Development and Testing of K-NN on a PIC18F87K22 Microprocessor, for High Speed Particle Collision Classification.

John Colling

***Abstract*—In this project, A K-Nearest Neighbours aglorithim was developed in PIC18 assembly for use on a PIC18F87K2 microprocessor for the binary classification of particle physics events. Its classification accuracy and speed were then assessed in order to determine its viability as a level one trigger at a particle collider. It had an accuracy of $68.4\pm0.5\%$ on sample data, an average classification time of 15.7 ± 8.0ms and data analysis rate of 1.5 ± 0.8Mbps, significantly underperforming both python equivalents and the benchmarks required to be used as a level one trigger. There were likely issues with the configuration of classification timing testing, and further investigation can be done to reduce uncertainty.**

I. Introduction

CONTEMPORARY particle physics is characterised by the detection of rare processes [1]. In order to offset this, particle colliders have high rates of collision in order to maximise the data produced in a given time-frame[2], and therefore increase the likelihood of a rare process occurring. The drawback of this method, however, is producing large quantities of data with majority of it not containing the rare events useful for analysis. Not only does this make the analysis more time consuming, but as the rate at which this data is created is very high, it can lead to both capacity and bandwidth issues when trying to store it.

In order reduce the amount of data for both bandwidth and storage reasons triggers are employed at different stages of the data pipeline. These are computers that take basic sensor readings, often calorimetry measurements, and classify the collisions based on the likelihood of them containing a rare process[3]. Collisions believed to contain a rare process are stored while the others are dropped. The initial trigger the data passes through is known as the level one trigger which often has to make this decision in a time in the order of microseconds[3].

The aim of this project was to develop a K-Nearest Neighbours (K-NN) algorithm on the provided PIC18F87K22 Microprocessor using PIC18 Assembly. This was done to use it to perform binary classification on particle physics data. Its classification accuracy and speed of classification were then measured in order to investigate its ability to act as level one trigger.

II. Data Structure

A. Overview

Fundamentally, K-NN is an algorithm that classifies an incoming data-point by calculating its nearest neighbours from the training data it has been supplied with. As a trade off between the number of training-points that can be stored, and the dimensionality of the feature space[1], it was decided that each training data-point would have 3 features and 1 associated classification label. Due to the 8-bit nature of the microprocessor, all data-points were scaled and rounded to 8-bit integers, the implications of this are talked about in *Accuracy Testing*.

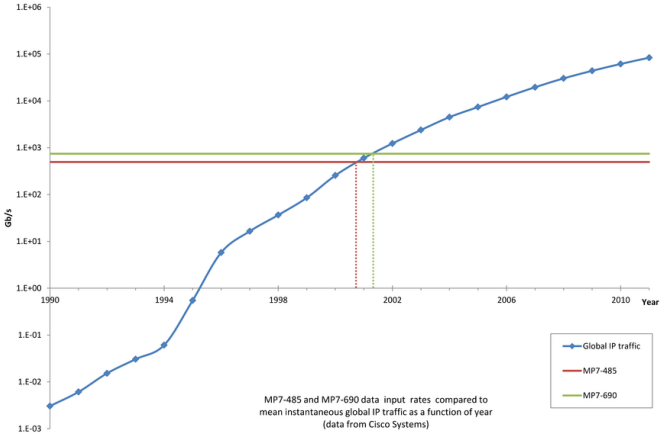

Fig. 1. A graph showing global internet traffic in comparison to the input/output capabilities of the MP7 Virtex-7 FPGA used as a level one trigger at CMS[4]. This is a testament to how important the trigger process is, as 200 of these cards are need to act as a trigger at CMS[5].

Generally, points were stored with the label in the memory address immediately after the 3 addresses storing the features, instead of in a separate area. This decision proved instrumental in the completion of the project, as it gave the label for each point an easily accessible predefined location. This halved the number of pointers required for swapping, which were invaluable as there were only 3 File Select Registers (FSR) on the microprocessor to act as pointers.

The layout of points in file memory makes use of all 16 banks in the file memory, with the exception of bank 15, as to not risk overwriting any special file register values stored there. Different types of data-points and distances were stored in different banks, allowing for easy memory allocation and preventing memory leaks.

[1]If each point has more associated features more complex classifications requiring more features to discriminate different classes can be performed.

B. Access Memory and Bank 0

The access memory, that is primarily composed of the first half of bank 0, is used miscellaneously, for subprocesses such as distance calculations, swapping points in memory, and 3-byte arithmetic.

The rest of bank 0 is allocated to the first k points that are loaded in, in order to calculate the first distances. This sets the limit on k, as it cannot be more than the last half of bank 0, divided by four (the size of a training point in bytes). On this microprocessor this limit is 32.

C. Bank 1: K Nearest Distances

This bank stores the distances to the k nearest points and their attached labels. As these are distances in 24-bit space[2], similar to the data-points they consist of three 8-bit addresses and fourth for the label.

D. Bank 2-14: Training Data

These banks store the training data-points that the algorithm calculates the distances to. Storing the points in subsequent banks made pointer calculations easier as the higher byte of the file select registers would increment when the lower byte would overflow, moving the pointer/FSR immediately to the next bank.

III. Program Architecture

A summary of the program architecture can be seen in Fig. 2.

A. Python Handler

The sending of training data and points for classification as well as the reception of classifications are all achieved using Universal Asynchronous Receiver/Transmitter (UART) protocol. A `python` handler was developed in order to first scale and round the data to 8 bit-integers, and then send and receive data to and from the microprocessor. This was achieved using the `pyserial` library, with the baud rate chosen to be 1324800 Bd.[3]

B. Initialisation and Modules

The program begins by making a series of initialisations ranging from enabling UART send and receive to setting up the 24-bit mathematics and data loader modules developed for this project. It then sends a message to the handler communicating that it is ready to begin receiving training data-points.

[2] 8 by 8 by 8 bit feature space.

[3] Inline with developing this for use as level one trigger, the speed of classification (and therefore the rate of data transmition) was maximised. This was balanced with the UART baud rate values allowed by the microprocessor, given by the formula $1.6 \times 10^7 \times \frac{1}{n+1}$, where n is a 2 byte integer, and the values supported by pyserial which are multiples of 9600. This value was chosen as it was tested to be the fastest reliable baud rate.

C. Data Loading and Training

Initially, k data-points are loaded onto the microprocessor and stored in bank 0, before the rest of the data is loaded on to banks 2-14. The splitting of this data allows for easier pointer arithmetic during the classification, while providing a group of k points in a fixed location. The need for the latter is expanded upon in *E. K-NN Classification* and *V. Improvements and Beta Product*. The is actually the complete training process for K-NN, as it only requires moving the training data into file memory.

D. Input

After the necessary initialisations and loading of training data, the program polls for a data-point to classify from UART. Although generally discouraged in favour of interrupts to reduce program downtime, it is appropriate here. This is because the program only classifies one data-point at a time, and will not request another point from the handler until it has performed the classification. It therefore does not have any tasks to perform while polling.

E. K-NN Classification

This subprocess performs the binary classification of data-points into either points to store or drop. It begins by calculating the distance from the data-point it is classifying to the k points in bank 0 and storing the results and labels in bank 1. It then performs a bubble sort of these k distances.

Next for each training point, it calculates the distance to the data-point. It then scans through the k distances from the beginning of bank 1, until it finds a one less then the calculated distance. It then inserts the calculated distance and associated label into its place, and removing the last distance as to keep it k points. After this scan and insert, or if it does not find a stored distance less than the calculated distance, it will skip to the next training point. This process means that after it has run through all the training points, bank 1 will store the k nearest distances, and their associated labels.

Finally the k nearest labels are then counted, and the class with the highest number determines the classification of the data-point.

F. Output

The classification is then sent via UART to the handler where it is recorded for data analysis later.

IV. Improvements Over Proposal

The primary improvement to the proposed architecture is the new K-NN algorithm. The proposed algorithm involved calculating the distance between the point of prediction and all training data points stored in memory. These distances would then be sorted using insert sort, with a time complexity of $\mathcal{O}(n \log n)$[6], where n is the number of training points. As mentioned, the new structure involves calculating the distances to the first k training data points, before comparing the distances to the distances to the rest of the

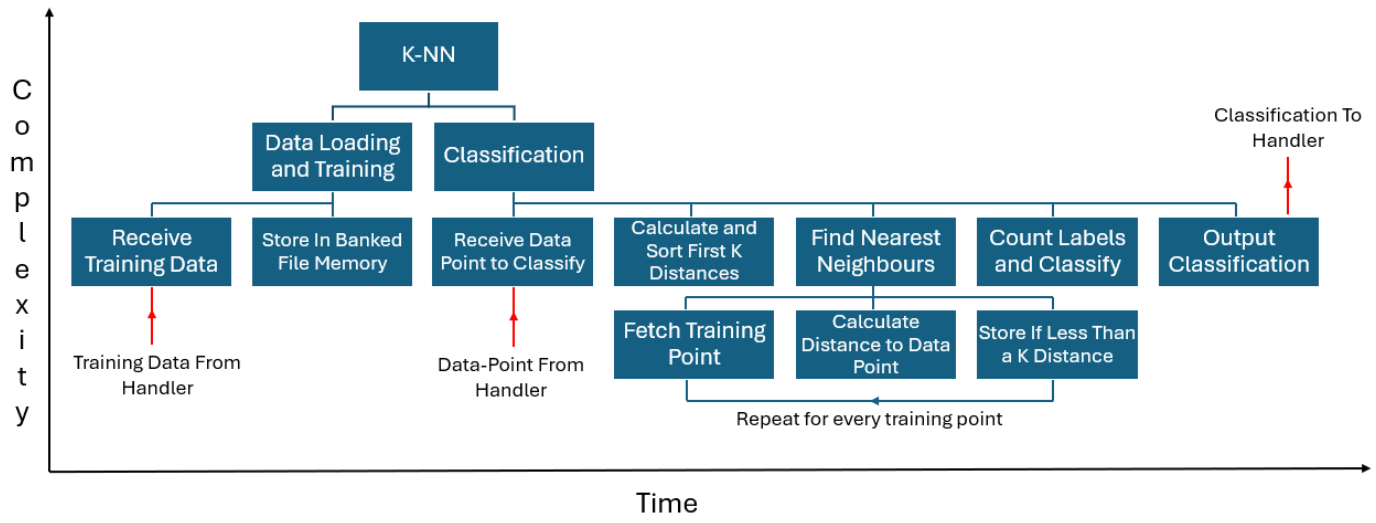

Fig. 2. A time-complexity diagram of the K-NN Architecture. Red arrows indicate the flow of data to and from the python handler.

training dataset. Although bubble sort[4] is used to sort the first k distances (in therefore $\mathcal{O}(k^2)$[6]), as only one distance calculation and comparison to the stored k distances for the rest of the dataset, it is clear that this scales in $\mathcal{O}(n)$ time. As k is necessarily less than n and in general significantly less than n, this therefore leads to a reduced classification time.

The new architecture also provides memory benefits. Over the proposed algorithm, which would have stored the distances to all points and the new algorithm only stores k distances at any point. As all these distances are three byte numbers, taking up almost the same space as the three features points, this change effectively doubled the quantity of training points the chip could store. Although the proposal suggested 480 training points was possible, the improvements allowed for it to run with a maximum of 832.

V. Performance Testing

The primary criteria for a suitable level one trigger are the classification accuracy and the speed of classification. A high classification accuracy is needed in order to prevent the dropping of rare events while not storing events that do not contain useful information. On the other hand, this classification needs to be made quickly (often in the realm of microseconds # cite), due to the rate at which these collisions are happening. Therefore a series of performance tests were carried out in order to evaluate the K-NN program on the microprocessor, against these criteria, in order to assess its viability. The dataset chosen for these tests was 5000 points of simulated background and super-symmetric events due to its binary classification particle physics nature (as would be the case for a level one trigger) although any binary classification dataset could be used for these tests. As this dataset contained more than three features (the number of features this K-NN was developed around), a search was done to find the best features for classification accuracy, with those features being used for testing.

A. Accuracy Testing

To assess the accuracy of the K-NN on the, the python handler was used to scale and round to 8-bit, and then send data-points to microprocessor. The microprocessor would then send the predicted classification to the handler, which then compared it with the known label to calculate an accuracy. In order to avoid over-fitting, the data was split into separate training and test data-sets, so the program did not have training data mapping specifically to any points it was tested on. This process was repeated, varying the value of k and the number of memory banks filled with training data. The results of this testing can be seen in Fig. 3 and Fig. 4

The points of comparison for this algorithm are a standard `scikit-learn` K-NN trained on data that has been scaled and rounded to be 8-bit integers (as to reflect the limitations of the microprocessor) and a `scikit-learn` K-NN trained on unscaled float data. As expected, the both the drop in feature space resolution and changing of distances from the multi dimensional[5] scaling changed the neighbours of certain points. This causes misclassification and therefore a drop in accuracy.

Investigating K-NN accuracy as the value of k is varied, alarmingly the microprocessor significantly underperformed even the scaled and rounded K-NN despite having performed perfectly in well clustered data. This is indicative of a bug, perhaps in the comparison of distances of similar magnitude. In order to verify this underperformance, I performed two sampled Kolmogorov-Smirnov (KS) statistical tests between the microprocessor's results and the scaled

[4]This was chosen due to its ease to implement over the proposed insert sort.

[5]The data had to be scaled in 3 dimensions so that all 3 features were between 0 and 255 to be rounded to 8-bit integers.

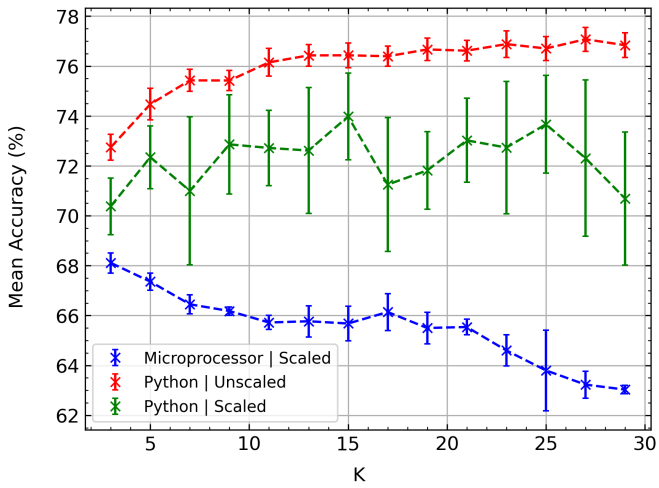

Fig. 3. A plot of the accuracy of a scaled and unscaled scikit-learn K-NN and the accuracy of the microprocessor K-NN against the value of k. The uncertainties plotted are the standard deviations of the accuracies for repeated runs for a given set of parameters. For this data, the microprocessor was trained on 13 banks of data. The microprocessor significantly underperforms the other two models.

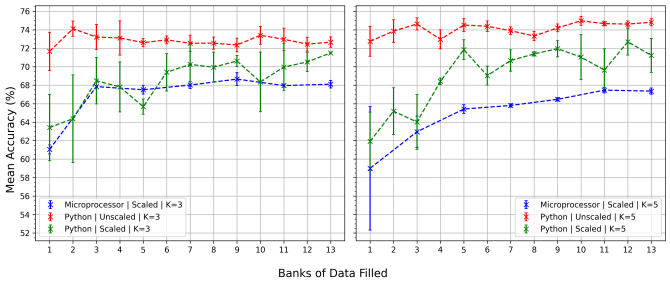

Fig. 4. Plots of the accuracy of a scaled and unscaled scikit-learn K-NN and the accuracy of the microprocessor K-NN against the banks of training data. The uncertainties plotted are the standard deviations of the accuracies for repeated runs for a given set of parameters. The microprocessor continues to underperform the scikit-learn models, but by less.

and unscaled K-NN. These are more appropriate than chi-squared goodness of fit tests as although K-NN is a complex algorithm, it is still completely deterministic and therefore has no inherent uncertainty. The uncertainties plotted stem from the random nature in how the data is split between training and testing sets, and are taken as the standard deviation for repeated measurements at a given set of hyper-parameters. The p values for these tests were in the order of magnitude of 10^{-8}, indicating it is very unlikely that the results from the microprocessor are from the same distribution as either of the other K-NNs.

This trend continues for accuracy as the number of banks filled with training data is varied. In these cases the (KS) tests returned marginally higher p values of $\sim 10^{-4}$ to the unscaled K-NN and 0.09(k=3) and 0.004(k=5) to the scaled K-NN. As it is over 0.05, this is a statistically significant result, and the behaviour in relation to the scaled models needs to be investigated more.

B. Speed of Classification Testing

To measure the time taken for the microprocessor to perform a classification, the handler was modified to create and store a `python datetime` object before and after the data-point had been sent, and another after it had received the classification. By analysing the time difference between these objects, the average time taken for data transmission and the average time taken for the program to make and send a classification. By subtracting the former from the later gives the average time taken for the classification, propagating the standard deviation of the averages as the uncertainties. The data is summarised in Fig. 5 and Fig. 6.

As explained in *IV. Improvements Over Proposal* when investigating the relationship between classification time and amount of training data, we expect a linear relation due to a time complexity of $\mathcal{O}(n)$. Even when fitted using `Iminuit` a straight line has a statistically insignificant p value[6]. Although, at first this could be mistaken for a bug in the K-NN, it is important to note two values the timing measurements take, 16 and 32. Given that these are powers of two commonly found in computing, it is possible that this is a precision limit in the `Datetime` module, especially as during the analysis it was not uncommon for the data transmission times to be calculated as zero microseconds.

This is further compounded by the analysis of the relationship between k and classification time. As mentioned this is expected to be $\mathcal{O}(k^2)$, and therefore a quadratic fit was performed. Although this fit with a p value of 1 `Minuit` achieved this by suppressing the quadratic element with a coefficient of 0 ± 0.11, providing no indication that this trend is actually quadratic. The high p value is likely caused by the relative error size in comparison to the almost zero change in values. This low change in values, especially around the 16 millisecond mark is further an indication that there is some issue with the timing setup as these were changes that were visibly noticeable on an oscilloscope waveform in the laboratory. Although `Datetime` claims to be precise to the microsecond, this warrants further investigation.

VI. Beta Product

A. Additional Features

As mentioned due to how the final classification is made when counting the classes of the k nearest points, this limits the values of k to only odd numbers. This could be implemented by performing a random choice, however this risks dropping the accuracy due to its random nature. Another approach would be to check the next furthest neighbour for the deciding class, which could be achieved by storing the last distance and label removed from the end of the k distances.

The number of training data-points stored is also limited to only full banks which may create problems with some datasets as it would involve the dropping of data the end part of the data so that it can all fit into 64 point groupings.

[6]This was calculated using chi-squared, as we are now performing physical measurement with some degree of both resolution and stochastic uncertainty.

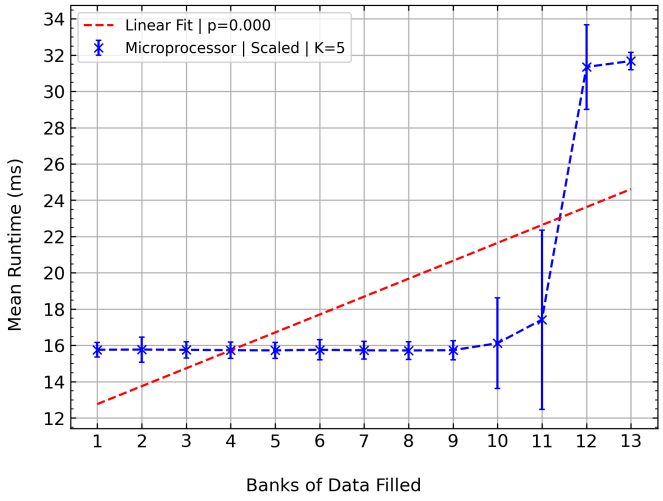

Fig. 5. Plot of program classification time against the number of banks filled with training data. The Poor linear fit is either an indication of a bug in the K-NN or timing setup.

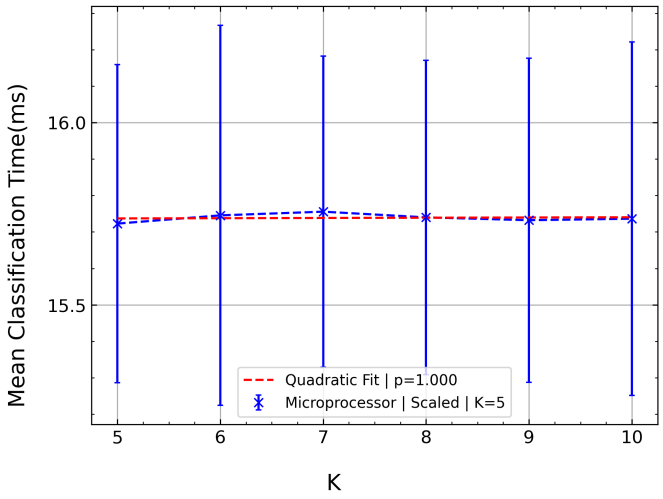

Fig. 6. Plot of program classification time against the value of K. Although it fits a quadratic to a high p value, this is with the quadratic coefficient suppressed, indicating a lack of quadratic feature in the data.

This can be overcome by instead of the data loader counting the number of banks to be filled, to count the number of individual points. This had been the planned approach for the project but had to be abandoned due to time constraints.

B. Performance Upgrades

The main modification that could be made to this project in order to increase its accuracy would be an increase in the amount of file memory. Although the proposal was outperformed in number of points stored, 832 points is still very low for modern machine learning. It is also important to remember that each point was only allocated 3 features, which is also low for modern machine learning. By increasing the file memory it would be possible to increase both of these, allowing for the K-NN to perform better on data that does not have a clear linear split, which is often the case in particle physics. Although an approach for this was considered during the project using a mikroelektronika™ click-board, these would have interfaced with the microprocessor using Serial Peripheral Interface (SPI) protocol. This would increase the classification time, as the data would need to be moved into file memory first using SPI, in comparison to the current method of just calling the memory address. An increase in memory would also allow for configuration that does not involve scaling and rounding the data, removing the inherit accuracy drop from that process.

A change in the algorithm that could lead to a decrease in total classification time would be the vectorisation of the process. Although very difficult to achieve in PIC18 assembly, this would reduce the overhead times, such the communication and polling time. This would also need an increase in file memory or reduction in training data. Primarily, however, a change in hardware would yield the biggest performance upgrade. The PIC18F87K2 microprocessor is a 64MHz single cored microprocessor, meaning that it has one core that can execute one program, at a rate of 16×10^6 assembly instructions per second. Even the addition of one more core to the microprocessor would likely lead to significantly faster classification times as one could be dedicated a K-NN program and another to data-communication to the handler. This could be done in a manner where there would be no polling time for the K-NN core and therefore no down time. Ultimately a move to an Field Programmable Gate Array (FPGA) would likely yield the best results. Due to its programmable hardware nature implementing K-NN onto an FPGA is not trivial and is still an area of active research[7]. For comparison, the MP7 Virtex™ FPGAs used at CMS for their level one trigger system, although running at only 100Mhz have hardware level parallelisation built into them. This means that in comparison to the 1.5 ± 0.8Mbps of data processing rate achieved by the microprocessor the MP7 can achieve 1.88Tbps[4].

VII. Viability

With its best hyper-parameters (k=3 and 9 banks of training data) the K-NN developed achieves an accuracy of $68.4 \pm 0.5\%$. This gives it a processing time of 15.7 ± 8.0ms, taking the uncertainty as the believed resolution of the timing setup given the previous discussion.

Although this accuracy is not ideal, it is important to note that the accuracy is still above 50%, indicating that there is still some level of classification taking place. With these hyper-parameters the microprocessor is also less than 3% away from the scaled `scikit-learn` K-NN. This is an indication that this low accuracy may be caused by K-NNs struggling to classify this data in general, rather than issues with the microprocessor. This is likely due to the use of only 3 features from the data as well as the scaling process, as when training an unscaled `scikit-learn` K-NN on the full set of features, it achieves an accuracy of 92.4%. This is still well below the over 99.9% accuracy required by triggers

at the LHC, perhaps indicating that in general K-NN may not be an appropriate classification method for this task.

The classification time is also subpar. As the rate of events at the LHC is 40MHz[2], this allocates 250 nanoseconds for each event. As mentioned, the microprocessor classification time is approximately 630,000 times more then that and is therefore not sufficient. This also means it would take approximately 630,000 microprocessors running in parallel in order to achieve this task, assuming there is no delay caused by the simultaneous management and aggregation of outputs of these microprocessors.

VIII. Conclusion

This project involved the development `python` handler and K-NN algorithm in PIC18 assembly in order to perform classification of particle physics collision data on a PIC18F87K2 microprocessor. Testing of its classification accuracy and speed was then performed in order to assess its viability as a level one trigger. In these tests it performed poorly, underperforming `scikit-learn` equivalents consistently, indicating an issue with the program. Its average classification time was also poor at 15.7 ± 8.0ms, which is orders of magnitude above what is required. The high uncertainty on this measurement is caused by issues in the timing setup, likely a precision limit on the `python Datetime` library used to perform the measurements. More investigation should be carried out in order to reduce this uncertainty as this could not be done during the project due to time constraints. The rate of data analysis performed by the chip is 1.5 ± 0.8Mbps, significantly less than the 1.88Tbps achieved by similar products[4]. The performance suggest that although this program and microprocessor combination can perform classification of particle physics events, they do not do so at a high enough accuracy or speed in order to be viable as a level one trigger.

References

[1] P. D. Group, P. A. Zyla, R. M. Barnett, and B. et al, "Review of particle physics," *Progress of Theoretical and Experimental Physics*, vol. 2020, no. 8, p. 083C01, 08 2020. [Online]. Available: https://doi.org/10.1093/ptep/ptaa104

[2] V. Gligorov, "Real-time data analysis at the lhc: present and future," in *Proceedings of the NIPS 2014 Workshop on High-energy Physics and Machine Learning*, ser. Proceedings of Machine Learning Research, G. Cowan, C. Germain, I. Guyon, B. Kégl, and D. Rousseau, Eds., vol. 42. Montreal, Canada: PMLR, 13 Dec 2015, pp. 1–18. [Online]. Available: https://proceedings.mlr.press/v42/glig14.html

[3] C. Foudas, "The CMS Level-1 Trigger at LHC and Super-LHC," https://cds.cern.ch/record/2232067/files/arXiv:0810.4133.pdf, [Accessed 29-04-2025].

[4] "MP7 — hep.ph.ic.ac.uk," https://www.hep.ph.ic.ac.uk/mp7/, [Accessed 29-04-2025].

[5] G. Hall, "A time-multiplexed track-trigger for the cms hl-lhc upgrade," *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, vol. 824, pp. 292–295, 2016, frontier Detectors for Frontier Physics: Proceedings of the 13th Pisa Meeting on Advanced Detectors. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0168900215011304

[6] A. kumar Karunanithi, "A survey, discussion and comparison of sorting algorithms," 2014. [Online]. Available: https://api.semanticscholar.org/CorpusID:18564359

[7] M. H. Yacoub, S. M. Ismail, L. A. Said, A. H. Madian, and A. G. Radwan, "Generic hardware realization of k nearest neighbors on fpga," in *2022 International Conference on Microelectronics (ICM)*, 2022, pp. 169–172.