FINESSD: Near-Storage Feature Selection with Mutual Information for Resource-Limited FPGAs

Nikolaos Kyparissas, Gavin Brown, Mikel Luján
Department of Computer Science, The University of Manchester

DOI: TBC

FINESSD: Near-Storage Feature Selection with Mutual Information for Resource-Limited FPGAs

Nikolaos Kyparissas, Gavin Brown, Mikel Luján
Department of Computer Science
The University of Manchester
{firstname.lastname}@manchester.ac.uk

***Abstract*—Feature selection is the data analysis process that selects a smaller and curated subset of the original dataset by filtering out data (features) which are irrelevant or redundant. The most important features can be ranked and selected based on statistical measures, such as mutual information. Feature selection not only reduces the size of dataset as well as the execution time for training Machine Learning (ML) models, but it can also improve the accuracy of the inference.**

This paper analyses mutual-information-based feature selection for resource-constrained FPGAs and proposes FINESSD, a novel approach that can be deployed for near-storage acceleration. This paper highlights that the Mutual Information Maximization (MIM) algorithm does not require multiple passes over the data while being a good trade-off between accuracy and FPGA resources, when approximated appropriately. The new FPGA accelerator for MIM generated by FINESSD can fully utilize the NVMe bandwidth of a modern SSD and perform feature selection without requiring full dataset transfers onto the main processor. The evaluation using a Samsung SmartSSD over small, large and out-of-core datasets shows that, compared to the mainstream multiprocessing Python ML libraries and an optimized C library, FINESSD yields up to 35× and 19× speedup respectively while being more than 70× more energy efficient for large, out-of-core datasets.

I. Introduction

Part of the growing success of Machine Learning (ML) during the last decade has been attributed to being able to collect and curate increasingly large datasets. These large datasets often can be interpreted as a table containing rows after rows of values and each column representing a variable, or *feature*. Reducing the dataset would naturally reduce the computational time for the training of a given ML algorithm. Thus, the recommended practice is to reduce the number of features (number of columns) that would be used for the ML model. Such reduction of features is often known as *dimensionanality reduction*. The *curse of dimensionality* and its effects on the process and accuracy of ML algorithms is well-established and understood in the literature. Dimensionality reduction is a fundamental part of data analysis in ML for preprocessing large datasets in a principled manner as to obtain curated datasets.

One of the most well-known dimensionality reduction algorithms is the Principal Component Analysis (PCA). Thus, the FPGA community has studied how to optimise such an important algorithm in multiple occasions. However, at its core PCA incurs a matrix factorization with real numbers as to compute the eigenvalues and the associated eigenvectors. The FPGA requirements of PCA (and other matrix-based factorizations) make them not to be the first choice for small FPGAs.

On the other hand, *feature selection* is the data analysis process which selects a subset of the original dataset by identifying a subset of features according to their relevance and them not being redundant (correlated with another feature in the set). The most important features can be ranked and selected relying on statistical measures, such as mutual information. Within feature selection, filter methods are often preferred over the alternatives because they are application and ML algorithm agnostic, generating high-quality reduced datasets regardless of where and how they will be used.

Information-theoretic feature selection is not based on matrix factorizations, but rather on counting events to measure event frequencies, and thus calculate probabilities. Thus, this kind of feature selection can be an ideal candidate for acceleration on a resource-constrained FPGA element. Nonetheless, mutual-information-based feature selection poses two main challenges. First, the statistical properties of the dataset, such as the probability distributions of each feature, have to be extracted from it, usually in the form of histograms. This process, depending on the filtering algorithm, requires one or multiple passes over the whole dataset, resulting in substantial amount of data transfers and random memory accesses for multiple histograms. Additionally, in the case of large datasets which do not fit in the memory, those passes produce I/O transfers from the much slower storage. Second, calculating the mutual information between variables is not trivial, with the mainstream approaches resorting to complex floating-point mathematical operations such as divisions and logarithms.

To tackle the complexity of feature selection and mutual information, past approaches either utilize power-hungrier computing devices such as GPUs, or trade accuracy for resources and speed with approximate computations. However, in the case of large datasets, transferring low-reuse data from storage remains a bottleneck.

When dealing with large datasets that do not fit in main memory, the main bottleneck tends to become the latency and bandwidth of accessing the storage devices. Nowadays, SSDs connected via PCIe (such as NVMe) to a multicore chip capture most desktop, laptops and server computing devices. To avoid the bottleneck of transferring data from the SSD via PCIe, there is a growing interest in bringing computation

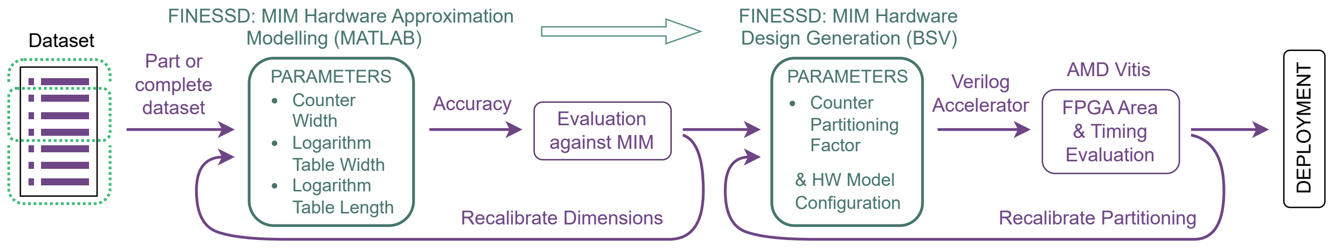


Fig. 1. The flow of FINESSD: a MATLAB model of the mutual information approximation and the hardware design dataflow is used to perform design space exploration, until a configuration with an acceptable final accuracy is found. The accelerator generator written in Bluespec SystemVerilog (BSV) uses that configuration along with other architectural configuration options to generate the Verilog code of the final MIM accelerator.

closer to the storage; *near-storage processing*. When compared with state-of-the-art chips found on servers, the computational capacity of the near-storage processing elements is much reduced and constrained.

To overcome the aforementioned challenges, we identify the Mutual Information Maximization (MIM) algorithm [1] for acceleration since it avoids multiple passes over the dataset. We propose FINESSD, a methodology for approximating MIM and generating accelerators for resource-constrained FPGAs (Figure 1). We deploy FINESSD on a Samsung SmartSSD, a reconfigurable computational storage device (CSD), and show how the generated FPGA design can fully harness the bandwidth of a modern SSD. By doing so, FINESSD offloads from the host the feature selection pipeline with significant end-to-end performance and energy gains.

This paper makes the following contributions:

- We propose FINESSD, a novel hardware-oriented methodology for approximating mutual information with resource-constrained FPGAs. We provide a thorough design space exploration, contingent on the resource constraints of FPGA-based CSDs, illustrating the trade-off between accuracy and hardware resources.
- Based on counters that record multiple feature samples at clock rate, the FPGA accelerator generated by FINESSD transforms the MIM algorithm from compute-bound to I/O-bound and filters the data with one sequential read, outperforming the baseline methods by tackling the data transfer and computation bottlenecks.
- We present the first near-storage application of feature selection, reducing data transfers of low-reuse data from storage to host over PCIe and increasing parallelism.
- We evaluate FINESSD using datasets of different dimensions and complexity, yielding up to 35× speedup over established, mainstream multi-processing tools and significant energy improvements.

II. Background

This section provides the context for feature selection and near-storage processing as to allow for further discussion thereafter. We introduce the fundamentals of feature selection, the role of mutual information as a scoring function, and the choice of MIM as an ideal candidate for resource-constrained FPGAs such as the ones found on CSDs.

A. Feature Selection and Mutual Information Maximization

In ML, many significant applications such as gene expression and text clustering may easily be comprised of several thousand variables, or *features* [1], [2]. Many of these features can be redundant or irrelevant, increasing the computation cost [3] and compromising the accuracy of the ML model, for example, by causing it to be prone to overfitting [1]. Hence, dimensionality reduction is an integral part of the ML pipeline.

Feature selection is a dimensionality reduction approach which, in contrast to feature extraction, preserves the data in their initial form by choosing a higher-quality subset of the initial features, excluding those which are redundant or exhibit low correlation. Preserving the original features is necessary, for example, when it comes to applications where model explainability is required, such as healthcare or safety-critical applications [4]. While there are several strategies to conduct feature selection, in this paper we will focus on *filter* methods. Filter methods use scoring mechanisms based on statistical measures to evaluate the usefulness of the input features relative to the output labels. The advantage of filter methods is that, in contrast to other feature selection strategies, they are independent of the ML algorithm that will be used, as they rely only on the statistical correlations found in the data. As a result, filter-based feature selection produces smaller, generic inputs of higher quality.

One of the prominent measures used for scoring the features is *mutual information* [1], [5]–[8]. In information theory, the mutual information between two random variables is the amount of information that is common in those two variables [9] or, in other words, the amount of information that is revealed about a random variable when observing the other one. Mutual information between two random variables, feature X and label Y, is defined as:

$$I(X;Y) = \sum_{\substack{x\in X\\ y\in Y}} p(x,y)\log\frac{p(x,y)}{p(x)p(y)} \tag{1}$$

where $p(x)$ is the probability of the random variable X having the value x, and is usually calculated with histogram estimators. With mutual information as a scoring criterion, the score of a feature X_k for a class label Y is defined as:

$$S_{MIM}(X_k) = I(X_k;Y)$$

Once we calculate $S_{MIM}(X_k)\ \forall\ k$, we rank the features based on their importance score and choose the K ones with the highest score. This method is known as feature selection based on Mutual Information Maximization (MIM).

MIM is widely used and, in most cases, has similar filtering-quality performance to other information-theoretic filtering algorithms such as Joint Mutual Information (JMI) and minimum Redundancy - Maximum Relevance (mRMR) [1], [10], [11]. However, MIM feature selection has the critical advantage of requiring only one pass over the dataset and simpler histograms, and for a dataset of N samples and M features its complexity is $O(NM)$ as opposed to $O(NM^2)$ for JMI and mRMR. Hence, MIM is preferred when more passes over the dataset would be inefficient or even impossible (e.g. in [12]). Near-storage deployment is such a scenario.

B. Near-Storage Processing and MIM Feature Selection

In a typical computer system, the host processor orchestrates the dataflow between the storage, the memory and the coprocessors or accelerators which, normally are connected via PCIe. Up until recently, in order for the coprocessor to process data found in storage, the host processor needed to load part of the file from the storage to memory and then copy the data from its address space to the address space of the accelerator. The latter step of time-consuming memory copies can now be bypassed with the use of an IOMMU [13] or protocols such as CXL [14] which provide a shared address space between the host processor and the accelerator.

An effective approach to improve coprocessor access to the storage, is to bring processing closer to it; a research area known as *near-storage processing*. The key idea behind this strategy is that there exists a direct link between the processing element and the storage device without the mediation of the host processor. Controller Memory Buffers (CMB) in NVMe enable a peer-2-peer (P2P) connection from one NVMe PCIe end point to another without using a system memory buffer. In recent years, significant efforts have been made to bring processing even closer to storage and process data within the storage device [13], forming a *computational storage device* (CSD). Bringing processing so close to the data has numerous advantages, such as decreased data transfers via the system PCIe bus, increasing parallelism by unloading the host processor, and reduced energy consumption.

However, CSDs, and especially reconfigurable CSDs, are characterized by a number of constraints. First, in most cases, they have limited hardware resources available for processing. Second, they are suitable for applications with high spatial and low temporal data locality, which are able to maximize the amount sequential accesses to the storage part of the CSD. As we mentioned in Section II-A, MIM has the critical advantage of needing only a single pass over the dataset and simpler hardware requirements for its histogram-based data structures compared to JMI and mRMR, making it the ideal candidate for near-storage acceleration and the resource-constrained FPGAs of CSDs if accelerated appropriately. While JMI and mRMR are able to take advantage of the re-use of features, unlike MIM, this eventually acts as their disadvantage against MIM. Mechanisms such as host page caching do not help for the mentioned algorithms in the cases of large and larger-than-memory datasets, where consecutive, complete passes are required over the dataset.

In this paper, we present FINESSD, our hardware-oriented approach on mutual information for accelerating MIM feature selection. The FPGA design generated by FINESSD accelerates MIM, causing it to become I/O-bound. When placed near storage, the FINESSD FPGA accelerator filters big datasets directly where data lie with a single sequential pass, resulting in significant time and energy savings by avoiding costly I/O transfers to and from the host.

C. Related Work

In recent years, several hardware acceleration approaches have been proposed for accelerating mutual information calculations, either independently or as part of feature selection and other algorithms. CUDA-JMI [15] and Fast-mRMR [16] use GPUs to speedup the two feature selection algorithms, JMI and mRMR respectively. Aside from the fact that GPUs are power-hungry, the main problem of these GPU implementations is that they are limited by the shared memory found on the device when counting contingency tables. Also, as we saw in Section II-B, a direct comparison between these GPU implementations and FINESSD for large and larger-than-memory datasets would not add any extra information.

FPGAs have also been used for calculating mutual information. A combination of reduced-precision arithmetic and look-up tables was used in [17], trading accuracy for parallelism to achieve a highly-parallel architecture on a multi-FPGA system. A different strategy for partitioning the problem in multiple parallel parts and using a combination of fixed and floating point precision was used in [18], with intensive processing over small data in mind, in contrast to our case. Approximate computing has been applied even more extensively by [19], where fixed-point arithmetic and narrow counters where used for feature selection, but with embedded platforms as a target and a limited set of selected features. Also, this analysis was not accompanied by a hardware implementation, which, as we will see later, comes with its own challenges.

As we mentioned earlier, counting histograms is a big part of calculating mutual information and constitutes one of the bottlenecks, which is why in the cases of CUDA-JMI [15] and [17] histogram counting was offloaded either partly or completely to the host CPU. Fast-mRMR [16] is using a different approach, based on changing the dataset access pattern from row-wise to column-wise (from sample-by-sample to feature-by-feature); a low-level trick that we also utilize in order to reduce the amount of random accesses to multiple histograms.

To our knowledge, FINESSD is the first near-storage feature selection solution and the first FPGA accelerator for the whole end-to-end process of feature selection.

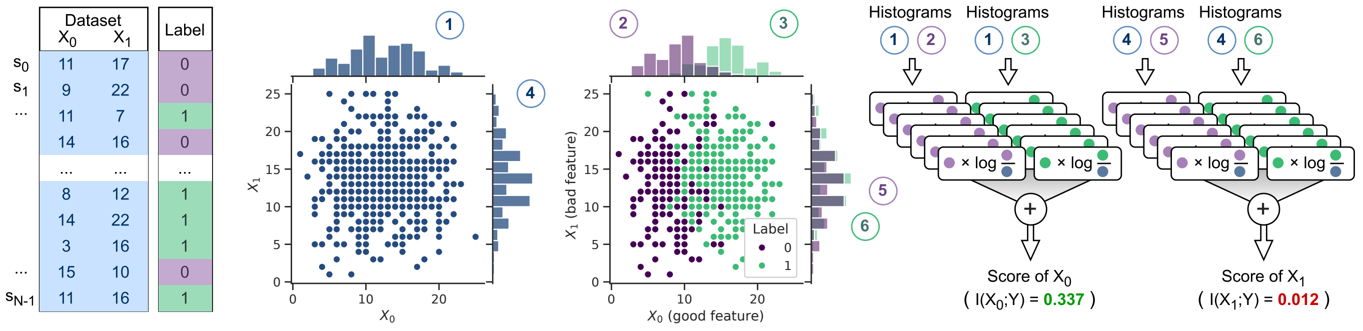


Fig. 2. A high-level overview of MIM: calculating and ranking the information-theoretic scores for a curated binary-classification dataset of N samples and two features, X_0 and X_1. Ⓐ The histogram of every feature and the histograms of the joint appearances of every feature with every class label must be calculated. Here, histograms 1, 2 and 3 refer to feature X_0, and histograms 4, 5 and 6 refer to X_1. After extracting the histograms, Ⓑ these have to be processed to calculate the information-theoretic score of each feature. Ⓒ A list of the best features can be extracted by the ranking of the feature scores.

III. FINESSD: Profiling, Approximation and Modelling

A. *Breaking Down MIM Feature Selection*

There are three key steps in MIM feature selection: Ⓐ estimate the probability distributions of the features and the label, Ⓑ calculate the mutual information between every feature and the label based on their probability distributions, and Ⓒ rank the features based on their mutual information scores. The process over a simple example dataset is shown in Figure 2.

The first important step consists of estimating the probability distributions of the dataset features. This is attributed to the fact that, practically, it is impossible to know the true probability distributions $p(x)$, $p(y)$ and $p(x,y)$ and we have to estimate them from the dataset. Estimating mutual information for continuous data is not trivial [20]–[22] as we need to measure the probability densities without the probability density functions. Therefore, in this paper, we use datasets discretized with Sturge's rule, a commonly-used algorithm that chooses the number of bins needed to approximate the original distribution of the samples based on the size of the dataset. For the scope of this paper, it is safe to assume that this process has been conducted offline as it refers to previous stages of a typical data preprocessing pipeline.

According to maximum likelihood estimation, the probability of an event can be estimated from the observed data as the number of event occurrences over the number of total events (i.e. the total number of samples): $\hat{p}(x) = [\text{count}(X = x)]/N$. Consequently, we estimate the probability distributions with histogram estimators, which give us the estimated frequency of the events.

Figure 2 shows that for every feature, numerous histograms have to be extracted from the data in order to calculate the score of that feature. This is the only pass required over the data for MIM feature selection, where each data element is only ready once. For example, in the simple example shown in Figure 2, we need one histogram for each feature, and two more histograms per feature for its joint appearances with the two classes of the label. From the joint-appearances histograms it can already be seen that X_0 is better at providing an indication about the classification result.

As we see in the formula of mutual information (Equation 1), for datasets with the same number of labels and bins, regardless of the number of samples, calculating mutual information between a feature and a label requires the same number of steps once we have counted the probability distribution histograms. However, the number of samples in a dataset is proportional to the time required for counting the histograms. We created two synthetic datasets with the same $N \times M$ size, but one is "short and wide" (few samples N, many features M) and the other is "tall and slim" (many samples, few features). Both are synthetic datasets created with Scikit-learn's *make_classification* function, having the same number of bins, labels, and probability distribution properties.

In Figure 3 we can see that for two datasets with the same characteristics, the time required for calculating the mutual information score of one feature (green rectangle) is significantly larger for the dataset with more samples, only because of the histogram calculations. That phenomenon is amplified when we apply MIM feature selection on large datasets with a large number of samples, since the number of samples is proportional to the effort needed for determining a feature's statistical properties.

B. *Breaking Down Mutual Information*

Considering that counting consumes most of the time we need for MIM feature selection, limiting the different objects that have to be counted affects greatly the performance of our approach. In order to minimize the amount of items we need to count, it is necessary to modify the second step of MIM feature selection, which consists of calculating the mutual information score for each feature. As mentioned before, mutual information is a measure used for scoring the features in order to rank them by importance. However, the actual value of the score in this application is irrelevant, as the ranking of the features is the only end goal. To that end, we will preserve only the parts of mutual information that are necessary for ranking. Mutual information can be expressed as $I(X;Y) = H(Y) - H(Y|X)$, where we can see that the

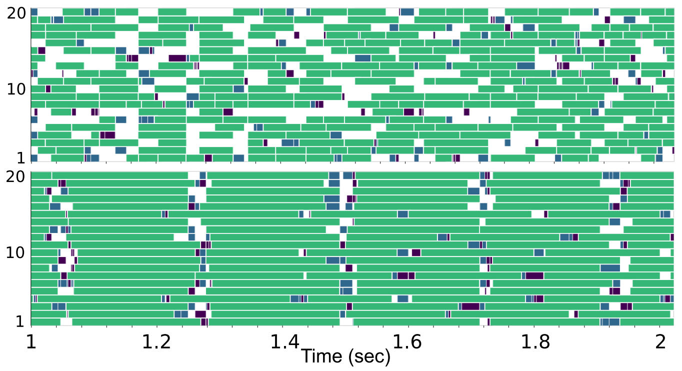


Fig. 3. Profiling a multi-process Python version of MIM on 20 processing cores for two datasets with the same number of bins and labels and different number of samples. On top, the time required for calculating mutual information for one feature (green rectangle) is significantly smaller compared to a dataset with more samples (bottom), solely due to histogram calculations.

first term of the two, $H(Y)$, is the entropy of the label. Notice that we can neglect $H(Y)$ without affecting the final ranking of the features, since its value depends only on the label and remains the same for every feature.

As a result, we can use the conditional entropy $H(Y|X)$ for scoring the features and have the same result. Estimating the conditional entropy with histogram estimators, given a dataset $D = \{([x_1,\ x_2,\ ...,\ x_m]_i,\ y_i);\ i = 1..N\}$ with N samples, where x_{ki} is the value of feature X_k in the i-th sample labeled y_i, conditional entropy can be estimated as:

$$\hat{H}(Y|X) = -\sum_{i=1}^{N} \hat{p}(x_i, y_i) \log \hat{p}(y_i|x_i) =$$
$$-\sum_{i=1}^{N} \big(\frac{\text{count}(x_i, y_i)}{N} \log \frac{\text{count}(x_i, y_i)}{\text{count}(x_i)}\big) =$$
$$-\frac{1}{N}\sum_{i=1}^{N} \big[\text{count}(x_i, y_i)\big(\log \text{count}(x_i, y_i) - \log \text{count}(x_i)\big)\big]$$

Notice that we can neglect multiplying the score of every feature by the constant $1/N$ without affecting the final ranking of the features. Also, by inverting the subtraction operands, since $\text{count}(x) \geq \text{count}(x, y)$ we do not need to negate the summation and we only need unsigned integers for our calculations. We do not have to count the frequency of how many times each value appears ($\text{count}(x_i)\ \forall\ i \in N$), since we have that information as the sum of all joint counts of a given $x\ \forall\ y \in Y$.

Thus, the final information-theoretic criterion that we can use to rank the features based on their importance relies only on counting the different joint appearances of X (feature bins) and Y (labels) for every feature:

$$I(X;Y) \approx \hat{I}(X;Y) \propto \hat{H}(Y|X) \propto S'(X) = \quad (2)$$
$$\sum_{i=1}^{N} \big\{ \text{count}(x_i, y_i) \big[\log \big(\sum_{y \in Y} \text{count}(x_i, y) \big) - \log \text{count}(x_i, y_i) \big] \big\}$$

Notice that the objects we have to count for the whole dataset form a cube whose dimensions are $features \times bins \times labels$

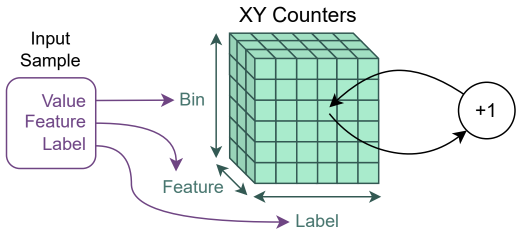


Fig. 4. A cube of counters is formed by all the different objects we have to count in order to estimate the mutual information between the features and the label for the whole dataset.

(Figure 4). Basically, for every feature we have a 2-D matrix which is its contingency table. Visualizing the counters this way provides an initial insight on how each dimension of the cube affects the amount resources needed for approximating the probability distributions on hardware.

However, as we mentioned earlier, with FINESSD we are resorting to the same low-level nuance as Fast-mRMR [16], reading the dataset in a feature-by-feature manner (column by column) instead of sample by sample (row by row). This allows us to alternate between only two copies of the aforementioned 2-D contingency tables (double buffering), having the need for only $2 \times bins \times labels$ counters instead of having the whole cube of counters available at any time in hardware. Hence, we can use one set of counters to calculate the score of feature X_k right after we have finished counting its histograms, while we use the other set to count the histogram of the next feature, X_k, in parallel. In this way, FINESSD is completely scalable as far as the features dimension is concerned, being able to handle an arbitrary number of features.

C. Design Space Exploration: Dimensioning and Trade-offs

FINESSD is configurable and can be completely adjusted according to the use case. To calculate the score for every feature as described in Equation 2, we introduce the dataflow of FINESSD shown in Figure 5. A look-up table is used to calculate the logarithm. There are some properties of the design that can drastically affect the area and the accuracy performance, such as the width of the counters and the logarithm look-up table length and width. However, not every configuration can perform well over any dataset.

Adjusting the width of the counters directly affects the logic resources needed for their implementation. Wider counters allow for a more accurate approximation of the different dynamics between the joint probability distributions, whereas narrow counters can still follow the distributions' trends. When a counter is about to overflow, all the counters are shifted right, avoiding the overflow and preserving the trends of the histograms. The number of counter overflow shifts depends on the number of samples in the dataset and how smooth or spiked the distributions are.

A similar intuition follows the logarithm look-up table. More precision is needed if the scores of features are very close; for example, in our synthetic dataset examples, all

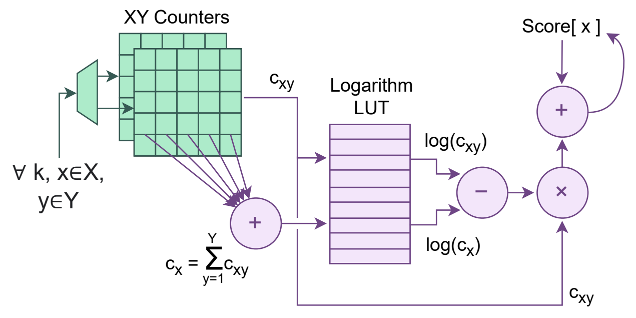


Fig. 5. The dataflow of FINESSD for approximating our information-theoretic scoring function consists of two seperate stages of the process, histogram counting (green) and processing (purple).

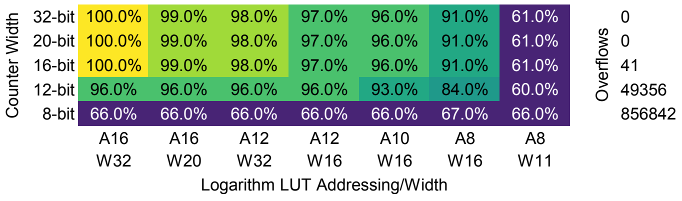


Fig. 6. The ranking accuracy when selecting 100 out of 2000 features of the "Epsilon" dataset compared to the baseline. Both the width of the counters and the dimensions of the logarithm look-up table affect the accuracy of FINESSD.

features follow very similar normal distributions. The table length, apart from the logarithm precision, also affects how much the counters need to be truncated for a look-up operation, while the table width affects the precision of the multiply-accumulate operations and thus that of the final score.

A detailed model of FINESSD has been created in MATLAB, enabling the quick design space exploration based on the aforementioned parameters over multiple datasets. The model reports the accuracy of FINESSD over the selected datasets, before the user proceeds to generate the design based on their needs. The results of such an example can be seen in Figure 6.

IV. FINESSD: Design and Implementation

In FINESSD, the three key steps of MIM feature selection (Ⓐ counting, Ⓑ calculating, Ⓒ sorting) are completely separate stages, making control data-driven and straightforward. The stages are pipelined, with each stage running in parallel to the next iteration of the previous stage. The architecture of the system can be seen in Figure 7. The design and function of its parts are analysed below.

A. Counting Histograms for Probability Distributions

There are two sets of counters for the contingency table of two features as we use double buffering to read the dataset input in a feature-by-feature manner. When FINESSD is counting the histograms of a feature, it is using the first set of counters. Once it reads the last sample of that feature, it copies the values of the counters to the second set in one clock cycle and uses the first set to count the histograms of the next feature.

As we mentioned earlier, histogram counting is one of the bottlenecks of calculating mutual information. For a near-storage solution, we need to be able to fully harness the bandwidth provided by a modern SSD, and in order to do that, we have to count fast. Accessing the data feature by feature instead of sample by sample, saves us a lot of counters (and hence resources and timing convergence effort), since we need to count only the joint appearances of values and labels for one feature at a time. Still, we need to do so at clock rate with a wide interface in order to reach high speeds for a generic near-storage solution applicable to faster storage devices.

Histogram FPGA designs in existing papers usually use either BRAM in a map-reduce fashion [18], [23], or elaborate encoding techniques which require substantial resources [24]. However, since we need to switch counting buffers in a few clock cycles between features, using two copies of multiple subcopies of a histogram would not be practical resource-wise. Also, as we explained in Section III-C, preventing the counters from overflowing is achieved by right-shifting them all when one counter is about to overflow. With multiple copies of counters in BRAM, that would require a LOAD-SHIFT-STORE operation sequence over every memory element.

Instead, we use hardware logic for all counters, being able to perform the aforementioned operations within a few clock cycles. FINESSD comes with a wide interface to fully harvest the high bandwidth of a storage device. Hence, we accept multiple inputs per clock cycle, and they all have to be counted without knowing how many common values we have in every read. Having all the counters checking if and how many of those values need to be counted in a clock cycle makes routing impossible.

For that reason, we stream the values through a Gearbox FIFO in order to reduce the processing frequency without sacrificing reading speed, providing routing with some slack. The overall performance of the feature selection pipeline is not affected by lowering the clock frequency, since processing is still completed before the next buffer change. Next, the stream of values is passed through shift registers. Every register is feeding a different subset of counters with values, and all the counters of that partition are checking all the values of that batch in one clock cycle, as shown in Figure 8. The width of the shift register is determined by the width of the interface, however the numbers of shift registers and counters per partition are customizable by the user.

In this way, we are achieving 20 to 40 times more samples counted per second for double the number of histogram bins compared to [24], depending on the interface width chosen by the user, as explained below.

B. Processing the Histogram and Sorting

Once a feature's joint probability distributions have been approximated by histogram counting, we copy the counts to the second pair of counter registers. The second pair of counts is used for calculating the feature's score, while the first pair of counters are reset and ready to be used for next incoming feature after 1 clock cycle.

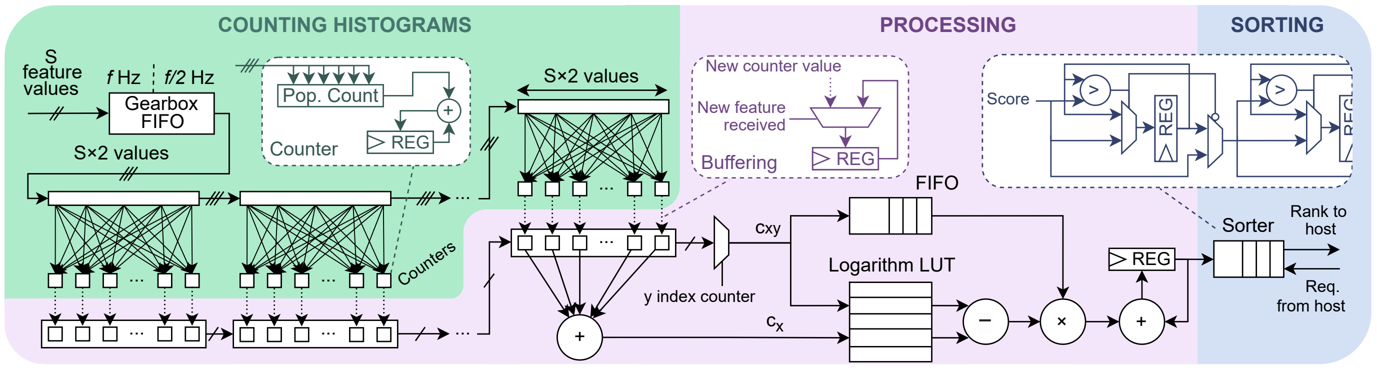


Fig. 7. The complete architecture of FINESSD. The three steps of MIM feature selection (A counting, B processing, C sorting) are seperate and independent from one another, with data moving in a stream-through fashion. Control signals are straight-forward and have been omitted in this figure, for example "Sort Enable" when Multiply-Accumulate is completed etc.

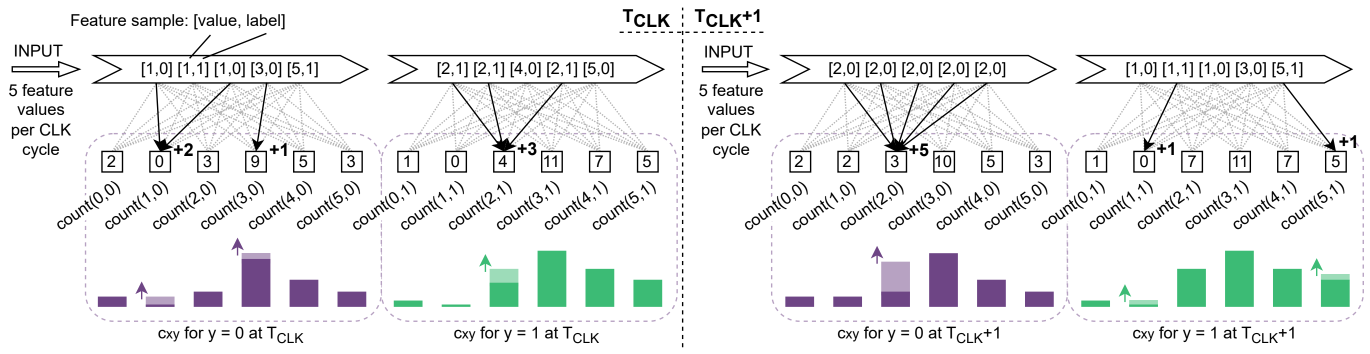


Fig. 8. An abstract representation of how FINESSD would partition counting the histograms for one feature of a 6-bin binary classification dataset, with the system accepting 5 sample inputs per clock cycle. For multiple values accepted per clock cycle and many counters, finding which values must be counted by which counter and how many of them are common makes routing impossible. We partition this population count task into batches of different counters for the same input and we propagate the input values through shift registers to limit routing congestion.

The processing part of FINESSD implements the dataflow shown in Figure 5 to calculate the information-theoretic score for every feature (Equation 2) in a completely pipelined manner, as shown in Figure 9. All the processing elements perform unsigned integer arithmetic operations. The logarithm is implemented as a look-up table in the FPGA's BRAM. A MATLAB script generates a suitable fixed-point representation and generates the look-up table, depending on the desired number of elements in the table. A scaling factor can be applied, in order to scale down the values as long as the scaling preserves as many distinctive values in the table as desired, before duplicates start to appear due to rounding. The granularity of the logarithm affects the final ranking of the features, especially when the distributions of the features are similar to one another.

As we can see in Figure 9, because of our approximate computing approach of transforming the calculation of mutual information into a few basic integer operations and look-ups, for calculations we only need a fraction of the time we need for counting. Hence, the whole processing and sorting part operates at half the frequency of counting, providing the place-and-route tools with some slack and saving resources.

Sorting is also conducted in a way that, as a module, it functions independently. As soon as there is a score input, a shift register with comparators in-between its registers transfers the new score value in its rank (Figure 7). We only accept a sorting value every time we have completed all the necessary calculations for a feature score, as denoted by the sole "sorting" stage in Figure 9.

C. Implementation

The system is implemented in hardware using Bluespec SystemVerilog (BSV) and tested as an RTL Kernel for AMD Vitis 2022.2. BSV combines the flexibility of High-Level Synthesis (HLS) with the explicitness of a Hardware Description Language (HDL). As a superset of SystemVerilog, it allows for complex design approaches that HLS does not provide, such as multiple clock domains within one module and custom interfaces between modules. At the same time, BSV's atomic rules offer a high level of concurrency, allowing for rapid design exploration and prototyping [25], [26].

FINESSD is integrated with the host application with the use of Xilinx Runtime Library (XRT), a set of libraries and drivers which enable not only the communication between the software and the hardware, but also the direct P2P connection between the SSD and the FPGA. For that reason, the hardware

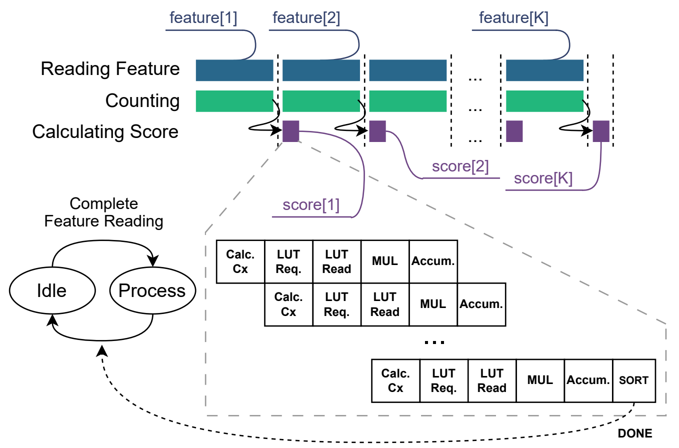


Fig. 9. The dataflow of processing a feature's histogram while counting the histogram of the next feature. Counting is performed directly when reading the dataset. Thanks to our approximate computing approach, processing requires only a few steps and it can be of a lower clock frequency. Therefore, we can focus routing and performance efforts on fast counting and make a difference for big datasets.

kernel of FINESSD supports the appropriate Block-Level Control Protocol required by XRT, which orchestrates our system's AXI4 interface. FINESSD can be called as a function in C/C++, accepting a pointer to the dataset file buffer and the dataset dimensions as an input. The dataset file is in NumPy's *.npy* binary format [27]. Column-wise reading is accomplished by simply storing the transpose of the dataset in storage using the corresponding built-in NumPy option.

The design is scalable and is automatically generated based on the aforementioned parameters of the logarithm table length and width, number of counters, counter width, counters per partition, and interface width. It can accept 16, 32, or 64 samples per clock cycle. For convenience and easier deployment, FINESSD comes with an optional AXI4 interface, either 512 or 1024-bit wide. With any of those two interface widths and any of the aforementioned number of inputs, FINESSD can process incoming data at 250 MHz, as long as there are enough samples per feature to be counted while processing the previous feature. The time it takes to process a dataset is the time needed to read it, plus the processing time and sorting for the last feature: $(N/S) \times M + 2 \times (bins \times labels + L_{MUL} + K + 2)$ cycles, where N is the number of samples, S is the number of values we accept in every cycle, M is the number of features, L_{MUL} the latency of the multiplier and K the number of features we want to select (final sorting).

D. Limitations

The limitation of FINESSD is its scalability, as there have to be enough counters in the FPGA implementation for any potential dataset dimensions and the corresponding histograms ($labels \times bins \times 2$). However, with the current technology, even with the modest-sized FPGA of the CSD used during evaluation (see Section V), FINESSD can fit 1600×2 32-bit counters, occupying only 66% of the resources; a configuration which should fit the vast majority of curated tabular datasets. Due to the stream-through pipeline of the FINESSD accelerator, the number of counters is primarily an area and not a timing issue.

V. Experimental Evaluation

A. Experimental Setup and Baseline

To evaluate FINESSD we use a SmartSSD, a CSD consisting of an SSD with onboard FPGA introduced by Samsung and AMD [28]. As shown in Figure 11-a, the SmartSSD consists of an NVMe SSD, an FPGA, FPGA DRAM and a dedicated, onboard PCIe switch. The FPGA DRAM is exposed as a common memory area to both the host and to the NVMe SSD. This configuration allows for direct P2P data transfers between the storage and the FPGA, and effectively minimizes PCIe traffic and the data transfer overhead from the SSD to the host and from the host to the FPGA (Figure 11-b), leaving to host only the initialization of P2P requests. The SSD's maximum reading bandwidth is 3.3 GB/sec while the maximum bandwidth of the FPGA DRAM is 15.4 GB/sec. The SmartSSD CSD has been used with great success in the past for other data-intensive applications such as sorting [29]–[31] and large-scale searching [32].

We install a SmartSSD in a computer system with an Intel Core i9-7900X chip running at 3.30 GHz (max 4.3 GHz) and 64 GB of RAM (DDR4 at 2133 MHz). The system runs Ubuntu 20.04.6 with Linux Kernel 5.15.0-79-generic. The version of the AMD tools (Vitis and XRT) is 2022.2. We compare FINESSD against mainstream and high-performance MIM software libraries and suites:

- A popular ML library, known as Scikit-learn [33], implemented in Python and its built-in MIM feature selection function, *SelectKBest(mutual_info_classif)*,
- A multi-process version of feature selection using Scikit-learn's *mutual_info_score* and *Dask* [34], which is a suite of tools for parallel, out-of-core computing in Python, supporting larger-than-memory datasets,
- FEAST [1], [35], an optimized feature selection suite implemented in C with MIM as a built-in option.

The metrics used in the evaluation are:

- Execution time and speedup: end-to-end time for reading the dataset from storage and producing the list with the top K selected features.

We only evaluate configurations with wide counters that produce 100% accuracy compared to the baseline, ensuring that FINESSD preserves the accuracy of the algorithm. We also feature a brief discussion on the FPGA resource allocation per example and the measured energy efficiency savings of FINESSD in contrast to the baseline.

FEAST accepts discretized datasets as inputs, stored in MAT v7.3 files. For Python we use NumPy files. NumPy files are high-performance binary files which can be used by all major ML Python tools such as TensorFlow, PyTorch and Scikit-learn [27]. Binary files are often preferred over hierarchical file formats such as HDF5 because they are splittable and have native support for the aforementioned ML suites.

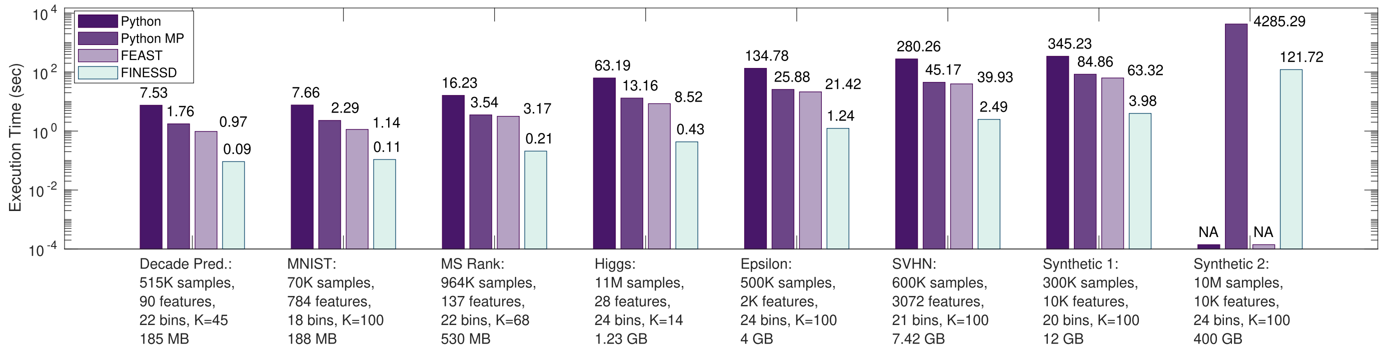
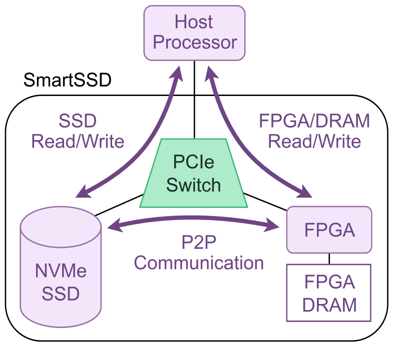


Fig. 10. Execution time of FINESSD compared to both mainstream and high-performance solutions running on a high-end desktop computer. The execution time includes the time to load the dataset from an SSD and select the K best features.



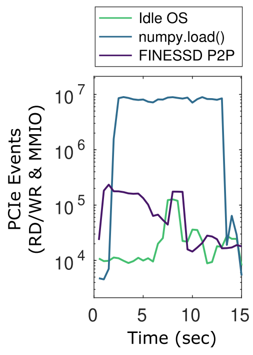


Fig. 11. a) Organizational overview of SmartSSD, b) PCIe events as perceived by the host CPU, measured by Intel Performance Counter Monitor (PCM) when the "Synthetic 1" dataset is being loaded by the host (NumPy) vs. FINESSD (P2P Connection).

While *SelectKBest* is the mainstream out-of-the-box solution, we deem that a fair approach should include a multiprocessing approach which does not require a lot of programming effort and would be what most users would do to gain performance. The numbers for the sequential version of Python are only mentioned to highlight the performance of the optimized version of Python using multiprocessing ("Python MP"). In the rest of the paper, we will only comment on the comparison between FINESSD, Python MP and FEAST.

We use the Enterprise-Class Samsung SSD (3.84 TB PCIe Gen3×4) found on the SmartSSD for the baseline software experiments and the near-storage processing experiments with FINESSD alike. To check whether this affects negatively the software baseline, we also run the software experiments using another SSD, a Samsung 970 EVO Plus 1 TB, instead of the SmartSSD. The results do not show any significant difference in the execution time, and thus for conciseness they are not presented.

B. Experiments and Results

For the experiments we used small, large and larger-than-memory datasets, to showcase the accuracy of our approximation and the effect of near-storage placement as the properties of the dataset change. The size and characteristics of the datasets vary in different dimensions, as shown in Figure 10.

TABLE I
COMPARATIVE PERFORMANCE OF FINESSD.

	Higgs	**Epsilon**	**SVHN**	**Synth. 1**	**Synth. 2**
Python	146.33×	109.02×	112.52×	114.39×	–
Python MP	30.47×	20.93×	18.13×	21.34×	35.21×
FEAST	19.72×	16.90×	16.03×	15.92×	–

The execution times of each case and the speedup of FINESSD can be seen in Figure 10 and Table I accordingly.

The datasets span several applications and fields, from digit classification on images (Modified National Institute of Standards and Technology – MNIST [36], Street View House Numbers – SVHN [37]), to particle physics (Higgs Boson [38]) and ranking models (Microsoft Learning to Rank [39]).

The first point of interest is that, even for relatively small datasets, FINESSD outperforms the baseline. This is attributed to the high throughput of the accelerator, which starts counting multiple samples per clock cycle right from the first AXI4 burst received. Second, we notice that FINESSD yields higher speedup for datasets with a large number of samples due to the more demanding histogram calculations.

In addition to real-world datasets, we evaluate FINESSD using synthetic datasets. Large datasets are in most cases proprietary, thus, it is common for the Big Data and ML communities to use synthetic datasets for evaluation [40]. In order to produce the dataset, we used Python's *dask_ml.datasets.make_classification* [41], a multiprocessing version of a widely-used dataset generation function. Synthetic datasets act as a "worst case scenario," as all feature samples follow similar normal distributions.

"Synthetic 2" is a larger-than-memory synthetic multiclass dataset, occupying 400 GB when stored as a NumPy binary file of 32-bit unsigned integers. As the dataset is larger than the host's memory, the multithreaded Python implementation reads data from storage when necessary through a memory-mapped file. Dynamic scheduling assures that the processor utilization is as high as possible, overlapping loading with computing in different cores. FINESSD shows its true power here, processing 400 GB in roughly 2 minutes, at a rate of 3.3 GB/sec, which is the maximum the SmartSSD can achieve,

TABLE II
RESOURCE ALLOCATION OF FINESSD ON THE SMARTSSD KINTEX ULTRASCALE+ KU15P FPGA

	Higgs	**MS Rank**	**Synth. 2**
LUT	12152 (3.08%)	25654 (6.50%)	55511 (14.07%)
LUTRAM	298 (0.20%)	258 (0.17%)	298 (0.20%)
FF	12477 (1.46%)	24612 (2.89%)	49975 (5.87%)
BRAM	19 (2.59%)	51 (6.94%)	73 (9.93%)

yielding a speedup of $35\times$ against Python MP, which requires over an hour for the same result. FEAST cannot run this dataset, as it does not fit in memory.

The FPGA resources required depend mainly on the number and size of counters, the logarithm table size and the number of features to select, which in turn affects the number of registers and comparators necessary during sorting. Some indicative results can be seen in Table II after the implementation of FINESSD for three of the datasets mentioned above. The FPGA resource allocation for the smallest, a medium and the largest design is shown. As we can see, even for the modest-sized FPGA of the SmartSSD, for all examples our implementation occupies at most 15% of the logic and 10% of the BRAM, leaving plenty of area for other kernels. A critical advantage of FINESSD is that the resources required remain the same regardless of the total number of features or the number of samples in the dataset.

From an energy consumption point of view, the SmartSSD utilizes a low-power FPGA requiring $\sim$7.5 W, $\sim$10 W including the FPGA DRAM according to documentation and relevant work [28], [30]. The SmartSSD card as a whole consumes a maximum of 25 W [28]. We used a power meter connected to the PSU of the computer system used in the experiments to measure the power consumption. When FINESSD is filtering "Synthetic2," the system consumes 106.4 W (1.34 variance). For the software baseline (Python MP) filtering "Synthetic2," the system consumes 222.45 W (10.7 variance). Comparing the energy consumed by the system for the larger-than-memory dataset, using FINESSD is $(P_{Baseline} \times t_{Baseline})/(P_{FINESSD} \times t_{FINESSD}) = 73.6\times$ more energy efficient.

Beyond PCIe Gen3 — The FPGA-based accelerator generated by FINESSD accepts either 32 or 64 16-bit unsigned integers, or alternatively, 16 or 32 32-bit unsigned integers per clock cycle at 250 MHz without the need to stall/throttle. Hence, FINESSD can process up to 32 GB/sec, provided that the memory interface and I/O bus can accommodate such a bandwidth. While that is more than enough for the SmartSSD's 3.3 GB/sec actual bandwidth, the fact that our design can be placed and routed with that configuration shows us that we can go beyond the speeds of PCIe Gen3 that the SmartSSD supports. With PCIe Gen4 still not having become a mainstream choice and the first PCIe Gen5 SSDs having been announced in 2023, FINESSD's potential speeds can still easily harness the bandwidth of the fastest available PCIe Gen5 SSDs of today at a bit under 15 GB/sec [42], [43].

VI. CONCLUSIONS

Feature selection and feature extraction represent the first stage for analysing large and complex datasets. Both reduce the dataset size as to accelerate the training of the appropriate ML algorithm as well as improving the accuracy of the inference. Feature selection has the advantage that it retains the meaning of the features and thus can be used with explainable ML algorithms.

FINESSD has analysed the preprocessing stage of ML pipelines from the point of view of near-storage feature selection and has focused on mutual information maximization (MIM) due to its low resource requirements while providing state-of-the-art results. We have deployed FINESSD on a Samsung SmartSSD, a computational storage device with an NVMe SSD, an FPGA, and a dedicated PCIe link connecting the two in the same package. The performance evaluation has shown how FINESSD can fully harness the bandwidth of a modern SSD and offload from the host processors the complete feature selection pipeline with significant end-to-end performance and energy gains.

FINESSD introduces a novel approximation for mutual information, and produces tailored designs which benefit from the polymorphic nature of FPGAs. Based on counters that record multiple feature samples at clock rate, FINESSD outperforms the baseline methods by tackling the data transfer and computation bottlenecks. FINESSD is the first near-storage application of feature selection, eliminating data transfers of low-reuse data from storage to host and increasing system-level parallelism. An important advantage of FINESSD is that the FPGA resources required remain constant regardless of the size of the dataset.

A comprehensive evaluation using datasets of different dimensions and complexity, has shown that FINESSD yields up to $35\times$ speedup over standard multiprocessing Python packages, and up to $19\times$ speedup over the FEAST optimized C library. Using FINESSD also provides important energy efficiency gains. The acceleration of feature selection was limited by current SmartSSDs using PCIe Gen3. The implementation of FINESSD can take advantage of PCIe Gen5. The evaluation has also provided the first thorough exploration on the approximation of mutual information with resource constraints derived for near-storage FPGAs, demonstrating the trade-off between accuracy and hardware resources.

Finally, mutual information has many applications beyond feature selection and ML. Other domains with large datasets where this approximation could be applied include Medical Imaging, Gene Analysis (reconstruction of gene networks or multisequence alignment), Cosmology, and Solar Physics.

ACKNOWLEDGMENTS

Nikolaos Kyparissas is supported by the Department of Computer Science Kilburn Scholarship. This work is partially funded by EPSRC EP/N035127/1 (LAMBDA project) and EP/T026995/1 (EnnCore project). Mikel Luján is supported by a Royal Society Wolfson Fellowship and an Arm/RAEng Research Chair Award.

References

[1] G. Brown, A. Pocock, M.-J. Zhao, and M. Luján, "Conditional likelihood maximisation: A unifying framework for information theoretic feature selection," *Journal of Machine Learning Research*, vol. 13, no. 1, p. 27–66, jan 2012.

[2] T. Liu, S. Liu, Z. Chen, and W.-Y. Ma, "An evaluation on feature selection for text clustering," in *Proceedings of the Twentieth International Conference on International Conference on Machine Learning*, ser. ICML'03. AAAI Press, 2003, p. 488–495.

[3] H. Liu and H. Motoda, *Computational Methods of Feature Selection (Chapman & Hall/CRC Data Mining and Knowledge Discovery Series)*. Chapman & Hall/CRC, 2007.

[4] A. A. Freitas, "Comprehensible classification models: A position paper," *SIGKDD Explor. Newsl.*, vol. 15, no. 1, p. 1–10, mar 2014. [Online]. Available: https://doi.org/10.1145/2594473.2594475

[5] J. Tang, S. Alelyani, and H. Liu, "Feature selection for classification: A review," in *Computational Methods of Feature Selection (Chapman & Hall/CRC Data Mining and Knowledge Discovery Series)*, H. Liu and H. Motoda, Eds. Chapman & Hall/CRC, 2014, ch. 2, pp. 37–64.

[6] V. Bolón-Canedo and B. Remeseiro, "Feature selection in image analysis: a survey," *Artificial Intelligence Review*, vol. 53, no. 4, pp. 2905–2931, Apr 2020. [Online]. Available: https://doi.org/10.1007/s10462-019-09750-3

[7] E. Hancer, B. Xue, and M. Zhang, "A survey on feature selection approaches for clustering," *Artificial Intelligence Review*, vol. 53, no. 6, pp. 4519–4545, Aug 2020. [Online]. Available: https://doi.org/10.1007/s10462-019-09800-w

[8] P. Dhal and C. Azad, "A comprehensive survey on feature selection in the various fields of machine learning," *Applied Intelligence*, vol. 52, no. 4, pp. 4543–4581, Mar 2022. [Online]. Available: https://doi.org/10.1007/s10489-021-02550-9

[9] C. E. Shannon, "A mathematical theory of communication," *Bell System Technical Journal*, vol. 27, no. 3, pp. 379–423, 1948.

[10] S. Zhang and Z.-Q. Lang, "Orthogonal least squares based fast feature selection for linear classification," *Pattern Recognition*, vol. 123, p. 108419, 2022. [Online]. Available: https://doi.org/10.1016/j.patcog.2021.108419

[11] L. Morán-Fernández and V. Bolón-Canedo, "Finding a needle in a haystack: insights on feature selection for classification tasks," *Journal of Intelligent Information Systems*, Nov 2023. [Online]. Available: https://doi.org/10.1007/s10844-023-00823-y

[12] S. Liu and M. Tian, "Mutual information maximization for semi-supervised anomaly detection," *Knowledge-Based Systems*, vol. 284, p. 111196, 2024. [Online]. Available: https://doi.org/10.1016/j.knosys.2023.111196

[13] A. Barbalace and J. Do, "Computational storage: Where are we today?" Jan. 2021, conference on Innovative Data Systems Research 2020.

[14] D. D. Sharma, R. Blankenship, and D. S. Berger, "An introduction to the compute express link (CXL) interconnect," 2023.

[15] J. González-Domínguez, R. R. Expósito, and V. Bolón-Canedo, "CUDA-JMI: Acceleration of feature selection on heterogeneous systems," *Future Generation Computer Systems*, vol. 102, pp. 426–436, 2020. [Online]. Available: https://doi.org/10.1016/j.future.2019.08.031

[16] S. Ramírez-Gallego, I. Lastra, D. Martínez-Rego, V. Bolón-Canedo, J. M. Benítez, F. Herrera, and A. Alonso-Betanzos, "Fast-mRMR: Fast minimum redundancy maximum relevance algorithm for high-dimensional big data," *International Journal of Intelligent Systems*, vol. 32, no. 2, pp. 134–152, 2017. [Online]. Available: https://doi.org/10.1002/int.21833

[17] K. Iordanou, S. M. Nikolakaki, P. Malakonakis, and A. Dollas, "A performance evaluation of multi-fpga architectures for computations of information transfer," in *Proceedings of the 18th International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation*, ser. SAMOS '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 1–9. [Online]. Available: https://doi.org/10.1145/3229631.3229635

[18] D. Conficconi, E. D'Arnese, E. Del Sozzo, D. Sciuto, and M. D. Santambrogio, "A framework for customizable FPGA-based image registration accelerators," in *The 2021 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 251–261. [Online]. Available: https://doi.org/10.1145/3431920.3439291

[19] L. Morán-Fernández, K. Sechidis, V. Bolón-Canedo, A. Alonso-Betanzos, and G. Brown, "Feature selection with limited bit depth mutual information for portable embedded systems," *Knowledge-Based Systems*, vol. 197, p. 105885, 2020. [Online]. Available: https://doi.org/10.1016/j.knosys.2020.105885

[20] L. Paninski, "Estimation of entropy and mutual information," *Neural Computation*, vol. 15, no. 6, p. 1191–1253, jun 2003.

[21] A. Kraskov, H. Stögbauer, and P. Grassberger, "Estimating mutual information," *Phys. Rev. E*, vol. 69, p. 066138, Jun 2004. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevE.69.066138

[22] B. C. Ross, "Mutual information between discrete and continuous data sets," *PLOS ONE*, vol. 9, no. 2, pp. 1–5, 02 2014. [Online]. Available: https://doi.org/10.1371/journal.pone.0087357

[23] R. Kastner, J. Matai, and S. Neuendorffer, "Parallel Programming for FPGAs," *ArXiv e-prints*, May 2018.

[24] S. A. Fahmy, "Histogram-based probability density function estimation on FPGAs," in *2010 International Conference on Field-Programmable Technology*, 2010, pp. 449–453.

[25] R. Nikhil, "Bluespec System Verilog: efficient, correct RTL from high level specifications," in *Proceedings. Second ACM and IEEE International Conference on Formal Methods and Models for Co-Design, 2004. MEMOCODE '04.*, 2004, pp. 69–70.

[26] T. Bourgeat, C. Pit-Claudel, A. Chlipala, and Arvind, "The essence of Bluespec: A core language for rule-based hardware design," in *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation*, ser. PLDI 2020. New York, NY, USA: Association for Computing Machinery, 2020, p. 243–257. [Online]. Available: https://doi.org/10.1145/3385412.3385965

[27] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, "Array programming with NumPy," *Nature*, vol. 585, no. 7825, pp. 357–362, Sep. 2020. [Online]. Available: https://doi.org/10.1038/s41586-020-2649-2

[28] J. H. Lee, H. Zhang, V. Lagrange, P. Krishnamoorthy, X. Zhao, and Y. S. Ki, "SmartSSD: FPGA accelerated near-storage data analytics on SSD," *IEEE Computer Architecture Letters*, vol. 19, no. 2, pp. 110–113, 2020.

[29] S. Salamat, A. Haj Aboutalebi, B. Khaleghi, J. H. Lee, Y. S. Ki, and T. Rosing, "NASCENT: Near-storage acceleration of database sort on SmartSSD," in *The 2021 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 262–272. [Online]. Available: https://doi.org/10.1145/3431920.3439298

[30] S. Salamat, H. Zhang, Y. S. Ki, and T. Rosing, "NASCENT2: Generic near-storage sort accelerator for data analytics on SmartSSD," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 15, no. 2, jan 2022. [Online]. Available: https://doi.org/10.1145/3472769

[31] W. Qiao, J. Oh, L. Guo, M.-C. F. Chang, and J. Cong, "FANS: FPGA-accelerated near-storage sorting," in *2021 IEEE 29th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2021, pp. 106–114.

[32] J.-H. Kim, Y.-R. Park, J. Do, S.-Y. Ji, and J.-Y. Kim, "Accelerating large-scale graph-based nearest neighbor search on a computational storage platform," *IEEE Transactions on Computers*, vol. 72, no. 1, pp. 278–290, 2023.

[33] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[34] Dask Development Team, *Dask: Library for dynamic task scheduling*, 2016. [Online]. Available: https://dask.org

[35] A. Pocock, *FEAST: A FEAture Selection Toolbox for C/C++ & MATLAB/OCTAVE, v2.0.0.*, 2017. [Online]. Available: https://github.com/Craigacp/FEAST

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

[37] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng, "Reading digits in natural images with unsupervised feature learning," in

NIPS Workshop on Deep Learning and Unsupervised Feature Learning 2011, 2011.

[38] P. J. Sadowski, D. Whiteson, and P. Baldi, “Searching for Higgs Boson decay modes with deep learning,” in *Advances in Neural Information Processing Systems*, Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Weinberger, Eds., vol. 27. Curran Associates, Inc., 2014.

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

[40] A. Kleerekoper, M. Pappas, A. Pocock, G. Brown, and M. Lujan, “A scalable implementation of information theoretic feature selection for high dimensional data,” in *Proceedings of the 2015 IEEE International Conference on Big Data (Big Data)*, ser. BIG DATA ’15. USA: IEEE Computer Society, 2015, p. 339–346. [Online]. Available: https://doi.org/10.1109/BigData.2015.7363774

[41] D. developers, “Dask API reference: dask_ml: datasets: make_classification,” Oct 2023. [Online]. Available: https://ml.dask.org

[42] S. Downing, “Crucial T700 SSD review: The temporary king,” May 2023. [Online]. Available: https://www.tomshardware.com/reviews/crucial-t700-ssd-review

[43] C. Robinson, “Sabrent shows progress building the fastest PCIe Gen5 M.2 SSD,” Jul 2023. [Online]. Available: https://www.servethehome.com/sabrent-shows-progress-building-the-fastest-pcie-gen5-m-2-ssd/