Ensemble Model Compression for Fast and Energy-Efficient Ranking on FPGAs

Veronica Gil-Costa[1], Fernando Loor[1], Romina Molina[1,2,3], Franco Maria Nardini[3], Raffaele Perego[3], and Salvatore Trani[3(✉)]

[1] Universidad Nacional de San Luis, San Luis, Argentina
[2] Università degli Studi di Trieste, Trieste, Italy
[3] ISTI-CNR, Pisa, Italy
salvatore.trani@isti.cnr.it

Abstract. We investigate novel SoC-FPGA solutions for fast and energy-efficient ranking based on machine-learned ensembles of decision trees. Since the memory footprint of ranking ensembles limits the effective exploitation of programmable logic for large-scale inference tasks, we investigate binning and quantization techniques to reduce the memory occupation of the learned model and we optimize the state-of-the-art ensemble-traversal algorithm for deployment on low-cost, energy-efficient FPGA devices. The results of the experiments conducted using publicly available Learning-to-Rank datasets, show that our model compression techniques do not impact significantly the accuracy. Moreover, the reduced space requirements allow the models and the logic to be replicated on the FPGA device in order to execute several inference tasks in parallel. We discuss in details the experimental settings and the feasibility of the deployment of the proposed solution in a real setting. The results of the experiments conducted show that our FPGA solution achieves performances at the state of the art and consumes from 9× up to 19.8× less energy than an equivalent multi-threaded CPU implementation.

Keywords: Learning to Rank · Model Compression · Efficient Inference · SoC FPGA

1 Introduction

This work investigates the use of cost-effective SoC-FPGA (System on Chip - Field Programmable Gate Arrays) devices for speeding-up inference tasks based on complex machine-learned ensemble models. Latency and throughput at inference time are critical aspects in many applications of machine learning where the rate of incoming requests is high and tight constraints on prediction quality impose the adoption of computationally-expensive models. In these cases, quality-of-service requirements entail the optimization of the accuracy of the models subject to performing inference in near real-time or within a limited time budget. As a use case where finding the best trade-off between model accuracy and inference time is definitely important and challenging, we consider the

task of ranking documents according to their relevance for user queries. Indeed, ranking for ad-hoc retrieval entangles challenging effectiveness and efficiency constraints in many online services deployed in large-scale Web search engines, e-Commerce platforms and online social networks [6].

We specifically study techniques for performing document ranking with SoC-FPGA devices at a competitive level of quality and speed with respect to the state of the art, but using a fraction of the energy. SoC-FPGA technology provides an energy-efficient alternative to traditional computing due to the possibility of adapting the design of the logic to a specific architecture optimized for the task addressed. The cost and power/performance competitiveness of SoC FPGA makes this technology very attractive for specific tasks such as ranking, where the high cost and power consumption of GPUs make their adoption prohibitive [27]. We claim that SoC-FPGA architectures can provide an efficient and sustainable solution for large-scale query-processing since they can offer efficient ranking capabilities based on state-of-the-art solutions at a fraction of the energy cost incurred by CPU-based or GPU-based solutions. Recently, Molina *et al.* followed the same research line and proposed SoC-FPGA solutions for speeding-up inference based on Learning-to-rank (LtR) ensembles of decision trees [26]. The study identifies in the memory footprint the main issue limiting the computational performance. In this paper, we address this limitation by investigating the use of binning and quantization techniques for reducing the memory occupation of both the ranking model and the feature vectors representing the document-query pairs to be scored. Reducing the memory footprint of the model allows to replicate the ranking logic on the FPGA device to execute several inference tasks in parallel. Furthermore, by compressing the document-query feature vectors, we minimize the transmission costs incurred for transferring them to the FPGA device. We discuss the feasibility of the deployment of the proposed solution in a real setting and evaluate its performance using publicly available LtR datasets. The experiments conducted show that our solution does not impact the quality of the ranking and it provides highly competitive computational performance with very low energy consumption.

The remainder of the paper is organized as follows. Section 2 discusses the related work. Section 3 describes SoC-FPGA technologies. Section 4 introduces ensemble models and the challenges related to their use in the learning to rank scenario. It then discusses the use of binning and quantization for reducing the memory footprint and effectively deploying the ranking process on SoC-FPGA devices. Section 5 discusses the efficiency and effectiveness of the SoC-FPGA deployment compared to the traditional CPU one. It then details an evaluation of the latency introduced by transferring data from the host machine to the SoC-FPGA device. Moreover, it also reports an analysis of the energy consumption provided by both SoC FPGA-based and CPU-based scoring solutions. Finally, Sect. 6 concludes the paper and draws some future work.

2 Related Work

Several effective LtR algorithms and libraries have been proposed in the last years to train complex models able to precisely rank the documents matching a query [9,16,19]. State-of-the-art LtR models include those based on additive ensembles of regression trees learned by Mart [11] and λ-MART [3,31] gradient boosting algorithms. Since such ranking models are made of hundreds of additive regression trees, the tight constraints on query response time require suitable solutions able to provide an optimal trade-off between efficiency and ranking quality [6]. Among the main contributions in the area of efficient ranking, we cite the algorithms for the efficient traversal of tree ensembles [1,10,20,34]. Alternative methods are concerned with: i) strategies for pruning the ensemble during or after the training phase [21,22,24], ii) budget-aware LtR algorithms [1,30], and iii) end-to-end learning of multi-stage LtR pipelines [8,12]. Furthermore, researchers investigated early termination heuristics aimed to reduce, on a document or query-level basis, the cost of the ensemble traversal process without (or minimally) impacting quality [4,5,25]. An analogous strategy was recently proposed to reduce the computational cost of neural re-ranking based on bi-directional transformer networks [32].

Previous work showed that SoC-FPGA devices can handle the complex computation of LtR training algorithms and provide high computing efficiency with low power consumption. Xu *et al.* describe the design of a FPGA accelerator for a LtR algorithm to reduce training time [33]. Gao and Hsu evaluate a LtR algorithm deployed on a FPGA and explore the design space of the implementation choices [13]. Similar to our work, Qiang *et al.* present a fixed-point quantization approach for LtR algorithms on FPGA [18]. Experimental results show that the FPGA-based algorithm achieve a 4.42× speedup over a GPU implementation but with 2% accuracy loss. Differently from these previous works focusing on the offline, LtR training phase, we are interested in the online inference phase, where the machine-learned model is deployed in a large infrastructure and used under tight latency constraints. To the best of our knowledge, only Molina *et al.* previously investigated this important aspect and highlighted the memory usage on the FPGA device as the main issue limiting the computational performance. This work addresses this limitation by exploiting binning and quantization to compress the ranking model and the feature vectors.

3 Using Programmable Logic for Ranking

The features of current SoC-FPGA devices allow their adoption for high-performance computing tasks such as inference under tight time constraints where they can provide an efficient and energy-efficient solution. A SoC-FPGA device integrates on the same chip a general-purpose Processing System (PS) and a Programmable Logic (PL) unit. The PS includes a processor and a memory of greater capacity than the memory available in the PL. The PL includes blocks of memories (BRAM), control and logic components like the Flip Flop (FF) or

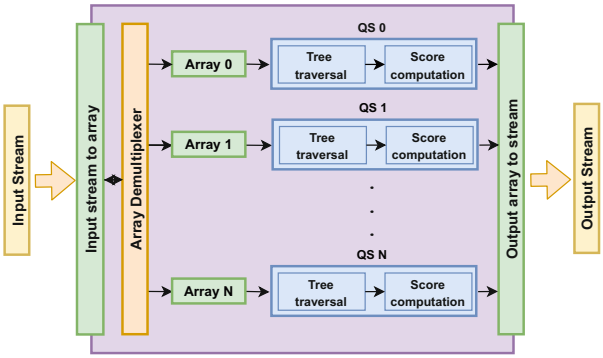

Fig. 1. Hardware design inside the IP block of the FPGA device.

the LookUp Table (LUT). These components are used to implement Intellectual Property (IP) blocks which actually execute the algorithms on the FPGA. The amount of components available in the FPGA is limited and this limitation in turn constrains the design and the deployment of the algorithm which usually involves a trade-off between processing speed and resource utilization. To take full advantage of FPGA logic, we need to process data in parallel, possibly avoiding jumps and recursive calls. To this end, High Level Synthesis (HLS) tools are used to create hardware from a high-level description, using directives to specify concurrency and pipelining opportunities. The HLS tool translates the code to a Register Transfer Level (RTL) specification of the hardware and also returns an estimation of execution latency and resource utilization. In this way, the designer is able to broadly evaluate the performance of different implementation strategies before actually deploying them on the hardware. In this phase the designer is also asked to detail the data communication occurring between the PS and PL. To exploit SoC-FPGA characteristics for ranking, we rely on QuickScorer (QS), the state-of-the-art algorithm for the traversal of large tree ensembles [10,20]. QS exploits a representation of the tree ensemble based entirely on linear arrays accessed with high locality. This characteristic permits a very fast traversal of the tree ensemble at inference time by effectively exploiting features and peculiarities of modern processors and memory hierarchies [17,23].

To estimate resource consumption and execution times several directives are inserted in the QS C++ code. Unrolling techniques are used to parallelize the execution of loop constructs such as for, while, do...while, repeat. These loops can be synthesized if and only if the loops bounds and the condition expression can be calculated during compilation time. In other words, the condition expression determining loop exiting cannot dynamically change at run-time. To tackle this problem, the QS algorithm has been modified by removing any dynamic condition expression from the loops and by including additional if-else statements to split the loops into sub-loops of fixed size which can be processed in parallel.

We propose a FPGA hardware design composed of a single Direct Memory Transfer (DMA) and one IP block responsible for accelerating the QS algorithm.

Both the IP and the DMA belong to the PL. In Fig. 1, we show how the QS algorithm is replicated into different Processing Elements (PE) inside the IP block to perform inference in parallel on different instances. To avoid off-chip memory transactions, the input of the IP is a stream composed of batches of instances (i.e., query-document feature vectors) to be predicted and the output is a stream formed by the actual predictions (i.e., the query-document scores producing the final ranking of the documents for a given query). We implement an array demultiplexer and apply the ARRAY_PARTITION directive to distribute the input feature vectors among the PEs.

We also implement task-level pipelining, allowing functions and loops to overlap in their operation, increasing the overall throughput of the design. The PIPELINE directive optimizes the insertion (push) and extraction (pop) of data from the stream. After inference, the final predictions are packed into an output stream adding the corresponding control signals. The PIPELINE directive is also used to speed-up the execution of the function admitting new inputs.

4 Ranking Model Compression

Ensembles of decision trees are among the most successful Machine Learning (ML) models and the winning solutions in many ML competitions.[1] However, inference with ensemble-based models can be computationally expensive since it requires the complete traversal of the tree ensemble, aimed at identifying all the tree leaves contributing to the prediction. To this end, each tree of the ensemble is visited from its root to a leaf by evaluating the splitting conditions (i.e., a test over a single feature with a learned threshold) associated with internal nodes. The contributions of all the leaves reached (i.e., a class label in case of a classification task or a numeric value in case of a regression task) are aggregated to compute the final prediction. This process has a complexity proportional to the number T of trees in the ensemble multiplied by the average depth d of the decision trees. For document ranking, the use case considered in this paper, typical LtR ensembles are made up of hundreds or even thousands of regression trees usually having each from 5 to 9 levels (corresponding to a number of leaves ranging from 32 to 512) [6]. For example, the winning solution of the Yahoo! Learning to Rank challenge used a linear combination of 12 ranking models, 8 of which were λ-MART boosted tree models each composed of about 3,000 trees for a total of 24,000 trees [7]. Since the traversal has to be repeated for each one of the K candidate documents to be scored for a user query, we have that the per-query ranking cost is proportional to $T \cdot d \cdot K$. Also the amount of memory needed for a LtR ensemble is quite large even if we do not consider the data structures needed to support the inference process. We can roughly estimate a lower bound for the space required to store an ensemble by considering the compact representation of each tree obtained by implicitly encoding its structure (parent/child nodes) in a linear array using a breadth-first order. Internal nodes require 8 bits for the feature identifier (assuming to have at most 256 features) and 32 bits for the

[1] https://dataaspirant.com/xgboost-algorithm/.

threshold value, summing up to 40 bits per node. Leaf nodes, on the other hand, are usually represented with 32-bit floating-point values. In the LtR case, these values represent the additive contribution of the specific tree to the ranking score predicted for the query-document pair. Let us consider for example an ensemble with $T = 1,000$ and $d = 9$: its compact representation requires about 4.4 MB of memory. Such size surely fits in the memory available on a low-cost FPGA device but reducing the memory requirements provides an interesting opportunity to fully exploit the FPGA logic and increase the number of inference tasks processed concurrently. To this regard, in this work we investigate the use of two popular techniques, namely binning and quantization, to lower the memory occupation of ensemble models and to parallelize the inference on FPGA devices.

- Binning consists in bucketing continuous feature values into discrete bins and representing each value with the index of its bin. It is commonly used to speed up training [16], but, at the best of our knowledge its usage for model compression at inference time has not been previously investigated. Specifically, we used binning to encode each internal node of the trees with only 16 bits: 8 bits for the feature identifier and 8 bits for the identifier of the bin associated with one of the possible 256 threshold values. Another advantage of binning the thresholds is that it allows to represent similarly with only 8 bit instead of 32 also the elements of the feature vectors representing the instances to be predicted. The splitting condition in the internal nodes of each decision trees moves than from $feature[i] <= threshold$ to $binned\_feature[i] <= bin$, with feature vectors values that can now be represented with a single byte storing the bin identifier in place of 32 bits. Let us consider one of the LtR datasets used for the experiments (Istella-S) where each query-document pair to be predicted is represented by 220 real-valued features for a memory occupation of 880 bytes. Binning these values into 256 bins results in a 3/4 reduction of the space needed, thus impacting both the number of instances that can be predicted in parallel on the FPGA and the cost of memory transfers.
- Quantization, on the other hand, consists in mapping continuous real values into a discrete set of finite values. This technique is popular for example to compress deep neural network models [14]. We apply quantization to the leaf values of each tree of the ensemble, so as to further lower the memory footprint. Specifically, we represent the 32-bits real value stored in each leaf of the original model with a 8-bits unsigned integer by mapping the min/max among the actual leaf values to the min/max in the range [0, 255]. This reduces of 3/4 also the space needed for storing the leaves of the ensemble.

The combination of the above binning and quantization techniques permit to represent the ensemble model previously mentioned by using only 1/3 of the original space. In Sect. 5, we will show experimentally that such compression does not introduce significant degradation in the resulting ranking effectiveness. On the other hand, we will show the benefits of compression in lowering the FPGA resources used and the data transmission time.

5 Experiments

We evaluate the impact of the proposed binning and quantization techniques for compressing ensemble models on two publicly available LtR datasets, namely MSLR-WEB30K-F1 (Fold 1)[2] [19], hereinafter simply abbreviated as MSN30K, and Istella-S[3] [22]. Both datasets contain more than $30K$ queries and about $3{,}5M$ query-document pairs, where each pair is represented with 136 features on MSN30K and 220 features on Istella-S. The query-document pairs in both datasets are labeled by relevance judgments ranging from 0 (irrelevant) to 4 (perfectly relevant). While the two datasets are comparable in size, they differ in the proportion between positive (label > 0) and negative (label $= 0$) examples. Indeed, in the MSN30K dataset about 48% of the documents are labeled with a positive judgement, while in the Istella-S dataset this proportion lower to 11%. The detailed characteristics of the two datasets are listed in Table 1.

Table 1. Characteristics of the two datasets used.

Dataset	MSN30K	Istella-S
queries	31,351	33,018
query-document pairs	3,771,125	3,408,630
features	136	220
positive examples	48.53%	11.39%

Each dataset is split in train, validation and test set according to a 60%-20%-20% scheme. We use training and validation sets to train ensemble models with the λ-MART [3,31] algorithm, while the test set is used for evaluating the performance of the model. The learning process of λ-MART is controlled by several hyper-parameters, some of them controlling the generalization power and the training speed of the learning phase, while others controlling the shape of the trees. Since our objective is to fasten the inference time by exploiting programmable SoC devices, which are limited in the amount of available resources, we start by finding the most compact model providing state-of-the-art performance, i.e., we investigate the optimal trade-off between model size and ranking effectiveness. To this end, we performed several grid searches by varying the hyper-parameters controlling the shape of the final model and allowing each grid exploits the remaining ones. In particular, we varied the number of leaves in $\{64, 128, 256, 512\}$ by keeping fixed the maximum number of trees to 800. We used the implementation of λ-MART available in the LightGBM library [16] for training and the HyperOpt library [2] for tuning the hyper-parameters. When comparing the performance of different models, we also evaluated statistical significance by using the randomization test with 10,000 permutations and p-value ≤ 0.05 [29].

[2] http://research.microsoft.com/en-us/projects/mslr/.
[3] http://quickrank.isti.cnr.it/istella-dataset/.

Table 2. Efficiency/effectiveness trade-off: NDCG@10 vs. model size in MB.

Dataset	Model Version	per-tree leaves			
		64	128	256	512
MSN30K	Full Precision	0.524 (0.30)	0.526 (0.87)	0.527 (1.59)	0.528 (2.73)
	Bin. + Quant.	0.524 (0.10)	0.526 (0.29)	0.527 (0.53)	0.528 (0.91)
Istella-S	Full Precision	0.771 (0.44)	0.775 (0.87)	0.779 (1.74)	0.781 (3.38)
	Bin.+ Quant.	0.770 (0.14)	0.776 (0.29)	0.779 (0.58)	0.782 (1.12)

5.1 Effectiveness Assessment

We report on the effectiveness of the resulting fine-tuned models on the MSN30K and Istella-S datasets as a function of the size of the models, with and without model compression. It is worth noting that only quantization can affect effectiveness. Binning in fact, even if not exploited previously for model and feature compression, is natively used by the LightGBM library to speed-up model training and has no impact on the quality of the model trained. Table 2 reports the value of NDCG@10 [15] and the size in MB of the full precision and compressed models trained on the two datasets. In terms of absolute effectiveness, the fine-tuned model with 64 leaves achieves a NDCG@10 equal to 0.524 on MSN30K (0.771 on Istella-S), while the best performing model with 512 leaves reaches 0.528 (0.781). We note that with a quality loss lower than 1% on MSN30K and 1.3% on Istella-S the models with 64 leaves are about 9× smaller than the ones with 512 leaves. Thus, these models largely offer the best effectiveness/space trade-off and are most suited for an efficient FPGA deployment in presence of strict memory constraints. By looking at the NDCG@10 values reported in each column of the table, we see that the impact of quantization on the ranking performance is limited. In most case we do not have differences in the NDCG@10 measured on the test set and in all the cases the differences are not statistically significant. To further investigate the impact of quantization on effectiveness, Fig. 2 reports the NDCG@10 of the ranking models with 64 leaves as a function of the number of trees. Each one of the plots in the figure shows three curves: one for the original, full precision models where the values associated with tree leaves are represented as 32-bit floating point values, and two for models exploiting quantization with 8 and 4-bit representations. The curves plotted confirm that quantization using 8-bit representations does not introduce performance penalty despite it permits to reduce of 4× the space needed for coding the leaves. On the other hand, the models using 4-bit representations perform slightly worse than the full precision ones, showing also a statistically significant difference. Considering the difficulties in working with binary representations smaller than a single byte, hereinafter we will consider models with 64 leaves and 8-bit quantization only. These fine-tuned models have 559 and 649 trees (out of the 800 maximum trees), for the MSN30K and Istella-S datasets, respectively.

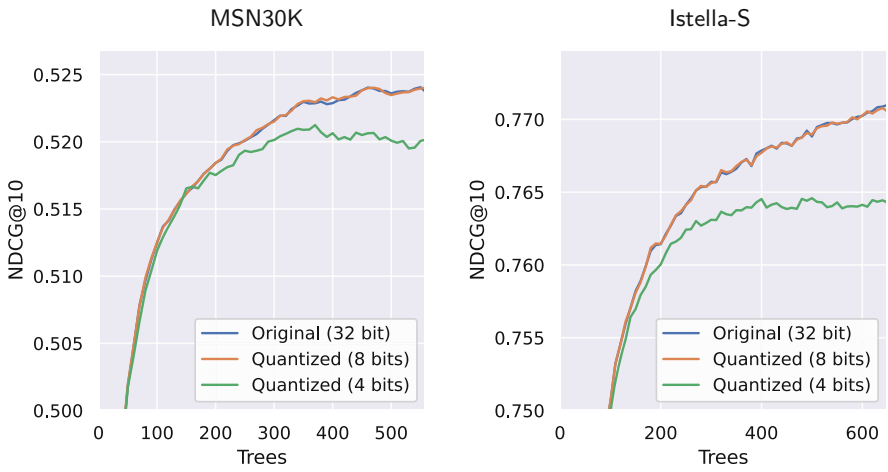

Fig. 2. Impact of quantization on NDCG@10 of λ-MART models.

5.2 Efficiency Assessment

We assess the impact of the proposed binning and quantization techniques on the efficiency of the scoring process by using two versions of QS: the original version [20] and a new one supporting binned and quantized models. Both the versions have been deployed on a high-end multi-core CPU and a SoC-FPGA device. The multi-threaded CPU version [17] runs on a server machine running Ubuntu 20.04 LTS and equipped with two Intel Xeon CPU E5-2630 v3 clocked at 2.40 GHz, 120 GB of RAM. The CPU exploits three levels of cache: 32 KB + 32 KB of L1 cache (data + instructions), 256 KB of L2 cache, and 20,480 KB of L3 cache. The code was compiled with GCC 7.5 with the `-O3` optimization flag. We also implemented the same two versions of QS on our SoC-FPGA device by using Vivado HLS 2019.2.1 to directly convert the annotated and optimized C++ code into Register Transfer Level (RTL) code for the FPGA logic.[4] We tested the FPGA implementation on a Zynq UltraScale+MPSoC ZCU102 device with a quad-core ARM CortexTM-A53 processor, dual-core Cortex-R5 real-time processor and Mali-400 MP2 graphics processing unit. The UltraScale FPGA consists of a PS and a PL block integrated on a single die and running independently.

We present the efficiency achieved by our QS deployments on CPU and FPGA for the MSN30K and Istella-S datasets in Table 3. Results are reported in terms of per-instance average inference time measured in µsecs by varying the replication factor (up to 12). In the case of the FPGA this indicates how many times we replicate the PEs inside the IP block (see Fig. 1). For the CPU implementation, it indicates instead the number of parallel threads used to predict the scores. In both cases the execution time is measured on the whole test

[4] Code available at https://github.com/hpclab/model_compression_for_ranking_on_fpga.

Table 3. Per-instance average inference time (µsec).

Dataset	QS version	Replication Factor				
		1	2	4	8	12
MSN30K	FPGA-FP	944	502.8	252.4	-	-
	FPGA-BQ	10.8	9.1	4.4	3.3	3.1
	CPU-FP	14.2	7.16	4.15	2.29	1.23
	CPU-BQ	12.8	6.45	3.88	2.15	1.10
Istella-S	FPGA-FP	1072	601.5	338.2	-	-
	FPGA-BQ	9.9	6.7	4.9	4.1	3.8
	CPU-FP	17.5	8.79	5.13	2.87	1.34
	CPU-BQ	16.1	8.11	4.71	2.63	1.24

set and then divided by the number of instances in the test set. The FPGA Full-Precision version (FPGA-FP in Table 3), i.e., the one that do not exploit binning and quantization, obtains an average instance scoring time of 252.4 µs for the MSN30K and 269.2 µs for the Istella-S with 4 replicas. We are not able to increase further the replication factor due to the over-utilization of the resources. The model compression techniques detailed in Sect. 4 allow instead to increase the number of replicas of the scoring logic up to 12. Moreover, the QS version exploiting binning and quantization also improve significantly the inference time due to the many optimizations introduced.

We report the results achieved with the optimized version (FPGA-BQ), which includes loop unrolling and additional if-else statements splitting the QS loops in fixed-size blocks of instructions processed in parallel by the FPGA hardware. By introducing all these optimizations, QS shows a significantly improved average scoring time ranging from 10.8 to 3.1 µs for MSN30K and from 9.9 µs to 3.8 µs for Istella-S. In both cases with 12 replicas we measure a resource utilization exceeding 82%. On the other side, the multi-threaded CPU version of QS that uses binning and quantization (CPU-BQ) techniques also outperforms the version implemented without these techniques (CPU-FP), thus proving that model compression is advantageous even on traditional hardware. Overall, with 8 or more threads running on different cores, the CPU-BQ version obtains lower scoring time than the optimized FPGA-BQ version. However, these slightly higher inference times are counterbalanced by a much lower power consumption as discussed in Sect. 5.4.

5.3 Data Transfer Assessment

The experimental evaluation reported in Table 3 does not include the time needed to transfer data from the host machine performing the retrieval of the candidate documents to the SoC-FPGA device aimed at re-ranking the list of candidates to produce the final results. We experimentally evaluated the impact of the data transfer by conducting additional tests with the MSN30K dataset on

an instance of Amazon AWS EC2 F1 node equipped with Xilinx UltraScale+ VU9P FPGAs. These devices are in fact connected to the host machine via a dedicated PCIe Gen3 $\times 16$ bus supporting data transfer with a maximum bandwidth of up to 16 GB/s. We used the AWS instance above for running a simulation measuring the actual bandwidth available when transferring: i) batches of query-document feature vectors from the AWS Amazon EC2 node to the SoC-FPGA, and ii) the resulting scores back from the FPGA to the server.

The result of this simulation shows that the PCIe interface connecting the host machine and the SoC-FPGA device allows to transfer each vector of 136 features, represented with only 136 bytes thanks to our lossless binning technique, with an average latency of 0.24 µs. This latency, although very low, is not an additional overhead to be included in the whole query processing system. Indeed, in a real-world scenario the host device and the SoC-FPGA device operate in pipeline. The host is aimed at retrieving candidate documents from the index and computing query-document feature vectors. The FPGA is instead responsible of inferring the final score to be assigned to each document by using the compressed ensemble model and the QS algorithm. The two operations can be easily pipelined. The resulting scores packed and returned back to the host device can be managed in a similar way. All these query processing operations can be thus overlapped during execution, and the final throughput is given by the slower stage of the pipeline. Since the data transfer time is one order of magnitude lower than the inference on the FPGA (0.24 µs $\ll$ 3.1 µs) we can conclude that the impact of transferring data from the host to the FPGA and viceversa is negligible on the total latency in a real-world production system.

5.4 Energy Consumption Assessment

We present the results of our energy consumption analysis for running on CPU and FPGA the inference task in Table 4. We report the results as the average energy (in μJoule) spent by the QS algorithm for scoring one single instance. For CPU implementations, we perform the analysis by employing the Mammut[5] library [28]. We use Mammut to read the total energy consumption of the CPU exposed by means of hardware energy counter registries available on Intel CPU architectures. All energy measures obtained are discounted by the energy spent by cores not used by the scoring process. For the FPGA implementation, we employ the Maxim Power Tool USB-to-PMBus Interface Dongle.[6] All energy measures obtained include the energy spent by all components in the PS and in the PL. The PL frequency is set at 200 MHz. Results in Table 4 show that the FPGA-BQ deployment significantly reduces on both datasets the energy consumption measured with respect to the CPU implementations. Specifically, on FPGA we measure an energy consumption for inference that is from 9$\times$ up to 19.8$\times$ lower than on the CPU. This large difference, consistent with measures

[5] https://github.com/DanieleDeSensi/mammut.

[6] https://www.maximintegrated.com/en/products/power/switching-regulators/maxpowertool002.html.

Table 4. Per-instance average energy consumption (μJoule).

Dataset	QS version	Replication Factor				
		1	2	4	8	12
MSN30K	FPGA-BQ	37	35	16	14	13
	CPU-BQ	492	316	239	180	173
Istella-S	FPGA-BQ	31.1	22	17.2	17.6	16.9
	CPU-BQ	616	398	294	224	217

reported in literature for other computing tasks [27], show that an accurate design of the most demanding components of the ranking pipeline with FPGA technology can impact significantly web-scale systems where energy is a major source of cost.

6 Conclusions and Future Work

Modern programmable logic provide an interesting energy-aware alternative to traditional servers for several high-performance tasks. In this paper we tackled the exploitation of SoC-FPGA devices for demanding inference tasks based on complex machine-learned models. We proposed to use binning and quantization to compress additive ensemble of decision trees and increase the number of inference tasks processed in parallel on the FPGA logic. The use case considered was ad-hoc retrieval with fully-optimized LtR models, where finding the best trade-off between accuracy and efficiency is definitely important. Reproducible experiments show that our model compression techniques do not impact the prediction accuracy in a statistically significant measure and that the deployment of a ranking solution based on state-of-the-art algorithms on a low-cost SoC-FPGA device achieves scoring times comparable to those measured on high-end multicore CPUs. We also showed that the data transfer supplied by the PCIe interface connecting the FPGA to the host machine contribute with a negligible latency with respect to the one of the scoring. On the other hand, the SoC-FPGA solution consumes one order of magnitude less energy in performing the inference. This result can impact significantly large-scale systems where inference is a key task and energy is a major source of cost.

As future work we will investigate the exploitation of SoC-FPGA architectures in the design of optimized algorithms for other inference tasks possibly benefiting from programmable logic. Specifically for the IR domain, we will study the feasibility of this technology for neural ranking.

Acknowledgements. This work was partially supported by the project HAMLET: Hardware Acceleration of Machine LEarning Tasks, funded by CONICET (Argentina) and CNR (Italy) 2017-2018 collaboration program, by the TEACHING project, funded by the EU Horizon 2020 Research and Innovation program (Grant agreement ID: 871385), and by the OK-INSAID project, funded by the Italian Ministry of Education and Research (GA no. ARS01_00917).

References

1. Asadi, N., Lin, J.: Training efficient tree-based models for document ranking. In: Serdyukov, P., et al. (eds.) ECIR 2013. LNCS, vol. 7814, pp. 146–157. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-36973-5_13
2. Bergstra, J., Yamins, D., Cox, D.: Making a science of model search: hyperparameter optimization in hundreds of dimensions for vision architectures. In: Proceedings of ICML, pp. 115–123. PMLR (2013)
3. Burges, C.J.: From ranknet to lambdarank to lambdamart: an overview. Learning **11**(23–581), 81 (2010)
4. Busolin, F., Lucchese, C., Nardini, F.M., Orlando, S., Perego, R., Trani, S.: Learning early exit strategies for additive ranking ensembles. In: Proceedings of SIGIR, pp. 2217–2221. ACM (2021)
5. Cambazoglu, B.B., et al.: Early exit optimizations for additive machine learned ranking systems. In: Proceedings of WSDM, pp. 411–420. ACM (2010)
6. Capannini, G., Lucchese, C., Nardini, F.M., Orlando, S., Perego, R., Tonellotto, N.: Quality versus efficiency in document scoring with learning-to-rank models. Inf. Process. Manag. **52**(6), 1161–1177 (2016)
7. Chapelle, O., Chang, Y.: Yahoo! Learning to rank challenge overview. J. Mach. Learn. Res. **14**, 1–24 (2011). Proceedings Track
8. Chen, R.C., Gallagher, L., Blanco, R., Culpepper, J.S.: Efficient cost-aware cascade ranking in multi-stage retrieval. In: Proceedings of SIGIR, pp. 445–454. ACM (2017)
9. Chen, T., Guestrin, C.: XGBoost: a scalable tree boosting system. In: Proceedings of ACM SIGKDD, pp. 785–794. ACM (2016)
10. Dato, D., et al.: Fast ranking with additive ensembles of oblivious and non-oblivious regression trees. ACM Trans. Inf. Syst. **35**(2), 15:1–15:31 (2016)
11. Friedman, J.H.: Greedy function approximation: a gradient boosting machine. Ann. Stat. **29**, 1189–1232 (2000)
12. Gallagher, L., Chen, R.C., Blanco, R., Culpepper, J.S.: Joint optimization of cascade ranking models. In: Proceedings of WSDM, pp. 15–23. ACM (2019)
13. Gao, R., Hsu, F.H.: An FPGA-based accelerator for LambdaRank in web search engines. ACM TRETS **4**, 1–19 (2011)
14. Han, S., Mao, H., Dally, W.J.: Deep compression: compressing deep neural network with pruning, trained quantization and Huffman coding. In: Proceedings of ICLR (2016)
15. Järvelin, K., Kekäläinen, J.: Cumulated gain-based evaluation of IR techniques. ACM Trans. Inf. Syst. **20**(4), 422–446 (2002)
16. Ke, G., et al.: LightGBM: a highly efficient gradient boosting decision tree. In: Proceedings of NIPS, pp. 3149–3157 (2017)
17. Lettich, F., et al.: Parallel traversal of large ensembles of decision trees. IEEE TPDS **30**(9), 2075–2089 (2019)
18. Li, Q., Wang, E., Fleming, S.T., Thomas, D., Cheung, P.: Accelerating position-aware top-k ListNet for ranking under custom precision regimes. In: Proceedings of FPL, pp. 81–87 (2019)
19. Liu, T.Y.: Learning to rank for information retrieval. Found. Trends Inf. Retr. **3**(3), 225–331 (2009)
20. Lucchese, C., Nardini, F.M., Orlando, S., Perego, R., Tonellotto, N., Venturini, R.: QuickScorer: a fast algorithm to rank documents with additive ensembles of regression trees. In: Proceedings of SIGIR, pp. 73–82. ACM (2015)

21. Lucchese, C., Nardini, F.M., Orlando, S., Perego, R., Silvestri, F., Salvatore, T.: X-CLEaVER: learning ranking ensembles by growing and pruning trees. ACM TIST **9**, 1–26 (2018)
22. Lucchese, C., Nardini, F.M., Orlando, S., Perego, R., Silvestri, F., Trani, S.: Post-learning optimization of tree ensembles for efficient ranking. In: Proceedings of SIGIR, pp. 949–952 (2016)
23. Lucchese, C., Nardini, F.M., Orlando, S., Perego, R., Tonellotto, N., Venturini, R.: Exploiting CPU SIMD extensions to speed-up document scoring with tree ensembles. In: Proceedings of ACM SIGIR, pp. 833–836 (2016)
24. Lucchese, C., Nardini, F.M., Orlando, S., Perego, R., Trani, S.: X-DART: blending dropout and pruning for efficient learning to rank. In: Proceedings of SIGIR, pp. 1077–1080. ACM (2017)
25. Lucchese, C., Nardini, F.M., Orlando, S., Perego, R., Trani, S.: Query-level early exit for additive learning-to-rank ensembles. In: Proceedings of SIGIR, pp. 2033–2036. ACM (2020)
26. Molina, R., Loor, F., Gil-Costa, V., Nardini, F.M., Perego, R., Trani, S.: Efficient traversal of decision tree ensembles with FPGAs. J. Parallel Distrib. Comput. **155**, 38–49 (2021)
27. Qasaimeh, M., Denolf, K., Lo, J., Vissers, K., Zambreno, J., Jones, P.H.: Comparing energy efficiency of CPU, GPU and FPGA implementations for vision kernels. In: Proceedings of IEEE ICESS, pp. 1–8 (2019)
28. Sensi, D.D., Torquati, M., Danelutto, M.: Mammut: high-level management of system knobs and sensors. SoftwareX **6**, 150–154 (2017)
29. Smucker, M.D., Allan, J., Carterette, B.: A comparison of statistical significance tests for information retrieval evaluation. In: Proceedings of CIKM. ACM (2007)
30. Wang, L., Lin, J., Metzler, D.: Learning to efficiently rank. In: Proceedings of SIGIR, pp. 138–145. ACM, New York (2010)
31. Wu, Q., Burges, C., Svore, K., Gao, J.: Adapting boosting for information retrieval measures. Inf. Retrieval **13**, 254–270 (2010). https://doi.org/10.1007/s10791-009-9112-1
32. Xin, J., Tang, R., Yu, Y., Lin, J.: BERxiT: early exiting for BERT with better fine-tuning and extension to regression. In: Proceedings of ACL, pp. 91–104. ACL, April 2021
33. Xu, N.Y., Cai, X.F., Gao, R., Zhang, L., Hsu, F.H.: FPGA acceleration of RankBoost in web search engines. ACM TRETS **1**(4), 1–19 (2009)
34. Ye, T., Zhou, H., Zou, W.Y., Gao, B., Zhang, R.: RapidScorer: fast tree ensemble evaluation by maximizing compactness in data level parallelization. In: Proceedings of SIGKDD, pp. 941–950. ACM (2018)