SUSTAIN Deliverable

D2.2: Probabilistic node intelligence: proof of concept on simulation level

Grant Agreement number	101071179
Action Acronym	SUSTAIN
Action Title	Smart Building Sensitive to Daily Sentiment
Type of action:	HORIZON EIC Grants
Version date of the Annex I against which the assessment will be made	28th March 2022
Start date of the project	1st October 2022
Due date of the deliverable	M13
Actual date of submission	31.10.2023
Lead beneficiary for the deliverable	AALTO
Dissemination level of the deliverable	Public

Action coordinator's scientific representative

Prof. Stephan Sigg
AALTO –KORKEAKOULUSÄÄTIÖ,
Aalto University School of Electrical Engineering, Department of Communications and Networking
stephan.sigg@aalto.fi

Funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or [name of the granting authority]. Neither the European Union nor the granting authority can be held responsible for them.

Authors in alphabetical order		
Name	**Beneficiary**	**e-mail**
Martin Andraud	Aalto University	martin.andraud@aalto.fi
Giovanni Iacca	Trento University	giovanni.iacca@unitn.it
Jelin Leslin	Aalto University	jelin.leslin@aalto.fi
Lingyun Yao	Aalto University	lingyun.yao@aalto.fi

Abstract

This deliverable D2.2 presents the progresses made mostly in the work package 2 (WP2) of the SUSTAIN project. It directly related to task T2.2: "System model and design of the probabilistic learning strategy". As stated in the project proposal, the SUSTAIN project envisages two levels of intelligence embedded in the target sensor nodes: a distributed intelligence controlled in the cloud and a node-level intelligence embedded in each node. In this context, WP2 focuses on the node-level intelligence, which intends to embed probabilistic machine-learning models on the node using an efficient hardware accelerator for that purpose. Towards this goal, the main objective of task 2.2 is to design a strategy for embedded probabilistic reasoning, in terms of model choice and possible hardware implementation. It closely related to tasks in WP3 as the node level intelligence is included in the distributed intelligence strategy and implementation. The circuit design and implementation containing this probabilistic strategy will be implemented in the remaining tasks of this WP, T2.3 (and associated deliverable D2.3) and T2.4 (and associated deliverable D2.4). In this deliverable, we start by depicting the envisaged node intelligence architecture in the SUSTAIN node. Then, we motivate the use of probabilistic models and in particular probabilistic circuits (PCs) for edge AI computation (and in SUSTAIN). We will detail our first contribution for more energy-efficient PC inference on hardware using approximate computing. We then report our progresses in parallel projects developing A-core, our own RISC-V processor that can also be used in a future implementation of the SUSTAIN node. We summarize our results and give an outlook towards the next task in the project. This deliverable is the first of a series of deliverables related to the node intelligence in SUSTAIN. We envisage it as a first step, and a baseline for the future work carried out in the SUSTAIN project.

Contents

1 Summary on project's objectives and tasks

1.1 Intelligence in sustAIn: summary of the envisaged solutions

The sustAIn project envisages two levels of intelligence for the system, as pictured in figure 1:

1. A Meta-level intelligence (mostly developed in WP3), which will control all nodes in the system and choose the main actions/behaviours to be taken depending on the context. This intelligence will run in a centralized manner, for instance on a server.
2. A node-level intelligence (mostly developed in WP2), which will be embedded in each node. The role of this intelligence is to gather local information about the current context, to transmit this information to the meta-level intelligence. At meta-level, it will be possible to configure the nodes so that the system adopts the intended behaviour.

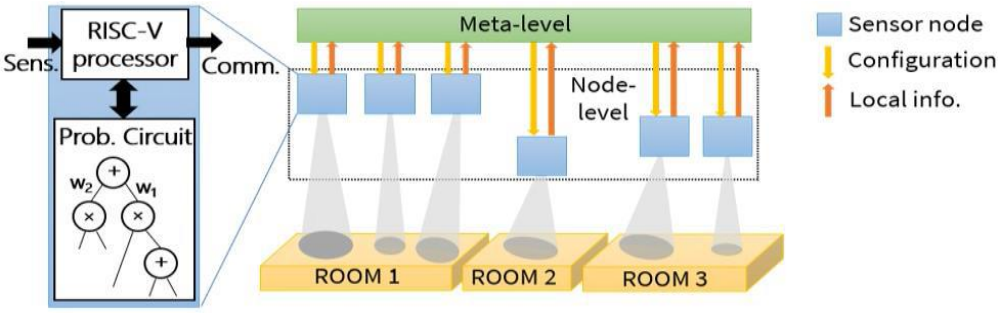

Figure 1 - general view of the envisaged intelligence in sustAIn

This WP focuses on developing the node intelligence. In that regard, the node intelligence block will be composed of two main parts:

1. A dedicated machine-learning accelerator to embed the intelligence model. The chosen machine-learning is a probabilistic machine-learning model, specifically a probabilistic circuit.
2. A RISC-V processor developed based on the work recently carried out in Aalto University. This processor can also be used for other tasks, such as communication or wireless sensing (WP4). It will be used to control the node and the accelerator.

1.2 Contents of this deliverable

In this deliverable we will go over the main developments in the two blocks composing the node-level intelligence: the use of probabilistic circuits as a machine-learning model, and the RISC-V processor that can be used as part of the node. First, section 2 motivates and explains the use of Probabilistic Circuits (PCs) as a baseline model in the project. The main goal is to obtain accurate, explainable, and hardware-efficient intelligence for the node. Second, section 3 details our first contribution towards building more efficient hardware for PC, using approximate computing. That way, we can reduce the energy inference of PCs by 100x-200x, depending on the tolerated error. Third, section 4 reports on advances carried on in parallel projects, with the development of Aalto's own RISC-V processor, A-core.

2 The model: probabilistic machine-learning and probabilistic circuits

2.1 Motivation

Devices populating the Internet-of-things (IoT), including the devices developed in the context of the sustAIn project, have a fundamental trade-off between energy and functionality. On the one hand, we wish to embed always more advanced functionalities, in particular embedded AI. On the other, power is becoming a clear bottleneck in these devices, and technology will not help as much reducing the power consumption of devices I the near future [Alioto17].

Deep Neural Networks (DNNs) architectures are de facto a standard model for embedded AI, yet they can be overconfident in their predictions [Hein19], limited to a single task (requiring retraining for each new task) [Ruder17], and even have been characterized as never truly reliable [Marcus20]. As an illustration, consider a multi-object detection task on the edge, as depicted in figure 2. A direct application could be multi-digit recognition [Stelzner19].

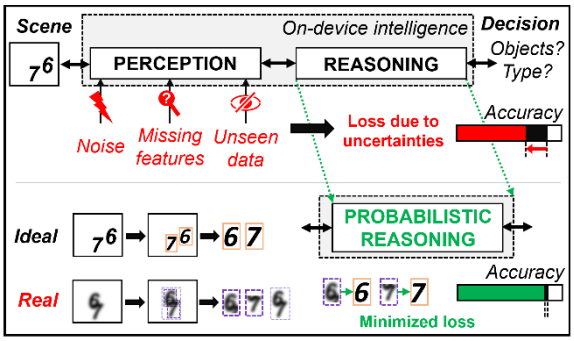

Figure 2 - Example of a probabilistic learning system

Using DNNs, the accuracy can be high for training samples, yet in real scenarios, the system's accuracy may drop as it is affected by various uncertainties. They can be e.g., noise in the data (a blurry image), missing features (an unusable sensor), or previously unseen data. In turns, a DNN is trained for one specific conditional distribution, i.e., giving an output according to a particular set of inputs. In this case, it could be more desirable for the model to encode a joint distribution (e.g., jointly model the labels and inputs). In the example depicted in figure 2, this means querying for the most probable digit according to the observed object. ***Probabilistic general-purpose reasoning*** allows for such queries, handling and quantifying uncertainty in a principled manner [Koller09]. With this model type, one can, for instance, compute predictions (conditional probability of the labels given the inputs) in case of missing values (e.g., in case of corrupted pixels in an image) by exactly marginalizing out the missing values. Such a computation is not possible in NNs as only the distribution conditioned on all inputs (e.g., pixels) is computable, and one would need to use additional imputation methods.

As a result, there is a need to explore alternative computationally efficient models, allow faithful and probabilistic general-purpose reasoning, and integrate well with existing deep NN frameworks. However, inferring answers in complex probabilistic models is typically intractable (computationally infeasible). These models are represented as Directed Acyclic Graphs (DAGs) with a relatively irregular structure; thus, they can't be easily translated into repetitive computation steps [Shah19]. To overcome this limitation, recent work on tractable probabilistic models, specifically on probabilistic circuits (PCs) [Choi22], poses a promising avenue as PCs 1.) exhibit high expressive efficiency (representational power), 2.) are tractable (computationally efficient) for many queries classes by design, and 3.) integrate well with state-of-the-art deep learning techniques.

Probabilistic models In SustAIn? SustAIn envisages the embedded node intelligence as transparent, accurate and energy efficient. PCs satisfy these three points: (1) probabilistic reasoning enables to have a level of confidence with the model, to take more clear and explainable decisions; (2) PCs have shown promising performances on various edge AI tasks (more details below); and (3) PCs have a representation already close to hardware levels, easy to translate into computation steps. yet, there are still challenges hampering the use of PCs at a large scale, which we wish to tackle in this work.

2.2 Introduction to probabilistic circuits

Notation: we use upper case letters to denote random variables (RVs) (e.g., X) and lower-case letters for realizations of RVs (e.g., x). Further, we use **bold font** for vectors (e.g., X, **x**) and matrices (e.g., **M**).

Probabilistic circuits (PCs) [Choi22] are a unifying framework of existing probabilistic models (e.g., [Darwiche03, Poon11, Rahman14, Kisa14]). They provide a concise language to represent and reason about tractable (i.e., exact, and efficient) probabilistic inference. The reader is referred to [Choi20] for an extensive review of PCs. In this introduction, we will go over the main characteristics only.

The model. Given a set of *d* RVs **X**, a probabilistic circuit is a function, typically a density or mass function, represented by a parameterized computational graph consisting of sum (**+**), product (**x**), and **leaf units**. An example of PC with three binary RV (T, V and P) is depicted in figure 3. The top node value represents the joint probability **P(T,V,P).** Each computational unit (sum, product, leaf) is defined over a set of variables, called its *scope* [Trapp19], and every non-leaf unit computes an algebraic operation over sub-circuits. The scope of each non-leaf unit is given by the union of the scopes of its sub-circuits (inputs). Essentially, sum units compute a weighted sum of sub-circuits with weight parameters being probabilities (**Θ**), product units multiply sub-circuits, and leaf units evaluate a tractably integrable function over its inputs. Leaf units can be e.g., univariate probability distributions in the case of continuous RVs, or binary indicators (**λ**), indicating if a value of a variable is observed or not in the case of binary/discrete RVs. Typically, we assume that product units compute binary products and sum units are normalized (the sum of weights arriving to a sum unit is 1).

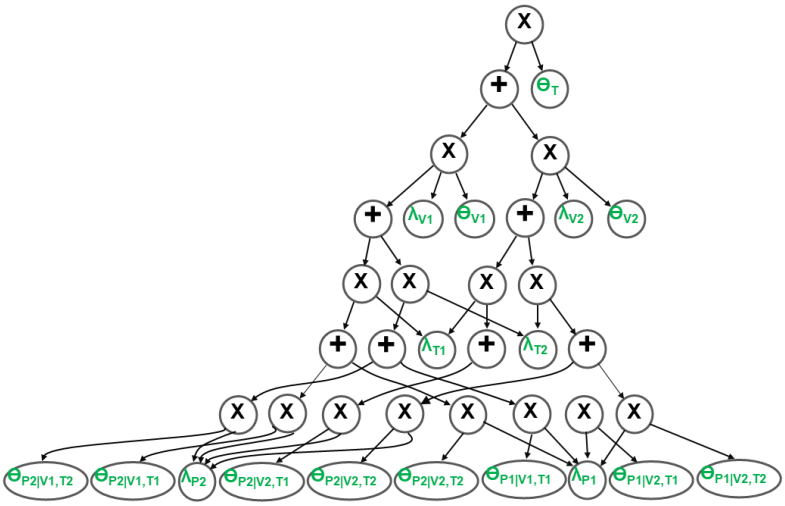

Figure 3 - Example of probabilistic circuit

PC learning. Learning a PC generally involves two steps: (1) learning the structure and (2) learning the parameters. The structure can be learned from data [Gens13, Trapp19] or chosen to be random but sufficiently large [Peharz19]. Parameters are typically learned by employing expectation maximisation (EM) [Peharz15], maximizing the likelihood under missing data. In comparison, the structure of deep neural

networks is typically fixed by design (number of layers, neurons per layer), thus training mostly involves parameter learning. Once learned, many queries can be answered without re-training.

Properties of PCs. As learning PCs involves first learning the structure of the model, constraints can be added to obtain tractable inference, i.e., obtain a manageable model in terms of computations even when the number of variable increases. This is particularly interesting for resource-efficient hardware acceleration. Hence, tractability is obtained by **constraining the structure**. Various constraints are available for the model, and the reader is referred to [Choi20] for more details on the topic (as learning PCs out of the scope of this introduction).

PC inference. Once the structure and parameters are learned, most inference routines are computed in one or several paths through the PC. The simplest type of inference is marginal query (MAR). MAR gives the probability of a certain event happening. It is computed with a bottom-up path through the PC, where leaf nodes are set to reflect the probability to be computed (e.g., leaf indicators set to '0' or '1' according to the target probability value) and the computations are carried out until the top of the PC. For instance, taking the PC represented in figure 3, computing *Prob(P=P1,V=V2,T=T1)* involves setting the indicators to the correct value for each variable (i.e., $\lambda_{P1}=\lambda_{V2}=\lambda_{T1}=1$ and $\lambda_{P2}=\lambda_{V1}=\lambda_{T2}=0$) and compute the top node value. Another type of inference that can be commonly used by probabilistic models is Maximum A Posteriori (MAP), also referred to as Most Probable Explanation. The aim of this query is to find out the most likely instantiation of a variable, or a group of variables, that are unknown. Following the example figure 3, MAP could give the most likely value of the variable P, in a certain configuration of variables V and T. MAP is computed by first carrying out a bottom-up path through the PC, replacing every sum node by a MAX node, to keep only the most probable case at every step. Here, the indicators of the unknown variable(s) are set all to '1' as do not assume any configuration. The bottom-up path can then be followed by a top-down path through the PC to recover the value of the unknown variable(s) corresponding to this most probable path. Using these queries, the confidence level of the model is also assessed, providing **explainable results**. For instance, a MAP query computes at the same time the most probable variable for certain evidence, and the probability associated with this event.

Application of PCs. PCs have proven to be competitive in several applications related to embedded reasoning, such as speech recognition [Nicolson20] or activity/action recognition from images [Amer20, Wang18]. They have for instance seen to be more **robust** and **compact** than convolutional NNs for speaker identification tasks, and competitive for more complex speech recognition and verification tasks [Nicolson20]. They have also been used for semantic mapping, e.g., robots exploring environments in large-scale areas [Zhang19]. Readers are referred to [Paris20] for a larger review of applications. PCs can as well **replace** other types of models in specific applications. This has been illustrated in [Stelzner19], where a variational autoencoder has been replaced by a PC, showing faster learning, and reducing the inference cost significantly (the number of parameters can be for instance divided by 2). This highlights the compactness of PCs in this context.

2.3 Probabilistic Circuits on Hardware

The deployment of PCs on hardware requires additional care compared to software inference: (1) defining the optimal format and computational resolution; and (2) specifying the hardware generation process. We will briefly review common approaches for these two points, with a focus on methods common for PCs.

(1) Format and resolution. As PCs use arithmetic with probabilities, each computed is in the [0;1] range. These values are successively added and multiplied, leading to small probabilities at the top layers. Thus,

computing PCs require **significantly higher resolution** than deep NNs (30-40 floating point (float) bits for medium-sized PCs, 5-8 integer bits for deep NNs). In software, PCs are trained using logarithmic computation to avoid underflow, as even a double representation (64-bit float) in a generic computer can lead to errors. Yet, on hardware, PCs are computed in the linear domain, using formats allowing for encoding large dynamic ranges, such as floating point or posit [Sommer20, Shah21]. The resolution (i.e., the number of exponent and mantissa bits) can be optimized depending on the PC structure with customized arithmetic blocks.

(2) Hardware generation. After fixing the arithmetic format and the resolution, a hardware representation needs to be generated. For that, a classical approach translates each computational unit (sum, product) of the PC into a separate hardware entity and connects multiple entities accordingly [Sommer18, Sommer20, Shah19]. This approach works well for FPGAs, because every PC will generate a different hardware configuration. A more advanced approach suitable for Application-Specific Integrated Circuits (ASICs) maps any PC to a generic processor [Shah22, Choi22h]. This processor contains several parallel paths (named processing elements), each computing part of the PC graph. This requires a dedicated graph compiler [Shah21].

We will give more details about existing PC accelerators and dive into their implementations in the next deliverable (D2.3), where the main focus will be to propose a framework to compile PCs on hardware and expand it for custom accelerators, with the help of our research in parallel projects.

3 First contribution: approximate computing for efficient PC inference

In this first contribution, we tackle the challenge related to the **computation resolution** (point (1) in section 2.3). Our objective is to evaluate how to increase the efficiency of probabilistic inference on hardware, to facilitate the implementation of PCs, using approximate computing blocks that are efficient on hardware.

3.1 Approximate computing for PCs: motivation

Approximate computing frameworks have flourished recently in deep learning applications, varying from quantization methods to approximate computing blocks trading off energy versus computation accuracy. As deep learning frameworks are relatively tolerant to errors, a large variety of techniques have been presented.

However, PCs have several specificities in that regard. As explained in section 2, computing multiple sums and products of probabilities requires a much higher resolution to avoid underflow. To develop a dedicated and effective approximate computing framework, we start with the following two observations:

1. In software, PC inference methods typically use a logarithm representation for the computation, alternating between logarithm multiplication and linear additions. In the logarithm domain, multiplications become additions, and as logarithm additions are complex, thus a technique called the "*log-sum-exp*" trick is used. Essentially, it transfers the operand back to the linear domain, add them, and convert them back to logarithm for further computation.
2. In hardware, most PC accelerators prefer using linear operators with higher resolution, to limit the risk of overflow, computing the complete PC in the linear domain [Sommer20, Shah22].

This discrepancy can be explained by hardware limitations. First, alternating between logarithm and linear domains would require specific hardware blocks for encoding/decoding, which would induce a higher cost and limit speed. Second, computing a PC fully in the logarithm domain is inefficient due to the prohibitive cost of logarithmic adders [Sommer20]. Instead, a full linear computation, using floating-point or Posit formats, is preferred. In this case, the hardware cost steadily increases with the model complexity to handle the increasing dynamic range of the PC. This cost is heavily dominated by multiplications, i.e., a floating-point multiplier consumes 6× more energy than an adder in a 45 nm process technology [Olascoaga19].

3.2 Approximation using Addition as Int (AAI)

A path towards more efficient PC inference on hardware is to perform inference as in software, i.e., alternating between logarithm and linear computations. To achieve this, we propose to build an approximate computing framework dedicated to PCs, leveraging Addition As Int (AAI) [Mogami20]. The complete approximation process is illustrated in figure 4.

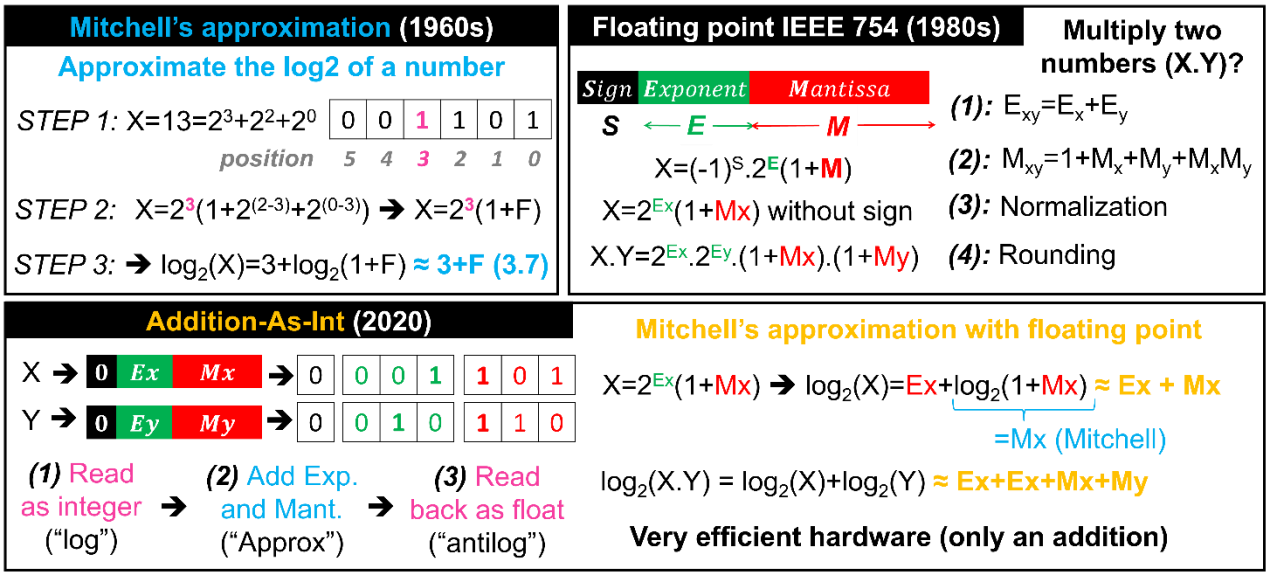

Figure 4 - Overview of the approximate computing methodology with Addition As Int

AAI approximates the logarithm of a floating-point number, extending the well-known Mitchell's approximation [Mitchell62], initially intended to provide a log approximation of integer numbers encoded in binary. To explicit it, consider an N-bit binary string, as represented in the top left part of figure 4. In binary, each symbol represents a power of two, i.e., the number X=13 is encoded as X='001101', corresponding to the value $X=2^3+2^2+2^0$. Mitchell's approximation starts by taking the leading '1' of the binary string and place a virtual decimal 'dot' at the position *k* of this leading one (position **3** in the example figure 4). Then, it is possible to factorize the represented number according to the power of two associated with the position of this leading '1'. This transforms the number into a floating point into: $X=2^k(1+F)$, *F* being the fractional part of this new number. Taking the logarithm of this new representation, we obtain $log_2(X)=k+log_2(1+F)$; Michell's approximates $log_2(1+F)$ by *F*, which means that to obtain the logarithm approximation, we only need to identify the leading one and extract the corresponding fractional part.

AAI extends this approximation technique for floating point multiplication. Assuming a floating-point representation with one sign bit, several exponent bits E and several mantissa bits M, the general multiplication process of two floating points X and Y is illustrated in figure 4 (right top). Specifically, it involves the multiplication of the two mantissas MxMy, which requires a high resolution. To avoid it, AAI proposes to apply Mitchell's idea to floating point representation, which is similar than the one of the transformed binary strings into fixed point. This gives:

$$X=2^{Ex}(1+Mx) \rightarrow log_2(X)=Ex+log_2(1+Mx) \rightarrow Ex+Mx \text{ (with Mitchell's approximation).}$$

Hence, the multiplication of two floating point numbers, becoming an addition in the logarithm domain, can be approximated by simply adding the two exponents and the two mantissas, which only requires an **extremely simple hardware.**

Generally, all multipliers can be replaced with AAI to save energy. However, that may have a dramatic impact on the accuracy of the model, because certain nodes require a very high resolution to be computed (as our experiments in section 3.3 and figure 5 show). Instead, in this work, we target the development of a **dedicated methodology** to use AAI approximate multipliers in an optimal way for efficient PC inference. This

work has been submitted in a workshop on tractable probabilistic modelling [Yao23] and recently at another AI-oriented conference. Here, we provide a quick summary of the methodology and results. Theoretical proofs and more details will be available upon final publication. The complete code will also be open-sourced.

3.3 First experiment: replace all multipliers with AAI

In a first experiment, we evaluated if we could reduce the number of bits of PC inference, and if replacing all multipliers by AAI approximate versions would have a large impact on the model's accuracy. We took four benchmarks as an example (NLTCS, Jester, DNA, Book), among the most used benchmarks used in the literature for probabilistic models. We evaluated two types of queries:

- Marginal query (MAR), which calculates the probability of a certain even happening.
- Maximum A Posteriori (MAP) which evaluates the most probable value of a missing variable (or a set of missing variables) under certain evidence.

The results, plotted according to the resulting model's energy, are displayed in figure 5. An energy of 1 corresponds to the energy of a double floating point format (64 bits). Plain lines represent an exact computation and dashed lines represent the approximate multiplication with AAI. We evaluated the results for various number of exponent bits (3 bits in blue, 5 bits in green and 8 bits in pink), each time varying the number of mantissa bits to obtain a full curve.

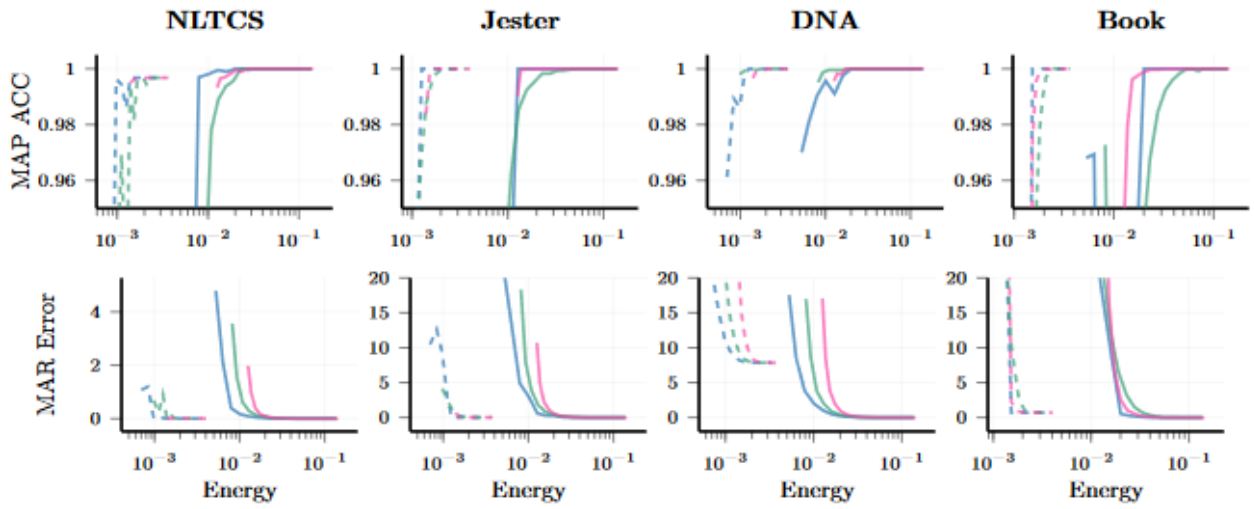

Figure 5 - Energy and accuracy comparison of exact versus AAI based mutipliers for various benchmarks

First, already for an exact floating-point representation, we can reduce the number of bits for accurate inference, gaining a significant energy consumption. This motivated the use of customized formats for efficient inference. Second, we can observe that AAI significantly reduces the inference cost on top of customized floating point, by around one order of magnitude in most cases. For MAR query, AAI may require fewer mantissa bits, as the two mantissas do not need to be multiplied compared to exact floating point multiplications. In the contrary, AAI tends to use more exponent bits to maintain a good accuracy. For MAP query, our experiments show that the inference can be done at very low cost for most benchmarks, without necessitating any error compensation mechanisms. This is because MAP inference cares about the rank between probabilities, as it gives the most probable values of missing variable(s). Hence, as long as the rank between probabilities is preserved, the absolute probability value can be relatively inaccurate. More

generally, compared to an initial 64-bit floating point computation, AAI can attain savings of almost 700x, by using a customized number of bits for the computation and simplify the hardware. For MAR query, the error tends to increase rapidly as the number of bits is reduced. The optimal energy can be traded-off with the tolerated error in this case.

3.4 Second experiment: safely replace multipliers.

In some cases, it is not possible to tolerate error in the model. Instead, we would like to be able to safely replace part of the multipliers in the PC while having no or a very limited impact on accuracy. That is why in the second experiment, we propose and evaluate an error compensation technique and dedicated replacement methodology, to safely replace multipliers by AAI also in the case of MAR query.

Error correction. The error introduced by AAI can result in substantial approximation errors in deep models as the error accumulates with an increasing number of multiplications. To reduce the error caused by AAI, [Saadat18] proposed an error correction by computing the *expected* error, assuming a uniform probability for all possible floating-point numbers. However, in PCs this assumption will typically not hold true. Therefore, as the PC represents a given probability distribution, we propose to correct for the expected error with respect to the ***probability distribution represented by the circuit***, and not only a uniform distribution. Essentially, we use a Monte-Carlo sampling method to compute the expected value at each node, comparing it to the ideal value with exact computation. After that, it is possible to implement a greedy approach that will gradually replace the multipliers introducing the lowest error on the final probability, i.e**., having the lowest influence on the probability distribution** learned by the model. In figure 6, we plotted the error observed on a marginal query (MAR), comparing 5 different random runs (i.e., where we randomly replace multipliers in the PC, in pink) with the proposed methodologies for two types of PCs (deterministic, in blue and non-deterministic in green). We plot the error against the energy of the PC inference, normalized with the energy of a standard implementation (double float, 64b). The error reflects how different the approximates PC is from the original using 64-bit computation.

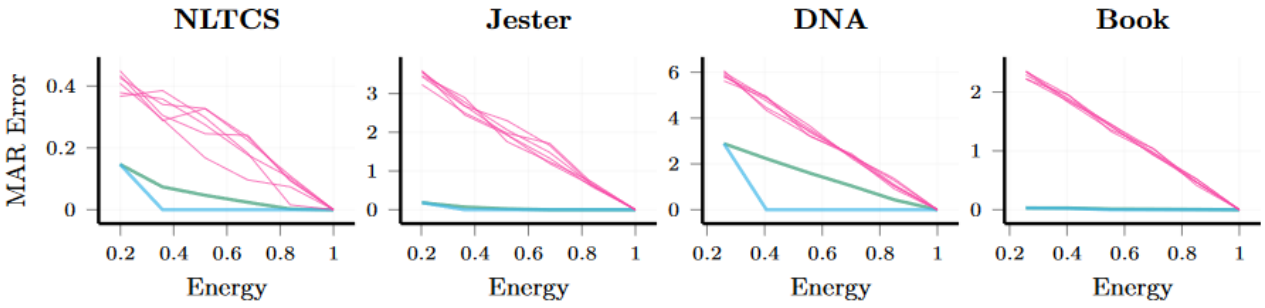

Figure 6 - Comparison of energy gains and error of random replacement of multipliers by AAI (pink, 5 runs), with the proposed methodology for two different types of PCs (blue and green)

As it can be seen, although a random replacement allows to save energy, the error quickly increases, in particular for complex datasets. In contrary, the proposed methodology can safely replace multipliers with limited impact on the accuracy, obtaining PCs with 40-60% less energy for most benchmarks.

4 RISC-V processor: A-core developments

In parallel of the development around approximate computing for PCs, Aalto University has successfully tested their first open-source RISC-V processor named A-core. The silicon chip has been taped-out in a 22nm technology in December 2022 and fully tested during the summer of 2023 (publications are ongoing). This processor can serve as a baseline for future integration withing the sustAIn project.

The Acore chip contains a RISC-V processor, fully developed at Aalto University. It is coupled with 2 accelerators, one dedicated to cryptographic tasks, and the other for AI tasks. The die photograph is illustrated in figure 7. Here, we will focus on the characteristics of the processor, yet it can be noted that this chip developed the competence of the laboratory to also include accelerators together with the processor core, connected through an AXI-4 bus. When designing the dedicated PC accelerator envisaged in sustAIn, this knowledge can be readily used.

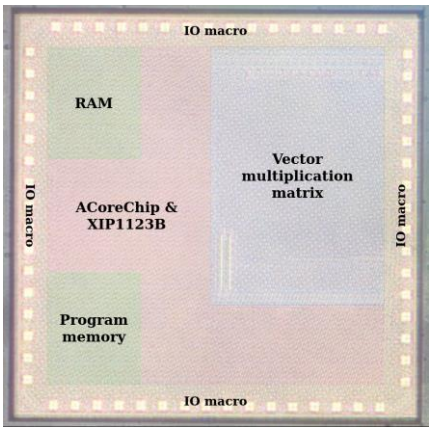

Figure 7 - Die photograph of the A-core chip

The processor is a 32-bit 7-stage pipeline architecture, equipped with 64 kB of program memory and 64bK of RAM. After first measurements, the processor has been successfully run with a 200 MHz clock. Performance benchmarks are giving similar performances than an ARM cortex M-0 processor.

A second version of the processor will be taped-out in 2024.

5 Outlook and conclusions

This deliverable detailed the first steps towards the complete node intelligence envisaged in sustAIn. It provides a motivation on the use of probabilistic circuits (PCs) for explainable and efficient embedded AI. It further details our first contribution around more energy-efficient inference for PCs using a dedicated approximate computing algorithm and custom hardware blocks. We also reported on parallel implementation of a RISC-V processor that could be used in the project at a later stage.

In the following task (T2.3), we will elaborate on our current developments for an accelerator targeting PC computing, yet not forgetting the possible acceleration of other models such as deep NNs. We are currently investigating an accelerator architecture that could execute several models on the same platform, to give more possibilities for the final sustAIn node.

6 References

References

[Alioto17] Massimo Alioto, “Energy-quality scalable adaptive VLSI circuits and systems beyond approximate computing”, In Proceedings of the Conference on Design, Automation & Test in Europe (DATE '17).

[Amer16] M. R. Amer and S. Todorovic, “Sum product networks for activity recognition,” IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 38, no. 4, pp. 800–813, 2016.

[Choi22] Choi, Y., “Probabilistic Reasoning for Fair and Robust Decision Making”, PhD thesis, 2022.

[Choi20] Choi, Y., Vergari, A., and Van den Broeck, G. “Probabilistic circuits: A unifying framework for tractable probabilistic models”, 2020.

[Choi22h] Choi, Y.-k., Santillana, C., Shen, Y., Darwiche, A., and Cong, “FPGA acceleration of probabilistic sentential decision diagrams with high-level synthesis”, ACM Trans. Reconfigurable Technol. Syst., 2022.

[Darwiche03] Darwiche, A. “A differential approach to inference in bayesian networks” J. ACM, 50(3):280–305, 2003.

[Gens13] Gens, R. and Pedro, D., “Learning the structure of sum-product networks. In International conference on machine learning”, pages 873–880, PMLR, 2013.

[Ghahramani15] Ghahramani, Z., “Probabilistic machine learning and artificial intelligence”, Nature, 521(7553):452–459, 2015.

[Hein19] Matthias Hein, Maksym Andriushchenko, and Julian Bitterwolf, “Why relu networks yield high-confidence predictions far away from the training data and how to mitigate the problem”, In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 41–50, 2019.

[Kisa14] Kisa, D., den Broeck, G. V., Choi, A., and Darwiche, A., “Probabilistic sentential decision diagrams”, In proc. 14th International Conference on Principles of Knowledge Representation and Reasoning, AAAI Press, 2014

[Koller09] D. Koller and N. Friedman, “Probabilistic graphical models: principles and techniques”, MIT press, 2009.

[Marcus20] Gary Marcus, “The next decade in AI: Four steps towards robust artificial intelligence”, arXiv preprint arXiv:2002.06177, 2020.

[Mitchell62] Mitchell, J. N., “Computer multiplication and division using binary logarithms”, IRE Transactions on Electronic Computers, (4):512–517, 1962

[Mogami20] Mogami, T. “Deep neural network training without multiplications”, arXiv preprint arXiv:2012.03458, 2020.

[Nicolson20] A. Nicolson and K. K. Paliwal, “Sum-product networks for robust automatic speaker identification,” in Proc. Interspeech 2020, pp. 1516–1520, 2020.

[Olascoaga19] Olascoaga, G. et al. (2019). Towards hardware-aware tractable learning of probabilistic models. Advances in Neural Information Processing Systems 32 (NeurIPS).

[Paris20] I. París, R. Sánchez-Cauce, and F. J. Díez, “Sum-product networks: A survey,” arXiv preprint arXiv:2004.01167, 2020.

[Peharz19] Peharz, R., Vergari, A., Stelzner, K., Molina, A., Trapp, M., Shao, X., Kersting, K., and Ghahramani, Z., “Random sum-product networks: A simple and effective approach to probabilistic deep learning”, In 35th Conference on Uncertainty in Artificial Intelligence (UAI), volume 115 of Proceedings of Machine Learning Research, pages 334–344, 2019.

[Poon11] Poon, H. and Domingos, P. M., “Sum-product networks: A new deep architecture”, In 27th Conference on Uncertainty in Artificial Intelligence (UAI), pages 337–346, 2011.

[Rahman14] Rahman, T., Kothalkar, P. V., and Gogate, V. (2014). Cutset networks: A simple, tractable, and scalable approach for improving the accuracy of chow-liu trees. In European Conference in Machine Learning and Knowledge Discovery in Databases ECML, volume 8725, pages 630–645, 2014

[Rooshenas14] Rooshenas, A. and Lowd, D., “Learning sum-product networks with direct and indirect variable interactions”, In International Conference on Machine Learning, pages 710–718. PMLR.

[Ruder17] Sebastian Ruder, “An overview of multi-task learning in deep neural networks”, arXiv preprint arXiv:1706.05098, 2017.

[Saadat18] Saadat, H., Bokhari, H., and Parameswaran, S., “Minimally biased multipliers for approximate integer and floating-point multiplication” IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 37(11):2623–2635, 2018.

[Seo22] Seo, J.-s., Saikia, J., Meng, J., He, W., Suh, H.-s., Anupreetham, Liao, Y., Hasssan, A., and Yeo, I., “Digital versus analog artificial intelligence accelerators: Advances, trends, and emerging designs”. IEEE Solid-State Circuits Magazine, 14(3):65–79, 2022.

[Shah21] Shah, N., Meert, W., and Verhelst, M., “Graphopt: constrained optimization-based parallelization of irregular graphs”, arxiv preprint, 2021.

[Shah22] Shah, N., Meert, W., and Verhelst, M., “DPU-V2: Energy-efficient execution of irregular directed acyclic graphs”, In 2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO), pages 1288–1307, 2022.

[Shah19] Shah, N., Olascoaga, L. I. G., Meert, W., and Verhelst, M., “Problp: A framework for low-precision probabilistic inference”, In Proceedings of the 56th Annual Design Automation Conference 2019, pages 1–6., 2019.

[Shah21b] Shah, N., Olascoaga, L. I. G., Zhao, S., Meert, W., and Verhelst, M., “9.4 piu: A 248gops/w stream-based processor for irregular probabilistic inference networks using precision-scalable posit arithmetic in 28nm”, In 2021 IEEE International Solid- State Circuits Conference (ISSCC), volume 64, pages 150–152, 2021.

[Sommer18] Sommer, L., Oppermann, J., Molina, A., Binnig, C., Kersting, K., and Koch, A., “Automatic mapping of the sum-product network inference problem to fpga-based accelerators”, In 2018 IEEE 36th International Conference on Computer Design (ICCD), pages 350 – 357, 2018.

[Sommer20] Sommer, L., Weber, L., Kumm, M., and Koch, A., “Comparison of arithmetic number formats for inference in sum-product networks on FPGAs”, in 2020 IEEE 28th Annual international symposium on field-programmable custom computing machines (FCCM), pages 75–83, 2020.

[Stelzner19] K. Stelzner, R. Peharz, and K. Kersting, “Faster attend-infer-repeat with tractable probabilistic models,” in Proceedings of the 36th International Conference on Machine Learning, ser. Proceedings of Machine Learning Research, vol. 97. PMLR, pp. 5966–5975, 2019.

[Trapp19] Trapp, M., Peharz, R., Ge, H., Pernkopf, F., and Ghahramani, Z., “Bayesian learning of sum-product networks” In proc. 32nd Conference on Neural Information Processing Systems (NeurIPS), pages 6344–6355, 2019.

[Vergari21] Vergari, A., Choi, Y., Liu, A., Teso, S., and den Broeck, G. V., “A compositional atlas of tractable circuit operations for probabilistic inference. In Ranzato, M., Beygelzimer, A., Dauphin, Y. N., Liang, P., and Vaughan, J. W., editors, In proc. 34th Conference on Neural Information Processing Systems (NeurIPS), pages 13189–13201, 2021.

[Wang18] J. Wang and G. Wang, “Hierarchical spatial sum-product networks for action recognition in still images,” IEEE Transactions on Circuits and Systems for Video Technology, vol. 28, no. 1, pp. 90–100, 2018.

[Yao23] L. Yao, M. Trapp, K. Periasamy, J. Leslin, G. Singh, M. Andraud, “Logarithm-Approximate Floating-Point Multiplier for Hardware-efficient Inference in Probabilistic Circuits”, 6th workshop on Tractable Probabilistic Modelling, collocated with the Uncertainty in Artificial Intelligence (UAI) conference, August 2023

[Zheng23] K. Zheng and A. Pronobis, “From pixels to buildings: End-to-end probabilistic deep networks for large-scale semantic mapping,” 2019