Exploiting Sum-Product Networks to Offload Database Query Cardinality Estimation to FPGA-based Smart Storage Devices

Lukas Weber[1 [0000−0003−0621−1151]], Yannick Lavan[1 [0000−0001−5309−4141]], Johannes Wehrstein[2 [0000−0002−7152−8959]], Torben Kalkhof[1 [0000−0002−4159−244X]], Carsten Heinz[1 [0000−0001−5927−4426]], Carsten Binnig[2 [0000−0002−2744−7836]], and Andreas Koch[1 [0000−0002−1164−3082]]

[1] Embedded Systems and Applications Group, TU Darmstadt, Germany
<surname>@esa.tu-darmstadt.de
[2] Systems Group, TU Darmstadt, Germany
<firstname>.<surname>@cs.tu-darmstadt.de

Abstract. Cardinality estimation is crucial for optimizing query performance in database systems. This study explores the application of Sum-Product Networks for estimating the cardinalities of database queries across various architectures. We analyze the capability of SPNs to handle different query types, highlighting their strengths and limitations.

Building upon existing research, we have developed a framework that creates both offload and smart storage hardware accelerators tailored for cardinality estimation. Compared to prior work, these accelerators utilize simplified fixed-point arithmetic to enhance resource efficiency. Our framework enables the generation of variants optimized for latency and throughput adaptable to diverse system architectures. Our approach now supports marginal and range-based queries essential for accurate cardinality estimation by extending prior functionality.

We applied our framework to generate accelerators and integrated them into two distinct architectures: a PCIe-based accelerator card for offloading tasks in large-scale general-purpose databases and a Near-Data Processing system within the COSMOS+ OpenSSD smart storage SSD. Our evaluation indicates that the PCIe-based architecture achieves over 165 million inferences per second, with latencies as low as 6.62 microseconds, and the Smart Storage device achieves latencies under 2 microseconds. Additionally, we analyzed the impact of our simplified fixed-point number system on resource efficiency and error margins, enabling a different trade-off between the different resources in a typical FPGA to make the existing framework more adaptable to different environments and applications.

Keywords: Sum-Product Networks · Probabilistic Models · Machine Learning · Cardinality Estimation · FPGA

1 Introduction

In recent years, machine learning and big data have become active research fields within computer science. With the development of models such as ChatGPT by OpenAI, interest in AI has increased significantly. More recently, companies like Google and Anthropic have started creating and training their respective models, and even more open alternatives like DeepSeek have become public, virtually sparking an AI race with researchers and corporations delivering improved models regularly. These models deal with an ever-increasing amount of data produced and stored daily, making data storage and management equally important. However, while there have been efforts to improve storage systems, they are much less in the public eye, and there is still significant potential for optimization. New GPUs and CPUs often come with specific optimizations for AI, with AI permeating into all sections of the hardware market, starting at data-center GPUs and desktop and mobile CPUs and GPUs.

While consumer storage devices have advanced in capacity, latency, and throughput, most of these advances are not specific to machine learning. One potential improvement to storage devices for machine learning applications is the use of Near-Data Processing (NDP), which offloads computational load from the CPU to the storage device. NDP is especially interesting in applications where the stored data can be pre-processed on the storage device using data-reductive operations such as selections. By performing these operations on the storage device, bandwidth on the PCIe bus can be freed up, and the transported data is more relevant to the application. For example, consider training a model on a vast dataset containing functionally dependent data (e.g., age and birth date). Removing such redundancies using NDP projection avoids inefficient data transfers and could increase performance if data movement is a bottleneck.

To implement NDP, smart storage devices such as the Samsung SmartSSD [3] or the Zynq-7000-based COSMOS+ OpenSSD [16] are employed. The COSMOS+ OpenSSD is a regular NVMe-based SSD, but its flash controllers are implemented in the programmable logic (PL) of the Zynq-7000 SoC, and the Cortex A9 cores are used to run firmware. To enable NDP, the hardware on the PL and firmware of the COSMOS+ can be extended with user-defined functionality. Thanks to the FPGA-based SoC, simple NDP operations can be realized in hardware or software, as shown in [20]. For more complex NDP operations, result handling becomes an important problem [17], as intermediary results must be materialized. Efficient materialization is only possible for results that fit into block RAM (BRAM) or dynamic RAM (DRAM), making the prediction of result sizes a relevant issue to determine the best result-handling strategy. Predicting result sizes is generally called Cardinality Estimation (CE).

In addition to NDP, CE is also relevant in more traditional database systems. Specifically, it is used in query optimization, translating complex database queries into a sequence of subqueries called an execution plan. Optimally, data-reductive selections and projections are performed *early* in an execution plan since this will reduce the runtime of more complex later operations like joins.

One relatively novel approach to CE is using Sum-Product Networks (SPNs), as demonstrated in DeepDB [8]. While this work provides an interesting proof of concept for using SPNs in CE, it lacks detail on how queries are estimated using SPNs. Instead, the paper focuses on additional applications of SPNs in database and storage systems. SPNs are a probabilistic graphical model used for various tasks, including classification, regression, and density estimation. In the context of CE, SPNs can be trained to estimate the cardinality of data sets, potentially improving performance and accuracy over traditional methods. However, further research is needed beyond DeepDB to explore the feasibility and effectiveness of this approach in practical applications.

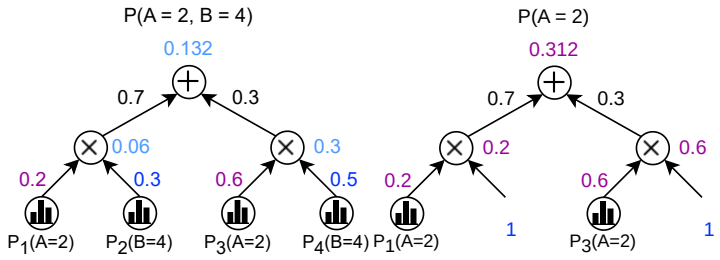
This paper has three specific contributions. First, we further elaborate the advantages and drawbacks of using SPNs for CE, explicitly focusing on the types of queries required for estimating the materialization within smart storage systems. While the underlying approach is similar to the prior work, we precisely define the types of queries and evaluate their suitability for usability in smart storage. Second, we implement an accelerator generation framework to allow for more complex queries by replacing the naive histogram probability lookup with a module capable of computing a multitude of sub-operations on the histogram. Lastly, we use a set of typical SPNs to generate several accelerator examples and drop them into multiple different architectures, evaluating the suitability of the approach in smart storage as well as general-purpose database systems. To this end, we evaluate the approach in detail on two specific platforms (COSMOS+ & AMD Alveo U280) while also providing a forward-looking What-If analysis using the Avnet Ultra96 as a stand-in for a potential UltraScale+ -based update of the limited COSMOS+ platform. We also include the Xilinx VC709 to allow a more direct comparison to prior work, which already employed that platform. Ultimately, our work provides new insights into the feasibility and effectiveness of using SPNs for CE in the context of practical NDP and general-purpose database applications on a wide range of different potential application spaces.

2 Background

2.1 Sum-Product Networks

SPNs [12] are probabilistic circuits represented as directed acyclic graphs (DAGs) that encode joint distributions over random variables. SPNs consist of three types of nodes: weighted sums, products, and leaves, which encode univariate or multivariate distributions over random variables. By performing a bottom-up pass through the DAG, joint and marginal inference can be efficiently performed on complete or partial evidence. Figure 1 provides an example of joint and marginal inference in an SPN.

SPNs can be constructed by hand for a specific purpose, or automatically trained on a given dataset. Different training approaches can be classified as structure learning, weight learning, or a combination of both. Random generation of SPNs with subsequent weight learning has also been shown to be a practical approach [11]. For example, an SPN can be learned from scratch by



(a) Joint inference (b) Marginal inference

Fig. 1: Inference example in an SPN, representing the joint probability distribution $P(A, B)$. In joint inference, all histograms output a corresponding value (a), while in marginal inference some histograms are marginalized and always output the value 1.0 (b).

performing structure learning to identify the relevant independent variables and their relationships, or refined by adjusting the weights of an existing SPN to improve accuracy. Relevant for this work, there is also research on learning SPNs from relational databases [8], with the goals of performing CE and Approximate Query Processing (AQP) in database and storage systems. This involves learning the underlying structure of the database schema and using that information to construct an SPN that accurately estimates the number of distinct values in a given query. The ability to automatically learn SPNs from data makes them a powerful tool for a wide range of applications in machine learning and artificial intelligence.

2.2 Cardinality Estimation

CE is a key operation in database and storage systems and used to predict the size (cardinality) of query results. In relational databases, CE specifically estimates the number of rows in a table or in an intermediary result. CE is often used in query optimization to rearrange subqueries for improved performance. Most databases use simple approaches for CE based on approximating data distributions using histograms and/or cost models. However, recent work suggests that machine learning and learned models are also a valid approach for CE [8]. SPNs are a promising approach for CE because they can perform very precise estimations of probabilities. Since CE does not require high precision, we are able to trade lower arithmetic for space efficiency.

3 Related Work

Cardinality Estimation The core idea of this work is derived from [8], which uses SPNs to perform CE, as well as AQP. In their work, Hilprecht et al. define

Relational SPNs (RSPNs), which are an extension of regular SPNs aimed towards relational databases. Specifically, RSPNs typically come in sets or ensembles which represent multiple datasets or database tables. Additionally, RSPNs support typical database-specifics like NULL values, handling of functional dependencies, and incremental training to keep the RSPN in sync with updates of the dataset. The authors show that CE and AQP using RSPNs is feasible. While there is prior work on CE for relational databases, most of those works do not use SPNs as model. The relatively extensive survey by Harmouch et al. discusses typical approaches for CE [5].

Near-Data Processing While first experiments towards NDP took place as early as the 1970s, most of these early approaches (like database machines [2] and ActiveDisk [1]) were rather unsuccessful, due to I/O and general bandwidth limitations. More recently, there has been work with Smart SSDs that exploit the higher bandwidth of typical flash memories [3].

Also, *bump-in-the-wire* processing has become more relevant, with a number of publications by Vincon et al. [20], which introduced the concept of cross-layer data formats in NDP-based key-value stores. By giving the storage device knowledge about the structure of the stored data, typical key-value store operations like GET and SCAN could be performed on a COSMOS+ OpenSSD. They describe corresponding implementations in [20], proving that more complex operations like SCAN can profit from software- and FPGA-based NDP. Furthermore, the work was later extended by an approach to generate the FPGA-based NDP operators *automatically* from annotated code [19]. Lastly, the authors have shown the importance of result-set handling and suggest possible solutions in [17].

Sum-Product Networks The first relevant paper considering the generation of specific hardware modules for SPN inference was published in 2018 by Sommer et al. [14]. It originally employed a compiler-like approach to read SPNs from a textual representation and generate corresponding accelerators using FloPoCo 64-bit floating point operators. In later works, this approach was extended to different custom data types like a logarithmic number system [18], as well as posit numbers and a custom floating point number system [15]. In more recent works, the SPN accelerators were integrated in different architectures enabling inference in 100G networks [6] and using fast on-chip HBM [21].

In parallel, Shah et al. developed a custom architecture for executing inference on probabilistic circuits [4, 13]. Their approach also features improvements to the learning process that ensure that intermediary results are as precise as possible given different configuration parameters of the overall architecture.

4 SPNs and Cardinality Estimation

Prior to discussing the hardware implementation of probabilistic cardinality estimation as FPGA accelerators, we present a brief overview of key concepts driving the implementation approach. For a comprehensive discussion on incorporating SPNs into database systems, we refer the reader to Hilprecht et al. [8]. In this study, we focus on applying FPGA-based cardinality estimation for data distri-

butions represented as SPNs over histogram leaf nodes. We assume *normalized* histograms, where each bucket signifies a probability rather than a density.

4.1 Estimating Query Cardinalities

In this section, we describe the probabilistic operations executed for corresponding database queries. Throughout this section, we represent the learned probability distribution for a database table T by $P(\mathbf{X})$, where $\mathbf{X}$ is the vector of random variables corresponding to each column in T. Generally, the estimated cardinality c of a query result is calculated as an expectation over the learned data distribution by determining the probability of the provided evidence vector $\mathbf{E}$ and multiplying it by the number of rows n in the table, as follows:

$$c = \mathbb{E}_{x \sim P(\mathbf{X}|\mathbf{E})}(\mathbf{X}) \approx n \cdot P(\mathbf{E}). \tag{1}$$

Next, we explain how to obtain $P(\mathbf{E})$ for various types of database queries.

Single-Column Equality Queries In the most trivial filtering case, we aim to estimate the outcome of filtering a table for a provided value in a specific column. The evidence provided to the SPN consists of a single random variable X_i corresponding to the column i of interest, resulting in the computation

$$c \approx P(X_i = x) \cdot n, \tag{2}$$

effectively marginalizing every other column. In SPNs, the marginalization is achieved by outputting the value 1.0 for leaf nodes of marginalized variables.

Range Queries In many applications, we are interested not only in single values for columns but also in ranges of data, such as "determine the number of scientists who have published *less* than five papers". Formally, we aim to evaluate the probability that the value of a random variable X_i falls within the range $[x_l, x_m]$ with $l < m$. Calculating the probability involves determining the probability of the or-event of several mutually exclusive events:

$$P(x_l \leq X_i \leq x_m) = \sum_{j=l}^{m} P(X_i = x_j). \tag{3}$$

Since non-leaf operations in SPNs remain constant w.r.t. j in Eq. 3, the sum propagates into the histogram leaves corresponding to X_i. This property enables us to compute the probability of single-column range queries without evaluating the entire SPN multiple times. Modifying the comparison operators within the query only changes the start and end indices for the sum operation.

AND Queries While the previous paragraphs focused on single-column queries, we can also combine queries. In this work, we restrict our scope to queries of the form `A <= 42 AND B = 123`, excluding queries like `A <= 42 OR B >= 10`, as the latter would require multiple SPN passes, since SPNs encode joint probabilities. To compute the corresponding query, at least three passes are required: One to determine `A <= 42`, and a second one for `B >= 10`. Since both subqueries might overlap on the underlying dataset, the overlap has to be

determined as well (`A <= 42 AND B >= 10`), to ensure that the overlap is not included twice. Furthermore, we limit the range queries to one predicate over a range and any number of equality predicates, as we cannot propagate the sum over all outcomes as before. Formally, we assess the probability $P(x_l \leq X_i \leq x_m, \mathbf{E} = \mathbf{e})$, with X_i corresponding to the column for the range computation and $\mathbf{e}$ representing evidence for other columns of interest $\mathbf{E}$. This probability is obtained by computing the sum inside X_i's histogram nodes, setting all other histograms for relevant columns to the corresponding evidence, and outputting *one* for all histogram nodes of marginalized columns. For practical purposes, we later empirically evaluate the effect of approximating cardinalities by applying the histogram summing of single range queries to multiple columns.

4.2 Hardware-Specific Model Optimization

While histograms with few buckets can be realized easily on FPGA through BRAM, histograms with a growing number of buckets, e.g. for 32-bit integers would either require LUTs with an unfeasible number of entries or merging of histogram buckets which may result in loss of representation accuracy. Due to the nature of SPNs to learn distributions over arbitrary data, we can make use of the underlying probabilistic semantics and simplify the resulting hardware.

Let X^n denote some n bit wide random variable. Then $P_L(X^n = x)$ denotes the probability output by a histogram leaf L for X^n taking the value x. Let us now assume w.l.o.g. that we slice X^n evenly into four bit chunks. Then we can view $P_L(X^n = x)$ as the joint probability
$\hat{P}_L(X^4_{\lfloor n/4 \rfloor - 1} = x_{n-1:4\cdot(\lfloor n/4 \rfloor - 1)}, ..., X^4_1 = x_{7:4}, X^4_0 = x_{3:0})$, with X^4_i denoting the random variable corresponding to the i-th nibble of X^n and $x_{m:n}$ corresponding to the concrete assignments of bits m through n of x.

Leveraging this insight, we can either retrain the entire SPN by pre-processing the training data and bit-slicing each data point to the desired BRAM size or by learning SPNs representing each histogram node and replacing the histogram nodes with the corresponding SPN.

4.3 Empirical Analysis

To evaluate the feasibility of SPNs for CE, we developed a software simulation that executes queries both exactly—by filtering the dataset—and approximately—via inference on a trained SPN. Our preliminary evaluation uses the NIPS dataset, which represents word frequencies in ML publications as a large table. Its many columns with small value ranges allow systematic query enumeration. While the approach can be applied to other datasets (see Section 4.2), the NIPS dataset offers variants with increasing column counts, which is particularly useful for later performance and hardware utilization analysis (Section 6).

Since SPN inference outputs query selectivity, we base our analysis on selectivity and derive cardinality by multiplying it with the dataset size. Queries are generated through range analysis per column, enabling enumeration of all single-column (SC) queries and their combination into multi-column (MC) queries. SC

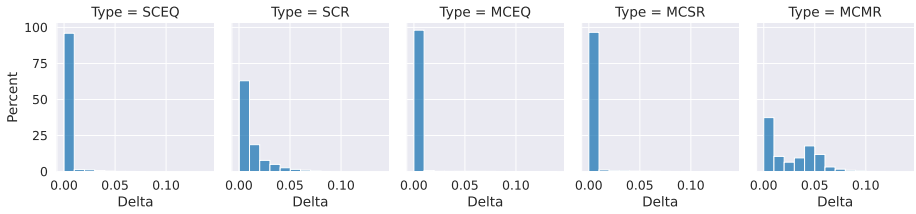


Fig. 2: Observed estimation error for range queries on all NIPS datasets.

queries are classified as equality (SCEQ) or range-based (SCR), while MC queries are grouped into equality-only (MCEQ), single-range (MCSR), or multi-range (MCMR) queries.

We restrict MC queries to at most three columns, as larger queries would typically be decomposed by query optimization. Additionally, we exclude queries with zero true results, since SPN inference yields near-zero selectivities with negligible absolute error in these cases, making them irrelevant for typical CE use cases. Table 1 summarizes the datasets and query counts used in our analysis.

Table 1: Datasets and the corresponding number of benchmark queries

	SC		MC			
Dataset	EQ	R	EQ	SR	MR	Overall
NIPS5	111	710	9,782	134,682	200,000	345,285
NIPS10	227	1,426	92,548	200,000	200,000	494,201
NIPS20	480	2,882	129,092	200,000	200,000	532,454
NIPS30	740	4,400	165,960	200,000	200,000	571,100
NIPS40	1,027	5,980	200,000	200,000	200,000	607,007
NIPS50	1,255	7,626	200,000	200,000	200,000	608,881
NIPS60	1,573	9,308	200,000	200,000	200,000	610,880

Single-Column Equality (SCEQ) First, we enumerate all SCEQ queries yielding at least one result. For each query, we then execute the query and the corresponding SPN inference to compute the actual and estimated selectivity. The leftmost subplot of Fig. 2 shows the distribution of the estimation error. It shows that the estimation error is below 0.05 for almost all queries over all datasets. The maximum estimation error is 0.075.

Single-Column Range (SCR) For single-column range queries, we take a similar approach. As we can see in Fig. 2, the distribution of estimation errors changes. While most estimation errors are still relatively small, there are less cases with an estimation error of less than 0.01. This is the case, since the histograms of the trained SPNs sometimes do not sum up to exactly one due to the used double-precision floating point numbers.

Multi-Column Equality (MCEQ) As discussed in Section 4, the error of this class of queries should not significantly increase over SCEQ, as the inference does not introduce any mathematical issues apart from potential precision errors due to the digital arithmetic. Accordingly, the estimation error behaves relatively similar to SCEQ. Additionally, the errors are also typically less than for the SCR queries, since they compute over more histogram buckets.

Multi-Column Single Range (MCSR) For these queries, the estimation error also behaves as expected. Interestingly enough, the impact of the range-based subquery is not as prevalent as in SCR. While the range-based subquery will introduce an estimation error, its impact is not as high, as the rest of the computation is still relatively precise.

Multi-Column Multiple Ranges (MCMR) This class is the most interesting. As discussed in Section 4, the estimation error in this class should be relatively high, since summing up multiple histograms actually violates the mathematical precedence. As expected, the violation of precedence leads to more queries yielding higher estimation errors. But while more queries have higher estimation errors, the worst-case error still remains below 0.1, and is thus still in the same magnitude as for the other classes. So empirically, it is possible to perform CE for the given queries on the given dataset.

5 SPN Accelerators

Although SPNs are a relatively new ML model, substantial prior work explores FPGA acceleration for SPN inference. We build a separate framework inspired by [14], but simplify the numeric encoding by using fixed-point arithmetic instead of the more complex schemes in prior work. Since SPN values are probabilities in the range $[0, 1]$, a single integer bit suffices, and the number of fractional bits is configurable in Chisel3 to trade precision for resource usage. Unlike prior work targeting very low relative errors 10^{-6} [15], CE tolerates larger errors, as SPN estimation errors are already small in practice [8]. This allows a simpler encoding and reduced hardware cost.

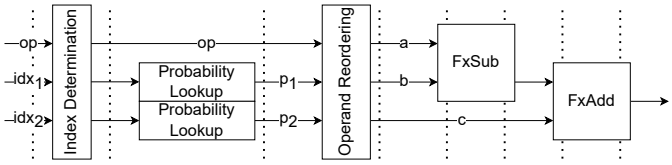


Fig. 3: FxHistogram Module. Dotted Lines indicate pipeline stages. The number of pipeline stages for FxSub and FxAdder depends on the used encoding.

SPN inference requires three hardware modules: FxAdder, FxMultiplier, and FxHistogram. The FxAdder performs saturated fixed-point addition, while the

FxMultiplier uses a parameterized tiling approach to map onto FPGA DSP resources. All modules are implemented as parameterized Chisel3 generators and instantiated by traversing the SPN DAG, yielding a fully spatial, pipelined accelerator with AXI Stream interfaces.

The accelerator supports both low-latency single inference via AXI4-Lite and high-throughput batch inference via AXI4, enabling use in smart storage and PCIe-based database accelerators.

FxHistogram Module The FxHistogram module is novel over prior work, which exclusively focused on *equality*-based bottom-up inference. It enables other compare operations ($\neq, <, \leq, >, \geq$), as well as a range- and a marginalization-operations.

Table 2: Functionality of FxHistogram and its submodules. b denotes the memory storing the accumulated histogram buckets.

		Probabilities		Operands		
Op	Calculation	p_1	p_2	a	$-b$	$+c$
$=$	$(b[i] - b[i-1])$	$b[id_1]$	$b[id_2]$	p_1	p_2	0
$\neq$	$1 - (b[i] - b[i-1])$	$b[id_1]$	$b[id_2]$	1	p_1	p_2
$<$	$(b[i-1] - 0)$	$b[id_1]$		p_1	0	0
$\leq$	$(b[i] - 0)$	$b[id_1]$		p_1	0	0
$>$	$1 - (b[i] - 0)$	$b[id_1]$		1	p_1	0
$\geq$	$1 - (b[i-1] - 0)$	$b[id_1]$		1	p_1	0
R	$(b[j] - b[i])$	$b[id_1]$	$b[id_2]$	p_2	p_1	0
M				1	0	0

The new module (cf. Fig. 3) enables additional operators, while the implementations from prior work could only compute equality-based inference. The first step in enabling the additional operations was the introduction of accumulated probabilities. In prior work, the probability lookup was limited to equality, so the value of each bucket could simply be stored in read-only memory using the index of the bucket as address. To enable the computation of ranges of buckets, we instead compute the accumulated probability. For each bucket, all probabilities up to and including the current one are summed and stored. By using this approach, the calculation for each of the corresponding operations can be done using two probability lookups. The corresponding addresses (idx_1 and idx_2) are determined in the Index Determination stage. Depending on the operation, up to two lookups happen concurrently, yielding the probabilities p_1 and p_2. The required calculations can be computed using three operands (a, b and c), where up to two operands are set to either zero or one, depending on the operation. The Operand Reordering stage will reorder the incoming probabilities and add zeroes or ones accordingly, so that the correct final result can be computed using the normalized computation $a - b + c$. The exact mathematical calculations, the required lookups and the final operands are shown in Table 2.

5.1 System Integration

As detailed earlier, the main accelerator can be integrated using the two provided wrappers. Additionally, it could also be used standalone in a more complex accelerator or in a network-attached use-case. Both provided wrappers use an AXI4Lite interface to expose control registers, as well as an interrupt signal to enable asynchronous execution. The throughput-optimized variant also provides an AXI4 interface for batch processing. Using the latency-optimized variant, we tested the accelerator on five setups for the NIPS40 dataset, achieving end-to-end execution times shown in Table 3. The software interface runs on the ARM cores in SoC-based platforms and on the host CPU in PCIe-based platforms.

Setup (a) integrates the accelerator in a *Near-Data Processing* (NDP) scenario on the COSMOS+ smart (computational) SSD. Execution of SPN inference takes 6.85 µs, when controlling the accelerator from a baremetal firmware running on the Cortex A9 of the Zynq 7000 SoC. Polling is used to determine whether the execution has finished. As the Zynq-7000 in the COSMOS+ has been superseded by more recent FPGAs, Setup (b) evaluates the accelerator on more recent hardware (Ultra96), but retains the rest of the COSMOS+ architecture, achieving a latency of just 1.6 µs.

Setups (c), (d), and (e) employ the accelerator in *offload* mode, instead of the baremetal NDP approach of (a) and (b). They use the TaPaSCo framework [7,9] for system integration and as runtime API. Setup (c) uses TaPaSCo on an Ultra96, running an embedded Linux on the PS of the Zynq UltraScale+. Setups (d) and (e) both use PCIe-based accelerator cards (VC709 and Alveo U280). These are connected to their respective host via PCIe Gen3 x8 and x16 respectively. While inference is still relatively fast, the added overheads of operating systems and interrupts increases latency significantly. This is especially visible for the Ultra96 with a latency of 26.7 µs. This is due to the OS running on the rather limited embedded CPUs. For the PCIe-based platforms, the impact of OSs and interrupts is less significant, due to the much higher performance of the host CPUs. Thus, the resulting latencies are better here than for setup (c), even though the latency includes PCIe transfers. While the U280 is more recent compared to the VC709, the VC709 still reaches a lower latency. This shows the impact of the employed host machine: The VC709 is connected to a workstation, while the U280 resides in a server with a more complex PCIe subsystem, which increases latency. We report the value based on the server-setup as it is more representative of real-world usage of a data-center FPGA card.

6 Evaluation

To evaluate our framework, we use the different variations of the NIPS dataset as benchmarks. Each variation comes with a trained SPN, which was trained using the *SPFlow* library [10]. As queries we use the enumerated and combined queries we generated for the empirical analysis of SPN-based CE. In addition to the different SPNs, we also want to evaluate the two accelerator objectives (latency- vs. throughput-optimized). Finally, we also want to evaluate the error

Table 3: Tested Architectures

Test Setup	**(a)**	**(b)**	**(c)**	**(d)**	**(e)**
Platform	COSMOS+	Ultra96	Ultra96	VC709	U280
Arch	Zynq 7000	Zynq US+	Zynq US+	PCIe	PCIe
Fabric	Kintex 7	UltraScale+	UltraScale+	Virtex 7	UltraScale+
Driver	none	none	TaPaSCo	TaPaSCo	TaPaSCo
Control	Polling	Polling	Interrupt	Interrupt	Interrupt
Freq.	200 MHz	440 MHz	440 MHz	200 MHz	420 MHz
Latency	6.85 µs	1.6 µs	26.7 µs	16.1 µs	21.3 µs

margins of the fixed-point encoding depending on the number of bits. Thus, we generate the accelerators using different encodings, resulting in 56 different accelerators (7 SPNs $\times$ 2 variants $\times$ 4 encodings).

6.1 Benchmarks

For our evaluation, we rely on the NIPS dataset. While not being a traditional database or cardinality estimation benchmark, it is advantageous for our study due to its scalability. The number of each benchmark indicates the size of the underlying entries in bytes. For example, a NIPS60 entry is comprised of 60 separate values that are encoded with a single byte each. Accordingly, the NIPS dataset is useful for showing increasing error margins, and decreasing throughput and latency, depending on the entry-sizes of the underlying data. It also makes our results comparable to prior art, such as the acceleration of key-value stores in smart storage devices [19,20], which employed the same dataset. As discussed in Section 4.2, our results are applicable to more traditional database benchmarks.

6.2 Fixed-Point Encoding

In a first step, we evaluate the arithmetic error introduced by the encoding. Since it is based on fixed-point numbers, the maximum error can be derived depending on the encoding and the SPN. Additionally, we use the enumerated and generated queries to gain an empirical perspective shown in Fig. 4. The plot shows the maximum theoretical error in light color and the empirical maximum error in darker color for all datasets and four configurations of the fixed-point encoding. We observe that theoretical and empirical maximum error always differ by at least a factor of 5. The worst case occurs if each column is queried using a range query. Additionally, it is clear that increasing the number of bits in the encoding will reduce the maximum error, since numbers can be represented more accurately.

Lastly, we observe an increased error towards the more complex versions of the NIPS dataset like NIPS50 and NIPS60. This is the case, because the corresponding SPNs are more complex with more histograms and more operations. While more histograms also add more potential for conversion errors during

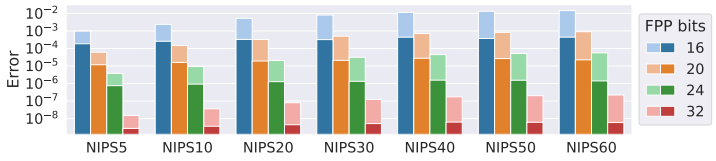


Fig. 4: Arithmetic errors of the encoding depending on the combination of SPN and encoding. The darker bar in front is the maximum measured error, while the lighter bar behind is the theoretical maximum error.

accelerator generation, additional operations introduce more potential for accumulating or multiplying conversion errors. Most importantly, the empirical error of the arithmetic is relatively small compared to the error introduced by CE (cf. Section 4.3). Thus, fixed-point encodings with 16 or 20 bits will most likely be enough for most CE use-cases.

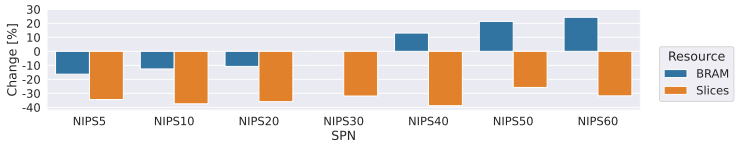


Fig. 5: Changes in resource utilization of a 24 bit fixed-point encoding compared against the closest matching CFP encoding from [15]. DSP utilization is identical for both encodings.

Resource Utilization To gauge the resource efficiency of our design, we used the 24 bit fixed-point accelerators and compared them against the Custom Floating Point (CFP) variants from [15]. We synthesize for the VC709 to achieve comparable results. The changes in resource utilization from [15] to our work are shown in Fig. 5. The diagram shows that there are no changes in DSP utilization. This is the case, since the multiplications in both variants are similar in size and can both be done using 2 DSP slices per multiplication. The BRAM utilization for smaller SPNs is slightly reduced, but will increase for bigger SPNs. This change is the result of three factors: 1) In our work, we need two BRAM lookups to enable the more complex operations in the FxHistogram, which doubles the required memory. 2) The encoding used in our work is more compact, as no exponent is stored. This reduces the required BRAM by about 30%. 3) The use of accumulated histograms and the Index Determination stage allows us to store the buckets without replication. Since 1) and 2) are more or less constant, the changes show mostly 3), which is SPN-dependant. Lastly, we see a reduction in Slice utilization. Overall, our accelerators are more resource efficient. While BRAM utilization is increased, this is not as problematic, as overall BRAM

utilization is below 5% for all SPNs. The reduced resource footprint is paramount for enabling the apporach on the COSMOS+ due to reduced size compared to data-center FPGAs used in prior work.

6.3 Performance

For brevity, we focus our performance evaluation on accelerators using a 24 bit fixed-point encoding, as the encoding has no relevant impact on performance.

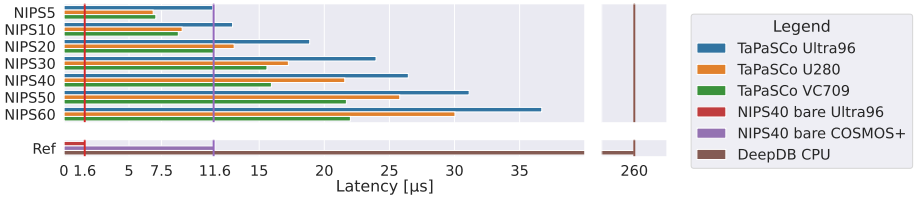


Fig. 6: End-to-End inference latencies (less is better) of TaPaSCo-based architectures in comparison against baremetal NIPS40-accelerators on Ultra96 and COSMOS+ and a RSPN CPU implementation from prior work [8].

Latency First, we want to look at end-to-end latency. For this, we rely on latency measurements that are comparable to real-world implementations, including necessary data movement. We achieve this by using the TaPaSCo-based system integrations (cf. test setups (c)-(e)). The corresponding measurements are shown in Fig. 6. The figure shows that bigger SPNs have higher latencies, which can be attributed to increased data-transfers, as the actual computation takes less than 1 µs. Interestingly, this impact is similar for PCIe-based and SoC-based platforms. While there is no overhead for data-transfers via PCIe in SoC-based systems, the latter are much slower due to the slow ARM cores of the Ultra96. While the latencies seem high compared to the baremetal implementations from Table 3, we still outperform prior CPU-only work [8] by up to 40x. Even for the biggest SPN (NIPS60), our achieved speed-up is still more than 10x. Note that the result reported in prior work is not based on the NIPS dataset. To keep the comparison fair, we thus always compare against their *overall best* reported result of 260 µs. Reproducing the numbers of the original work for the well-known Job-Light benchmark even yields a latency of 910 µs on an Intel Xeon Platinum 8268, showing that the FPGA-based approach can significantly improve the latency.

Throughput Prior CPU-based work does not report throughput, so we compare against existing SPN implementations. Although [15] was later improved via replication, HBM, and 100G networking [6, 21], these effects are hard to isolate. We therefore restrict the comparison to a single accelerator on PCIe platforms, excluding the Ultra96 due to limited memory bandwidth.

Results (Fig. 7) show that, despite 4x larger inputs per inference (1→4 bytes) due to range-supporting histograms, throughput does not drop proportionally.

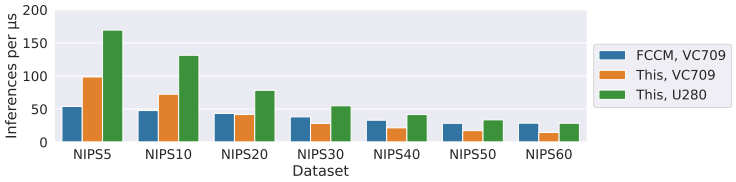


Fig. 7: Throughput of the throughput-optimized accelerator variants on VC709 and Alveo U280 in comparison to single-core prior work published at FCCM [15].

Instead, improved TaPaSCo and SoC design—using optimized vendor IP over custom AXI-Stream interconnects—enable higher throughput for smaller SPNs. On a PCIe Gen3 x16 U280, this yields up to 3x speedup (NIPS5). While gains decrease for larger models, performance remains competitive up to NIPS60.

7 Conclusion

In this work, we extend the use of SPNs for CE [8] and evaluate FPGA-based acceleration. Our framework generates accelerator variants for diverse architectures, including smart storage devices and FPGA cards in database systems, achieving up to 40x latency speedup over prior CPU-based work. On COSMOS+, CE is accelerated by 100x, with newer hardware enabling up to 400x for latency-optimized designs. Compared to prior throughput-oriented approaches, we achieve competitive or better performance despite supporting more complex queries: similar worst-case performance and up to 2x higher throughput, reaching 165 million inferences per second.

Overall, our work shows the overall applicability of the approach in very different system architectures and shows the potential not only in theory, but also in practice on a multitude of different target platforms, including mainstream datacenter cards like the Alveo U280, as well as the more niche architectures such as smart storage, here in the form of the COSMOS+ Smart SSD.

Acknowledgments. AI tools were used for language refinement. This work was partially funded by the Hessian Center for Artificial Intelligence, Germany.

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

References

1. Acharya, A., Uysal, M., Saltz, J.: Active disks: Programming model, algorithms and evaluation. In: Proc. ASPLOS 1998 (1998)
2. Boral, H., DeWitt, D.J.: Parallel architectures for database systems. In: Database Machines, chap. Database Machines: An Idea Whose Time Has Passed? A Critique of the Future of Database Machines, pp. 11–28. Springer (1989)

3. Do, J., Patel, J., DeWitt, D., al, e.: Query processing on smart ssds: Opportunities and challenges. In: Proc. SIGMOD 2013 (2013)
4. Galindez Olascoaga, L.I., Meert, W., Shah, N., Verhelst, M., Van den Broeck, G.: Towards hardware-aware tractable learning of probabilistic models. In: Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., Garnett, R. (eds.) Advances in Neural Information Processing Systems. vol. 32. Curran Associates, Inc. (2019)
5. Harmouch, H., Naumann, F.: Cardinality estimation: An experimental survey. Proc. VLDB Endow. **11**(4), 499–512 (dec 2017). https://doi.org/10.1145/3186728.3164145, https://doi.org/10.1145/3186728.3164145
6. Hartmann, M., Weber, L., Wirth, J., Sommer, L., Koch, A.: Optimizing a hardware network stack to realize an in-network ml inference application. In: 2021 IEEE/ACM International Workshop on Heterogeneous High-performance Reconfigurable Computing (H2RC) (2021)
7. Heinz, C., Hofmann, J., Korinth, J., Sommer, L., Weber, L., Koch, A.: The tapasco open-source toolflow. Journal of Signal Processing Systems (May 2021). https://doi.org/10.1007/s11265-021-01640-8
8. Hilprecht, B., Schmidt, A., Kulessa, M., Molina, A., Kersting, K., Binnig, C.: Deepdb: Learn from data, not from queries! Proceedings of the VLDB Endowment, Vol. 13, No. 7 **13**(7), 992–1005 (mar 2020). https://doi.org/10.14778/3384345.3384349
9. Korinth, J., Hofmann, J., Heinz, C., Koch, A.: The TaPaSCo Open-Source Toolflow for the Automated Composition of Task-Based Parallel Reconfigurable Computing Systems. In: Applied Reconfig. Comp. (2019)
10. Molina, A., Vergari, A., Stelzner, K., Peharz, R., Subramani, P., Mauro, N.D., Poupart, P., Kersting, K.: Spflow: An easy and extensible library for deep probabilistic learning using sum-product networks (2019)
11. Peharz, R., Vergari, A., Stelzner, K., Molina, A., Shao, X., Trapp, M., Kersting, K., Ghahramani, Z.: Random Sum-Product Networks: A Simple but Effective Approach to Probabilistic Deep Learning. In: Proceedings of the Thirty-Fifth Conference on Uncertainty in Artificial Intelligence (UAI) (2019)
12. Poon, H., Domingos, P.: Sum-Product Networks: a New Deep Architecture. Proc. of UAI (2011)
13. Shah, N., Galindez Olascoaga, L.I., Meert, W., Verhelst, M.: Acceleration of probabilistic reasoning through custom processor architecture. In: 2020 Design, Automation & Test in Europe Conference & Exhibition (DATE). pp. 322–325 (2020). https://doi.org/10.23919/DATE48585.2020.9116326
14. Sommer, L., Oppermann, J., Molina, A., Binnig, C., Kersting, K., Koch, A.: Automatic Mapping of the Sum-Product Network Inference Problem to FPGA-Based Accelerators. In: 36th Intl. Conf. on Computer Design (ICCD) (Oct 2018)
15. Sommer, L., Weber, L., Kumm, M., Koch, A.: Comparison of arithmetic number formats for inference in sum-product networks on fpgas. In: Intl. Symposium on Field-Programmable Custom Computing Machines (FCCM) (2020)
16. Song, Y.H., Jung, S., Lee, S.W., Kim, J.S.: Cosmos+ openssd: A nvme-based open source ssd platform. Flash Memory Summit (2016)
17. Vinçon, T., Knödler, C., Bernhardt, A., Solis-Vasquez, L., Weber, L., Koch, A., Petrov, I.: Result-set management for ndp operations on smart storage. In: 18th International Workshop on Data Management on New Hardware (DaMoN) (2022). https://doi.org/10.1145/3533737.3535097

18. Weber, L., Sommer, L., Oppermann, J., Molina, A., Kersting, K., Koch, A.: Resource-Efficient Logarithmic Number Scale Arithmetic for SPN Inference on FPGAs. In: Intl. Conference on Field-Programmable Technology (FPT) (2019)
19. Weber, L., Sommer, L., Solis-Vasquez, L., Vinçon, T., Knödler, C., Bernhardt, A., Petrov, I., Koch, A.: A framework for the automatic generation of fpga-based near-data processing accelerators in smart storage systems. In: 2021 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW). pp. 136–143 (2021). https://doi.org/10.1109/IPDPSW52791.2021.00028
20. Weber, L., Vincon, T., Knödler, C., Solis-Vasquez, L., Bernhardt, A., Petrov, I., Koch, A.: On the necessity of explicit cross-layer data formats in near-data processing systems. Distributed and Parallel Databases (2021). https://doi.org/10.1007/s10619-021-07328-z
21. Weber, L., Wirth, J., Sommer, L., Koch, A.: Exploiting high-bandwidth memory for fpga- acceleration of inference on sum-product networks. In: 2022 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW) (2022)