A measurement-based calibration approach for highly scalable timing and energy modeling of EdgeAI multi-core systems

Quentin Dariol, Sébastien Le Nours, Sébastien Pillement, Ralf Stemmer, Domenik Helms, Kim Grüttner

A Measurement-Based Calibration Approach for Highly Scalable Timing and Energy Modeling of EdgeAI Multi-Core Systems

Quentin Dariol[a,b], Sébastien Le Nours[a], Sébastien Pillement[a], Ralf Stemmer[b], Domenik Helms[b], Kim Grüttner[b]

[a]*IETR UMR CNRS 6164, Nantes Université, France*
[b]*German Aerospace Center (DLR), Oldenburg, Germany*

Abstract

Deploying Artificial Neural Networks (ANNs) on embedded multi-core platforms requires precise models for estimating and optimizing timing and energy, which is crucial for enabling novel Artificial Intelligence (AI) applications. However, predicting non-functional properties (timing, power) is challenging due to degrees of parallelism in ANNs and complex effects in execution platforms (e.g. contentions at shared resources, dynamic power management). This article presents an Electronic System-Level (ESL) timing and energy modeling flow and the associated calibration methodology for optimizing ANN deployment on multi-core platforms. The proposed flow leverages SystemC simulation to offer both speed and accuracy while ensuring high scalability in many dimensions, such as platform resources modeling. Analytical models are used for ANN layer computation and communication delays as well as power consumption and energy cost. We propose a measurement-based calibration approach to these models which enables high prediction accuracy while guaranteeing high re-usability. The calibrated models can be used across different settings without the need to re-perform a calibration phase. We validate our flow against real measurements of ANN implementations on a prototype multi-core platform. Results demonstrate over 97% accuracy in timing and 93% in energy for 54 mappings of different ANNs tested with and without the use of power management on the platform, with an evaluation time under 2s per mapping. Furthermore, we illustrate that our flow is suitable for Design Space Exploration (DSE), allowing up to 24% improvement in inference time and 16% in energy compared to baseline implementation.

1. Introduction

Artificial Intelligence (AI) and more specifically Artificial Neural Networks (ANNs) have garnered significant interest in recent years. The fast evolution of hardware platforms has rendered possible the training and execution of complex ANNs featuring millions of parameters, such as AlexNet [1], thus opening the door to a wide range of novel applications across various domains. Current trends aim at bringing ANNs at the edge part of Internet-of-Things (IoT) applications, thus removing an important number of timing and energy-costly transfers compared to their execution in the cloud [2]. At the edge, various types of embedded platforms are available, such as MicroController Units (MCUs) [3, 4], MultiProcessor Units (MPUs) [5, 6], Field Programmable Gate Arrays (FPGAs) [7, 8] and even Application-Specific Integrated Circuits (ASICs) which have emerged for the efficient processing of ANNs [9, 10]. Most of these platforms feature systems with multiple Processing Elements (PEs) combined with a communication infrastructure that allows cores to share processed data. Runtime management is needed to further optimize power consumption in such platforms. Programming ANNs on embedded platforms with limited processing and memory resources, while complying with time and energy constraints, is challenging. This design process is also constrained by development time, which limits possible optimization of these systems.

Early optimization of hardware resources and software deployment is thus essential to deliver architectures that fully meet both functional (such as ANN classification accuracy) and non-functional requirements (such as timing and energy). In this context, Electronic System-Level (ESL) [11] modeling is key to enable early evaluation and Design Space Exploration (DSE) of implementation candidates. ESL frameworks were developed to assess the performance and energy of applications deployed on multi-processor platforms early in the design process. They allow preparing optimized configurations of software (i.e. workload splitting, mapping) and hardware (i.e. number of PE, memory size) resources of ANNs. The deployment of ANNs algorithms on multi-core platforms can take benefit of data (pipeline execution of different layers) and spatial (splitting the workload inside layers between multiple PEs) parallelism. However, possible contention for accessing shared resources and dynamic power management render the modeling and prediction of non-functional properties difficult on such systems [12]. Significant efforts must be spent to correctly abstract low level details which can limit the efficiency of ESL approaches for AI systems. The calibration phase is thus critical to deliver efficient ESL models with limited effort.

This article presents a measurement-based calibration approach to facilitate the creation of efficient performance and energy models for ANNs mappings onto multi-core platforms. The proposed approach, illustrated in Figure 1, is designed for high scalability across multiple dimensions. We define by scalability the fact after being calibrated once, the models provide fast-yet-accurate assessment of timing and energy for a wide

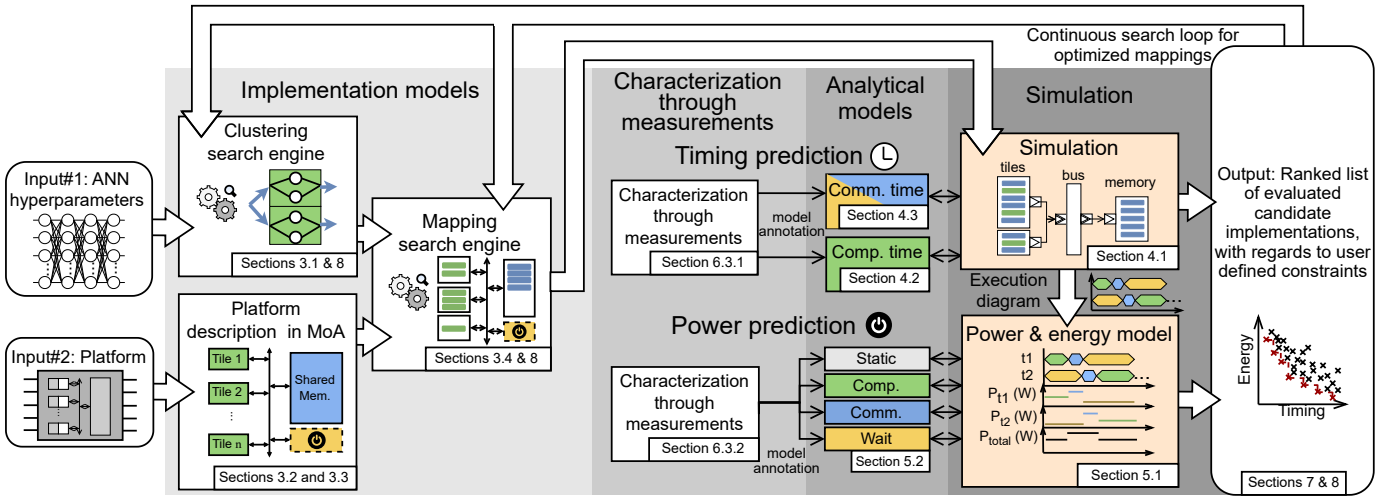


Figure 1: Overview of the proposed flow. The contributions of this article correspond to the proposed analytical models and their characterization procedure. We use the following legend in the remainder of this article: computations are represented in green color, communications in blue color, and waiting in yellow color.

range of different settings. The scopes of scalability include ANN mappings leveraging varying degree of data and spatial parallelization and different levels of communication workload and shared resource contention. Furthermore, the proposed flow scales with the physical resources of tile-based multi-core platforms, including the number of PEs, tile memory size, and dynamic power management capabilities. The full evaluation of the scalability of the flow and its limitations can be found in Section 7.3. To offer predictive models with good level of scalability, the flow relies on the Synchronous Data Flow (SDF) Model of Computation (MoC) and a tile-based Model of the Architecture (MoA). Based on these assumptions, the contribution of this article are:

- Timing and power models and the related characterization procedures are proposed for computation and communication delays and power consumption according to possible tiles phases (computation, communication, low-power mode).
- A measurement-based approach for the calibration of models to capture intrinsic platform characteristics, including power management effects. With the proposed approach, the models offer high scalability i.e. without the need to re-perform calibration for considering multiple applications partitionings and mappings.
- The validation of the proposed hybrid modeling approach through the evaluation of the prediction speed and accuracy against real implementations of 54 mappings of different ANNs on multi-core platforms of varying dimensions. The results demonstrate that the established workflow delivers high level of accuracy (more than 97 % on timing and 93 % on power and energy) with an evaluation time under 2 s per mapping.
- A demonstration of the proposed models in a DSE approach to find architecture optimization of ANNs on multi-core platforms, leading up to respectively 24 % and 16 % improved inference time and energy. Our modeling flow, DSE framework and the results presented in this article are publicly available on the project's open-source Git repository [13].

The remainder of this article is organized as follows: Section 2 gives a review of relevant literature regarding the evaluation of ANN implementations under timing and energy constraints. Section 3 presents the fundamental principles of our approach. Sections 4 and 5 describe our methodology to obtain a fast yet accurate prediction flow for timing and power/energy. Section 6 presents the calibration methodology and the experiments led to extensively evaluate our proposed workflow. Section 7.3 discusses the level of scalability of the flow in the tested dimensions, and its limitations. Section 8 presents how the modeling approach can be used to perform efficient DSE. In the last section, we conclude and give perspectives on this work.

2. Related work

Electronic System-Level (ESL) [11] modeling approaches were proposed to facilitate the high level abstraction modeling and evaluation of multiprocessor architectures. Fundamentally, they rely on the representation of a system architecture with three complementary views: application, platform and mapping descriptions. For the purpose of early performance and power evaluation, applications are described as workload models. Timing and power estimations are considered to model the influence of computation and communication loads on platform resources. In this article, we specifically compare to the methods developed to evaluate performance and power of AI systems. Approaches for the evaluation of non-functional properties of ANN mappings on edge platforms can be separated into three main categories:

1) Rapid prototyping: it focuses on systematic implementation of ANNs and measurement of tested metrics during the execution on real hardware.

2) *Analytical modeling:* it aims at formally define models to abstract low level platform details and allows early prediction of non-functional properties.
3) *Simulation-based modeling:* it relies on the virtualization of ANN mappings in order to run and observe its execution and analyze the occupation of platform resources.

2.1. Rapid prototyping

Measuring quantities such as latency and energy at test on a real hardware platform offers the highest possible evaluation accuracy. The approaches in [5, 4] provide evaluation and optimization flows for ANN implementations respectively on NVIDIA Jetson GPU and MCUs under classification accuracy, latency and energy constraints. Neural Architecture Search (NAS) [14] is included in these flows to optimize along with the implementation the design of ANNs in regards to their architectural features (e.g. number of layers, number of neurons inside layers). NAS is performed with a feedback loop using quantities measured after training and implementation on the real hardware platform. Due to the important exploration time necessary to perform NAS, these approaches are limited in exploring the degree of parallelism, mapping and scheduling of the ANN on the targeted hardware.

The authors of [15, 6] propose DSE flows to find CNN deployments that optimize latency and energy respectively on platforms featuring multi-core systems and AI accelerators, which are instrumented to measure timing and energy to rank mappings. As a general limit to rapid prototyping approaches, the systematic evaluation of ANN implementations through measurement requires an important, time-consuming effort. This type of approach also restricts the possibilities in regards to architectural exploration due to the necessity of using a fixed implementation platform.

In our work, we carry out a calibration phase through measurements for our high level of abstraction models. The calibration effort is limited due to the considered modeling rules adopted for the applications and the platform. This allows capturing characteristics from the hardware platform while still maintaining reduced development time and guaranteeing high scalability.

2.2. Analytical modeling

Analytical models are fast to execute, thus overcoming the challenges regarding evaluation time faced by rapid prototyping approaches. This type of models establish a relationship between the value of an evaluated quantity (e.g. timing, power and energy), ANN hyperparameters (e.g. layer types and numbers, number of neurons) and platform settings (e.g. number of PEs). AutoDice [16] is a framework for fast exploration of ANN mappings onto distributed heterogeneous edge devices with possible automatic generation of implementable C code. It leverages high level analytical models which formulate for each ANN layer the relationship between the number of cores usable on the hardware target and the resulting delay and power/energy cost. PreVIous [17] proposes analytical models for timing and energy regressed from measurements on two edge multi-core platforms. Both approaches [16, 17] propose simple communication time models which capture data retrieval from external memory and layer input/output communications between cores, but they don't take into account shared resource contentions.

The authors of [18] propose a DSE flow for CNNs deployed on FPGA, which allows exploring several sources of parallelism within convolution layers and evaluates possible mappings using analytical models for timing. It enables an important speedup of AlexNet's execution [1] compared to the state-of-the-art. fpgaConvNet [19] is an evaluation and optimization framework enabling fine-grained partitioning, analysis and timing prediction (with more than 93 % accuracy) of CNNs described as SDF graphs [29] on FPGA. Both approaches [18, 19] however do not propose power and energy prediction. Other approaches such as Timeloop [20], NNest [21], HALF [22] and MAESTRO [23] propose automatized evaluation and exploration of ANN mappings on FPGA accelerators and GPUs to formulate the relationship between performance, utilized area, and energy.

As general criticism to solely analytical modeling approaches, they show limitations when applied to multi-core platforms, due to the important overheads on timing and energy caused by concurrent accesses of cores to shared resources. In our work, we include fast yet accurate simulation to capture the inter-core communication and the contention effects. We complete our analytical models with measurements to deliver precise estimates with a limited effort of characterization.

2.3. Simulation-based modeling

Simulation-based approaches conveniently allow studying how cores contend for shared resources. This allows providing high scalability in regards to the number of cores and amount of communications on the platform, to the expense of low level description and thus longer evaluation time. The authors of [25] propose a simulation-based approach for platform-aware NAS for ANNs implemented on Neural Processing Units (NPUs). The simulation aspect allows modeling the complex data movement and processing through various specialized units and memory hierarchies inside NPUs, but does not focus on modeling shared resource contentions. NNSim [26] is a SystemC Transaction Level Modeling (TLM) simulator for the Eyeriss [9] ANN accelerator architecture. It is focused on providing a fast functional model to help designers verify and validate ANN implementations and does not offer power and energy prediction.

SECDA-TFLite [27] is an open-source framework which uses SystemC simulation for co-designing CPU systems with dedicated ANN accelerators. The SystemC models are calibrated by retrieving number of cycles and power consumption metrics directly from Register Transfer Level (RTL) system description. Once calibrated, the SystemC models can be re-used to explore the design space while abstracting fine implementation details, thus offering significant speed-up with high prediction accuracy. However the targeted architectures in [27] are composed of single– or double–thread only host processor, and on– and off–chip communications are performed with

Table 1: Summary of main properties and features of approaches from the state-of-the-art.

Type	Approach	Platform	Evaluation speed	Evaluated quantities		Considered form of parallelism		Platform properties	
				Timing	Power/energy	Application	Spatial	Shared resource contention	Power management
Rapid prototyping	[5]	GPU	✗	✓	✓	✓	✗	✓	✓
	[15, 4]	VPU	✗	✓	✓	✓	✗	✓	✗
	[6]	MPU + GPU	✗	✓	✓	✓	✗	✓	✗
Analytical models	AutoDice [16]	All	✓	✓	✓	✓	✓	✗	✗
	PreVIous [17]	MPU	✓	✓	✗	✓	✓	✗	✗
	[18, 19]	FPGA	✓	✓	✗	✓	✓	✗	✗
	Timeloop [20]	FPGA, GPU	✓	✓	✓	✓	✓	✗	✗
	NNest [21]	ASIC	✓	✓	✓	✓	✓	✗	✗
	HALF [22]	FPGA	✓	✓	✓	✓	✓	✗	✗
	MAESTRO [23]	ASIC	✓	✓	✓	✓	✓	✗	✗
Simulation	[24]	NPUs	✓	✓	✗	✓	✓	✗	✗
	[25, 26]	NPUs	✓	✓	✗	✓	✓	✗	✗
	SECDA-TFLite [27]	MCU + NPU	✓	✓	✓	✓	✓	✗	✗
	[8, 28]	ASIC	✓	✓	✗	✓	✓	✓	✗
	Our work	MPU	✓	✓	✓	✓	✓	✓	✓

Legend: In this table, the symbol ✗ marks properties that we deemed not acceptably satisfied and ✓ marks properties that we deem acceptably satisfied. Regarding evaluation speed, we deem that rapid prototyping approaches do not allow offering fast enough evaluation time compared to modeling approaches.

AXI stream bus, which severely limits the shared resource contention effects that we are tackling in our approach.

The approach in [8, 28] consists in an IP-based design methodology to find optimized accelerator architectures for the inference of CNNs described in the Kahn Process Network (KPN) MoC [30]. SystemC simulation is used to model shared resource contention, providing an evaluation time in the order of seconds up to the order of hundreds of seconds (excluding model compilation time) depending on the data size processed inside ANN layers and number of neurons in layers. This approach achieves more than 92 % accuracy on timing prediction. In contrast to this approach, our work is focused on tuning platform parameters and optimizing software deployment of ANNs on multi-core platforms rather than selecting hardware blocks from an IP bank to build optimized accelerators. Our models also offer a higher accuracy on timing, overall faster prediction time, and also enable power and energy analysis.

Table 1 summarizes papers discussed in this section. The originality of our work relies on the combination of analytical, simulation and measurement approaches in a hybrid modeling approach that demonstrate then fast yet accurate capabilities. The considered modeling assumptions, presented in the following section, especially allow the measurement effort to be limited but still with good level of scalability.

3. Fundamentals and system modeling assumptions

In this work we consider two of the main ANN algorithms considered in both industry and academia: the Feed-forward Neural Networks (FNNs, also commonly called multi-layer perceptrons) [31] and the Convolutional Neural Networks (CNNs) [32]. FNNs are composed of only one type of layer: the dense layer, also called fully-connected layer, used to classify data. In addition to dense layers, CNNs also contain convolutional and pooling layers generally placed before the dense ones. An example of CNN is provided in Figure 2 ⓐ, which illustrates the different types of layers.

3.1. Model of Computation (MoC)

To ease the analysis and optimization process of ANNs implemented on embedded platforms, dataflow-oriented MoCs are appropriately used [9, 33]. They allow expressing the dataflow with different degrees of parallelism. In this work we rely on Synchronous Data Flow (SDF) [29]. SDF is used in many works to model the data-flow of ANNs [19, 34, 35]. It offers a clear and well-known semantic for the description of applications, which:

- offers the separation of computation and communication phases, thus making possible the characterization and analysis of these two aspects independently,
- offers the possibility to express an application's parallelism through its partitioning in sets of actors,
- allows static analysis to prevent inter-blocking in the application, taking into account the defined token production/consumption rates for communications,
- is adapted to pipeline execution (spatial parallelism),
- supports real implementation on multi-processor systems.

SDF also offers formal models which can be used to formally describe hardware/software architectures. The interested reader can refer to [19, 36] for more details. An example of SDF Graph (SDFG) is provided in Figure 2 ⓑ. Actors, represented in green color, model computation. Their behavior is separated into three phases: read, compute and write. During the compute phase, no interference with any other actor can occur. The dataflow between actors, represented in blue color, is modeled as bounded communication channels which are buffers containing data labeled as tokens exchanged under the First In First Out (FIFO) principle. For every communication channel in the SDFG, each actor has a defined token production rate and a token consumption rate. Without loss of genericity, we assume in this work that for every channel, the token production and consumption rates are equal.

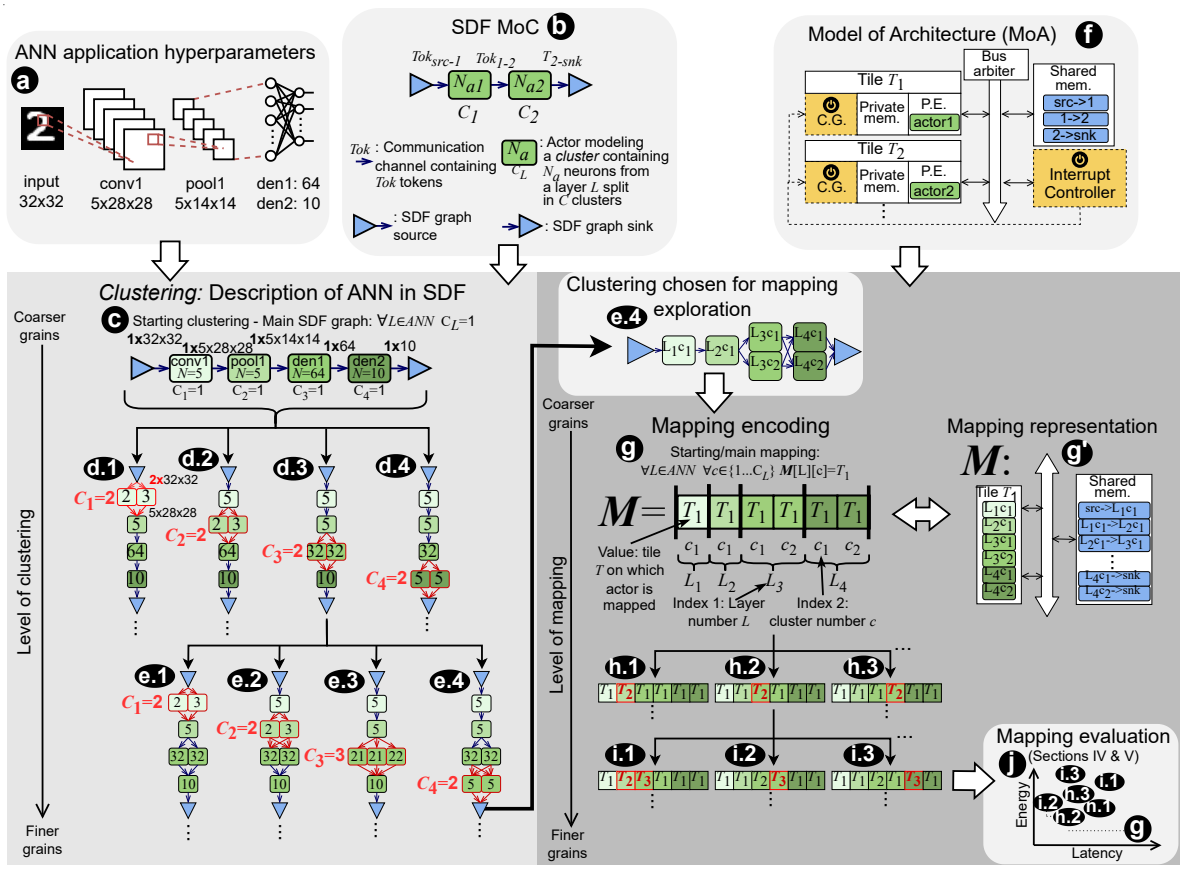


Figure 2: ANNs (a) are described in SDF (b) through a process called *"clustering"* (c, d and e). SDFGs issued from the clustering can be mapped (g, g' h and i) onto platforms that satisfy our MoA hypothesis (f). Mappings can then be evaluated with the proposed modeling flow (j).

SDF allows expressing different degrees of parallelism of ANNs with respect to the computations and communications performed. The authors of [35] present a way of capturing ANNs in SDF using the layer level of granularity, where each layer of the ANN is modeled as an actor, and the neuron level of granularity, where each neuron is modeled as an actor. In this work, we allow intermediate levels of granularity between layer and neuron grains and we also allow mixing the level of granularity inside a model: layers can be described with a number of actors that differ from one another. To properly define intermediate levels of granularity, we introduce the notion of *"clustering"* of a layer L in SDF, and we denote C_L the number of actors generated from L. All actors issued from the clustering of a layer contain roughly the same number of neurons. In Figure 2, the number inside actors represent the number of neurons it contains. The communication channels number and token consumption rate are also dependent on the clustering as illustrated in the different examples of CNN clusterings in Figure 2. c corresponds to the main SDFG for the ANN in a. It represents the layer level of granularity, where every layer L in the ANN is represented by a single actor (i.e. $C_L = 1$). The computation phase of actors corresponds to the sequential execution of the entire layers without using spatial parallelization. From the main SDFG, the individual clustering of layers C_L can be increased to create finer grained representations of the ANN. For example, in d.1, the number of neurons (in this case convolution filters) inside each cluster of the first layer is 2 and 3. Two input channels containing both the entire input image from the source are generated to provide each actor with the complete input necessary to process each neuron. Two output channels are also generated. Their size is determined by the number of neurons contained in the actor ($2 \times 28 \times 28$ tokens for the actor with 2 neurons and $3 \times 28 \times 28$ for the other). Increasing the clustering of a layer enables parallel computations with possible inference time enhancements, at the cost of increasing the communication workload, leading to possible contention and timing overheads. Each SDFG d.1 to d.4 can be refined by increasing one layer's clustering C_L by 1. We provide the example of d.3 from which e.1 to e.4 can be generated. Exploring the grain used to model ANN layers is necessary to find the best compromise between parallelism expression and communication overheads to optimize latency and energy.

3.2. *Model of Architecture (MoA)*

The considered MoA is a tile-based model in which each tile is one single-core processor with private data and instruction memories. Executing instructions from this private memory causes no interference with other tiles. Data exchanges between different tiles are performed through a shared memory. The shared memory accesses are done using shared communication

bus, which integrate an arbiter to manage concurrent accesses to the shared resources by tiles. This MoA allows respecting the separation of computation phases (performed privately inside tiles) and communication phases (which involve accesses to the shared memory) required by the MoC used. More details regarding the communication protocol in this MoA can be found in [37]. This MoA is fully compositional, i.e. adding tiles does not disturb the properties of other tiles. The use of this MoA then allows to propose performance and power models that are also scalable with respect to the number of tiles used in the implementation platform. A diagram that positions the different components of the MoA is given in Figure 2 (f). This MoA is applicable to several commonly used platforms in the industry, such as Digital Signal Processing (DSPs) SoCs (e.g. NxP MSC8156 featuring 6 tiles [38]), integrated many-core platforms (e.g. Kalray MPPA DPU - 80 tiles [39] and Intel SCC - 48 tiles [40]) and most modern platforms with AI accelerators (e.g. Coral Dev. Board - NXP i.MX 8M SoC with 5 tiles [41]). These platforms also feature additional elements that are not considered in our MoA: external memories such as DDRRAM with their access infrastructure, caches and specific AI-accelerators. In this article, we concentrate thus on ANNs with limited level of complexity as the consideration of external memory effects on performance and power is out of the scope of this work.

3.3. Power management

During the execution of ANNs on multi-core platforms, when tiles require unavailable tokens to execute, they enter a waiting state until the tokens are available. Waiting tiles can therefore be switched to a lower power consumption mode. Several techniques can be used in order to manage and reduce the power consumption of multi-core systems. In this work, we choose to implement clock gating as presented for example in [42]. In order to evaluate the impact of clock gating on ANN execution on tile-based multi-core platforms, we considered two different communication modes, as depicted by the dotted components in Figure 2 (f):

1. One mode with an active wait implemented as polling-based communications without the use of clock gating. When a tile needs unavailable data to execute, it polls the shared memory until the data is available,
2. One mode with a passive wait implemented as interrupt-based communications with the use of clock gating (C.G.). When a tile needs unavailable data to execute, it enters the low power mode (clock gating) until its clock is re-enabled by an interrupt signal, which indicates that the data is now available.

The interrupt-based platform introduces an interrupt controller to manage the interrupt signal and clock gating controllers which are represented in yellow. When finishing a read/write transaction, tiles enables the interrupt signal through the use of the interrupt controller peripheral. When the interrupt is enabled, clock gated tiles exit the low power mode and resume their execution. The additional components cause overheads in latency and increase the system's power consumption. However they enable the usage of low power mode instead of polling on the shared memory which can lead to power consumption enhancements. The trade-off in latency and energy between the two communication modes must therefore be evaluated on a case-by-case basis for each considered mapping.

3.4. Mapping

The actors are mapped on the tiles available on the platform and each tile implements the code related to mapped actors into its private memory. Communication channels between actors are mapped on the shared memory. Tiles read (*ReadTokens()* statement) and write (*WriteTokens()* statement) the data necessary for the execution of actors in communication channels. During the execution of actors' computation phase, PE cannot be interrupted. In our work, the application is self-scheduled: the scheduling is established based on the dependency between actors. We illustrate in Figure 2 how several mappings of a given SDFG are possible with the example of the ANN in (a) modeled as the SDFG (e.4). As shown in (g), **M** corresponds to the mapping in which all actors are mapped on the same tile T_1. We provide the encoding of **M** with significance of each index, as well as a graphical representation of the mapping in (g'). Similarly to the clustering refinement process, the mapping can be progressively modified by allocating actors to different tiles. For example, mappings (h.1), (h.2) and (h.3) are variations from (g), in which, one actor was mapped to T_2 while others remain mapped on T_1. Mapping actors from the same or different layers on different tiles allows leveraging data parallelism. To leverage spatial parallelism expressed by the clustering of layers, different actors from the same layer must be mapped on different tiles (e.g. (h.2) and (h.3)). Mappings can be further modified as shown in (i.1), (i.2) and (i.3). In general using more tiles enable a parallel execution of the application but it also generates more communications, with impact on the application's latency. Similarly, using more cores will increase the power consumption but the use of passive wait can significantly reduce it. As illustrated in (j), comparing both clusterings and mappings allows finding solutions that optimize timing and power properties and the number of tiles required.

4. Timing analysis flow

4.1. Shared resource contention modeling using simulation

The approach considers workload models that capture the influence of running the ANNs on processing and communication resources. Computation and communication effects are modeled with delays caused by the usage of platform resources. Our workload models are created with the SystemC framework with the same organization as in [43]. The SystemC model allows describing all the tiles and the shared resources composing the system, and predicting the impact of possible contentions on the latency of the application. The bottom part of Figure 3 provides an illustration of the hierarchy of the SystemC models with a simple application mapping. The SystemC models are built accordingly to the SDF and MoA models. Analytical models of computation and communication time are integrated in a behavioral description of each tile, which describes

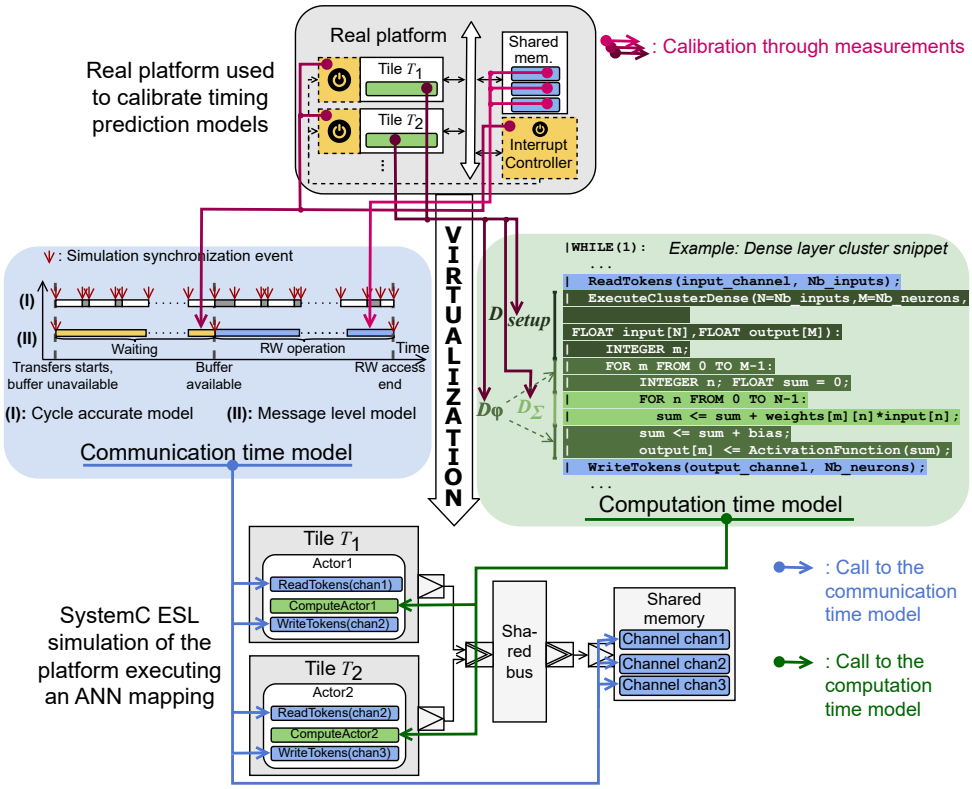


Figure 3: Illustration of the timing modeling flow's elements. The SystemC ESL simulation at the figure's bottom is used to abstract the behavior of the real platform executing ANNs as seen on the figure's left. Computation and communication statements inside tiles, respectively in green and blue, use the associated models represented in the figure's center.

the sequence of the mapped computation and communication statements. During simulation, when an actor is in the compute phase, the analytical computation time model is set to predict the corresponding delay. During communications through channels, the communication time model is called to compute the delays of communications on the platform considering the ongoing communication workload and contentions for shared resources. The simulation emulates the state of tiles (wait, read, write or compute) and shared resources during the processing of the ANN. It allows considering the two communication modes: polling and interrupt-based. Once the simulation terminated, the estimated execution traces contain the information about tiles and shared resource states as well as the actor being executed by tiles at any given time of the simulated execution. The latency and throughput of the ANN mapping can be extracted from these execution traces.

4.2. Analytical model for ANN layer computation time

The proposed analytical model can be used to predict the computation delay of any actor containing a set of neurons issued from the clustering of ANN layers. In this article we focus on convolutional, pooling and dense layers, but the proposed methodology can be used to obtain similar models for other layers. The center-right part of Figure 3 illustrates how three elementary delays: D_Σ, D_φ and D_{setup} can be identified in the provided snippet based on the operations executed to compute a dense layer. Equation 1 presents the resulting formula for the delay needed to compute a cluster of neurons from a dense layer D_{dense}.

$$D_{\text{dense}}(M,N) = M \cdot N \cdot D_\Sigma + M \cdot D_\varphi + D_{\text{setup}}. \quad (1)$$

D_Σ is the delay needed to perform the Multiply-ACcumulate (MAC) operation: the multiplication of the input of the neuron by the weight and its addition to the output of the neuron. D_φ corresponds to the delay needed to compute the activation function of the neuron but also to add the bias. In addition to these two delays, the initialization of the variables at the beginning of the function *ExecuteClusterDense* and the call and return procedures of the function causes a delay denoted D_{setup}. Due to the adopted MoA, all the data (neuron's weights, inputs) and instructions are available in the local memory of tiles. The model is thus scalable in consideration of the number of neurons in the actor M and the number of tokens in input N. Using the same approach, we propose an analytical computation time model for actors issued from the clustering of convolution and pooling layers in Equations 2 and 3.

$$D_{\text{conv}}(F,I_w,I_h,K_x,K_y) = F{\cdot}I_w{\cdot}I_h{\cdot}K_x{\cdot}K_y{\cdot}D_* + F{\cdot}I_w{\cdot}I_h{\cdot}D_\varphi + D_{\text{setup}}. \quad (2)$$

$$D_{\text{pool}}(F,I_w,I_h,K_x,K_y) = F \cdot I_w \cdot I_h \cdot K_x \cdot K_y \cdot D_{\max} + D_{\text{setup}}. \quad (3)$$

D_{conv} depends on the number of neurons (convolution filters) F in the cluster, the convolution input image width I_w and height I_h, and the convolution filters' width K_x and height K_y.

It also depends on the elementary delays D_* needed to perform the convolution operation, D_φ to add the bias and compute the activation function and D_{setup} to call the *ExecuteClusterConv* function. The delay needed to compute actors issued from the clustering of pooling layers depends on the number of filters F, the input image's width I_w and height I_h, and the maximum filter's width K_x and height K_y. It depends on the elementary delays $D_{\max}$ needed to parse the input image and compute the maximum or average of $K_x \cdot K_y$ pixels and D_{setup} needed to call the function *ExecuteClusterPool*.

The base delays D_Σ, D_*, D_φ, $D_{\max}$ and D_{setup} are calibrated through measurements as presented in Section 6.3. It can be observed on the center-right of Figure 3 that extra-delays are captured as well, linked to the implementation in code of the ANN, such as the initialization and increment of variables used inside *for* loops. The use of measurement-based model calibration allows to model these additional delays with accuracy. It also allows accounting for the effect caused by compiler optimizations and hardware, such as PE's Instruction Set Architecture (ISA).

4.3. Message level communication time model

In [37] a message level communication time model was proposed for timing prediction of dataflow applications on multi-core platforms compatible with the proposed MoA. It is composed of analytical models to predict the delay of tiles when writing/reading tokens in shared memory. The analytical models are combined with a high level abstraction executable model of channels. The resulting model allows an activity-sensitive description of communication phases during write and read accesses to the shared memory. They are performed in three phases: check token availability, buffer access (read/write) and token status update, rendering possible the prediction of communication resources use even in the case of contention due to concurrent accesses. The model is calibrated through measurements of elementary delays in the different communication phases, when the cores access the bus. After calibration, the proposed model was compared to measured delays and to a cycle-accurate communication time model, showing significant prediction speedup without loss of accuracy. The center-left part of Figure 3 illustrates how the proposed model allows predicting communication time with a reduced number of simulation synchronization events compared to the cycle accurate model, thus enhancing simulation speed. The characterization and validation process was done with polling-based communications on platforms under our MoA hypothesis. In this work, we use this model to predict the communication delays between clusters of ANN layers while considering the influence of platform shared resources. We use an extension of the communication bus model's to support interrupt-based communications with the use of clock gating presented in [44].

5. Power and energy analysis flow

5.1. Power and energy analysis using simulation

To accurately assess power and energy consumption, we propose a state-based power model. We define a *phase* as an interval with start and end times of a specific state of tiles. In this work, tile phases can either be compute, read, write, poll or clock gated. The knowledge of the phases in which tiles are throughout the execution of ANNs and the shared resources' state is obtained from the execution traces from the simulation. The execution traces contain time markers with the current phase for each tile. A new timestamp is generated when a tile changes phase during the execution. The execution traces allow thus knowing the state of all cores and shared resources at any time of the execution. Figure 4 provides an example of traces output by the SystemC simulation and the predicted system's power consumption. The proposed power modeling flow is used as post processing of the execution traces output by the SystemC simulation. It allows predicting the evolution of power consumption during the execution of the ANN. The total energy cost of the ANN's mapping is then deduced by integrating the predicted system's power consumption over the estimated inference time.

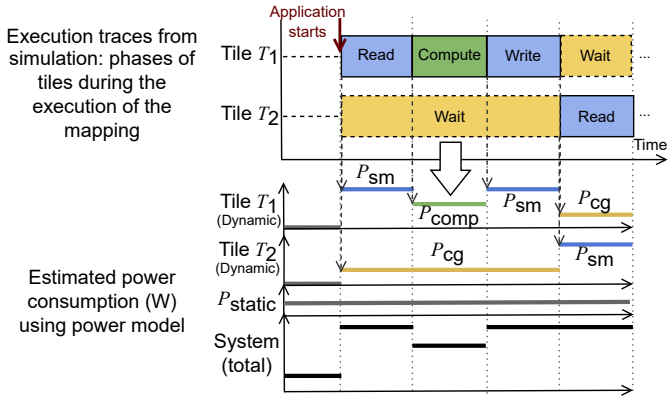


Figure 4: The execution traces from the simulation are used to predict power consumption. The computation and communication time and power models are calibrated through measurements. The phase in green corresponds to computations, phases in blue to active communications (read, write) and phases in yellow to passive waiting in clock gating mode.

5.2. Proposed power model

5.2.1. Coarse-grained model

To accurately determine the power consumption $P(t)$, both static and dynamic power consumption must be accounted for. In integrated circuits, static power consumption is due to the leakage current that flows continuously as long as transistors are powered. This leakage current is present even when the transistors are not actively switching states. Dynamic power consumption, on the other hand, is associated with the switching activity of the transistors [42]. We consider in this work that the static power consumption P_{static} is the power consumption of the multi-core system when supplied with power but not clocked. When using power management, additional components are implemented to support interrupt-based communications and clock gating. We thus introduce different models for the static power consumption of the platform: with power management $P_{\blacktriangle,\,\text{static}}$ and without $P_{\triangle,\,\text{static}}$. Regarding dynamic power consumption, when executing ANNs, tiles execute com-

putation and communication activities, which are strictly separated due to the SDF MoC and our MoA. During the execution of ANN mappings, the different possible tile phases are illustrated on the execution traces in Figure 4 when using power management. We describe the dynamic power consumption of the platform executing ANNs using the following terms:

- $P_{\text{comp}}(t)$, the total power consumption of tiles in computation (comp.) phase at time t.
- $P_{\blacktriangle,\,\text{comm}}(t)$, the total power consumption in communication (comm.) phase at time t when using power management, and $P_{\triangle,\,\text{comm}}(t)$ when not using power management.

To keep our models simple, we make the assumption that dynamic power consumption models $P_{\text{comp}}(t)$ and $P_{\blacktriangle/\triangle,\,\text{comm}}(t)$ do not depend on tile private memory sizes. In Section 7.3 we discuss the validity of this hypothesis and show some limitations. To summarize, the total power consumption of the system at time t is provided in Equations 4 and 5 respectively with (▲) and without (△) power management.

$$P_{\text{total},\blacktriangle}(t) = P_{\blacktriangle,\,\text{static}} + P_{\text{comp}}(t) + P_{\blacktriangle,\,\text{comm}}(t). \tag{4}$$

$$P_{\text{total},\triangle}(t) = P_{\triangle,\,\text{static}} + P_{\text{comp}}(t) + P_{\triangle,\,\text{comm}}(t). \tag{5}$$

5.2.2. *Refinement of $P_{comp}(t)$*

When tiles are in computation phase, they are performing ANN computations (i.e. processing neurons from layers). In this phase, tiles are independent from one another as all necessary data is contained in the private memory of the tile. The proposed model for $P_{\text{comp}}(t)$ is thus linear in consideration of tiles in computation phase. The model for $P_{\text{comp}}(t)$ is provided in Equation 6.

$$P_{\text{comp}}(t) = \sum_{i=1}^{T} P_{\text{comp},i}\ \alpha_{\text{comp},i}(t),$$

$$\text{with}\quad \alpha_{\text{comp},i}(t) = \begin{cases} 1 & \text{if tile } i \text{ is in computation phase at time } t, \\ 0 & \text{otherwise.} \end{cases} \tag{6}$$

In this equation, T denotes the number of tiles in the platform. $P_{comp,i}$ is the power consumption of tile i in computation phase, with $1 \leq i \leq T$. $\alpha_{comp,i}(t)$ is a factor indicating if tile i is in computation phase at time t. The $\alpha_{\text{comp},i}(t)$ are obtained from the execution traces of the simulation.

5.2.3. *Refinement of $P_{comm}(t)$*

When tiles are in the communication activity, they either perform a read or a write in the shared memory or wait for the availability of data. With power management, tiles enter clock gated mode when waiting for data, and without, tiles perform active waiting by polling the shared memory. When clock gated, tiles are not supplied with the clock and should theoretically contribute only to the static power consumption of the system $P_{\blacktriangle,\,\text{static}}$. However, on some platforms - as observed on our FPGA-based experiment platform -, tiles in clock gated mode still bear a contribution to dynamic power which must be taken into account in the model, e.g. due to internal components such as the private memory still being supplied with the clock. Read, write and polling phases are interleaved and correspond to the total power consumption of tiles accessing the shared memory at time t. When several tiles try to access a shared memory simultaneously, the bus arbiter gives the access to only one tile and the others are paused until the shared memory is available. Shared memories can thus be accessed by only one tile at any given time t. To simplify our model, we assume that tiles in read, write and polling phases share the same dynamic contribution to power denoted P_{sm}, which corresponds to the power consumption of tiles accessing the shared memory. According to that, the power consumption $P_{\text{comm}}(t)$ is either equal to P_{sm} when at least one tile is accessing the shared memory, or 0 otherwise. The proposed models for $P_{\text{comm}}(t)$ without and with power management are given respectively in Equations 7 and 8:

$$P_{\triangle,\,\text{comm}}(t) = P_{\triangle,\,\text{rwp}}(t) = P_{\text{sm}} \left(1 - \prod_{i=1}^{T} \left(1 - \alpha_{\text{rwp},\,i}\,(t)\right)\right) \tag{7}$$

$$= \begin{cases} P_{\text{sm}} & \text{if at least one tile is reading, writing,} \\ & \text{or polling on shared memory at time } t, \\ 0 & \text{otherwise.} \end{cases}$$

$$P_{\blacktriangle,\,\text{comm}}(t) = P_{\blacktriangle,\,\text{rw}}(t)\ + P_{\blacktriangle,\,\text{cg}}(t) \tag{8}$$

$$= P_{\text{sm}} \left(1 - \prod_{i=1}^{T} \left(1 - \alpha_{\text{rw},\,i}\,(t)\right)\right) - \sum_{i=1}^{T} P_{\text{cg},i}\ \alpha_{\text{cg},i}(t).$$

In these equations, T denotes the number of tiles in the platform. P_{sm} is the power consumption of one tile accessing the shared memory. $\alpha_{\text{rwp},\,i}\,(t)$ is a factor indicating if tile i is accessing the shared memory at time t, when not using power management. $\alpha_{\text{rw},\,i}\,(t)$ is defined similarly when using power management, thus excluding the waiting phases. P_{cg} is the power consumption of tiles when clock gated and $\alpha_{\text{cg},\,i}\,(t)$ indicates if i is clock gated at time t.

5.2.4. *Possible refinement of P_{static}*

P_{static} is usually fixed as constant in power models. In this work, we propose a possible refinement of this part which extends the model's scalability to platforms that respects our MoA but feature different dimensions in regards to the number of tiles and size of private memories. The experiments and results presented in this article use this optional refinement. In Equation 9 and in Equation 10 we propose the refined model for $P_{\triangle,\,\text{static}}$ and for $P_{\blacktriangle,\,\text{static}}$ respectively.

$$P_{\triangle,\,\text{static}}(T, \mathcal{M}) = \sum_{i=1}^{T} P_{\text{tile, mem}}\ \mathcal{M}_i + T P_{\text{tile, core}} + P_{\text{circuit}}. \tag{9}$$

$$P_{\blacktriangle,\,\text{static}}(T, \mathcal{M}) = P_{\triangle,\,\text{static}}(T, \mathcal{M}) + P_{\blacktriangle,\,\text{components}}. \tag{10}$$

In these equations, T is the number of tiles in the considered platform. The term $\mathcal{M}_i$ corresponds to the private memory size

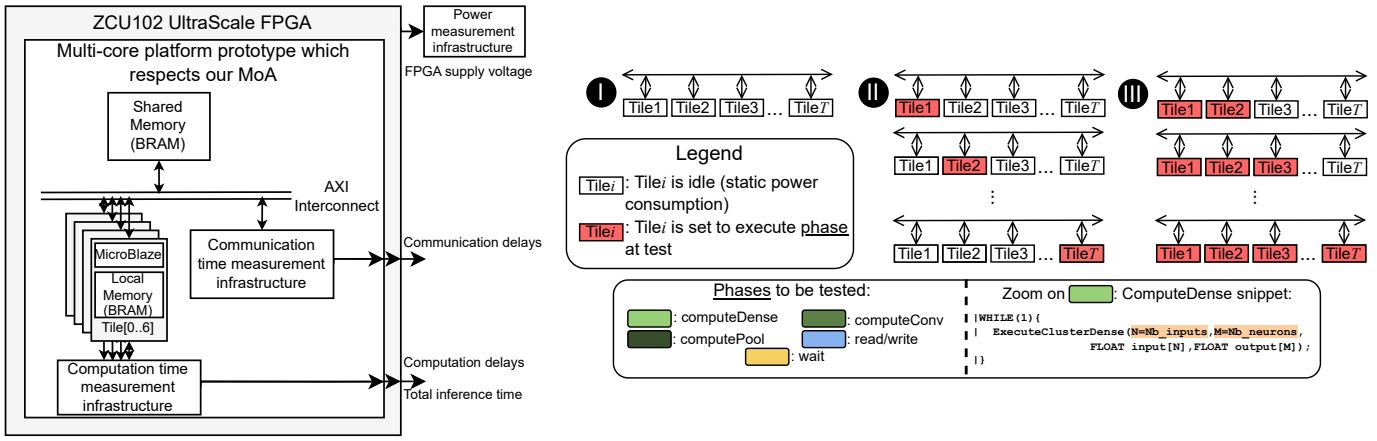


Figure 5: On the left: implementation platform and measurement infrastructure used for model calibration and validation. On the right: illustration of the configurations tested during the models' calibration.

of tile i. The private memory of a tile is used to store its instructions and data. The vector $\mathcal{M} = \{\mathcal{M}_1, \mathcal{M}_2 \ ... \ \mathcal{M}_T\}$ of size T corresponds to the private memory size of each tile. $P_{\text{tile, mem}}$ is the base contribution to static power consumption of a unit of private memory. $P_{\text{tile, core}}$ is the contribution of the processor core of tiles to the static power consumption, which is independent of the private memory size. P_{circuit} is the contribution of the remainder of the circuit to the static power consumption, which corresponds to the shared memory and shared interconnect. Finally, $P_{\blacktriangle, \text{components}}$ is the contribution to the static power consumption of the additional components used to support power management, i.e. the interrupt and clock gating controllers. The base power consumption terms $P_{\text{comp},i\in\{1,2...T\}}$, $P_{\text{cg},i\in\{1,2...T\}}$, P_{sm}, $P_{\text{tile, mem}}$, $P_{\text{tile, core}}$, P_{circuit} and $P_{\blacktriangle, \text{components}}$ are obtained during the measurement-based calibration of the model, as presented in Section 6.3.

6. Experimental setup

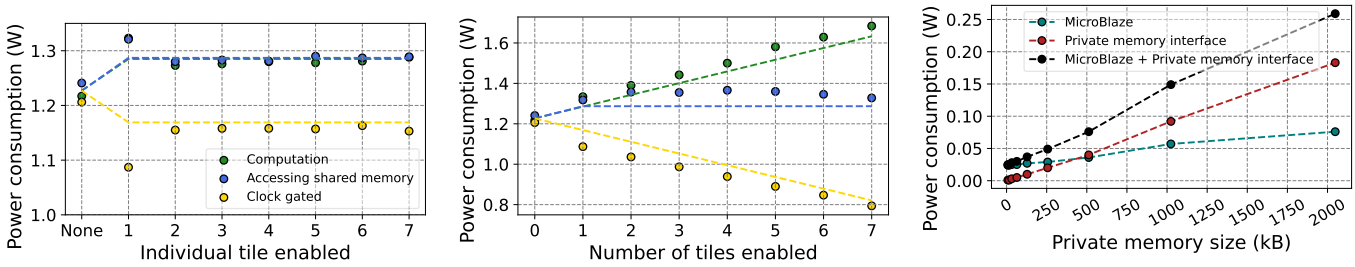
6.1. Platform

In order to perform the calibration and validation of the proposed modeling flows, we implemented a tile-based multi-core platform which respect the MoA. We implemented one platform with interrupt-based communications with clock gating and one with polling-based communications without clock gating. The platforms are implemented on a AMD ZCU102 board, which features a UltraScale MPSoC+ FPGA [45]. They are implemented on the programmable logic section of the FPGA. An illustration of the implementation platform is provided in the left part of Figure 5. We selected this type of experiment platform in place of a Commercially Available Off-the-Shelf (COTS) platform to fully control the hardware resources (number of tiles, power management) and the mapping of software resources. This also allowed us to accurately monitor the occupation of resources.

The processing core of the tiles is a MicroBlaze [46] equipped with Floating Point Units (FPU) and multiplier ISA extensions and running at 100 MHz. The private memory of tiles and the shared memory are implemented as Block Random Access Memories (BRAMs), which are internal to the FPGA SoC. The communication medium to access the shared memory is an Advanced eXtensible Interface (AXI) bus. In the clock gating version, the interrupt controller and the clock gating controllers are IPs provided by AMD [46]. The main version of the implemented platforms is composed of 7 tiles, one with 1MB of private memory (T_1) and others with 256kB of private memory. T_1 contains a bigger private memory than the others to enable the execution of single-core scenarios, which are memory-intensive.

6.2. Measurement infrastructure

We use the timing measurement infrastructure presented in [47]. It relies on code instrumentation to issue a start signal at the beginning of the SDFG's execution and a stop signal at the end, requiring the execution of a single processor instruction for both. The overhead on timing when using this infrastructure is thus marginal. Based on the elapsed time between the two signals, the SDFG's execution time is measured. In addition to this, to perform an accurate calibration of communication times, we used Integrated Logic Analyzers (ILAs) [48] which allows probing AXI signals. The targeted FPGA features several power supplies. We probed *VCCINT*, which corresponds to the supply voltage of the programmable logic of the FPGA, and *VCCBRAM*, which corresponds to the supply voltage for the BRAM. We observed marginal variation of the power consumption on *VCCBRAM* when performing our calibration. In fact, the activity linked to memory usage is observed rather on the *VCCINT* power supply, as the memory controllers are implemented on the programmable logic part of the FPGA. For this reason, we focus only on *VCCINT* power consumption in this study. The power measurements are obtained using the R&S HMC8012 Digital Multimeter [49] at a sampling rate of approximately 100 samples per second. We took 10 000 samples for each considered measurement in order to obtain a representative average of the system consumption.



(a) Power model, configurations II

(b) Power model, configurations III

(c) XPE estimates of the dynamic power consumption of a tile based on its private memory size.

Figure 6: Graphs (a) and (b) display power consumption data including both static and dynamic components for configurations II and III defined in Figure 5. The legend of (a) is the same for (b). Graph (c) displays the static power model refinement for a single tile based on private memory size of tile.

6.3. Model calibration through measurements

6.3.1. Timing models

To calibrate the computation time models, we measure delays required for the execution of clusters with varying workloads, e.g. in regards to the number of inputs N and number of neurons M for dense layers, as highlighted in orange color in the snippet in the bottom right of Figure 5. The measured data is then used to retrieve the elementary delays identified in the models using multi-linear regression. The calibrations are done when using the FPUs and multipliers extensions of the MicroBlaze's ISA and compiler optimization setting *-O0*. Calibrating another sets of elementary delays when using different hardware extensions and compiler settings is possible using the same methodology, and would allow engineers using our flow to test several settings. We calibrated the computation time elementary delays, given in processor cycles (p.c., 10 ns in our setup):

- Convolution layers: $D_* = 77$ p.c., $D_\varphi = 631$ p.c. and $D_{\text{setup}} = 28$ p.c.
- Dense layers: $D_\Sigma = 50$ p.c., $D_{\varphi,ReLU} = 106$ p.c., and $D_{\text{setup}} = 31$ p.c.
- Max pooling layers: $D_{max} = 25$ p.c. and $D_{\text{setup}} = 106$ p.c.

The advantage of characterization through measurements over estimations is to offer higher accuracy, as it accounts for hardware/software implementation details. These include the intricate behavior of the platform (e.g., use of FPUs), code structure (e.g., variable initialization), and compiler optimization effects. To compare, let us assume that each statement in the snippet provided in Figure 3 requires three instructions, ignoring pipeline optimizations, with each instruction taking one cycle to execute. Under this assumption, the dense layers delays would be $D_\Sigma = 30$ p.c., $D_\varphi = 61$ p.c., and $D_{\text{setup}} = 6$ p.c., which is a significant underestimation of the values obtained through measurements.

The communication time model is also calibrated through the regression of measured data regarding the time spent in communication, when performing read and write tokens statements and waiting for the availability of data. These delays are measured by probing the communication bus on the targeted platform using ILAs. On the one hand, the calibration shows that the interrupt-based communication with power management support introduce extra delays to enter and exit waiting phases. These are due to the setup of the interrupt controller and clock gating mode. They are not observed with polling-based communication without power management. On the other hand, polling-based communication cause extra-delays linked to the constant polling of shared memory by tiles in waiting phases, which are not observed with interrupt-based communication.

6.3.2. Power model

The power model calibration process involves measuring the power consumption of the targeted multi-core system under various operating conditions and using multi-linear regression to determine the base power consumption terms (e.g. P_{comp}). The operating conditions' specifications are illustrated on the right of Figure 5. They consist in I: measure the static power consumption of the system when all tiles are idle, II: measure the power consumption of the system when each tile in the platform is set individually in the operating condition at test and III: measure the system's power consumption when tiles are progressively set to execute simultaneously. Only I and II are necessary to perform the calibration of the model, but additional configurations like III allow increasing the accuracy. The calibration data and calibrated models are illustrated in Figure 6 (a) and (b). We calibrate $P_{\text{comp}} = 0.058$ W, $P_{\text{cg}} = 0.058$ W and $P_{\text{sm}} = 0.060$ W for our setup. Our models are based on the assumption that dynamic power consumption models do not depend on tile private memory sizes. This hypothesis is valid for smaller private memory sizes, within the 1024 kB threshold. As it can be seen on Figure 6 (a), tile 1 has a slightly increased power consumption compared to others. This is due to the fact tile 1 is equipped with a 1024 kB private memory, others only having 256 kB. The difference in dynamic power consumption is however observed to be marginal (maximum 6 % difference between tile 1 and the others) as the private memory size remains under 1024 kB.

To calibrate the refined model for static power consumption presented in Equations 9 and 10, we created designs of multi-core platforms with different number of tiles and allocated private memory sizes, with and without power management. Due to our implementation technology, we had the possibility to alleviate the data retrieval effort by using instead chip provider estimates from the Xilinx Power Estimator (XPE) presented in [50], which is reliable for predicting static power consumption. We then performed a linear regression of the estimated

data from XPE. We provide in Figure 6 (c) a graph containing a subset of the gathered data for the evolution of tile power consumption based on private memory size, which shows contribution to static power consumption of tiles evolving linearly with the private memory size. We obtain $P_{\text{tile, mem}} = 1.16 \times 10^{-3}$ W, $P_{\text{tile, core}} = 0.031$ W and $P_{\blacktriangle\text{, components}} = 0.033$ W through the regression of XPE estimates. To conveniently set P_{circuit}, we calculate the difference between the measured static power consumption of our main platform prototype obtained from ❶ and the model obtained from XPE estimates. Through this process we obtain $P_{\text{circuit}} = 0.678$ W.

7. Validation results & discussions

7.1. Tested criteria

The calibrated modeling flow, tested setups and results are available on our open source Git repository [13]. In order to evaluate our modeling flow, we tested and stressed it in the following regards:

1. Consideration of applications of different complexity. We considered four ANNs to test our flow, including three FNNs and one CNN, as illustrated at the top of Table 2. The FNN1 has minimal computation workload, thus testing our flow in high communication-to-computation ratio situations. The FNN2 and FNN3 have three layers and allow testing intermediate computation and communication workloads. They are trained on different data-sets: MNIST [32] and GTSRB [51], thus also testing different input, output, source and sink sizes. The CNN also implements convolution and pooling, thus testing the applicability of our flow to these additional types of layers. Despite the tested applications' simplicity, they allow testing a significant number of representative situations. They also have a low number of parameters and thus reduced memory requirements, thus enabling their implementation on our prototype multi-core platform on FPGA with limited private memory sizes possibilities. We used SDFGs with a low and high amount of actors and communication channels, respectively 2 and 3 at the lowest, and 22 and 113 at the highest.
2. Consideration of a total of 54 mappings using 1 up to 7 tiles. The multi-core mappings leverage both data and spatial parallelism.
3. Consideration of mappings with high and low amount of communications. We define for this the notion of communication rate as the percentage of time spent by all tiles on average in communication (including read/write and wait phases). The lowest observed communication rate is 2 % and the highest is 70 %.
4. Mappings are tested with and without power management.

Prediction errors are computed using the formula $\epsilon = \frac{X_{\text{P}} - X_{\text{M}}}{X_{\text{M}}}$ where X_{P} is the prediction and X_{M} is the measurement. The number of iterations simulated by the SystemC model is set by the user and must be high enough to properly evaluate the effect of spatial parallelism leveraging. In our experiments, for each considered mapping, we empirically set the total number of iterations to 100. Figure 8 in appendix provides all the detailed evaluation setups and results.

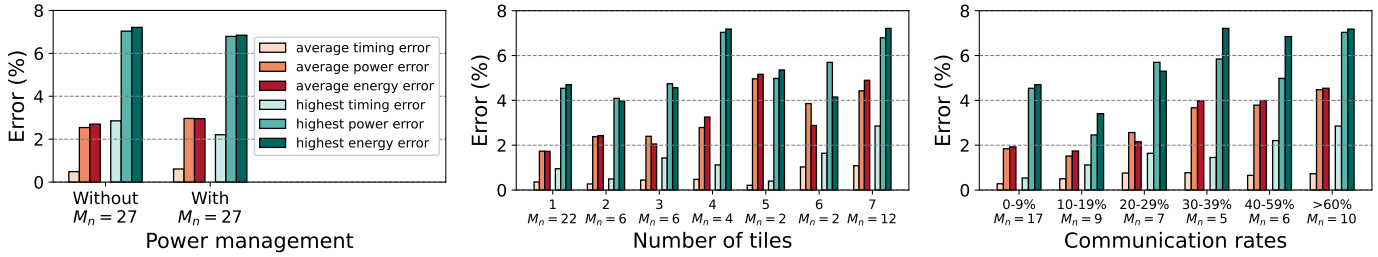
7.2. Observations and discussions

Results overview. Overall, the proposed hybrid modeling flow allows predicting timing with more than 97 % prediction accuracy, and power and energy with more than 92 %. The worst observed error regarding timing is 2.85 % on latency, 7.03 % on power and 7.21 % on energy. The model is also fast with an evaluation time of both timing and power/energy under 2 s for 100 simulated iterations per mapping, observed on all the 54 tested mappings. The fast-yet-accurate evaluation is made possible by the hypothesis of strict separation of computations and communications using SDF and the MoA, which allows building separate computation time models for ANN layers and communication time model.

The analytical computation time model is validated by the high accuracy, more than 99 % on average, obtained on execution time prediction of single-core scenarios, which have a limited amount of communications. The use of simulation and the message level communication time model offer an accurate modeling of communications on the platform and contentions when multiple cores access simultaneously the shared memory. This allows offering high accuracy on multi-core mappings. The power consumption linked to the different phases of tiles are correctly modeled through the use of the execution traces from our simulation-based timing modeling flow. The decrease in prediction accuracy between timing and power is due to the power measurement infrastructure having reduced accuracy compared to the time one, and to the propagation of error from the execution traces predicted with the timing modeling flow on power predictions.

Evaluation speed. On the 54 tested ANN mappings, the evaluation time per mapping is on average 1.76 s and at most 2.27 s on an AMD Ryzen 7 PRO 4750U CPU @ 1.70 GHz. This duration includes the compilation time of the scenario and running the simulation. The power and energy estimation is performed as post-processing of the simulation. Its duration is observed to be marginal compared to the compilation and simulation time. By ways of contrast, our automatized timing and power measurement infrastructure used for the calibration and validation of the models takes up to 120 s including also compilation and FPGA programming time, but excluding ANN training, FPGA design synthesis, Board Support Package (BSP) generation and source code development. Our SystemC models offer a significant speedup, allowing engineers to perform fast and reliable evaluation of ANN mapping candidates under timing and energy constraints, while alleviating important effort spent on the development of such applications on the targeted platform.

Scalability regarding ANN type and complexity. Table 2 provides the average and highest (max) prediction accuracy of the models for the four ANNs. The FNN1 is the application with the highest timing error (0.83 % on average over all tested mappings, 2.85 % at highest). The slightly higher timing prediction



(a) Error by use of power management

(b) Error by number of tiles used in the mapping

(c) Error by communication rates

Figure 7: Average and highest observed prediction error on timing, power and energy in regards to different metrics. The legend of (a) is the same for (b) and (c). M_n denotes the number of mappings in each category.

Table 2: Summary of the average (avg) and highest (max) observed prediction error by ANN for every tested metric.

Application	FNN1	FNN2	FNN3	CNN1
	IN: 1x28x28 → 10 dense1 → 10 dense2 → OUT: 10	IN: 1x28x28 → 32 dense1 → 16 dense2 → 10 dense3 → OUT: 10	IN: 1x24x24 → 30 dense1 → 30 dense2 → 43 dense3 → OUT: 43	IN: 1x28x28 → 5 conv1 → 5 pool1 → 64 dense1 → 10 dense2 → OUT: 10
timing error % (avg)	0.83%	0.31%	0.62%	0.42%
power error % (avg)	2.19%	2.46%	2.58%	3.96%
energy error % (avg)	2.08%	2.54%	2.68%	4.20%
timing error % (max)	2.85%	1.13%	1.64%	1.45%
power error % (max)	4.88%	4.94%	6.78%	7.03%
energy error % (max)	6.85%	5.30%	6.22%	7.21%

error on this application is due to FNN1 having the highest communication workload ratio compared to the computation one. The prediction errors for power and energy are observed to be slightly higher on the CNN1 application. This is due to the different types of computations inside layers (convolution, pooling and dense). However this difference remains minimal. Globally, the results show no major influence of the application type on the prediction error.

Scalability regarding the use of power management. Figure 7 (a) illustrates the prediction error with and without power management. It can be observed that the prediction error on the three tested quantities is slightly higher when using power management, but this difference is marginal. This observation is made when testing 27 different mappings for both modes, which shows that the proposed models offer scalability in regards to the use of power management.

Scalability regarding the number of cores used and communication rates. Charts are provided in Figure 7 (b) and (c) depicting how the prediction error of the models evolve based on the number of cores used and the communication rates, which are intrinsically linked. For both timing and power, the error raises in an apparent linear way with the amount of communication and core number. The error reaches up to 2.85 % for timing and 7 % for power and energy for mappings with approximately 70 % communication rate. It is important to note that scenarios where all tiles spend on average more than 60 % of the execution in communication are exceptional: such scenario are not desirable as the platform's PEs are significantly underutilized. When performing a linear regression of the highest observed error by the number of cores and communication rate, the resulting models observe that the worst case error remains bounded under the threshold of 10 % for mappings with up to 10 cores and 100 % communication rate. Within these boundaries, the models remain confident.

The observed phenomenon is due to the communication model used in the proposed approach, which was presented in [37]. This model has some limitations due the way low level communication details are abstracted. The estimated durations only consider the status of the communication infrastructure at the beginning of each communication statement. This neglects possible evolution of the infrastructure status over the computed duration. The limitations of the communication time model are further detailed in [52], where possible enhancements are proposed.

Baseline comparison. As means of comparison with state-of-the-art analytical modeling approaches presented in Section 2 such as Timeloop [20] and MAESTRO [23], we built analytical models that enable modeling time and power / energy for ANN mappings described in our MoC and mapped on platforms which respects our MoA. The analytical models are composed of our calibrated communication / computation time and power models, respectively presented in Sections 4.2, 4.3 and 5.2, but excluding the simulation flow. In addition to what is offered in related works, they also support power management and model data dependency while describing exchanges of data through interconnect-bus types as used in our approach. They are also calibrated for the tested platform. However, without their integration into the proposed complete simulation-based approach, they cannot describe shared resource contention as accurately.

We evaluated the 54 mappings with the analytical models. Results show that these models provide a very fast evaluation speed in the order of milliseconds. They are reliable on tim-

ing, power and energy prediction for mappings with and without power management, with low communication regimes and reduced number of cores. However, the error grows in a seemingly exponential trend with communication rates and number of cores, reaching up to 28 % on timing and 22 % on power and energy on mappings with more than 60 % communication rates. The analytical models show limitations especially when predicting for mappings that leverage both data and spatial parallelism, for which important shared resource contention effects occur. When applied to multi-core platforms which present shared resource contention problems, this type of models prove unreliable compared to the proposed simulation-based flow. The interested reader can find more details on this comparison in [36].

Table 3: Power and energy prediction error regarding different platform dimensions. Each row corresponds to a different platform. On the left, the private memory size per tile is indicated in kB — grayed out boxes indicate that the tile is not implemented on the platform. In the center, the error on static power consumption is provided. Then the error on power and energy when running mappings is provided, including static and dynamic power. M_n denotes the number of tested mappings, *avg* denotes the average prediction error and *max* denotes the highest observed prediction error.

Platform (number of tiles, private memory size per tiles in kB)							Error static power (%)	M_n	Error running mappings (%)			
									Power		Energy	
T_1	T_2	T_3	T_4	T_5	T_6	T_7			avg	max	avg	max
1024	256	256	256	256	256	256	1.54	54	2.54	7.03	2.70	7.21
512	256	256	128	128			0.53	2	2.50	2.53	2.71	2.87
64	64	64					3.33	4	3.56	4.93	3.95	5.13
64	64						4.74	2	0.95	1.25	0.54	0.75
512							2.19	6	4.74	5.25	4.32	4.85
1024							4.22	6	7.56	8.36	7.16	7.97
2048							1.57	6	9.21	8.82	10.71	10.32

Scalability regarding different platform dimensions. Table 3 shows summarized results on the applicability of our power and energy modeling flow to multi-core platform with different number of tiles and private memory sizes. We tested 4 multi-core platforms, including the main platform used to test all 54 mappings, and 3 single-core platforms with important private memory size. These platforms are dimensioned to support the execution of sub-sets of the 54 mappings. There are no variation regarding timing behavior on these different platform versions (timing results are therefore not presented in the Table 3). This validates the scalability of the timing model to platforms of different sizes subscribing to our MoA.

We observe that the error when predicting the static power term only lies below 5 % for all tested platforms. This allows validating the static power model presented in Equations 9 and 10 and the proposed calibration approach. The prediction error on both power and energy is on average 2.5 % for all tested mappings on multi-core platforms, which means that the flow is highly accurate. The error gets however exceptionally high for mappings run on single-core platforms, reaching up to 10.32 % on energy at highest on the one with a 2048 kB private memory. We observe in fact that the error grows steadily with the private memory size on single-core platforms. Since the static power consumption model is validated, our hypothesis that private memory size does not influence the dynamic power consumption terms of tiles is proven to be not applicable when focusing on single-platforms with important private memory size. On the FPGA used to implement our platform, BRAMs are placed in rows along the programmable logic sections. The tile's private memory tends thus to be inefficiently distributed surrounding the PE, requiring significant routing resources and causing the observed marked increase in dynamic consumption during memory accesses. We further discuss this phenomenon in [36]. We observe however that the error remains bounded below 10 % on single and multi-core platforms with tiles equipped with at most 1024 kB of private memory. This effect is therefore mitigated under this threshold, as specified in Section 6.3. Within the 1024 kB private memory size boundary, the power and energy modeling flow is scalable in regards to different platform dimensions.

7.3. Summary on the scalability of the proposed models

In Section 7.2, we have evaluated the proposed timing and energy modeling flow in regards to many dimensions. Table 4 provides a summary of the results of this evaluation, and the conclusion regarding the scalability of the flow. We define by scalability the fact after being calibrated once, the models provide fast-yet-accurate assessment of timing and energy for a wide range of different settings. When deploying our modeling flow to a new platform, the models needs to be calibrated once following the procedure described in Section 6.3. They can then be used to efficiently estimate the timing, power and energy of ANN mappings with scalability in regards to many dimensions.

Regarding ANN hyperparameters, our experiments show that the timing model is scalable in regards to layer types, layer number and neurons number. Extending the proposed flow to other types of layers (other than convolution, pooling and dense) and different activation functions (other than ReLU) is possible and requires building new computation time models with the methodology described in Sections 4 and 6.3. The results also show that the power and energy model is scalable in regards to any ANN application hyperparameters, including layer type as the same base power consumption term is used for all layer types without important error variation. Regarding clustering / mapping and number of tiles, the results show that the models are scalable for mappings with small and large number of actors and channels and with up to 7 tiles. The timing models are proven to be scalable in regards to the private memory size of tiles. The power and energy models show however limitations when applied to platforms with large memory size. For memory of less than 1024 kB, the model provides acceptable prediction accuracy. Regarding power management, the timing and power models have proven to be scalable when applied to clock gating. However, other power management approaches come with different complex phenomenon not currently captured in our models. For example modeling Dynamic Voltage and Frequency Scaling (DVFS) requires capturing several voltage and frequency modes with important power con-

Table 4: Summary of the scalability of the proposed timing and power models in regards to several dimensions.

Scalability of model / Scope	ANN hyper parameters	Clustering / mapping	Number of tiles	Private memory size	Power management	Compiler optimization settings	ISA extensions	Communication medium (e.g. bus)
Timing models	Scalable. New layer types need new model.	Scalable	Scalable, tested up to 7 tiles	Scalable	Scalable for clock gating. Other power management techniques require a new model.	Re-calibration necessary	Re-calibration necessary	Scalable for interconnect-bus type. Other communication medium require a new model
Power and energy models	Scalable	Scalable	Scalable tested up to 7 tiles	Scalable within private memory size threshold	Scalable for clock gating. Other power management techniques require a new model.	Re-calibration necessary	Re-calibration necessary	Scalable for interconnect-bus type. Other communication medium require a new model

sumption and timing behavior difference. By following the presented model characterization and calibration methodology, similar fast-yet-accurate models could be developed within the proposed hybrid simulation-based framework. Our flow can capture and compare the effects of compiler optimization settings (e.g. *-O0* compared to *-O3*). To achieve this, different sets of elementary delays should be calibrated for each setting. Our flow can also be used to evaluate open-hardware platforms, such as RISC-V-based, testing different ISA extensions and assessing their contributions and costs in terms of energy and inference time. Different sets of elementary delays and base power consumption terms should be calibrated for the tested settings, thus enabling a case-by-case assessment of ISA extensions' effects for considered ANNs. Regarding communication medium types, the timing and energy models can be reused as such for AXI bus with interconnects. Re-calibration is necessary to extend the applicability of the model to similar bus, such as Wishbone or Intel Avalon. However, the current modeling flow will not support other medium such as Networks on Chip (NoC), which require different dedicated models to be supported.

The proposed modeling flow should be applicable, with re-calibration, to Commercial Off-The-Shelf (COTS) platforms provided that the signals required for calibration are observable. On such platforms, computation and communication time model base terms shall be retrieved by instrumenting the cores. The targeted platform should include cycle counters or platform management units to enable timing measurements necessary for the calibration of the computation and communication time models base terms. Shunts resistors that can be instrumented are necessary for power measurements for the power model's calibration. To be able to re-use our flow, engineers should follow the methodology to calibrate the proposed models as described in Section 6.3.

8. Design Space Exploration (DSE)

We illustrate the use of our modeling flow to explore and optimize ANN mappings in a DSE flow available on our open source Git repository [13]. The DSE module allows to automatically search for optimized clusterings and mappings. Candidate solutions are successively generated, compiled and run to be evaluated and then compared. The exploration duration can be specified by the user. The list of evaluated mappings is saved in either *.json* or *.csv* format. The tooling offers the possibility to generate the C code to implement any of the evaluated mappings on embedded multi-core platforms. The exploration is enhanced by utilizing the Branch & Bound (BB) exhaustive search algorithm presented in [53]. The BB algorithm is used two times: first to perform the search of clusterings, then of mappings. The clusterings and mappings are progressively upgraded using BB following a greedy approach: the selected next branch is the one with the immediate best score. The upgrading process follows the principles presented in Sec. 3.1 and 3.4, and illustrated in Figure 2. For each considered mapping, the DSE tool generates the SystemC file to simulate the mapping, and evaluates it using the proposed modeling flow as presented in Sections 4 and 5 and validated in Section 7. Our models provide in particular three estimated metrics per mapping: the execution time, average power consumption and energy per inference. In our experiments, the BB algorithm selects at each step of the DSE the mapping with the lowest energy, and generates new mappings from it. We refer thus in our DSE approach to the energy as the score to optimize. Different quantities and scoring functions can be specified by the user depending on the optimization strategy.

We have left the flow automatically dimension the platform's number of tiles under the limit of $T_{\max} = 7$ tiles and memory sizes based on the needs of each mapping. To compare the results of our flow, we also evaluated a so-called *baseline* implementation of each ANN, in which each layer's clustering is maximized (up to 7 clusters due to the constraint $T_{\max} = 7$) and each cluster is mapped on a different tile. This baseline mapping also always implements power management. Figures 9 (a) to (d) in appendix show the distribution of explored mappings in graphs with the latency in x-axis and the energy in y-axis. Tables 5 to 8 show a summary of the DSE results from Figure 9. In the tables, we use the notion of *rank*, which refers to the ranking of a mapping based on its score (i.e. the energy in our approach) compared to the other mappings. For example the mapping ranked 1 has a score which is superior to all the other mappings (i.e. has the lowest energy cost per iteration in our approach).

For each ANN, we ran the evaluation flow for 4-hours, with a total of 4140 mappings per ANN evaluated on average during that time-frame. This corresponds to an evaluation speed of 17 mappings/min., which is less than the observed 30 mappings/min. (2 s per mapping) in Section 7.2. This is due to over-

Table 5: Results of the DSE process for the FNN1.

Rank	Mapping	⏻	Energy	Latency
1	C_{dense1}=5 C_{dense2}=5 1 2 3 4 5 1 2 3 2 3	✗	1.40 mJ	97 289 cycles
2	C_{dense1}=5 C_{dense2}=5 1 2 3 4 5 1 2 2 3 3	✗	+0.07%	+0.09%
3	C_{dense1}=5 C_{dense2}=5 1 2 3 4 5 1 2 2 4 4	✗	+0.13%	+0.09%
...				
143	C_{dense1}=7 C_{dense2}=5 1 2 3 4 5 6 7 1 2 3 4 5	✓	+5.71%	+9.36%
...				
212	C_{dense1}=7 C_{dense2}=7 1 2 3 4 5 6 7 1 2 3 4 5 6 7	✓	+16.08%	+23.56%

Table 6: Results of the DSE process for the FNN2.

Rank	Mapping	⏻	Energy	Latency
1	C_{dense1}=7 C_{dense2}=3 C_{dense3}=5 1 2 3 4 5 6 7 1 2 3 1 2 3 4 5	✓	3.43 mJ	224 054 cycles
2	C_{dense1}=7 C_{dense2}=3 C_{dense3}=5 1 2 3 4 5 6 7 1 2 3 1 2 3 4 5	✗	+3.78%	+5.33%
3	C_{dense1}=7 C_{dense2}=4 C_{dense3}=5 1 2 3 4 5 6 7 1 2 3 4 1 2 3 4 5	✓	+4.67%	+7.37%
...				
16	C_{dense1}=7 C_{dense2}=6 C_{dense3}=7 1 2 3 4 5 6 7 1 2 3 4 5 6 7 1 2 3 4 5 6 7	✓	+6.61%	+8.76%

heads when using the DSE toolflow caused by the loading and saving of the database containing all mappings description and scores, which is done at every step. Due to the expanding data saved in the database, loading and saving it causes increasing overheads. Additional delays are also due to the comparison of mappings scores done to select the best mapping, the creation of new mappings from the selected one, and the generation of the SystemC experiments descriptions. The DSE experiments are conducted when using caching on the host processor.

Our experiments consistently showed that the top-ranked mapping (rank 1) outperforms others in optimizing both energy and latency. While the selection and optimization process in our DSE flow focuses solely at each step on the mapping with the best score i.e. the lowest energy in our approach, we observe that the latency gets also optimized in the same process. This is due to the energy quantity being intrinsically related to the latency as explained in details in Section 5.1. As illustrated in Table 5, the identified optimized ANN mappings achieve energy savings of up to 16 % and latency reductions of up to 24 % compared to the baseline implementation which is ranked 212. The results also reveal that higher-ranked mappings exhibit non-trivial configurations. They do not necessarily employ the highest clustering for each layer and often feature complex mapping strategies, as observed in the top three mappings of FNN1 shown in Table 5. Interestingly, gains from using power management differs among applications and mappings. For instance, for the FNN1, the highest-ranked mapping using power management is at rank 143, consuming 5.71 % more energy and requiring 9.36 % more time per inference than the 1st-rank, which does not use power management. The opposite is observed for the FNN2. The 1st-ranked mapping uses power management, and the 2nd-ranked mapping is the same one without power management. The 2nd-ranked mapping requires 3.78 % higher energy consumption and 5.33 % longer inference time compared to its counter-part with power management. The information from our flow allow engineers to make informed trade-offs when deciding whether to implement power management, which may require additional resources but offer potential gains. Overall, these results highlight the effectiveness of the proposed flow in optimizing ANN implementations.

Table 7: Results of the DSE process for the FNN3.

Rank	Mapping	⏻	Energy	Latency
1	C_{dense1}=7 C_{dense2}=6 C_{dense3}=7 1 2 3 4 5 6 7 1 2 3 4 5 6 1 2 3 4 5 6 7	✗	2.86 mJ	192 008 cycles
2	C_{dense1}=7 C_{dense2}=6 C_{dense3}=7 1 2 3 4 5 6 7 1 2 3 4 5 6 1 2 3 4 5 6 7	✓	+0.60%	+1.67%
3	C_{dense1}=7 C_{dense2}=5 C_{dense3}=7 1 2 3 4 5 6 7 1 2 3 4 5 1 2 3 4 5 6 7	✗	+0.72%	+0.95%
...				
6	C_{dense1}=7 C_{dense2}=7 C_{dense3}=7 1 2 3 4 5 6 7 1 2 3 4 5 6 7 1 2 3 4 5 6 7	✓	+1.55%	+2.54%

Table 8: Results of the DSE process for the CNN.

Rank	Mapping	⏻	Energy	Latency
1	C_{conv1}=5 * C_{dense1}=7 C_{dense2}=5 1 2 3 4 5 1 1 2 3 4 5 6 7 1 2 3 4 5	✓	41.80 mJ	3 117 941 cycles
2	C_{conv1}=5 * C_{dense1}=7 C_{dense2}=6 1 2 3 4 5 1 1 2 3 4 5 6 7 1 2 3 4 5 6	✓	+0.04%	+0.02%
3	C_{conv1}=5 * C_{dense1}=7 C_{dense2}=7 1 2 3 4 5 1 1 2 3 4 5 6 7 1 2 3 4 5 6 7	✓	+0.16%	+0.16%
...				
6	C_{conv1}=5 * C_{dense1}=6 C_{dense2}=5 1 2 3 4 5 1 1 2 3 4 5 6 1 2 3 4 5	✗	+11.63%	+3.78%

9. Conclusion & prospectives

This article presents a novel energy and timing optimization flow for mapping ANNs onto multi-core platforms. The originality of this work lies in the integration of a modeling approach with simulation and analytical models calibrated through measurements, achieving both high evaluation speed and accuracy. The proposed models were calibrated using a prototype multi-core platform implemented on FPGA and validated against real measurements of 54 ANN mappings. The flow demonstrated an evaluation time of less than 2 s per mapping, with a prediction accuracy reaching 97 % for timing and 93 % for power and energy. These results validate the flow's scalability in regards to many dimensions without the need for re-calibration. The models are scalable in regards to ANN application types and complexity, clustering, mapping strategies with and without power management, high and low communication rates, and platform resources (number of tiles, private memory sizes within 1024 kB threshold.). The evaluation results and DSE demonstration validate the proposed flow as a fast yet accurate solution for evaluating and optimizing ANN mappings on

multi-core platforms.

This flow is particularly well-suited for exploring various hardware/software settings. For instance, it could be used to evaluate open-hardware platforms, such as RISC-V-based, testing different ISA extensions and assessing their contributions and costs in terms of energy and inference time. With the increasing adoption of edge platforms incorporating dedicated AI accelerators, shared caches, and external memories, a notable prospect of this work could be to extend the flow to these more complex architectures. Since the proposed flow does not require training ANNs to evaluate mappings, it could also be integrated with Neural Architecture Search (NAS) [14] to optimize both algorithms under classification accuracy constraints and their implementations under timing and energy constraints.

References

[1] A. Krizhevsky, I. Sutskever, G. E. Hinton, ImageNet Classification with Deep Convolutional Neural Networks, in: Advances in Neural Information Processing Systems, 2012.

[2] W. Shi, J. Cao, Q. Zhang, Y. Li, L. Xu, Edge Computing: Vision and Challenges, IEEE Internet of Things Journal (2016).

[3] T. Garbay, P. Dobias, W. Dron, P. Lusich, I. Khalis, A. Pinna, K. Hachicha, B. Granado, CNN Inference Costs Estimation on Microcontrollers: the EST Primitive-based Model, in: IEEE International Conference on Electronics, Circuits, and Systems (ICECS), 2021.

[4] L. Heim, A. Biri, Z. Qu, L. Thiele, Measuring What Really Matters: Optimizing Neural Networks for TinyML (2021). arXiv:2104.10645, doi:10.3929/ethz-b-000514817.

[5] I. Galanis, I. Anagnostopoulos, C. Nguyen, G. Bares, D. Burkard, Inference and Energy Efficient Design of Deep Neural Networks for Embedded Devices, in: IEEE Computer Society Annual Symposium on VLSI (ISVLSI), 2020.

[6] O. Spantidi, I. Galanis, I. Anagnostopoulos, Frequency-based Power Efficiency Improvement of CNNs on Heterogeneous IoT Computing Systems, in: IEEE 6th World Forum on Internet of Things (WF-IoT), 2020.

[7] Y. Umuroglu, N. J. Fraser, G. Gambardella, M. Blott, P. Leong, M. Jahre, K. Vissers, FINN: A Framework for Fast, Scalable Binarized Neural Network Inference, ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (ISFPGA) (2017).

[8] S. Sombatsiri, J. Yu, M. Hashimoto, Y. Takeuchi, A Design Space Exploration Method of SoC Architecture for CNN-based AI Platform, Workshop on Synthesis And System Integration of Mixed Information technologies (SASIMI) (2019).

[9] Y. H. Chen, T.-J. Yang, J. Emer, V. Sze, Eyeriss v2: A Flexible Accelerator for Emerging Deep Neural Networks on Mobile Devices, IEEE Journal on Emerging and Selected Topics in Circuits and Systems (2018).

[10] R. Machupalli, M. Hossain, M. Mandal, Review of ASIC Accelerators for Deep Neural Network, Microprocessors and Microsystems (2022).

[11] A. Gerstlauer, C. Haubelt, A. D. Pimentel, T. P. Stefanov, D. D. Gajski, J. Teich, Electronic System-Level Synthesis Methodologies, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCADICS) (2009).

[12] S. Rahim, P. P. Nair, R. Devaraj, A Survey of Machine Learning-Driven Task Scheduling Approaches for Multiprocessor Systems, Journal of Systems Architecture (2026).

[13] pSSim4AI open source Git Repository, last accessed: 21.12.2025. URL https://gitlab.univ-nantes.fr/lenours-s/pssim4ai-open

[14] T. Elsken, J. H. Metzen, F. Hutter, Neural Architecture Search: A Survey, Journal of Machine Learning Research (JMLR) (2019).

[15] F. Tsimpourlas, L. Papadopoulos, A. Bartsokas, D. Soudris, A Design Space Exploration Framework for Convolutional Neural Networks Implemented on Edge Devices, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCADICS) (2018).

[16] X. Guo, A. D. Pimentel, T. Stefanov, Automated Exploration and Implementation of Distributed CNN Inference at the Edge, IEEE Internet of Things Journal (2023).

[17] D. Velasco-Montero, J. Fernandez-Berni, R. Carmona-Galan, A. Rodriguez-Vazquez, PreVIous: A Methodology for Prediction of Visual Inference Performance on IoT Devices, IEEE Internet of Things Journal (2020).

[18] M. Motamedi, P. Gysel, V. Akella, S. Ghiasi, Design Space Exploration of FPGA-based Deep Convolutional Neural Networks, in: Asia and South Pacific Design Automation Conference (ASP-DAC), 2016.

[19] S. I. Venieris, C.-S. Bouganis, fpgaConvNet: Mapping Regular and Irregular Convolutional Neural Networks on FPGAs, IEEE Transactions on Neural Networks and Learning Systems (2019).

[20] A. Parashar, P. Raina, Y. S. Shao, Y.-H. Chen, V. A. Ying, A. Mukkara, R. Venkatesan, B. Khailany, S. W. Keckler, J. Emer, Timeloop: A Systematic Approach to DNN Accelerator Evaluation, in: IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), 2019.

[21] L. Ke, X. He, X. Zhang, NNest: Early-Stage Design Space Exploration Tool for Neural Network Inference Accelerators, in: Proceedings of the International Symposium on Low Power Electronics and Design (ISLPED), 2018.

[22] J. Ney, D. Loroch, V. Rybalkin, N. Weber, J. Krüger, N. Wehn, HALF: Holistic Auto Machine Learning for FPGAs, in: International Conference on Field-Programmable Logic and Applications (FPL), 2021.

[23] H. Kwon, P. Chatarasi, V. Sarkar, T. Krishna, M. Pellauer, A. Parashar, MAESTRO: A Data-Centric Approach to Understand Reuse, Performance, and Hardware Cost of DNN Mappings, IEEE Micro (2020).

[24] X. Yi, J. Yu, Z. Wu, X. Xiong, D. Xu, C. Chen, J. Tao, F. Yang, NNASIM: An Efficient Event-Driven Simulator for DNN Accelerators with Accurate Timing and Area Models, in: IEEE International Symposium on Circuits and Systems (ISCAS), 2022.

[25] J. Lee, J. Park, S. Lee, J. Kung, Implication of Optimizing NPU Dataflows on Neural Architecture Search for Mobile Devices, Transactions on Design Automation of Electronic Systems (ACM TODAES) (2022).

[26] Y. C. Lee, T.-S. Hsu, C.-T. Chen, J.-J. Liou, J.-M. Lu, NNSim: A Fast and Accurate SystemC/TLM Simulator for Deep Convolutional Neural Network Accelerators, in: International Symposium on VLSI Design, Automation and Test (VLSI-DAT), 2019.

[27] J. Haris, P. Gibson, J. Cano, N. Bohm Agostini, D. Kaeli, SECDA-TFLite: A Toolkit for Efficient Development of FPGA-based DNN Accelerators for Edge Inference, Journal of Parallel and Distributed Computing (2023).

[28] S. Sombatsiri, Y. Takeuchi, M. Imai, An Efficient Performance Estimation Method for Configurable Multi-Layer Bus-Based SoC, Information and Media Technologies (2015).

[29] E. A. Lee, D. G. Messerschmitt, Synchronous Data Flow, Proceedings of the IEEE (1987).

[30] G. Kahn, The Semantics of a Simple Language for Parallel Programming, Information processing (1974).

[31] F. Rosenblatt, The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain, Psychological review (1958).

[32] Y. Lecun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, L. D. Jackel, Backpropagation Applied to Handwritten Zip Code Recognition, Neural Computation (1989).

[33] L. Mei, P. Houshmand, V. Jain, S. Giraldo, M. Verhelst, ZigZag: Enlarging Joint Architecture-Mapping Design Space Exploration for DNN Accelerators, IEEE Transactions on Computers (2021).

[34] S. Minakova, E. Tang, T. Stefanov, Combining Task- and Data-Level Parallelism for High-Throughput CNN Inference on Embedded CPUs-GPUs MPSoCs, in: Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS), 2020.

[35] D. Luenemann, M. Fakih, K. Gruettner, Capturing Neural-Networks as Synchronous Dataflow Graphs, in: Methods and Description Languages for Modelling and Verification of Circuits and Systems (MBMV), 2020.

[36] Q. Dariol, Early Timing and Energy Prediction and Optimization of Artificial Neural Networks on Multi-Core Platforms, Ph.D. thesis, Nantes Université (2023).
URL https://theses.hal.science/tel-04390337

[37] H.-D. Vu, S. Le Nours, S. Pillement, R. Stemmer, K. Grüttner, A Fast Yet Accurate Message-level Communication Bus Model for Timing Prediction of SDFGs on MPSoC, in: Asia and South Pacific Design Automation Conference (ASP-DAC), 2021.

[38] Six-Core Digital Signal Processor MSC8156 Datasheet, rev. 6, 07.2013. Last accessed: 21.12.2025.
URL https://www.nxp.com/docs/en/data-sheet/MSC8156.pdf

[39] Kalray MPPA® Manycore: A Massively Parallel Processor Array Architecture, last accessed: 21.12.2025.
URL https://www.kalrayinc.com/products/kalray-processors/

[40] Intel Labs - The SCC Platform Overview, rev. 0.7, 24.05.2010. Last accessed: 21.12.2025.
URL https://www.intel.cn/content/dam/www/public/us/en/documents/technology-briefs/intel-labs-single-chip-platform-overview-paper.pdf

[41] Coral Development Board Datasheet, version 1.7, December 2022. Last accessed: 21.12.2025.
URL https://coral.ai/docs/dev-board/datasheet/

[42] J. Rabaey, Low Power Design Essentials, Springer New York, NY, 2009.

[43] R. Stemmer, H.-D. Vu, S. Le Nours, K. Grüttner, S. Pillement, W. Nebel, A Measurement-Based Message-Level Timing Prediction Approach for Data-Dependent SDFGs on Tile-Based Heterogeneous MPSoCs, MDPI Applied Sciences (2021).

[44] Q. Dariol, S. Le Nours, D. Helms, R. Stemmer, S. Pillement, K. Grüttner, Fast Yet Accurate Timing and Power Prediction of Artificial Neural Networks Deployed on Clock-Gated Multi-Core Platforms, in: Rapid Simulation and Performance Evaluation for Design Optimization: Methods and Tools (RAPIDO), 2023.

[45] AMD, UltraScale™ Architecture and Product Data Sheet: Overview (DS890), rev. v4.8, 19.11.2025. Last accessed: 21.12.2025.
URL https://docs.xilinx.com/v/u/en-US/ds890-ultrascale-overview

[46] AMD, MicroBlaze Processor Reference Guide, uG984 (v2022.1) May 25, 2022.

[47] C. Schlaak, M. Fakih, R. Stemmer, Power and Execution Time Measurement Methodology for SDF Applications on FPGA-based MPSoCs, International Workshop on High Performance Energy Efficient Embedded Systems (HIP3ES) (2017).

[48] AMD, Xilinx Integrated Logic Analyzer v2.0 Data Sheet (DS875), rev. 25.06.2012. Last accessed: 21.12.2025.
URL https://docs.amd.com/v/u/en-US/ds875-ila

[49] Rohde & Schwarz, Rohde & Schwarz HMC8012 Digital Multimeter User Manual, version 0.6, 18.03.2020. Last accessed: 21.12.2025.
URL https://scdn.rohde-schwarz.com/ur/pws/dl_downloads/pdm/cl_manuals/user_manual/5800_4505_01/HMC8012_UserManual_de_en_06.pdf

[50] A. K. Sultania, C. Zhang, D. K. Gandhi, F. Zhang, A. K. Sultania, C. Zhang, D. K. Gandhi, F. Zhang, Designing with Xilinx® FPGAs: Using Vivado, Springer International Publishing, 2017, Ch. Power Analysis and Optimization, pp. 177–187.

[51] S. Houben, J. Stallkamp, J. Salmen, M. Schlipsing, C. Igel, Detection of Traffic Signs in Real-World Images: The German Traffic Sign Detection Benchmark, in: International Joint Conference on Neural Networks, 2013.

[52] S. Le Nours, H.-D. Vu, R. Stemmer, S. Pillement, A Hybrid Message-level Modeling Approach for Fast Yet Accurate Simulation of Multiprocessor Shared Bus Effects on Data Flow Applications Execution, Journal of Signal Processing Systems (2026).

[53] A. H. Land, A. G. Doig, An Automatic Method of Solving Discrete Programming Problems, Econometrica (1960).

Quentin Dariol received in 2019 the Master degree of engineering in electronics and digital technologies from the graduate school of engineering of Nantes University (Polytech'Nantes) in France. He then received in 2023 the PhD degree from Nantes University for his research work on modeling and optimizing AI algorithms implementation on edge multi-core platforms under timing and energy constraints.

Sébastien Le Nours is an associate professor at Polytech Nantes since 2004. He is currently a research member of the ASIC Research Team of the IETR Lab. (Research Institute in Electronic and Telecommunication). He received a PhD and Habilitation degrees in Electronic from the INSA Rennes and Nantes University, France, respectively. His research interests include electronic system level design, extra-functional properties evaluation and optimization of embedded systems and related hardware-software architectures.

Sébastien Pillement (IEEE member) received the PhD degree and Habilitation degrees in computer engineering, respectively, from the University of Montpellier II and the University of Rennes 1. Since 2012, he is a full professor at Ecole Polytechnique of Nantes University, France. From 1999 to 2012 he was with IUT in Lannion, the subdivision of the University of Rennes 1, France, and was also a research member of the CAIRN INRIA Research Team of the IRISA Lab. (Research Institute in Computer Science and Random Systems). He is now a member of the IETR laboratory, UMR CNRS 6164. His research interests include dynamically reconfigurable architectures, system on chips, design methodology and Network on Chip (NoC)-based circuits. He focuses his research on designing flexible and efficient architectures managed in real time. He is the author or coauthor of about 120 journal and conference papers.

Ralf Stemmer did his PhD in simulation based execution time analysis of multi-processor systems using measured delay distributions at the University of Oldenburg. He then worked for the German Aerospace Center (DLR) on runtime monitoring of complex scenario-based requirements for autonomous driving functions and modular software architectures for software-defined vehicles.

Domenik Helms studied theoretical physics till 2001 and graduated in Microelectronics in 2009 (Leakage Models for High Level Power Estimation). Till 2018, he led a research group on low power electronics at OFFIS research institute. Currently, his research group at the German Aerospace Center (DLR) develops techniques for hardware aware optimization of embedded AI systems.

Kim Grüttner studied computer science (Diploma 2005, PhD 2015) and currently is head of the department of "System Evolution and Operation" within the German Aerospace Center (DLR) – Institute of Systems Engineering for Future Mobility. Before joining DLR he was working at OFFIS – Institute for Information Technology in various positions as group leader "Hardware-/Software Design Methodology", leader of the competence center "Embedded Design Automation" and finally as co-director of the R&D-Division Transportation. His background is in technical computer science and embedded systems with a focus on systems engineering for safety relevant transport systems.

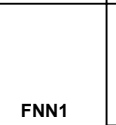
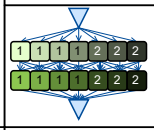
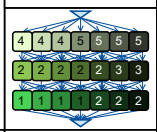
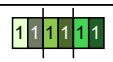
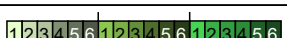
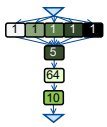
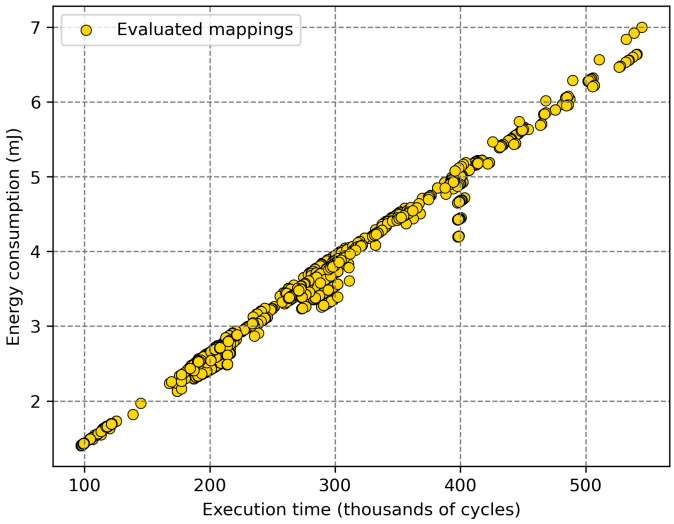
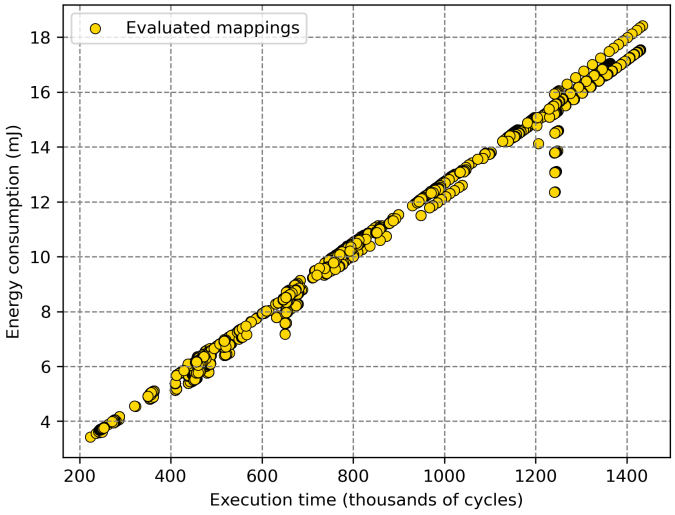
Application	Clustering	Mapping		Comm. rate	Timing error	Power error	Energy error
FNN1			✗	5%	0.21%	-0.49%	-0.27%
			✓	5%	0.37%	-0.53%	-0.16%
			✗	51%	0.21%	-3.11%	-2.92%
			✓	52%	0.34%	-2.22%	-1.88%
			✗	13%	0.29%	-0.88%	-0.59%
			✓	14%	0.56%	-1.64%	-1.09%
			✗	26%	-0.03%	-2.02%	-2.05%
			✓	28%	1.43%	-1.42%	-0.01%
			✗	70%	-2.85%	-3.13%	-5.89%
			✓	68%	0.82%	-4.88%	-4.10%
			✗	25%	0.61%	-0.72%	-0.11%
			✓	31%	+0.95%	-2.17%	-1.24%
			✗	56%	0.74%	-2.72%	-2.00%
			✓	55%	-2.21%	-4.74%	-6.85%
FNN2			✗	2%	+0.08%	-0.41%	-0.33%
			✓	2%	+0.13%	-1.84%	-1.71%
			✗	66%	+0.19%	-4.74%	-4.56%
			✓	66%	+0.39%	-3.65%	-3.28%
			✗	5%	-0.15%	-0.96%	-1.11%
			✓	6%	-0.06%	-1.97%	-2.03%
			✗	8%	-0.25%	-0.93%	-1.17%
			✓	9%	+0.38%	-1.62%	-1.25%
			✗	60%	-0.60%	-4.23%	-4.81%
			✓	59%	+0.35%	-4.94%	-4.61%
			✗	11%	-0.33%	-0.57%	-0.90%
			✓	15%	-0.21%	-2.46%	-2.66%
			✗	28%	+0.04%	-1.85%	-1.81%
			✓	28%	-1.13%	-4.22%	-5.30%
FNN3			✗	3%	-0.32%	-1.35%	-1.66%
			✓	4%	-0.23%	-2.50%	-2.72%
			✗	4%	-0.49%	-1.15%	-1.64%
			✓	4%	-0.32%	-2.37%	-2.69%
			✗	70%	+0.99%	-4.87%	-3.93%
			✓	69%	+0.61%	-6.79%	-6.22%
			✗	7%	-0.54%	-0.80%	-1.34%
			✓	9%	-0.40%	-2.03%	-2.43%
			✗	14%	-0.28%	-1.32%	-1.60%
			✓	14%	-1.11%	-2.32%	-3.40%
			✗	10%	-0.75%	-0.49%	-1.23%
			✓	14%	-0.58%	-2.38%	-2.95%
			✗	20%	+0.42%	-2.02%	-1.61%
			✓	24%	+1.64%	-5.70%	-4.15%
CNN1			✗	2%	-0.17%	-4.07%	-4.23%
			✓	2%	-0.17%	-4.54%	-4.70%
			✗	33%	+0.13%	-4.09%	-3.97%
			✓	33%	+0.13%	+1.35%	+1.48%
			✗	66%	-0.16%	-7.03%	-7.18%
			✓	66%	+0.33%	+0.50%	+0.83%
			✗	2%	+0.43%	-3.84%	-3.43%
			✓	2%	+0.44%	-1.52%	-1.09%
			✗	52%	-0.40%	-4.98%	-5.36%
			✓	52%	-0.03%	-4.94%	-4.97%
			✗	37%	-1.45%	-5.84%	-7.21%
			✓	37%	-1.20%	-4.87%	-6.01%

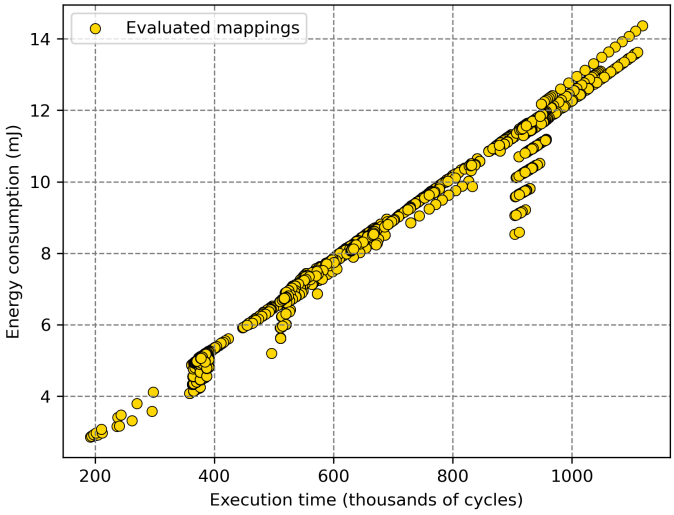
Figure 8: Detailed evaluation results of our flow.



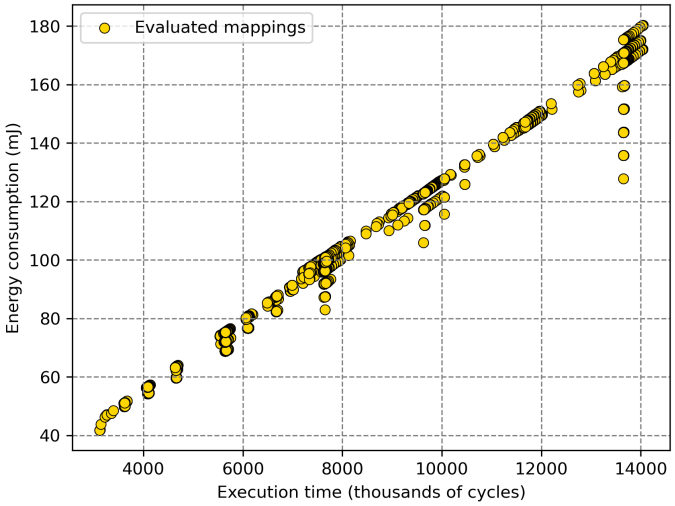
(a) FNN1



(b) FNN2



(c) FNN3



(d) CNN

Figure 9: Pareto plots of the energy by the latency, obtained during the DSE process for the tested ANNs.