion-based Racetrack Memory (SK-RTM), incorporating LiM capabilities. We provide an optimal mapping based on the quadratic assignment problem (QAP) algorithm.
- We enhance RTSim, a popular architecture-level RTM simulator, to facilitate design exploration specific to skyrmion-based memories with integrated LiM functionalities.
- We conduct extensive simulations via the enhanced RT-Sim to evaluate the proposed mappings, considering realistic metrics specific to SK-RTMs, demonstrating the

This work has been partially supported by the EU TRISTAN project (project number 101095947), which has received funding from the Key Digital Technologies Joint Undertaking (KDT JU) and its members.

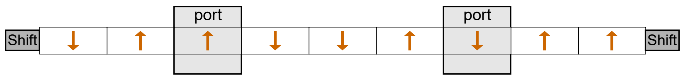


Fig. 1: DW-RTM cell with two access ports and nine domains.

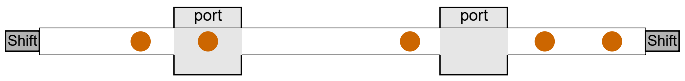


Fig. 2: SK-RTM cell with two access ports.

effectiveness of our approach.

The remainder of the paper is structured as follows: Section II gives the necessary background. In Section III, the simulation framework is discussed, including our enhancements to RT-Sim. Afterward, several mapping strategies are proposed in Section IV. The effectiveness of these mapping strategies is evaluated in Section V. Lastly, Section VI concludes this work.

II. Background and Related Work

In this work, we aim to efficiently map the data of Quickscorer onto an LiM-accelerated RTM. This section reviews RTMs, skyrmion logic, and the Quickscorer algorithm to contextualize our study.

A. *Racetrack Memories*

RTMs come in two main types: Domain Wall Racetrack Memory (DW-RTM) and SK-RTM. DW-RTMs utilize a ferromagnetic nanowire divided into magnetic domains, with each domain's magnetization direction representing either '0' or '1'. SK-RTMs, in contrast, use skyrmions—nanoscale spin configurations—as data carriers, where the presence of a skyrmion encodes a '1' and its absence encodes a '0' [10]. Figure 1 and Figure 2 depict a DW-RTM cell and a SK-RTM cell, respectively.

RTMs require multiple access ports for read/write operations. Reading involves Magnetic Tunnel Junctionss (MTJs) to detect the direction of magnetization of the domain walls in DW-RTMs or the presence of skyrmions in SK-RTMs through magnetoresistance effects. To read data accurately, racetracks arranged in parallel, need synchronization to avoid misalignment errors [3]. Synchronization issues in SK-RTMs are mitigated using Voltage-Controlled Magnetic Anisotropy (VCMA) gates, which control skyrmion movement by creating energy barriers [15].

Writing in DW-RTM is done via spin-transfer torque (STT) currents, moving domain walls along the track to set the magnetization at desired positions. In SK-RTMs, skyrmions are injected or removed at specific positions using localized currents (depinning currents) without rewriting the entire track. A phenomenon known as the Skyrmion Hall Effect causes skyrmions to move in multiple directions; solutions include curbed tracks to confine their motion. Skyrmion-based memories can be denser and more energy-efficient than domain wall memories due to the smaller size of skyrmions, although their detection is more challenging [16].

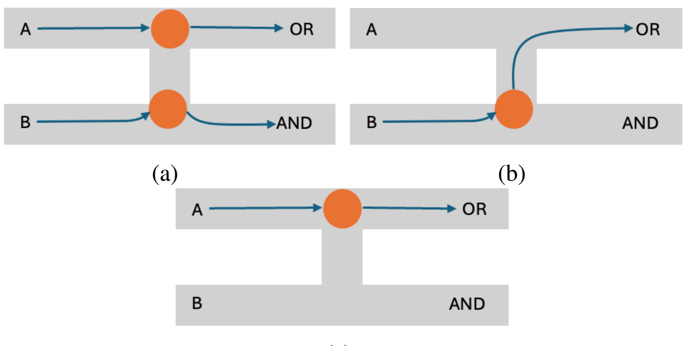


Fig. 3: Skyrmion AND/OR gate. (a) A skyrmion from port A moves directly to the OR output, while the skyrmion from port B is first deflected upward by the Skyrmion Hall Effect and then repelled into the AND output by the first skyrmion. (b) A skyrmion from port B moves to the OR output due to the Skyrmion Hall Effect. (c) A skyrmion from port A moves directly to the OR output.

B. *Skyrmion logic*

The Skyrmion Hall Effect has been used to develop reversible logic gates, as demonstrated in previous research [7]. By designing racetracks with specific geometries and junctions, and by leveraging the repulsion between skyrmions, various logic functions can be implemented. Two key gate types are the NOT/COPY gate and the AND/OR gate depicted in Figure 3. The AND/OR gate takes two skyrmion inputs and produces both AND and OR outputs. The NOT/COPY gate uses a data input and a control input to generate three outputs: two identical copies of the data input (COPY) and one inverted output (NOT). The control input manages skyrmion flow, ensuring a consistent number of skyrmions enter and exit the gate.

C. *Quickscorer*

Tree ensembles, particularly Gradient-Boosted Regression Trees (GBRT) and LambdaMART, have demonstrated exceptional performance in ranking search engine query results [17], [18]. However, these methods are computationally intensive due to the need to traverse numerous trees. To address this, Luchesse et al. proposed Quickscorer, an efficient algorithm designed to improve the performance of Learning-to-Rank (LtR) models such as the aforementioned GBRT and LambdaMART [12].

Quickscorer uses a bitvector representation of tree structures to enhance ranking efficiency. Instead of traversing each tree individually in a root-to-leaf manner, Quickscorer performs an interleaved traversal across the entire tree ensemble using bitwise logical operations. This method allows Quickscorer to evaluate multiple trees' conditions simultaneously, updating bitvectors that represent candidate exit leaves, thereby avoiding unnecessary computations.

As detailed in Algorithm 1, Quickscorer processes a tree ensemble T and a feature vector x. Nodes are grouped by features and sorted by thresholds, with `result bitvectors` indicating reachable leaves. As $x[k]$ is compared with each

Algorithm 1 Quickscorer (features x, ensemble T)

```
 1: // Set all result bitvectors to 1
 2: for $h = 0$ to $h = |T|$ do
 3:     $result\_bitvectors[h] \leftarrow 11...11$
 4: end for
 5: // Iterate through features
 6: for $k \leftarrow 0$ to $k \leftarrow |F|$ do
 7:     for $i \leftarrow offsets[k]$ to $i \leftarrow offsets[k+1]$ do
 8:         if x$[k] >$ thresholds$[i]$ then
 9:             $h \leftarrow tree\_ids[i]$
10:             $result\_bitvectors[h] \leftarrow result\_bitvectors[h] \wedge bitvectors[i]$
11:         else
12:             break
13:         end if
14:     end for
15: end for
16: $score = 0$
17: for $h = 0$ to $h = |T|$ do
18:     $i \leftarrow$ first bit index set to 1 in $bitvectors[h]$
19:     $score \leftarrow score + leaf\_values[h * |L| + i]$
20: end for
```

threshold in ascending order, the tree `bitvectors` are updated using bitwise AND operations to eliminate unreachable leaves. If $x[k]$ is below a threshold, the algorithm skips the remaining nodes for that feature. This process continues until it identifies the exit leaf for each tree to compute the final score.

III. Simulation Framework

Due to the absence of commercial off-the-shelf SK-RTMs, our methodology consists of the following steps. First, we log the memory access patterns of Quickscorer during inference. Next, we map these memory traces onto an SK-RTM architecture—either including or omitting an LiM accelerator—using several mapping strategies proposed in this work. Finally, we perform architectural-level memory simulations of the mapped memory traces to evaluate Quickscorer's performance under the different mapping strategies.

A. Hardware architecture

Using an HDL framework introduced by Gnoli et al. [8], which tracks skyrmion positions within a racetrack and supports AND/OR and COPY/NOT logic functions, we developed a proof-of-concept LiM architecture. The requirements for the SK-RTM-LiM architecture were as follows: (1) There need to be two racetracks, one for the bitvector input and one for the result of the AND operation; (2) The original bitvectors should be immutable since the data is reused between the scoring of different documents.

Our SK-RTM-LiM architecture, shown in Figure 4, can perform a bitwise AND between two memory tracks, representing the binary AND in the Quickscorer algorithm, and store the result in one of the original locations. The layout comprises three main components: two storage racetracks—one for input `bitvectors`, one for `result bitvectors`—and the logic. VCMA gates control skyrmion positions and enable synchronized shifting within the tracks. A COPY/NOT gate, connected to T-shaped junctions, copies skyrmions without

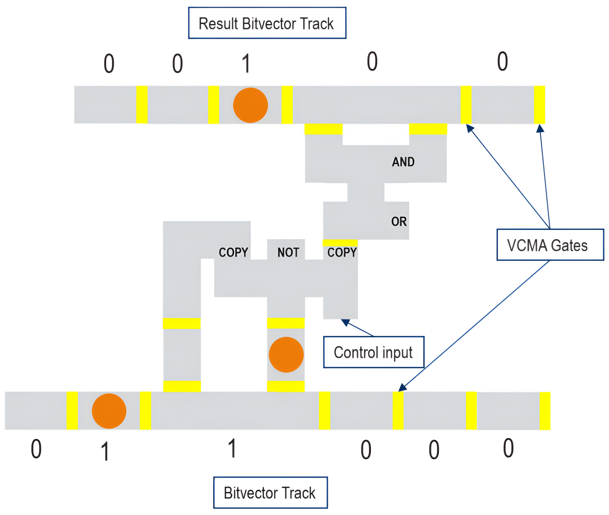


Fig. 4: LiM Architecture for Quickscorer.

altering the original bitvector. The copied skyrmion feeds into an AND/OR gate, while the original returns to its racetrack.

B. Memory simulation

We used RTSim [19] as an architectural-level memory simulator. RTSim is a versatile, cycle-accurate simulation framework specifically designed for evaluating the performance of RTMs. Built on top of NVMain2.0 [20], RTSim provides comprehensive support for modeling the unique characteristics of racetrack memory, such as read/write and shift operations as well as access port management [21].

The essential structure of an RTM is shown in Figure 5. It consists of Domains grouped into Domain Block Clusters (DBCs). A domain references a specific bit within said DBC. All tracks within a DBC shift together in order for words to stay aligned. Upon a read/write request to a DBC, RTSim automatically dispatches a Shift request, finds the closest access port to the requested memory location and then updates all port locations within the said DBC to reflect the number of shifts.

RTSim was originally designed to support DW-RTMs, not SK-RTMs. To adapt RTSim for this work, several enhancements were implemented. First, Insert and Delete operations were added, along with automatic Shift requests, functioning similarly to Write and Read operations. Additionally, a metric was introduced to monitor skyrmion creation and destruction, such as by comparing data during Write operations. For simplicity, we retained the naming conventions used in DW-RTMs: DBC refers to tracks, and Domain refers to bits.

Second, LiM operations were introduced, shifting two DBCs simultaneously without automatic Shift requests. In these operations, the request address refers to the input bitvector, and the new data field encodes the result bitvector's address, minimizing changes to the request structure. LiM operations shift both input and result DBCs to the nearest access port, computing skyrmion metrics as in other operations. A mask allows selective activation of DBCs in parallel LiM operations.

Lastly, a new metric for total shift duration was introduced to address the limitation of counting shifts, which doesn't account

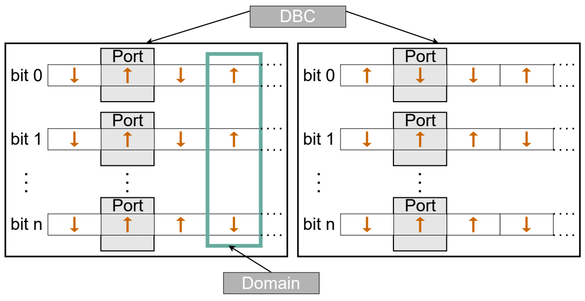


Fig. 5: Structure of an RTM considered in RTSim

for parallel track shifts. This metric records the duration of the longest shift. For LiM operations, it captures the longer of the two shifts, which is the one that bottlenecks the operation.

C. Memory access traces generation

The inputs to RTSim include a configuration file as well as memory traces. For this work, Quickscorer was trained on The Microsoft Learning to Rank dataset [22] using the XG-Boost framework [23]. We developed a C++ class to generate memory access traces from Quickscorer runs, creating RTSim-compatible trace files. This class includes methods for skyrmion memory instructions (Read, Write, Insert, Delete, LiM) and computes memory addresses based on specified DBC/Domain values.

IV. Quickscorer on Skyrmion

We mapped Quickscorer to SK-RTMs using several different strategies. In all cases, memory traces were generated solely for the ranking algorithms, omitting the data preparation phase, as the latter is performed only once and is independent of the number of documents being ranked.

A. Mapping without LiM

As one of the baseline approaches, we mapped Quickscorer onto an SK-RTM architecture without LiM. Each input array is assigned to a DBC, resulting in eight DBCs, each containing 32 subtracks to match the word size. During runtime, any access or modification of the data arrays is logged in a trace file, along with the corresponding instruction and the data states before and after the operation, allowing RTSim to calculate the number of skyrmions created or destroyed.

B. Mapping with LiM

In the second mapping, we incorporate the aforementioned LiM accelerator to perform the AND operations necessary for computing the `result bitvectors`. This mapping requires bits to be in close physical proximity to enable skyrmion logic operations. However, this introduces a shift overhead, which depends on the number of racetrack access ports. Using LiM can increase this overhead up to the number of trees in the ensemble, as bitvector n might belong to the first tree while bitvector $n+1$ belongs to the last tree.

To decrease the overhead in shifts caused by the LiM implementation, a more efficient mapping of the trees in the

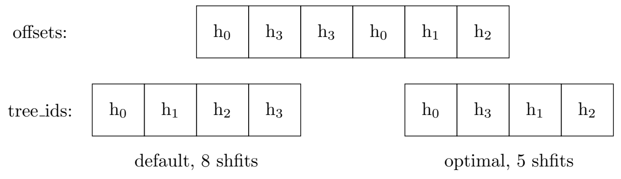


Fig. 6: Example of the default and an optimal mapping of trees to the racetrack. Given an access pattern $t = (h_0, h_3, h_3, h_0, h_1, h_2)$, the default mapping of trees the racetrack (i.e. h_0, h_1, h_2, h_3) leads to $3+3+1+1 = 8$ shifts in total. The optimal mapping (i.,e h_0, h_3, h_1, h_2) that places h_3 right behind h_0 only leads to $1+1+2+1 = 5$ shifts.

result bitvectors is needed. Recall, that Quickscorer uses a look-up-table called `tree_ids` to determine which result bitvector belongs to a certain node. The order in which each tree is accessed is determined by the `offsets` array (c.f. line 7 in Alg. 1), whose order is ultimately determined by the sorting of thresholds. Put differently, the access pattern to each tree is determined by the sorting of thresholds. However, the order in which trees are sorted onto the RTM can greatly impact the number of shifts that occur. An example can be found in Figure 6. Here, 4 trees (h_0 to h_3) are sorted in two different orderings onto the racetrack. Given the access pattern (i.e. the `offsets` array) of these trees, the default mapping results in a total of 8 shifts, whereas the optimal layout only requires 5 shifts. Clearly, the optimal layouting of `tree_ids` can drastically reduce the number of shifts required.

We now formulate the layouting of trees on the racetrack to minimize shifts as a QAP [24]. Conceptually, let
$P = \{p_1, \dots, p_T\}$ be a set of T facilities (i.e., the trees in our case) and let $L = \{0, 1, \dots, T-1\}$ be a set of T locations (i.e., positions on the track). We define a distance matrix $D \in \mathbb{N}^{T \times T}$ between the locations on the track by the number of shifts between two positions i and j, i.e.

$$D_{i,j} = |i - j| \tag{1}$$

Moreover, given a certain access pattern $t = (p_1, p_2, \dots, p_n) \in P^n$ of n facilities (i.e., trees), we define the flow $F_{i,j}(t)$ between trees p_i and p_j as the number of changes from p_i to p_j in the entire pattern:

$$F_{i,j}(t) = \sum_{k=1}^{n-1} 1\{t_k = p_i \land t_{k+1} = p_j\} \tag{2}$$

Recall, that Quickscorer iterates feature by feature. Let t_l be the access pattern of feature $l = 1, \dots, d$, then the flow between positions i and j across all features is defined as:

$$F_{i,j} = \sum_{l=1}^{d} F_{i,j}(t_l) \tag{3}$$

Last, let $\Pi_n = \{X \in \{0,1\}^{n \times n}\}$ be the set of $n \times n$ permutation matrices with the following properties:

- $\sum_{i=1}^{n} x_{i,j} = 1, j = 1, \dots, n$
- $\sum_{i=1}^{n} x_{i,j} = 1, i = 1, \dots, n$

- $x_{i,j} = 1$ if i is mapped to j and 0 otherwise.

Then, the goal is to find that permutation matrix X that minimizes the following objective:

$$\min_{X \in \Pi_n} tr(FXD^T X^T) \tag{4}$$

where tr denotes the trace of the matrix.

Solving a QAP has been proven to be NP-complete, meaning there is no optimal solution achievable in polynomial time unless $P = NP$ [24]. Nevertheless, there are approximations such as the Fast Approximate QAP (FAQ) algorithm available that we can use to solve this problem efficiently [25]. For this QAP formulation, it is interesting to note that we use the entire access pattern of all nodes and features, i.e., all shifts across all thresholds and all features have the same impact on the flow. It is well-known, however, that due to the stochastic nature of decision trees, in which observations may follow a different path through a tree, the optimal layouting of trees should favor more likely paths (see e.g. [26]–[28]). Put differently, there might be a chance that for most observations we only iterate over a few thresholds instead of all thresholds (c.f. line 8 and the break instruction in line 12 in Alg. 1). Hence, we should place more emphasis on these 'early' thresholds to optimize the layout for the more common access patterns.

The training data used to build the initial forest can serve as a guideline here. Specifically, in addition to the number of changes from p_i to p_j, we also weight each flow by the number of how often we visited the corresponding node in the training data. This way, nodes that are visited more often during training (and hence hopefully are also visited more often during deployment) get a larger weight during optimization. To do so, we simply replace the flow calculation in Eq. 2 with its weighted counterpart. Let $b = (b_1, \ldots, b_n)$ be the nodes that are visited during the access pattern t and let $s(b)$ be the number of samples in the training data that visited the node b, then the weighted flow is defined as

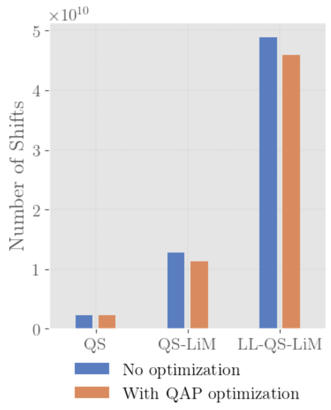
$$\widehat{F}_{i,j}(t, b) = \sum_{k=1}^{n-1} 1\{t_k = p_i \wedge t_{k+1} = p_j\} s(b_k) \tag{5}$$

As a further iteration to the mapping strategies, we expanded upon the QAP mapping solution and explored a so-called **parallel** LiM mapping. For this mapping approach, instead of processing a single document for each iteration, multiple documents are processed concurrently. Here, LiM instructions handle multiple bitvectors at once, increasing required DBCs to N for bitvectors and result bitvectors. In both cases, we maintain a 32-bit word size for read and write operations.

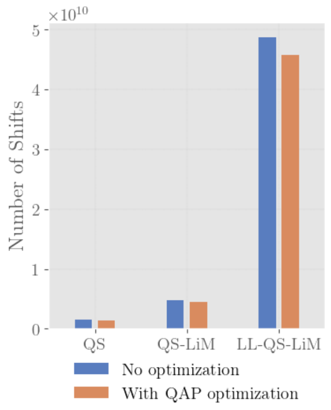
Consequently, we established three mapping strategies: **QS** (base mapping without LIM), **QS-LiM** (base mapping with LIM), and **LL-QS-LiM** (parallel mapping with LIM).

V. Evaluation

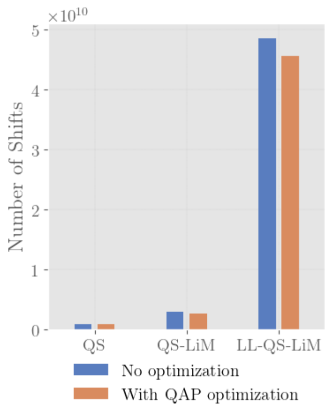
To evaluate the effectiveness of the proposed mapping strategies, several experiments were conducted via RTSim under various parameter settings. For each variation, we report the performance metrics for both the default layout of `tree_ids` and the optimized layout obtained after applying the QAP



(a) 128 Ports



(b) 512 Ports

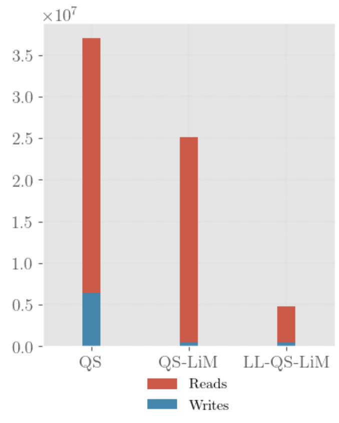


(c) 1024 Ports

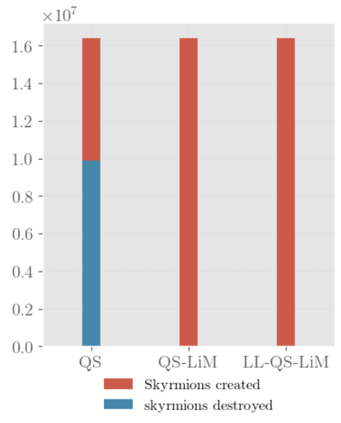
Fig. 7: Total number of shifts over different mappings and number of ports, with and without QAP optimizations.

optimization algorithm. We evaluated all mappings—i.e., QS, QS-LiM, and LL-QS-LiM—with port counts of 128, 512, and 1024. In RTSim, the shifting policy was set to *lazy*; thus, after shifting a skyrmion to an access port, the track remains in that state until the next instruction is executed. Each mapping utilized 32,768 domains per DBC and eight DBCs, except for the parallel LiM mapping, which employed 24 DBCs to accommodate increased parallel operations.

The following metrics, selected for their relevance to SK-RTMs' performance, are reported: the **number of shifts**, **shift duration**, the **number of read/write** operations, the **number of created/destroyed skyrmions**, and the **total energy consump-tion** for each Quickscorer run. While the total number of shifts represents all shifts executed during a single Quickscorer run, shift duration more accurately reflects the delay introduced by shifting skyrmions, as shifts can occur in parallel. For example, shifting skyrmions by 10 positions on a track with a word size of 32 involves 320 shifts in total but results in a delay equivalent to only 10 shifts. Furthermore, the total energy consumption is calculated as the sum of the energy required for reading/writing operations and the energy consumed during skyrmion shifting, as determined by RTSim using the energy model outlined in [21].



(a) Read/Write operations



(b) Created/destroyed skyrmions

Fig. 8: I/O Operations, where each read or write targets a 32-bit word. The number of created/destroyed skyrmions associated with these operations is also reported.

A. *Total number of Shifts and shift duration*

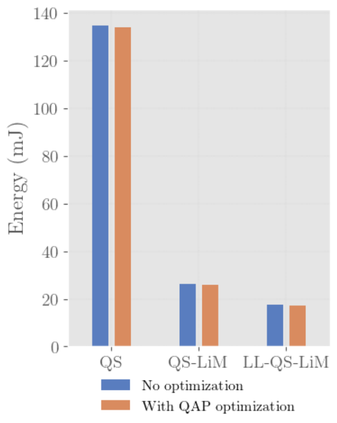
As shown in Figure 7, the total number of shifts increases significantly in the LiM-enabled mappings (QS-LiM and LL-QS-LiM) compared to the baseline QS mapping. While the number of shifts decreases across all mappings as the number of access ports increases from 128 to 1024, this parameter has minimal impact on shift operations in the LiM-enabled parallel mapping, LL-QS-LiM. A critical factor of the outcomes from the QAP optimization in the LiM-enabled mappings is the placement or the order in which the bitvectors and result bitvectors are placed onto the tracks. Post QAP optimization, the QS-LiM and LL-QS-LiM mappings exhibit the most substantial reductions in the total number of shifts, with average improvements of 8.71% and 6.23%, respectively. Moreover, the shift duration metric exhibits a trend consistent with the total number of shifts, scaled down by at least a factor of 32.

B. *Read/write operations and number of created/destroyed skyrmions*

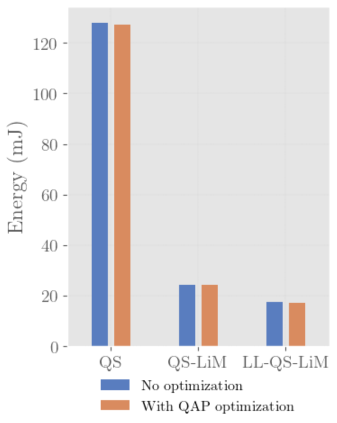
Figure 8a shows that QS-LiM and LL-QS-LiM mappings significantly reduce read and write operations compared to the base QS mapping. Additionally, as observed in Figure 8b, the number of skyrmions created remains consistent across all mappings. This is because, in our simulations, we configured RTSim to reuse skyrmions wherever feasible. Skyrmions are created once during the initialization phase of the Quickscorer algorithm, when all result bitvectors are set to 1. In the base QS mapping, skyrmions are destroyed and recreated for each read/write operation, whereas LiM mappings preserve them throughout the computation.

C. *Total energy consumption*

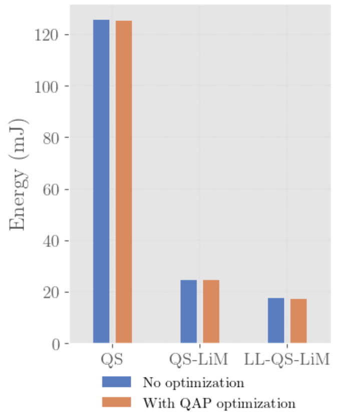
As shown in Figure 9, the total energy consumption decreases markedly in the LiM-enabled mappings compared to the base mapping: 5.1 and 7.3 times smaller, on average. This reduction primarily results from a corresponding decrease in read and write operations in these mappings. Although the LiM-enabled mappings incur an increased number of shift operations, the energy required for shifting skyrmions along the tracks is an order of magnitude lower than that needed to read, write, destroy, or create skyrmions.



(a) 128 Ports



(b) 512 Ports



(c) 1024 Ports

Fig. 9: Total energy consumption over different mappings and number of ports, with and without QAP optimizations.

VI. Conclusion

In this paper, we introduced TrackScorer, an optimized implementation of Quickscorer, a widely-used tree-based ranking model, mapped onto a skyrmion-based LiM accelerator. To validate our approach, we developed a proof-of-concept LiM architecture integrated with RTSim, a cycle-accurate racetrack memory simulator. Additionally, we demonstrated that the layout of Quickscorer on racetrack memory can be framed as a quadratic assignment problem, which can be efficiently addressed through approximations, as confirmed by our simulation results. These results show that LiM-enabled mappings substantially lower the number of read and write operations, as well as the overall energy consumption, compared to a baseline without LiM. For future work, we aim to explore broader applications across different LiM architectures, enhancing the potential of these emerging technologies to address diverse computational challenges.

References

[1] A. Chen, "A review of emerging non-volatile memory (nvm) technologies and applications," *Solid-State Electronics*, vol. 125, pp. 25–38, 2016, extended papers selected from ESSDERC 2015. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0038110116300867

[2] R. Venkatesan, V. J. Kozhikkottu, M. Sharad, C. Augustine, A. Raychowdhury, K. Roy, and A. Raghunathan, "Cache Design with Domain Wall Memory," *IEEE Transactions on Computers*, vol. 65, no. 4, pp. 1010–1024, Apr. 2016. [Online]. Available: http://ieeexplore.ieee.org/document/7349153/

[3] W. Kang, X. Chen, D. Zhu, X. Zhang, Y. Zhou, K. Qiu, Y. Zhang, and W. Zhao, "A Comparative Study on Racetrack Memories: Domain Wall vs. Skyrmion," in *2018 IEEE 7th Non-Volatile Memory Systems and Applications Symposium (NVMSA)*. Hakodate: IEEE, Aug. 2018, pp. 7–12. [Online]. Available: https://ieeexplore.ieee.org/document/8537687/

[4] S. S. P. Parkin and S.-H. Yang, "Memory on the racetrack." *Nature nanotechnology*, vol. 10 3, pp. 195–8, 2015. [Online]. Available: https://api.semanticscholar.org/CorpusID:5531119

[5] R. Bläsing, A. A. Khan, P. C. Filippou, C. Garg, F. Hameed, J. Castrillon, and S. S. P. Parkin, "Magnetic racetrack memory: From physics to the cusp of applications within a decade," *Proceedings of the IEEE*, vol. 108, no. 8, pp. 1303–1321, 2020.

[6] W. A. Wulf and S. A. McKee, "Hitting the memory wall: implications of the obvious," *SIGARCH Comput. Archit. News*, vol. 23, no. 1, p. 20–24, mar 1995. [Online]. Available: https://doi.org/10.1145/216585.216588

[7] M. Chauwin, X. Hu, F. Garcia-Sanchez, N. Betrabet, A. Paler, C. Moutafis, and J. S. Friedman, "Skyrmion Logic System for Large-Scale Reversible Computation," *Physical Review Applied*, vol. 12, no. 6, p. 064053, Dec. 2019. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevApplied.12.064053

[8] L. Gnoli, F. Riente, M. Vacca, M. Ruo Roch, and M. Graziano, "Skyrmion Logic-In-Memory Architecture for Maximum/Minimum Search," *Electronics*, vol. 10, no. 2, p. 155, Jan. 2021. [Online]. Available: https://www.mdpi.com/2079-9292/10/2/155

[9] C. Hakert, A. A. Khan, K.-H. Chen, F. Hameed, J. Castrillon, and J.-J. Chen, "BLOwing Trees to the Ground: Layout Optimization of Decision Trees on Racetrack Memory," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*. San Francisco, CA, USA: IEEE, Dec. 2021, pp. 1111–1116. [Online]. Available: https://ieeexplore.ieee.org/document/9586167/

[10] Y.-S. Hsieh, P.-C. Huang, P.-X. Chen, Y.-H. Chang, W. Kang, M.-C. Yang, and W.-K. Shih, "Shift-Limited Sort: Optimizing Sorting Performance on Skyrmion Memory-Based Systems," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 11, pp. 4115–4128, Nov. 2020. [Online]. Available: https://ieeexplore.ieee.org/document/9211559/

[11] Y. Wang, L. Ni, C.-H. Chang, and H. Yu, "DW-AES: A Domain-Wall Nanowire-Based AES for High Throughput and Energy-Efficient Data Encryption in Non-Volatile Memory," *IEEE Transactions on Information Forensics and Security*, vol. 11, no. 11, pp. 2426–2440, Nov. 2016. [Online]. Available: http://ieeexplore.ieee.org/document/7484726/

[12] C. Lucchese, F. M. Nardini, S. Orlando, R. Perego, N. Tonellotto, and R. Venturini, "QuickScorer: A Fast Algorithm to Rank Documents with Additive Ensembles of Regression Trees," in *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*. Santiago Chile: ACM, Aug. 2015, pp. 73–82. [Online]. Available: https://dl.acm.org/doi/10.1145/2766462.2767733

[13] ——, "Exploiting cpu simd extensions to speed-up document scoring with tree ensembles," in *Proceedings of the 39th International ACM SIGIR Conference on Research and Development in Information Retrieval*, ser. SIGIR '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 833–836. [Online]. Available: https://doi.org/10.1145/2911451.2914758

[14] T. Ye, H. Zhou, W. Y. Zou, B. Gao, and R. Zhang, "RapidScorer: Fast Tree Ensemble Evaluation by Maximizing Compactness in Data Level Parallelization," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. London United Kingdom: ACM, Jul. 2018, pp. 941–950. [Online]. Available: https://dl.acm.org/doi/10.1145/3219819.3219857

[15] B. Liu, S. Gu, M. Chen, W. Kang, J. Hu, Q. Zhuge, and E. H.-M. Sha, "An Efficient Racetrack Memory-Based Processing-in-Memory Architecture for Convolutional Neural Networks," in *2017 IEEE International Symposium on Parallel and Distributed Processing with Applications and 2017 IEEE International Conference on Ubiquitous Computing and Communications (ISPA/IUCC)*. Guangzhou, China: IEEE, Dec. 2017, pp. 383–390. [Online]. Available: https://ieeexplore.ieee.org/document/8367291/

[16] W. Kang, B. Wu, X. Chen, D. Zhu, Z. Wang, X. Zhang, Y. Zhou, Y. Zhang, and W. Zhao, "A Comparative Cross-layer Study on Racetrack Memories: Domain Wall vs Skyrmion," *ACM Journal on Emerging Technologies in Computing Systems*, vol. 16, no. 1, pp. 1–17, Jan. 2020. [Online]. Available: https://dl.acm.org/doi/10.1145/3333336

[17] N. Asadi and J. J. Lin, "Training efficient tree-based models for document ranking," in *European Conference on Information Retrieval*, 2013. [Online]. Available: https://api.semanticscholar.org/CorpusID:11498939

[18] S. Tyree, K. Q. Weinberger, K. Agrawal, and J. Paykin, "Parallel boosted regression trees for web search ranking," in *Proceedings of the 20th International Conference on World Wide Web*, ser. WWW '11. New York, NY, USA: Association for Computing Machinery, 2011, p. 387–396. [Online]. Available: https://doi.org/10.1145/1963405.1963461

[19] [Online]. Available: https://github.com/tud-ccc/RTSim

[20] M. Poremba and Y. Xie, "NVMain: An Architectural-Level Main Memory Simulator for Emerging Non-volatile Memories," in *2012 IEEE Computer Society Annual Symposium on VLSI*. Amherst, MA, USA: IEEE, Aug. 2012, pp. 392–397. [Online]. Available: http://ieeexplore.ieee.org/document/6296505/

[21] A. A. Khan, F. Hameed, R. Blasing, S. Parkin, and J. Castrillon, "RTSim: A Cycle-Accurate Simulator for Racetrack Memories," *IEEE Computer Architecture Letters*, vol. 18, no. 1, pp. 43–46, Jan. 2019. [Online]. Available: https://ieeexplore.ieee.org/document/8642352/

[22] T. Qin and T. Liu, "Introducing LETOR 4.0 datasets," *CoRR*, vol. abs/1306.2597, 2013. [Online]. Available: http://arxiv.org/abs/1306.2597

[23] T. Chen and C. Guestrin, "XGBoost: A Scalable Tree Boosting System," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Aug. 2016, pp. 785–794, arXiv:1603.02754 [cs]. [Online]. Available: http://arxiv.org/abs/1603.02754

[24] R. E. Burkard, E. Çela, P. M. Pardalos, and L. S. Pitsoulis, *The Quadratic Assignment Problem*. Boston, MA: Springer US, 1998, pp. 1713–1809. [Online]. Available: https://doi.org/10.1007/978-1-4613-0303-9_27

[25] J. T. Vogelstein, J. M. Conroy, V. Lyzinski, L. J. Podrazik, S. G. Kratzer, E. T. Harley, D. E. Fishkind, R. J. Vogelstein, and C. E. Priebe, "Fast approximate quadratic programming for graph matching," *PLOS ONE*, vol. 10, no. 4, pp. 1–17, 04 2015. [Online]. Available: https://doi.org/10.1371/journal.pone.0121002

[26] S. Buschjäger and K. Morik, "Decision tree and random forest implementations for fast filtering of sensor data," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 65-I, no. 1, pp. 209–222, Jan. 2018. [Online]. Available: https://doi.org/10.1109/TCSI.2017.2710627

[27] S. Buschjager, K.-H. Chen, J.-J. Chen, and K. Morik, "Realization of Random Forest for Real-Time Evaluation through Tree Framing," in *2018 IEEE International Conference on Data Mining (ICDM)*. Singapore: IEEE, Nov. 2018, pp. 19–28. [Online]. Available: https://ieeexplore.ieee.org/document/8594826/

[28] K.-H. Chen, C. Su, C. Hakert, S. Buschjäger, C.-L. Lee, J.-K. Lee, K. Morik, and J.-J. Chen, "Efficient realization of decision trees for real-time inference," *ACM Transactions on Embedded Computing Systems*, Dec. 2021. [Online]. Available: https://doi.org/10.1145/3508019