High-Throughput FPGA-Based Inference of Gradient Tree Boosting Models for Position Estimation in PET Detectors

Karl Krueger, *Member, IEEE*, Florian Mueller, *Member, IEEE*, Pierre Gebhardt, Bjoern Weissler, David Schug, and Volkmar Schulz, *Senior Member, IEEE*

***Abstract*—In modern large-scale positron emission tomography (PET) systems, transferring the digitized raw detector data at high count rates to a centralized processing unit is a challenge. Processing data on field programmable gate arrays (FPGAs) close to the detectors can reduce data early on and improve scalability of the PET system. We present and evaluate an FPGA implementation of gradient tree boosting (GTB) for 1-D position estimation of gamma interactions in the scintillator. GTB is a supervised machine learning algorithm based on building ensembles of binary decision trees. Models were trained offline and inferred in an FPGA (XC7K410T-2FFG676 Kintex-7). Input features and GTB parameters influencing both positioning performance and model size were varied while evaluating the inferred models concerning data throughput and FPGA resource consumption as well as positioning performance. We achieved throughputs per detector between 2.94×10^6 and 4.55×10^6 gamma interactions per second. For an optimized GTB model, resource consumption could be reduced by factors of 17 and 10 to less than 1 % (2.51×10^3 look-up tables) of available logic and 1.26 % (20 BRAMs) of memory resources, while maintaining a positioning performance of 98.63 % when compared to the model with the best positioning performance. The presented framework can be easily adapted to other photosensors and scintillator influencing.**

***Index Terms*—Data processing, decision trees, field programmable gate arrays (FPGAs), gradient tree boosting (GTB), positron emission tomography (PET).**

I. Introduction

POSITRON emission tomography (PET) is a functional, tracer-based imaging modality, offering a high sensitivity for the imaging of metabolic processes and is used widely in clinical and preclinical applications [1], [2]. A positron, emitted from a radioactively labeled tracer, annihilates with an electron, emitting two 511-keV gamma photons. Scintillation crystals convert these gamma photons into optical photons, which are detected by photosensors, typically silicon photomultipliers (SiPMs).

To perform image reconstruction, information about the timing, energy, and spatial position of the gamma photon interaction needs to be calculated from the acquired raw data. The raw data can consist of the individual channel responses of the photosensor [3] or the outputs of a multiplexing scheme between the individual channels (e.g., row and column summing [4], [5]). The data processing can be done either using dedicated processing servers [6], [7] or directly within the hardware of the data acquisition system [8], [9], [10], [11]. In the first case, this requires either large data storage solutions in case of offline or large server solutions in case of online processing. Especially in total-body PET [12], [13], [14] with its large number of detectors, large amounts of data would have to be stored and/or processed in servers, increasing system costs significantly. Instead, the field programmable gate arrays (FPGAs) used to collect data from a single or from multiple detectors in the data acquisition system can be used for the necessary processing steps for timing, energy, and position estimation. Since costs for FPGAs with high processing power can be significant, algorithms for data processing need to be efficient to be implementable in small and therefore cheap FPGAs. Furthermore, processing the PET raw data in FPGAs reduces the amount of data early in the data acquisition system and thereby reduces the system complexity needed to transfer large amounts of data.

This work focuses on one aspect of the aforementioned necessary processing steps: the estimation of the interaction position of the gamma photon in the scintillator, which is needed to accurately determine the annihilation point of the positron.

Manuscript received 21 November 2022; revised 10 January 2023; accepted 13 January 2023. Date of publication 23 January 2023; date of current version 3 March 2023. This work was supported in part by the Helmholtz Validation Fund under Grant 0051, and in part by the European Union's Horizon 2020 Research and Innovation Programme under Grant 667211. *(Corresponding author: Karl Krueger.)*

This work did not involve human subjects or animals in its research.

Karl Krueger and Florian Mueller are with the Department of Physics of Molecular Imaging Systems, Institute for Experimental Molecular Imaging, RWTH Aachen University, 52074 Aachen, Germany (e-mail: karl.krueger@pmi.rwth-aachen.de).

Pierre Gebhardt is with the Department of Physics of Molecular Imaging Systems, Institute for Experimental Molecular Imaging, RWTH Aachen University, 52074 Aachen, Germany, and also with Bruker Biospin GmbH, 76275 Ettlingen, Germany.

Bjoern Weissler and David Schug are with the Department of Physics of Molecular Imaging Systems, Institute for Experimental Molecular Imaging, RWTH Aachen University, 52074 Aachen, Germany, and also with Hyperion Hybrid Imaging Systems GmbH, 52074 Aachen, Germany.

Volkmar Schulz is with the Department of Physics of Molecular Imaging Systems, Institute for Experimental Molecular Imaging, and the Physics Institute III B, RWTH Aachen University, 52074 Aachen, Germany, also with Hyperion Hybrid Imaging Systems GmbH, Aachen, Germany, and also with the Fraunhofer Institute for Digital Medicine MEVIS, 52074 Aachen, Germany.

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TRPMS.2023.3238904.

Digital Object Identifier 10.1109/TRPMS.2023.3238904

Different methods have been developed to estimate this positin, depending on the scintillator/photosensor configuration. In one-to-one coupled scintillators, the planar interaction position is either directly associated with the individual readout channels or can be obtained from look-up tables (LUTs) in the case of multiplexed signals. In high-resolution scintillators, the planar interaction position can be calculated from the light distribution using, e.g., center-of-gravity (CoG or Anger) algorithms [15], [16], [17]. Depth-of-interaction (DOI) information can be obtained by, e.g., staggered designs and CoG [18] or neural networks [19]. In monolithic detectors, the interaction in planar and DOI direction can be estimated using, amongst others, maximum-likelihood searches [20], [21], neural networks [22], [23] or gradient tree boosting (GTB) [24], [25].

Efforts have been made to adapt FPGA implementations of more complex positioning algorithms, such as maximum-likelihood position estimation [26], [27], [28] or neural networks [29], [30], to overcome the limitations of simpler methods like one-to-one coupling or CoG [31]. Many of these adaptations suffer from a high memory requirement, making them unfeasible for smaller (and cheaper) FPGAs, or a low throughput, requiring further processing before positioning. For early data reduction, the position estimation should be performed at an early stage, using small FPGAs close to the detectors, where higher throughputs are required and FPGA memory and logic resources are scarce.

Recently, our group has shown that the GTB algorithm provides high spatial resolution in planar [24] as well as in DOI [25] direction and could be optimized concerning its complexity and computation cost, which enabled high throughputs in a software-based inference [32].

Here, we aim at developing an FPGA-based inference of GTB that is designed to be integrated as one step of the required data processing in a PET data acquisition system. As such, the implemented GTB framework must meet the aforementioned requirements of high data throughputs and a low consumption of FPGA resources, so that it can be included in the (optimally) small readout FPGAs close to the detectors of the PET system. Previous work on FPGA implementations of GTB for different applications exists. Some of these works implement models of fixed sizes for a specific applications [33], [34], [35]. Others optimize different models for a high throughput and a low latency [36], [37], [38], [39]. All implementations have in common that they are designed as stand-alone FPGA algorithms, which is why their main focus is not on the FPGA resource consumption. In contrast, we show a combination of high throughputs and low resource consumption. The latter is achieved by executing different steps of the GTB algorithm in the same FPGA logic and pooling their necessary information in the same memory spaces. High throughputs are realized by executing large parts of the model in parallel. We further investigate the tradeoff between resource consumption and positioning performance for different model configurations to show the feasibility of including GTB in small FPGAs of large-scale PET systems.

The implementation in this work is shown with an example of a photosensor based on digital SiPMs and a high-resolution scintillator array. Since the data used for training the GTB models is not specific to the detector composition, the implementation can be easily adapted to other detector types, such as analog SiPMs and/or monolithic scintillators.

II. Materials and Methods

A. Data Acquisition

A detailed description of the detector used for data acquisition can be found in [3] and [17], therefore only a short summary is given here. The detector consists of a high-resolution scintillator coupled to a sensor tile via a lightguide. The scintillator array is made up out of 30 × 30 LYSO crystal segments with a pitch of 1 mm and a height of 12 mm. The array was mounted on a 2-mm thick glass plate to spread light onto a 32.6 × 32.6 m^2m sensor tile. The sensor tile consisted of 16 digital SiPMs (PDPC DPC 3200-22) [40], [41]. Each DPC comprised 2 × 2 readout channels, resulting in a total of 64 channels. Every DPC provides an independent, customizable two-level trigger scheme, that was set to trigger on 2.33 and 17 photons for the first and second threshold, respectively. The collected information from all channels of the triggered DPCs corresponding to one gamma photon interaction is called an event.

GTB needs training data to build predictive models (see Section II-B). This data as well as test data for evaluation were acquired in a benchtop coincidence calibration setup [42]. The sensor tile was placed on a translation stage and the scintillator array was irradiated in both planar directions with a fan-beam collimator by a gamma photon beam created by a slit of 0.4-mm width with two ^{22}Na sources, each with an activity of 5.5 MBq. The translation stage was moved 1 mm at a time, so that every row or column of scintillator segments was irradiated centrally. We acquired a total of 600 000 events as the training data and 180 000 events as the test data.

The acquired raw data was preprocessed using the tool developed and described in [17]. A time window of 40 ns was used to form events from the individual readout channel photon counts. The time stamp for each event was determined as the time stamp of the earliest channel that was read out for this event. While this is currently done in software, an FPGA implementation is under development. A sliding coincidence window of 10 ns was applied to find coincident events. No further preprocessing steps were applied to the raw photon counts. Since each DPC can trigger independently, not every channel sends out data for each event. Such missing photon counts were set to −1, so that every event consisted of 64 values representing the 64 channels of the detector.

It is noteworthy that the training and test data after preprocessing is not specific to the used photosensor architecture. Each event only consists of a collection of the timely correlated outputs of the triggered channels of the photosensor that represent the measured light from one gamma interaction in the scintillator. We chose the above-mentioned photosensor architecture with a pixelated scintillator, as it provides an easy to understand measure to evaluate the FPGA implementation of GTB models (see Sections II-B–II-D) and allows a comparison to a previous work on software-based inference of GTB [32].

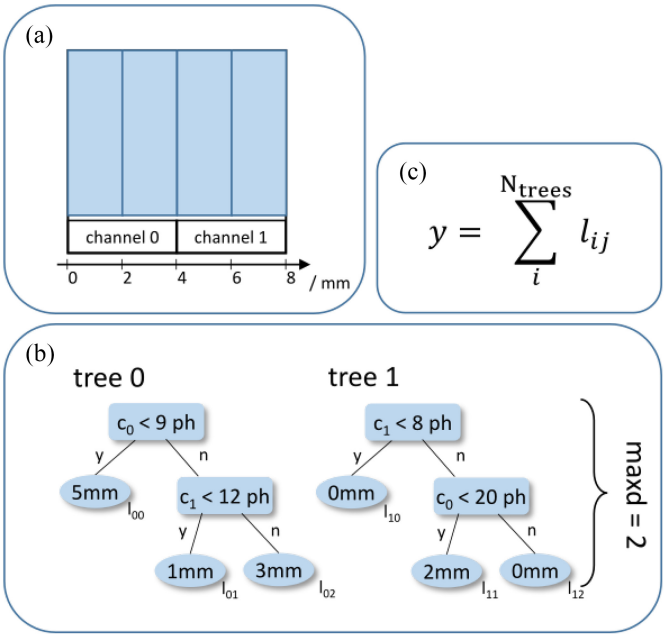

Fig. 1. (a) Simple example of a 1-D detector with two channels and four scintillator segments and (b) GTB model for predicting the interaction position. The model consists of two decision trees ($N_{\text{trees}} = 2$), which has a maximum depth (maxd) of 2 and the photon counts of the two channels (c_0, c_1) as its input features. In each node (rounded rectangular shape), one of the input features is compared to a trained value, each leaf (oval shape) represents a position output. (c) Prediction y is the sum of the reached leaf values l_{ij} from each tree.

B. Gradient Tree Boosting

GTB was introduced in detail for planar and DOI positioning of the gamma photon interaction in the scintillator in [24] and [25]. Therefore, only its main features and the characteristics important for an FPGA inference are presented here. GTB is a supervised machine learning algorithm that builds predictive models based on a set of training data (in this case the measured events with their known irradiation positions). A GTB model consists of an ensemble of binary decision trees (see Fig. 1 for an example tree), where each tree can be seen as a sequence of simple tests with two possible outcomes [43], [44]. The ensemble is built by adding new decision trees sequentially. The first tree is trained on the irradiation position, each subsequent tree seeks to minimize the error of the estimated position versus the irradiated position of the former ensemble. We used RMSE as training loss for the objective function. The resulting model can be used as a predictor by iterating a data point through all trees of the ensemble. Single trees are independent of each other and can thus be evaluated in parallel. The prediction for the event under test is then the sum of predictions of the individual decision trees. In this case, the output value is a 1-D prediction of the interaction position in the scintillator. Therefore, the prediction of the spatial position needs a separate model for each direction.

To investigate tradeoffs between positioning performance and model size, and therefore FPGA resource consumption, we varied the following hyperparameters (parameters that control the learning process) for GTB model training.

TABLE I
SETS OF INPUT FEATURES

Features	# of Feat.	Description
raw	64	raw channel photon counts
raw + $mainChan$ + $mainDPC$	66	raw + channel with highest ph. count + DPC contain. $mainChan$
raw + CoG + $phSum$	67	raw + center of gravity + photon sum
$rowColSum$	16	row and column sums of photon counts
$rowColSum$ + CoG + $phSum$	19	row and column sums + CoG + photon sum

1) *Maximum Depth:* The maximum number of comparisons in a decision tree from the root node to a leaf (the example tree in Fig. 1 has a maximum depth of two).
2) *Number of Decision Trees:* The number of decision trees in the GTB model.
3) *Learning Rate:* The learning rate is a factor that weighs the ensemble error prior to adding a new decision tree in each training step.

Models of larger maximum depth and higher number of decision trees generally show an improved performance for positioning of gamma photons [24] (before overfitting occurs). At the same time, increasing the model size increases the computational and memory requirements. The number of nodes and leaves (N_{nodes}, see Fig. 1) in a GTB model is dependent on the maximum depth (maxd) and the number of decision trees (N_{trees}) as follows:

$$N_{\text{nodes}} = N_{\text{trees}} * \left(2^{\text{maxd}+1} - 1\right). \quad (1)$$

A high learning rate gives large influence to single trees, which can lead to a good performance for models with a small number of decision trees, but to overfitting for larger models. GTB models can be trained with arbitrary input features. We used the raw photon counts of the sensor's readout channels in combination with calculated features based on the light distribution to investigate their effect on positioning accuracy and, by means of reducing the number of input features, on resource consumption. Based on the hyperparameter space investigated and the results obtained in [24] and [25], we trained models with combinations of maximum depths {4, 6, 8}, number of trees {20, 60, 100, 140, 180, 220}, learning rates {0.1, 0.2, 0.4, 0.7}, and sets of input features (see Table I). We calculated the additional features *mainChan*, *mainDPC*, *CoG*, *phSum*, *rowColSum* for two reasons. Many detector designs do not digitize each photosensor channel individually, but reduce the number of input signals by, e.g., row and column summing. Furthermore, the calculated features can contain more information per feature about the gamma interaction position than the raw photon counts and can therefore be beneficial for positioning performance [24], [25].

GTB models were trained as regression models that output a continuous estimation of the gamma interaction in the scintillator in one spatial direction. Since the data labels (the irradiated positions) are limited to the irradiated scintillator segment and thus form a discrete system, the output values were mathematically rounded. Model training was performed offline with the 600 000 events of the training data set and the

TABLE II
PARAMETERS OF THE XC7K410T-2FFG676 KINTEX-7 FPGA [47]. EACH BLOCK RAM (BRAM) COMPRISES 18 kb OF MEMORY

LUTs	Register	DSP slices	Total RAM (kb)	BRAMs
254 200	508 400	1540	28 620	1590

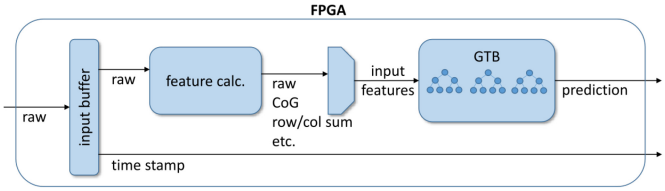

Fig. 2. Schematic overview of FPGA implementation. Raw data is loaded from a control PC. Additional features are calculated and the right set of input features is fed to the GTB model. The prediction output and a time stamp are sent back to the control PC.

known irradiation positions as data labels using the framework *XGBoost* [45] and only the resulting models were implemented and tested on an FPGA. Each event contains the input feature values for the chosen feature set in Table I, which are directly calculated from the measured raw photon counts of the readout channels. Model building is analogous to [32], where the same detector block was used, and to [24], where models were built in the same way for a monolithic detector block. These works describe the training and building of GTB models in detail.

C. FPGA Implementation

The FPGA inference of GTB models was developed using the hardware description language VHDL which gives the most flexibility during the implementation process. For further flexibility and the possibility to implement models over a wide range of hyperparameters, the implementation, including the calculation of features from the raw photon counts and the GTB models, was done as a stand-alone algorithm on the FPGA and not inside a PET data acquisition architecture. We used a custom-made board, designed as the mainboard in our PET data acquisition and processing platform [46] and equipped with an XC7K410T-2FFG676 Kintex-7 FPGA. Table II shows the available logic and memory resources of the FPGA.

The raw photon counts of the gamma events of the test data were loaded into the FPGA from a control PC via a 10 Gbit Ethernet connection and stored inside input buffers. Once the buffers were filled, their data were fed to the positioning framework as fast as the framework could accept new inputs. A time stamp was generated each time the model accepted a new data point. The prediction output generated by the framework was sent back to the control PC together with the time stamp. A schematic of this data flow in the FPGA is shown in Fig. 2. The positioning framework was implemented at a clock frequency of 100 MHz. We chose this frequency for two reasons: 1) it is commonly used in our data acquisition system to collect raw sensor data and 2) the higher the frequency, the more difficult the routing between the configurable logic of the FPGA becomes during synthesis. We considered 100 MHz as a rather conservative frequency choice, where no routing issues were expected.

In the case of the *raw* feature set, the 64 raw photon counts were fed to the GTB models as input features directly. In all other cases, additional features were calculated in the FPGA first. The main channel, main DPC as well as the photon sum, row sums and column sums were calculated in the FPGA logic using LUTs. For the CoG calculation, the necessary multiplications were moved into dedicated digital signal processing (DSP) slices. The division was implemented using the Divider Generator v5.1 from Xilinx [48]. All features were calculated in parallel and in each clock cycle, photon counts from four channels could be processed. Thus, it took 16 clock cycles to calculate features for a gamma event with 64 photon counts.

During the offline training, all model parameters were represented as floating point values. These were translated into fixed point values for the FPGA implementation, meaning a fixed number of bits were allocated to represent the fractional part of each value. All feature values (input features and test values) were represented as 13 bit (b) words. The CoG position was represented with nine fractional bits, all other features with one fractional bit. Prediction values were represented as 32 b words with 23 fractional bits. With these bit widths, no prediction loss could be observed compared to the offline inference.

The basis of each node is the comparison of one of the input features to a trained test value. Based on the outcome of this comparison, the tree is traversed to either of the child nodes. In a naive FPGA implementation, each node of every tree in the ensemble could be mapped to FPGA logic. This could maximize throughput as it allows for data streaming and could minimize the usage of on-chip memory, as all node information could be hard-wired in the FPGA logic. However, with (1) and the hyperparameter space described in Section II-B, even the smallest model trained in this work comprises over 600 and the largest one more than 100 000 nodes. Mapping this amount of nodes directly to FPGA logic would exceed the available resources, making this approach unfeasible. Therefore, a different approach that balances resource utilization and throughput was searched for. For each data point, only a subset of nodes in a single tree can be traversed. Corresponding to the path through the tree, only one node per level (see Fig. 1) can be visited in each step. Thus, each tree was implemented by a single node structure at each level, as displayed in the left part of Fig. 3, similar to [33] and [35]. On the one hand, this drastically reduced the number of node structures that needed to be implemented, especially at higher depths (e.g., at a depth of eight, $2^8 - 1 = 255$ nodes could be omitted per tree). On the other hand, the complexity to implement a node structure was increased.

A schematic of a node structure implementation can be seen on the right-hand side of Fig. 3. Iterating a data point through a node structure took three clock cycles with the following operations.

1) *Load:* Node information is loaded to the output of the BRAM based on the provided memory address (memory index).

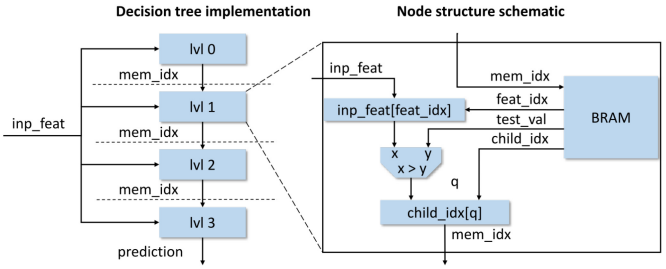

Fig. 3. Left: Implementation of a decision tree with one level per node. Right: Schematic of a node structure. Based on a memory index provided by the previous level, node information are loaded from BRAM. The loaded feature index (*feat_idx*) indicates which input feature is compared to the loaded test value (*test_val*). Based on the outcome of this comparison and the loaded index of where the child nodes are stored in the memory of the following level (*child_idx*), a memory index is output to the next level. In case of a leaf node, the node information consists only of a prediction value, which is loaded and output.

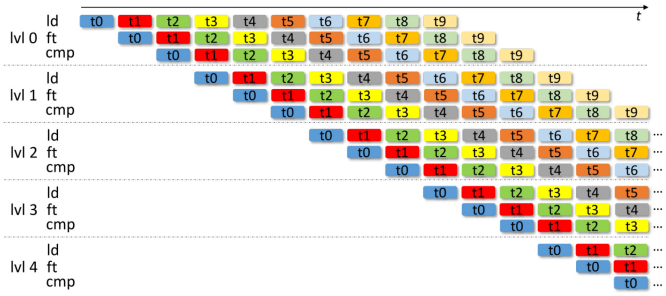

Fig. 4. Operations in a subensemble of a GTB model with maximum depth of 4 for one data point, *ld* = load, *ft* = fetch, *cmp* = compare, *pdt* = predict, *tn* = tree n. The different colors depict operations in different trees of the subensemble. Since operations in a level are independent of each other, the *load* operation of tree 1 could be executed during the *fetch* operation of tree 0 and so forth. The *compare* operation of tree 9 in level 0 finishes before the *ld* operation of tree 0 in level 4. Therefore, the same node structure can be used for level 0 and level 4. Operations continue until tree 9 is iterated through level 4.

2) *Fetch:* Node information could be fetched from the BRAM output.
3) *Compare/Predict:* One of the input features was compared to the test value and the memory index to the following level had to be output. In case of a leaf node, the prediction value had to be output.

These operations could be executed independently of each other, so that a new data point could theoretically still be processed at every clock cycle. To achieve this, the input features of each data point would need to be registered at each level, leading to a high register resource consumption. Instead, the input features of the current data point were directly routed to each node structure. However, the independence of operations in a node structure was used to share one set of node structures between multiple trees of the ensemble and iterate one data point through multiple trees in a pipelined way, as depicted in Fig. 4. We grouped ten decision trees in such a subensemble. In this case, the *compare* operation of level 0 of the last tree was executed before the *load* operation of level 4 of the first tree. It was therefore not necessary to implement a node structure for each level. Instead the node structures that were used for lower levels could be reused for higher levels. This further reduced the amount of logic resources utilized by this implementation.

Each node structure was connected to a dedicated BRAM. In this memory, information of all nodes in the levels that the node structure was used for were stored for all ten trees in one subensemble. Node information included a test value, an identifier, which input feature had to be compared against the test value, and an index, indicating at which memory address the child nodes in the following level were stored, or, in case of a leaf node, only a prediction value. The memory of the Kintex-7 FPGA can be configured to blocks with a minimum size of 18 kb, organized in 1024 18 b memory spaces. In each clock cycle, two memory spaces in one BRAM can be accessed. For each node, 35 b of information needed to be stored (32 b in case of a leaf node), so that each node occupied exactly two memory spaces in BRAM. Reusing node structures of lower levels for higher levels proved highly beneficial for memory usage, as there were only few nodes at lower level and BRAMs would have been largely unoccupied at this level. Storing the node information of multiple levels together, while guaranteeing that two levels never access the same BRAM at the same time and therefore keeping the BRAM output nonambiguous, used the blocks of memory more efficiently.

D. Evaluation

The implemented models were evaluated regarding their performance and resource consumption using the following metrics.

1) *Prediction Accuracy:* The prediction accuracy is defined as the rate of correctly identified scintillator segments, meaning that the predicted interaction position matches the irradiated scintillator segment. A perfect prediction accuracy of one is not achievable due to Compton scatter inside the scintillator array. However, the parameter still provides a simple measure to compare the positioning performance of different models. Furthermore, we included an *extended prediction accuracy*, which is defined as the fraction of events, that are either positioned in the correct scintillator segment or positioned in one of the neighboring scintillator segments.
2) *Mean Absolute Error (MAE):* The MAE is the mean of the absolute positioning error and was included to evaluate the error distribution of the positioning.
3) *Throughput:* The throughput is the number of events that can be positioned per second. It depends on the number of clock cycles until a new event can be accepted by the model and the clock frequency, with which the model is running on the FPGA. The reported throughputs in Section III-E were calculated from the time stamps that were generated each time a new data point was accepted into the model.
4) *Logic Resource Usage:* FPGAs are equipped with a limited amount of LUTs and registers, which are used to implement arithmetic and logic functions. We evaluated the dependence of the utilization of these resources on the size of the GTB models.
5) *Memory Resource Usage:* Apart from logic resources, FPGAs are also equipped with on-chip random access memory. This memory is organized in blocks (BRAMs)

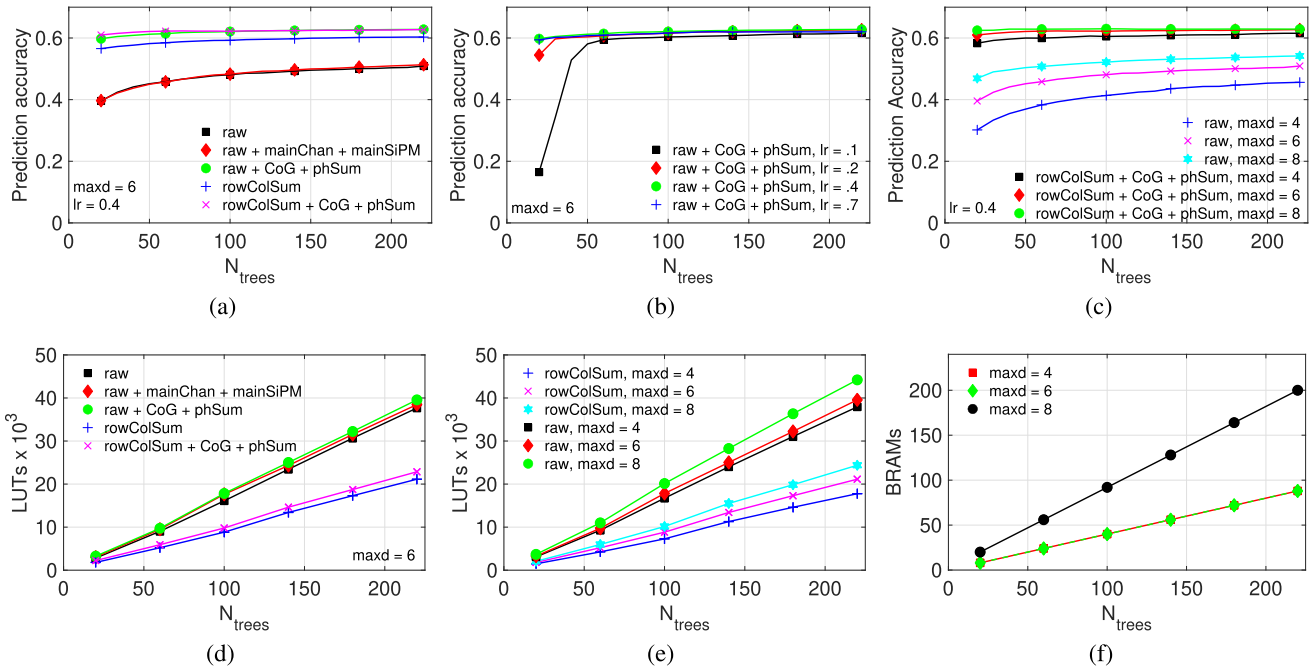

Fig. 5. Exemplary results. For all cases, the evaluated parameter is shown against the number of decision trees (N_{trees}. If shown, resource consumption always includes the resources needed for feature calculation. (a) Prediction accuracy for all feature sets with fixed maximum depth (maxd) and learning rate (lr) of 6 and 0.4, respectively. (b) Prediction accuracy of different learning rates for the feature set *raw* + *CoG* + *phSum* and a fixed maximum depth of 6. (c) Prediction accuracy of different maximum depths for the feature sets *rowColSum* + *CoG* + *phSum* and *raw* for a fixed learning rate of 0.4. (d) LUT consumption of all feature sets for a fixed maximum depth of 6. LUT consumption is independent of the learning rate. (e) LUT consumption of different maximum depths for the feature sets *raw* and *rowColSum*. (f) Memory consumption of different maximum depths. Memory consumption is independent of the learning rate and the feature set.

of fixed size. Single memory address spaces in each BRAM can be accessed in two clock cycles. However, only two memory address space in a BRAM can be accessed at one time.

III. Results

A. Positioning Performance

The prediction accuracy of the different input feature sets is compared in Fig. 5(a). The lowest prediction accuracies in the observed hyperparameter space are observed with the feature sets *raw* and *raw* + *mainChan* + *mainDPC*. Using the calculated row and column sums (*rowColSums*) instead of the raw photon counts (*raw*) improves prediction accuracy between around 10 % and 67 %. However, the positioning performance of input feature sets, including the features CoG and phSum, exceeds that of sets without these features.

Fig. 5(b) shows how the prediction accuracy can be influenced by the learning rate. Especially for smaller models with less than 100 trees, learning rate has a high impact on the prediction accuracy, where a higher learning rate leads to higher prediction accuracies. For larger models, the positioning performance deteriorates slightly for higher learning rates due to overfitting.

Fig. 5(c) shows the prediction accuracy for different depths for the feature sets *raw* and *rowColSum* + *CoG* + *phSum*. Increasing the maximum depths increases the prediction accuracy. This improvement is high for the *raw* feature set (up to 58.80 %), but only small for the feature set based on calculated features (a maximum improvement of 8.32 % from a maximum depth of 4–8 at 20 decision trees).

TABLE III
Logic Resource Consumption for Calculating Additional Features

Feature	LUT	Register	DSP slices
CoG, phSum	340	550	6
rowColSum	262	478	0
mainChan, mainDPC	105	87	0

The best prediction accuracy of the trained models of 0.63 was achieved for a model with 180 trees and a maximum depth of 8, trained with a learning rate of 0.2 and the feature set *raw* + *CoG* + *phSum*. This corresponds to about 96 % of the maximum achievable prediction accuracy, which was found offline at about 0.66 when training much larger models of depths of 12 or 14.

Fig. 6 shows the extended prediction accuracy and the MAE for the different input feature sets. The extended prediction accuracy shows the same behavior as the prediction accuracies with improved performance for row and column sums over raw photon counts and for feature sets, including CoG and phSum. The best extended prediction accuracy was found to be 0.85. The MAE shows the inverse behavior to the prediction accuracy with lower errors for row and column sums over raw photon counts and for feature sets, including CoG and phSum. The lowest MAE achieved was 0.88.

B. Logic Resources

Table III shows the resources consumed for calculating features from the raw photon counts. The results for the individual GTB models shown in the following include the resources

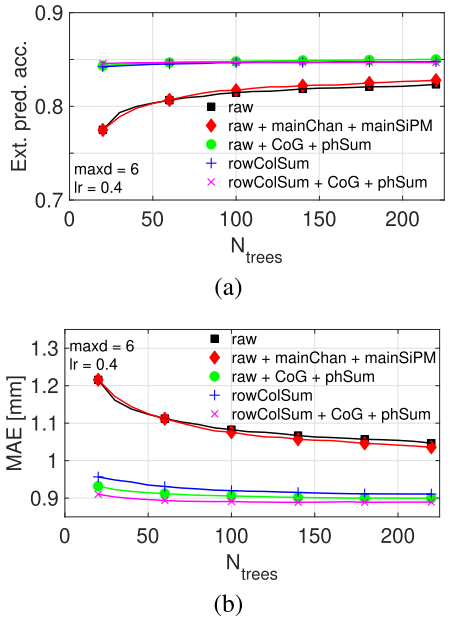

Fig. 6. Exemplary results for (a) extended prediction accuracy and (b) MAE for all feature sets with fixed maximum depth (maxd) and learning rate (lr) of 6 and 0.4, respectively.

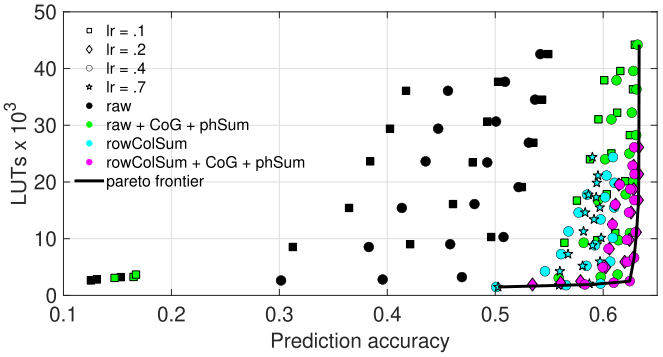

Fig. 7. Prediction accuracy versus LUTs for selected feature sets and learning rates. Different learning rates are distinguished by different shapes, different feature sets by different colors. To keep the figure distinguishable, not all combinations of implemented feature sets and learning rates are shown. For each shown feature set and learning rate, models for all depths and number of trees are shown. The Pareto frontier is shown by the black line.

needed for these calculations. DSP slices are only used for the calculation of the CoG and not for the implementation of the GTB models, therefore a maximum of six (0.39 % of available DSP slices in the Kintex-7) are occupied by the positioning framework.

As can be seen in Fig. 5(d), the number of input features strongly influences the logic resource consumption of GTB models. A lower number of input features decreases the logic resource consumption. For example, using 16 row and column sums instead of 64 raw photon counts brings a reduction between 35.74 % and 52.83 %.

Increasing the maximum depth of the GTB models increases the amount of consumed logic resources [see Fig. 5(e)] due to some overhead when node structures are reused in different levels. Fig. 5(d) and (e) shows that LUT consumption grows linearly with the number of trees. The LUT consumption ranges from 1.48×10^3 LUTs for the smallest model (feature set *rowColSum*, maximum depth 8, 20 decision trees) to 44.21×10^3 LUTs for the largest model (feature set *raw + CoG + phSum*, maximum depth 4, 220 decision trees). This corresponds to 0.58 % and 17.39 % of all available LUTs in the Kintex-7 FPGA.

The utilization of registers is independent of the amount of input features. Register consumption follows a similar linear trend as LUT consumption for an increasing number of decision trees and increases with higher maximum depth. However, the overall register consumption is much lower than LUT consumption, ranging between 0.28 % of all available registers for the smallest and 3.82 % for the largest model.

C. Positioning Performance Versus Logic Resources

Fig. 7 shows the prediction accuracy versus the logic resource consumption for implemented models and the Pareto frontier. Points on the Pareto frontier form a set of optimal solutions, where one parameter cannot be improved without worsening the other [49]. The best prediction accuracy of 0.63 is achieved for a consumption of 44.21×10^3 LUTs. A prediction accuracy of 99.62 % of the best achieved performance can be achieved with 25.20 % (11.14×10^3 LUTs) of the LUT consumption and a prediction accuracy of 98.63 % with 5.67 % (2.51×10^3 LUTs) of the LUT consumption.

TABLE IV
NUMBER OF CLOCK CYCLES UNTIL A NEW EVENT COULD BE PROCESSED AND RESULTING THROUGHPUT AT 100 MHz FOR DIFFERENT MAXIMUM DEPTHS

Maximum Depth	Clock Cycles	Throughput
4	22	4.55 MEvents/s
6	28	3.57 MEvents/s
8	34	2.94 MEvents/s

D. Memory Resources

The number of memory resources used by the GTB models only depends on the number of nodes (and therefore on the maximum depth and the number of decision trees) but not on the number of input features. The amount of 18 kb BRAMs utilized for the implemented models is shown in Fig. 5(f). Memory resources increase linearly with increasing number of decision trees. Models of depths four and six consume the same amount of memory resources. A maximum depth of eight increases the memory consumption by a factor of more than two, however. In total, the memory utilization ranges from 8 BRAMs (0.5 % of available BRAMs) for the smallest model to 200 BRAMs (12.58 %) for the largest model.

E. Throughput

The number of clock cycles until a new input can be processed is dependent on the number of decision trees in each subensemble and the maximum depth of the model. With ten decision trees in each subensemble and a clock frequency of 100 MHz, the throughputs for the three maximum depths are shown in Table IV.

At a clock frequency of 100 MHz, the largest maximum depth of eight still had a throughput of almost 3×10^6 events per second and the smallest maximum depth of four exceeded 4.5×10^6 events per second that could be processed.

F. Power Consumption

No in-depth analysis of power consumption of the design was performed. Instead, power consumption was estimated from the post-implementation reports of the Xilinx Vivado Design Suite software that was used for synthesizing the FPGA design. Estimated power analysis at maximum throughput yielded a power consumption between about 13 mW (1 % of total power consumption of the FPGA design) for a model with 20 trees and about 150 mW (11 % of total power consumption of the FPGA design) for a model with 220 decision trees.

IV. Discussion

The GTB positioning framework presented here is not specific to the employed photosensor. Since GTB can be trained with arbitrary input features, any inputs that carry information about the interaction position in the scintillator can be employed. While we used the outputs of digital SiPMs in this work as inputs to the framework, inputs could also be raw energy values of individual channels of an analog SiPM or, in the case of multiplexed channel signals, e.g., row and column sums. The changes to the positioning framework when a different sensor architecture is used are minimal. For example, the number of inputs or the bit width of single inputs can be set via global parameters.

Most events can be positioned either in the correct scintillator segment (about two thirds) or in the neighboring scintillator segment (about 85 %) with an MAE around 0.85 mm. Since few quality cuts were applied to the acquired data, only a 40-ns window to cluster single channel photon counts and a sliding coincidence window of 10 ns, we did not expect a higher positioning performance.

The choice of features calculated from the raw input values and used as inputs to the GTB model strongly influences both the logic resource consumption and the positioning performance. Since each node structure is shared between multiple nodes and even levels across trees, all input features have to be available at all node structures. At each node structure, the correct input feature must be chosen for comparison to the currently loaded test value. The necessary multiplexing consumes a high amount of resources for a higher amount of input features. By using 16 row and column sums instead of 64 raw photon count values, the logic resource consumption can be reduced by up to 50 % for models with the same number of decision trees and maximum depth.

At the same time, using row and column sums instead of raw photon values improves the positioning performance. The GTB models implemented here have a maximum of eight comparisons per tree. In this range of maximum depths, GTB models profit more from fewer features with more information per feature than from many features, e.g., row and column sums versus raw photon counts. Positioning performance can be improved further by adding a 2-D CoG position as an input feature, as serves as a first estimate of the actual interaction position of the gamma photon and thus contains a lot of information on the value to be predicted. Models including the CoG feature and trained with higher learning rate, which places a larger influence on single trees, can achieve high positioning performances for a small number of trees. These results can lead, compared to the model with the best positioning performance, to a reduction by a factor of 17 in logic and a factor of 10 in memory resources while still achieving more than 98 % of the maximum achievable prediction accuracy (99.5 % of the maximum extended prediction accuracy and a deterioration of MAE of less than 0.035 mm).

It has to be noted that the focus of this work was not to try to maximize the positioning performance of the pixelated scintillator by employing GTB. In fact, similar positioning results might be expected with computationally simpler algorithms, such as CoG, but this comparison remains part of future work. However, we chose the given detector configuration in this work as the results are easy to interpret and it allows a comparison with previous work, where the same models were inferred in a software-based approach [32]. Since the training data for the GTB models is not specific to the detector configuration, in this case, the pixelated scintillator coupled to digital SiPMs, the shown implementation of GTB models, can be easily adapted to different detector types, e.g., monolithic scintillators. In [24] and [25], similar feature sets were used in a software-based GTB positioning approach for monolithic scintillators. The results showed that calculated features, such as row and column sums and CoG, lead to improvements in the positioning performance for models of the same size as shown in this work. These results indicate that the results concerning the reduction in resource consumption with minimal or no loss in positioning performance could be transferred to other scintillator configurations, although this remains to be investigated in detail in further work.

The number of nodes in a tree and in an ensemble grows exponentially with tree depth. However, due to the organization of memory on the FPGA into blocks of RAM, the implementation of models of maximum depths of four and six requires the same amount of memory resources. At a depth of four, large parts of the 18 kb blocks of memory are not occupied. However, due to the parallel execution of different levels, it is not possible to distribute node information into BRAMs more efficiently while still ensuring only one *load* operation happens on the same memory block in one clock cycle, which is necessary to keep the output data of the BRAM nonambiguous. As described in Section II-C, the minimum amount of BRAMs that is needed per subensemble is four. This minimum amount of BRAMs can still store all node information for models of a depth of six. The exponential increase in nodes becomes apparent for models of a depth of eight. Here, each subensemble of ten trees requires three times (12 BRAM) the amount of memory as models of depths four and six.

The models on the Pareto frontier in Fig. 7 form a set of optimal solutions for the tradeoff between resource consumption and positioning performance. Based on the requirements of the specific system, e.g., a PET system with a small FPGA on detector level with few resources to spare, a model with the best possible positioning performance that still fits the resource requirements can be chosen from this set of solutions. Given that the tradeoff in positioning performance is low, a model with a low resource consumption can most likely serve in most PET readout systems to save significant costs. For example, the model with the best positioning performance would consume almost 70 % of logic resources in a smaller Artix-7 100T FPGA,

which could be used for the read-out of individual detectors. This number is not feasible considering the model only predicts a position in one dimension and should be embedded in a framework that handles multiple processing steps. However, with 98.63 % of the best positioning performance, only less than 4 % of logic resources in the Artix-7 100T would be occupied. To allow for flexibility in choosing the right model, the GTB implementation is easily adaptable. Without specific knowledge of hardware description languages, the models can be changed, e.g., by increasing the number of subensembles to increase the number of trees in a model.

As expected, no routing issues occurred during synthesis for any of the implemented models. An estimated power analysis showed a low power consumption, where the dynamic part of the consumption is proportional to the design frequency, for all models. Therefore, an increase in the clock frequency to, e.g., 150 MHz or even 200 MHz, could be possible which would increase throughputs by a factor of 1.5 or 2, respectively. It has to be noted that power consumption was only estimated from the Vivado Design Suite software in a post-implementation vectorless analysis. Power consumption was not analyzed in detail since it is heavily dependent on the overall FPGA design and the positioning framework was implemented as a stand-alone algorithm in this work to maximize flexibility during the implementation.

Previous work showed that the Hyperion II^{D} scanner, which is equipped with 60 detector stacks of the kind used in this work, saturated at a data rate of about 860 MB/s–950 MB/s [6]. Above these thresholds, either the individual detector stacks or the mainboards that collect data from multiple detector stacks start to randomly discard events. We found the average size of a gamma event to be 83.17 B, leading to maximum numbers of 0.18×10^6 to 0.20×10^6 gamma events per second that could be processed per detector stack. A recently developed high-throughput software-based GTB framework achieved maximum data rates around 9.5 GB/s, corresponding to 2.04×10^6 gamma events per second that can be processed per detector stack. Details on the software framework are given in [32]. Our implementation achieves throughputs between 2.94×10^6 and 4.55×10^6 gamma events per second, which are one order of magnitude higher than the saturation rate of the evaluated detector and at least 44 % higher than in the high-throughput software-based approach. More importantly than this increase over the software framework, the FPGA implementation is easily scalable to large systems. The work shown here can be implemented directly on detector level on the FPGAs used for the detector readout, therefore, no additional processing hardware needs to be added when the numbers of detectors is increased. In contrast, a processing server has an upper limit of the amount of data that can be processed. In large systems, such as total-body PET, this either limits the data rate or requires larger (or more) processing servers.

The implemented GTB models output a gamma interaction position in one spatial direction. For planar positioning for the detector shown here, two models have to be implemented per detector. For certain DOI-capable detectors, where the DOI-position is not directly encoded in the planar position, an additional GTB model is necessary for DOI estimation. With two (or three in case of DOI) parallel models, throughput would stay constant, but logic and memory resource consumption would be doubled (tripled). The high throughputs suggest that it might be possible to use the same FPGA logic for all models and position the different directions consecutively. To reduce the memory consumption further, GTB models could be compressed, similar to the method in [50]. Furthermore, it might be possible to use the same GTB model in both planar directions, eliminating half (one third in case of DOI) of the needed memory resources in a consecutive positioning approach. These optimizations are part of further research.

V. Conclusion

In this work, we have presented an FPGA inference of GTB models for the estimation of the gamma interaction in the scintillator in PET detectors. The positioning framework could process throughputs between 2.94×10^6 and 4.55×10^6 events per second, achieving throughputs one order of magnitude higher than the saturation throughputs of the used detector and 44 % higher than a software-based inference developed for high throughputs. At the same time, when choosing the right hyperparameters and providing calculated features based on the photon counts, we could achieve reductions of FPGA resources by factors of 17 and 10 for logic and memory resources, respectively, with less than 2 % loss in positioning performance. The resulting model occupies less than 1 % of logic and 1.26 % of memory resources in the employed Kintex-7 FPGA. Calculated features based on the light distribution, in particular the CoG, improved the positioning performance, especially for smaller models. The proposed framework can be implemented directly on the readout FPGAs of the PET system and thus scales easily with the PET system size. Due to its high throughput, low resource consumption, and scalability, it is of particular interest for large systems, such as total-body or whole-body PET, where smaller and cheaper FPGAs can cut down costs significantly.

In future work, the possibility of sharing the same model for positioning in different spatial directions will be evaluated. Clock frequency will be increased to further increase throughputs. Furthermore, techniques to reduce the memory requirements in large models will be implemented.

Acknowledgment

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests with the work reported in this article. Bjoern Weissler, David Schug, and Volkmar Schulz are founders of Hyperion Hybrid Imaging Systems GmbH.

References

[1] M. E. Phelps, *PET: Molecular Imaging and Its Biological Applcations*. New York, NY, USA: Springer, 2004.

[2] R. Myers, "The biological application of small animal PET imaging," *Nucl. Med. Biol.*, vol. 28, no. 5, pp. 585–593, 2001.

[3] B. Weissler et al., "A digital preclinical PET/MRI insert and initial results," *IEEE Trans. Med. Imag.*, vol. 34, no. 11, pp. 2258–2270, Nov. 2015.

[4] A. L. Goertzen et al., "Design and performance of a resistor multiplexing readout circuit for a SiPM detector," *IEEE Trans. Nucl. Sci.*, vol. 60, no. 3, pp. 1541–1549, Jun. 2013.

[5] J. Jung, Y. Choi, K. Park, Y. Kim, and J. H. Jung, "A diode-based symmetric charge division circuit with grounding path to reduce signal crosstalk and improve detector performance," *IEEE Trans. Radiat. Plasma Med. Sci.*, vol. 6, no. 7, pp. 788–793, Sep. 2022.

[6] B. Goldschmidt et al., "Software-based real-time acquisition and processing of PET detector raw data," *IEEE Trans. Biomed. Eng.*, vol. 63, no. 2, pp. 316–327, Feb. 2016.

[7] W. Krzemien, A. Gajos, K. Kacprzak, K. Rakoczy, and G. Korcyl, "J-PET framework: Software platform for PET tomography data reconstruction and analysis," *SoftwareX*, vol. 11, Jan.–Jun. 2020, Art. no. 100487.

[8] Y. Lv et al., "Mini EXPLORER II: A prototype high-sensitivity PET/CT scanner for companion animal whole body and human–brain scanning," *Phys. Med. Biol.*, vol. 64, no. 7, 2019, Art. no. 075004.

[9] W. W. Moses et al., "OpenPET: A flexible electronics system for Radiotracer imaging," *IEEE Trans. Nucl. Sci.*, vol. 57, no. 5, pp. 2532–2537, Oct. 2010.

[10] P. Gebhardt, "Design and investigation of an FPGA-based data acquisition and control architecture with MRI RF interference reduction capabilities for simultaneous PET/MRI systems," Ph.D. dissertation, Dept. Comput. Sci., King's College London, London, U.K., 2017.

[11] G. Sportelli et al., "The TRIMAGE PET data acquisition system: Initial results," *IEEE Trans. Radiat. Plasma Med. Sci.*, vol. 1, no. 2, pp. 168–177, Mar. 2017.

[12] R. D. Badawi et al., "First human imaging studies with the EXPLORER total-body PET scanner*," *J. Nucl. Med.*, vol. 60, no. 3, pp. 299–303, 2019.

[13] J. S. Karp et al., "PennPET explorer: Design and preliminary performance of a whole-body Imager," *J. Nucl. Med.*, vol. 61, no. 1, pp. 136–143, 2020.

[14] I. Alberts et al., "Clinical performance of long axial field of view PET/CT: A head-to-head intra-individual comparison of the biograph vision quadra with the biograph vision PET/CT," *Eur. J. Nucl. Med. Mol. Imag.*, vol. 48, no. 8, pp. 2395–2404, 2021.

[15] R. Wojcik, S. Majewski, B. Kross, V. Popov, and A. G. Weisenberger, "Optimized readout of small gamma cameras for high resolution single gamma and positron emission imaging," in *Proc. IEEE Nucl. Sci. Symp. Conf. Rec.*, vol. 3, 2001, pp. 1821–1825.

[16] C.-Y. Liu and A. L. Goertzen, "Improved event positioning in a gamma ray detector using an iterative position-weighted centre-of-gravity algorithm," *Phys. Med. Biol.*, vol. 58, no. 14, pp. 189–200, 2013.

[17] D. Schug et al., "Data processing for a high resolution preclinical PET detector based on philips DPC digital SiPMs," *IEEE Trans. Nucl. Sci.*, vol. 62, no. 3, pp. 669–679, Jun. 2015.

[18] M. Ito et al., "A four-layer DOI detector with a relative offset for use in an animal PET system," *IEEE Trans. Nucl. Sci.*, vol. 57, no. 3, pp. 976–981, Jun. 2010.

[19] A. LaBella, P. Vaska, W. Zhao, and A. H. Goldan, "Convolutional neural network for crystal identification and gamma ray Localization in PET," *IEEE Trans. Radiat. Plasma Med. Sci.*, vol. 4, no. 4, pp. 461–469, Jul. 2020.

[20] X. Li, C. Lockhart, T. K. Lewellen, and R. S. Miyaoka, "A high resolution, monolithic crystal, PET/MRI detector with DOI positioning capability," in *Proc. Annu. Int. Conf. IEEE Eng. Med. Biol. Soc.*, 2008, pp. 2287–2290.

[21] S. España, R. Marcinkowski, V. Keereman, S. Vandenberghe, and R. Van Holen, "DigiPET: Sub-millimeter spatial resolution small-animal PET imaging using thin monolithic scintillators," *Phys. Med. Biol.*, vol. 59, no. 1, pp. 3405–3420, 2014.

[22] P. Bruyndonckx et al., "Neural network-based position estimators for PET detectors using monolithic LSO blocks," *IEEE Trans. Nucl. Sci.*, vol. 51, no. 5, pp. 2520–2525, Oct. 2004.

[23] P. Conde et al., "Determination of the interaction position of gamma photons in monolithic Scintillators using neural network fitting," *IEEE Trans. Nucl. Sci.*, vol. 63, no. 1, pp. 30–36, Feb. 2016.

[24] F. Müller, D. Schug, P. Hallen, J. Grahe, and V. Schulz, "Gradient tree boosting-based positioning method for monolithic scintillator crystals in positron emission tomography," *IEEE Trans. Radiat. Plasma Med. Sci.*, vol. 2, no. 5, pp. 411–421, Sep. 2018.

[25] F. Müller, D. Schug, P. Hallen, J. Grahe, and V. Schulz, "A novel DOI positioning algorithm for monolithic scintillator crystals in PET based on gradient tree boosting," *IEEE Trans. Radiat. Plasma Med. Sci.*, vol. 3, no. 4, pp. 465–474, Jul. 2019.

[26] D. DeWitt et al., "Design of an FPGA-based algorithm for real-time solutions of statistics-based positioning," *IEEE Trans. Nucl. Sci.*, vol. 57, no. 1, pp. 71–77, Feb. 2010.

[27] N. G. Johnson-Williams, R. S. Miyaoka, X. Li, T. K. Lewellen, and S. Hauck, "Design of a real time FPGA-based three dimensional positioning algorithm," *IEEE Trans. Nucl. Sci.*, vol. 58, no. 1, pp. 26–33, Nov. 2011.

[28] Y. Wang, Y. Xiao, X. Cheng, L. Deng, and L. Wang, "An FPGA-based real-time maximum likelihood 3D position estimation for a continuous crystal PET detector," *IEEE Trans. Nucl. Sci.*, vol. 63, no. 1, pp. 37–43, Feb. 2016.

[29] W. Yonggang, D. Junwei, Z. Zhonghui, Y. Yang, Z. Lijun, and P. Bruyndonckx, "FPGA based electronics for PET detector modules with neural network position estimators," *IEEE Trans. Nucl. Sci.*, vol. 58, no. 1, pp. 34–42, Feb. 2011.

[30] P. Carra et al., "A neural network-based algorithm for simultaneous event positioning and timestamping in monolithic scintillators," *Phys. Med. Biol.*, vol. 67, no. 13, 2022, Art. no. 135001.

[31] T. Ling, K. Lee, and R. S. Miyaoka, "Performance comparisons of continuous miniature crystal element (cMiCE) detectors," *IEEE Trans. Nucl. Sci.*, vol. 53, no. 5, pp. 2513–2518, Oct. 2006.

[32] C. Wassermann et al., "High throughput software-based gradient tree boosting positioning for PET systems," *Biomed. Phys. Eng. Exp.*, vol. 7, no. 5, 2021, Art. no. 055023.

[33] B. van Essen, C. Macareg, M. Gokhale, and R. Prenger, "Accelerating a random forest classifier: Multi-core, GP-GPU, or FPGA?" in *Proc. IEEE 20th Int. Symp. Field Program. Custom Comput.*, 2012, pp. 232–239.

[34] J. Oberg, K. Eguro, R. Bittner, and A. Forin, "Random decision tree body part recognition using FPGAs," in *Proc. 22nd Int. Conf. Field Program. Logic Appl. (FPL)*, 2012, pp. 330–337.

[35] F. Saqib, A. Dutta, J. Plusquellic, P. Ortiz, and M. S. Pattichis, "Pipelined decision tree classification accelerator implementation in FPGA (DT-CAIF)," *IEEE Trans. Comput.*, vol. 64, no. 1, pp. 280–285, Jan. 2015.

[36] J. R. Struharik, "Implementing decision trees in hardware," in *Proc. IEEE 9th Int. Symp. Intell. Syst. Inf.*, 2011, pp. 41–46.

[37] R. Kułaga and M. Gorgon, "FPGA implementation of decision trees and tree ensembles for character recognition in Vivado Hls," *Image Process. Commun.*, vol. 19, pp. 71–82, May 2014.

[38] W. Song et al., "Design of a flexible wearable smart sEMG recorder integrated gradient boosting decision tree based hand gesture recognition," *IEEE Trans. Biomed. Circuits Syst.*, vol. 13, no. 6, pp. 1563–1574, Dec. 2019.

[39] S. Summers et al., "Fast inference of boosted decision trees in FPGAs for particle physics," *J. Instrum.*, vol. 15, May 2020, Art. no. P05026.

[40] C. Degenhardt et al., "The digital silicon photomultiplier—A novel sensor for the detection of scintillation light," in *Proc. IEEE Nucl. Sci. Symp. Conf. Rec.*, 2009, pp. 2383–2386.

[41] T. Frach, G. Prescher, C. Degenhardt, and B. Zwaans, "The digital silicon photomultiplier—System architecture and performance evaluation," in *Proc. IEEE Nucl. Sci. Symp. Med. Imag. Conf.*, 2010, pp. 1722–1727.

[42] R. Hetzel, F. Mueller, J. Grahe, A. Honné, D. Schug, and V. Schulz, "Characterization and simulation of an adaptable fan-beam collimator for fast calibration of radiation detectors for PET," *IEEE Trans. Radiat. Plasma Med. Sci.*, vol. 4, no. 5, pp. 538–545, 2020.

[43] J. H. Friedman, "Greedy function approximation: A gradient boosting machine," *Ann. Stat.*, vol. 29, pp. 1189–1232, Oct. 2001.

[44] S. B. Kotsiantis, "Decision trees: A recent overview," *Artif. Intell. Rev.*, vol. 39, pp. 261–283, Jun. 2013.

[45] T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system," in *Proc. 22nd ACM SIGKDD Int. Conf. Knowl. Disc. Data Min. (KDD)*, 2016, pp. 785–794.

[46] B. Weissler et al., "Hyperion III—A flexible PET detector platform for simultaneous PET/MRI," in *Proc. IEEE NSS/MIC*, 2019, pp. 1–9.

[47] *7 Series FPGAs Data Sheet: Overview*, Xilinx, San Jose, CA, USA, Sep. 2020.

[48] *PG151—Divider Generator v5.1 Product Guide (v5.1)*, Xilinx, San Jose, CA, USA, Feb. 2021.

[49] D. T. Luc, "Pareto optimality," in *Pareto Optimality, Game Theory and Equilibria*, A. Chinchuulun, P. P. M., A. Migdalas, and L. Pitsoulis, Eds. New York, NY, USA: Springer, 2008.

[50] A. Nakamura and K. Sakurada, "An algorithm for reducing the number of distinct branching conditions in a decision forest," in *Proc. Mach. Learn. Knowl. Disc. Databases Eur. Conf. (ECML PKDD)*, Sep. 2019, pp. 578–589.