Performance Modeling and Estimation of a Configurable Output Stationary Neural Network Accelerator

Ali Oudrhiri*†, Emilien Taly*‡, Nathan Bain*‡, Alix Munier†, Roberto Guizzetti*, Pascal Urard*
*STMicroelectronics, Crolles, France
†Sorbonne Université, CNRS, LIP6, F-75005 Paris, France
‡Univ. Grenoble Alpes, CNRS, Grenoble INP, TIMA, Grenoble, France

***Abstract*—Neural network accelerators are designed to process Neural Networks (NN) optimizing three Key Performance Indicators (KPIs): latency, power, and chip area. This work is based on the study of Gemini, an industrial prototype near memory computing inference accelerator designed using a high-level synthesis technique. Gemini is an output stationary configurable accelerator that achieves its performance based on two structural parameters. The measurement of the KPIs requires simulations that are time-consuming and resource-intensive.**

This paper presents a high-level practical estimator that can instantly predict the KPIs depending on the NN and the Gemini configuration. The latency is accurately derived using an analytical model based on the architecture, the operators scheduling and the NN characteristics. The power and the chip area are computed analytically and the models are calibrated using simulations. Finally, we show how to use the estimator to derive Pareto optima for choosing the best Gemini configurations for a VGG-like NN.

***Index Terms*—Neural network accelerator, output stationary, estimation, latency, power, area.**

I. INTRODUCTION

Deep Neural Networks (NN) have become incredibly popular [1]. We can find NN-based solutions in every field, which led it to become a field on its own. The principle behind NNs is far from being new. However, they are recently thriving due to hardware progress [2]. NNs require a tremendous computational complexity which was not available a few decades ago. CPUs, FPGAs and especially GPUs participated in the recent resurgence of NNs as they overcame this computational need [1]. However, NN Application-Specific Integrated Circuits (ASIC) accelerators become the NN hardware best candidates. These chips dedicated to NN processing are especially advantageous for inference. They can further enhance latency while having a small area and low power consumption. NNs present various structures and have different hardware requirements [2]: some applications need very low latency chips, such as cloud computing, while others require low power and small area, such as the edge computing market. Then, for each application, the designer has to always find a compromise between the 3 KPIs: latency, chip area and, power.

Gemini is designed as an industrial Near Memory Computing (NMC) solution to meet this challenge for the inference (all the NN weights are precalculated). It supports feed forward NNs (convolution layers, depthwise, pooling, and Fully Connected (FC)). The architecture of Gemini was primarily designed to be streamlined and highly configurable, facilitating effortless adaptation to various applications. Therefore, achieving pure performance on a specific NN was not the objective. Gemini is a configurable output stationary NN accelerator [3] with mainly two structural architectural parameters. Chip area, latency, and power consumption depend on both the NN to be used and the two architectural parameters. The configurability of Gemini allows it to adapt to the NN structure.

Choosing the best configuration according to the NN for Gemini is too time-consuming. There are around 1000 possible Gemini configurations. For a fixed NN, measuring the KPIs requires simulating the NN execution. It cannot be done for all the configurations in a reasonable time. However, using accurate KPIs models, one could rapidly estimate all the possible configurations for a fixed NN. Thus, the challenge lays in obtaining the best KPIs estimation depending on the NN and the two structural parameters. In this article, we consider only the scenario where the entire NN can fit into on-chip RAM. Consequently, only the accelerator's performance will be investigated. Considerations regarding off-chip communications are not taken into account since they are not influenced by the choice of the Gemini configuration.

A. Related works

There is a large research community working on NNs accelerators. Several surveys list the trends and the performances of state-of-the-art accelerators [3]–[5]. The accelerators' KPIs are usually directly measured on the system for specific NNs without any need for high-level KPIs estimations. Another important research area is the one dealing with the design space exploration of accelerators (generally using FPGAs) [6]–[10]. Their objective is to find the best architectural parameters according to KPIs. They use KPIs models and optimizing algorithms to find the best design solutions.

Most of the authors evaluate the latency using analytical formulas based on operations scheduling, accelerator architecture, and NN parameters [11], [12]. For example, Erdem et al. [12] evaluate the latency of the computation according to channel and kernel parallelizations.

For consumption, the strategies are often based on the power estimation of components [8], [13], [14]: for example, Wu

et al. [14] developed Accelergy, a tool that evaluates the energy of different architectures accelerators. Firstly, a designer describes the architecture with compound components characterized by primitives components for which the power is known; RAMs power is evaluated with CACTI [15] and other primitives such as Multiplications And Accumulations (MACs) are given by libraries. Secondly, the designer lists the actions of each component and their use rate. Accelergy estimates the total energy by combining all these data. Zhao et al. [8] also evaluate the consumption by listing the accelerator components but with more simplified energy models. They use also CACTI for RAMs power estimation and they consider registers, MACs, and communication networks for the other components.

Concerning the chip area, Shahshahani et al. [16] rely on machine learning models to predict it. The main drawback of this method is its lack of interpretability. For instance, the impact of each resource is difficult to estimate. Wu et al. [17] and Tang et al. [13] simply consider the area contribution of each component to evaluate the chip area.

B. Contributions

This paper presents a method to estimate KPIs of an NN output stationary accelerator based on its configuration and NN parameters. The study aims to provide insights into the performance metrics without optimizing the architecture.

The proposed estimation methodology can be utilized by anyone using output stationary accelerator architectures. This is due to the fact that the estimation methods rely on principles inherent to this type of architectures, which are universal across all accelerators classified as such.

In Gemini, the latency is estimated analytically depending on the architecture, the operators scheduling, and the NN parameters. This estimation comes from the predictability and the regularity of the operations schedule.

In this paper, we choose to model the power rather than the energy. The energy is impacted by the power of the system as well as its latency. Considering then the energy is less efficient when dealing with trade-offs between consumption and speed (the energy combines both of them). The power will be split into leakage and dynamic power. The leakage is the power dissipated when the device is powered up but the gates are not toggling; it does not depend on the inputs. The dynamic one is the power dissipated when the gates switch their states; it depends on the inputs. Splitting the power allows us to estimate the power as a function of the clock frequency because the dynamic power scales linearly with the clock frequency while the leakage remains constant [18]. This statement holds true since we are operating with a constant voltage VDD. Below the maximum frequency (chosen during the synthesis), there is no requirement to adjust VDD in order to achieve the desired frequencies.

The power consumption of Gemini cannot be measured using tools such as Accelergy [14]. Indeed, the computing part of Gemini is designed using High-Level Synthesis (HLS); the number, the type and the use rate of components are then difficult to predict because operators schedule and optimizations (such as resource sharing) depend on the configurations. However, we assume that main compound operators such as registers, MACs or multiplexers must be synthesized during the HLS. A power model for Gemini is then exhibited based on a linear equation of the complexity of main operators and calibrated through simulations of NNs executions. An advantageous characteristic of this model resides in its inherent simplicity, as it necessitates a minimal quantity of data regarding the architecture and the NN (accessible from a high level of abstraction) for its effective utilization. Conversely, this model also possesses the advantage of being explainable. The power estimator is based on gate-level simulations, which is sufficient to have accurate power values to compare several configurations.

Finally, the chip area will also be modeled with the area contribution of main operators multiplied by constants.

To evaluate the estimator accuracy, it is chosen to consider the Root Mean Square Error (RMSE). It has the advantage to be homogenous to the modeled parameter. The estimated RMSE for area and leakage is 0.005 mm² and 0.57 µW, respectively. The latency and power models of an NN are derived from the models of its individual constituent layers, encompassing all potential parameters. Therefore, these models are validated and universally applicable to any feed-forward NN. Latency is generally estimated with an error of less than 10 cycles, and dynamic power with a RMSE of less than 20 µW. We illustrate our results on a VGG-like NN, as presented in Fig.1, which is inspired by VGG-16 [19]. This network offers the advantage of encompassing diverse NN layer types and, notably, is extensively employed for evaluating NN accelerator performance [8], [20].

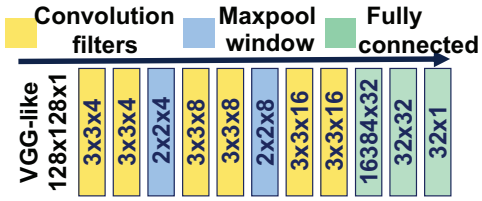

Fig. 1: VGG-like network structure

This paper is organized as follows. Section II presents the configurable architecture of Gemini. Section III exposes the simulation environment used to gather data used in determining KPIs model. Section IV details the estimation model of the KPIs as well as its accuracy. Section V illustrates how the configuration can be chosen once the performances have been estimated. Finally, the conclusion is made in section VI.

II. Gemini configurable Architecture

In this section, we start by presenting the two structural parameters of Gemini, then the architecture of the accelerator will be described in a bottom-up fashion from the processing elements to the whole accelerator.

A. Presentation of Gemini structural parameters

Gemini is composed of a Tensor Processing Unit (TPU) and two SRAM modules: the features maps (*fmaps*) RAM and the *weights* RAM. The TPU contains the block in charge of calculations called the processing elements (PEs) array. There are NPE parallel PEs organized in 2D $(WPAR, MPAR)$ with $NPE = WPAR \times MPAR$. These two structural parameters are configurable before the logic synthesis. They size all the designs from the PEs array to top-level RAMs and they fix the scheduling of the operations. They were introduced to optimize the convolutions processing. $WPAR$ stands for width parallelization of the output feature map (*ofmap*) and $MPAR$ is the filter parallelization (since the number of filters is usually called M in literature [21]). Fig.2 illustrates the notations.

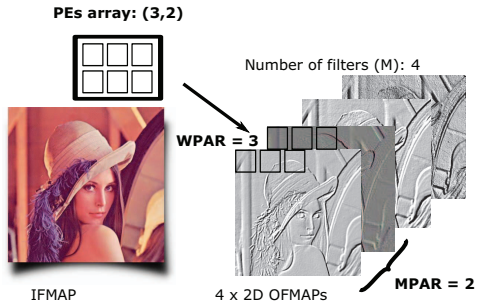

Fig. 2: NPE organization for convolution

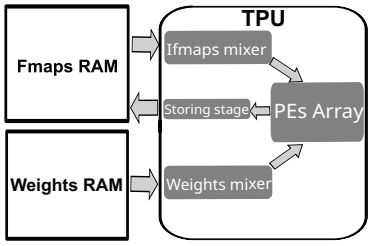

Fig. 3: Accelerator architecture

B. TPU architecture

The TPU architecture is composed of the PEs array, the input mixers and the storing stage. Fig.3 shows the TPU blocks in the top-level architecture.

The processing elements array is composed of NPE PEs. Concerning the architecture of one single processing element, each PE has two pipelined stages: the first one is the output computation stage performing MACs with *fmaps* pixels and *weights*. The second one is the quantization stage. It is triggered when *ofmaps* pixels are calculated (when all the MACs are performed). Its role is to put the *ofmaps* pixels in the desired range of quantization by multiplying them by a scaling factor and taking only *fmap* pixels bits from the most significant bits. The quantization stage always takes 5 clock cycles.

PEs are organized following an output stationary dataflow, each processor then computes an *ofmap* pixel. The partial sum contributing to the output is stored in the accumulator (register) of each PE when new *fmaps* and *weights* are broadcasted to the PEs every cycle. This paradigm is described in [3] and used by several accelerators such as [22] or [23].

The mixers are combinational blocks that take as input a disordered dataset and output the sorted data. The input mixers are the *ifmaps* mixer and the *weights* one.

Finally, the storing stage is located at the output of PEs array. It eliminates some useless PEs computations that should not be written in the *fmaps* RAM. For convolutions, the PEs array calculates *ofmaps* pixels corresponding to horizontal *padding* and *strides* even if they are not necessary for the *ofmap* (they are eliminated by the storing stage). A design choice allowing a few useless operations done by PEs was made to simplify the mixers and to optimize the power and area.

The storing stage also writes the quantized outputs inside the *fmaps* RAM in the correct order. This function is mapped by a mixer. This stage is pipelined with the PEs.

Ultimately, abstracting from Gemini-specific characteristics, the architecture remains consistent across any output stationary accelerator. In all such accelerators, the PEs compute parallelized outputs, necessitating mixers and a storing stage to facilitate data transfer between RAMs and PEs for read and write operations.

C. Layers execution scheduling

The TPU executes the NN layer by layer. It supports convolutions and FC layers with different operations scheduling.

For convolutions, for each *ofmap* in 2D, $WPAR$ pixels are calculated simultaneously. This paradigm is duplicated in $MPAR$ to process 2D *ofmaps* simultaneously (as a reminder M is the number of filters). This parallelization is illustrated in Fig.2 where $WPAR$ and $MPAR$ are respectively equal to 3 and 2.

At every cycle, one filter *weight* is selected by the *weights* mixer and broadcasted to all PEs. It is then multiplied by NPE *fmaps* selected by the *ifmap* mixer (each PE receives a different *fmap* pixel) and accumulated inside the PE register. When all filter MACs are done, the NPE outputs are quantized and then processed by the storing stage.

For FC layers, NPE from N_{out} output neurons are processed simultaneously. N_{out} are the FC *ofmaps* flattened. At every cycle, one input neuron from N_{in} (N_{in} are the FC *ifmaps* flattened) is broadcasted to the PEs by the *ifmap* mixer and multiplied with NPE *weights* (chosen by the *weights* mixer) and accumulated inside the PE accumulator every cycle. Then, as it is done for convolution, the NPE quantized outputs go into the storing stage that writes them inside the *fmap* RAM.

Depthwise layers have the same scheduling as convolutions. Pooling layers follow also the same scheduling replacing the *weight* multiplication by the pooling operation.

D. Top-level architecture

The *weights* RAM contains the NN *weights* as well as information on the network (such as layer types) in a compacted way. The *fmaps* RAM contains the feature map pixels: before the beginning of the executions, this RAM contains *ifmaps* pixels; during the executions, intermediary *fmaps* are also stored inside this RAM overwriting non-meaningful data, and finally, at the end of the execution, it contains the final *ofmaps*.

Both *fmaps* and *weights* SRAMs are sized according to the NN to be supported and according to $(WPAR, MPAR)$. The *fmaps* RAM is composed of $WPAR$ memory banks of $MPAR \times fmapbits$ width for each bank. The *weights* RAM is composed of one memory bank of $NPE \times weighstbits$. The number of bits used for the *fmaps* and *weights* pixels are respectively $fmapbits$ and $weightsbits$. As a reminder, in the scope of this study, it is required that the entire NN fits into the on-chip RAMs, including weights and intermediate ifmaps.

III. Simulation environment

The objective is to gather data on KPIs (latency, area, and power) via simulations for several NNs and $(WPAR, MPAR)$ couples. These data will be used to build an estimator based on analytical models predicting those indicators. For our utilization, both $WPAR$ and $MPAR$ vary from 2 to 32 (NPE could then vary from 4 to 1024). Each $(WPAR, MPAR)$ couple is called a configuration. As aforementioned, it was chosen to perform simulations at the gate level stage directly before the P&R.

We choose to work with 8 bits for both $fmapbits$ and $weightsbits$ since it is the most used quantization mode.

The SRAM capacity is fixed, only the aspect ratio between RAMs width and depth undergoes variation across different configurations.

The technology chosen is CMOSC40. The simulation environment is summarized in Fig.4.

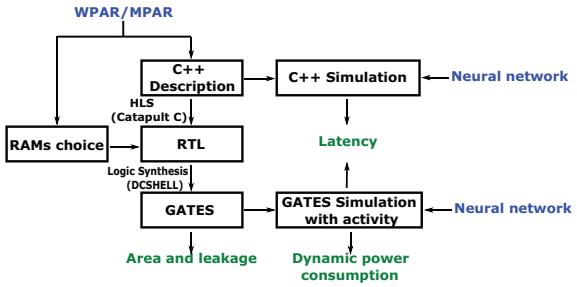

Fig. 4: Simulation environment

The design of the TPU is described in C++ and the HLS is performed by *SIEMENS CATAPULT®*. The TPU execution is described using loops ensuring that one loop execution in C++ corresponds to one clock cycle. Then the latency of a NN processing can already be measured at this level by counting the number of loops.

Once the TPU RTL is obtained for each configuration, the top level of the accelerator is built instantiating the TPU and its corresponding RAMs. If several cuts are possible for one RAM, we choose the option giving the smallest area. Once the full RTL is ready (including TPU and RAMs), we operate the logic synthesis using the *SYNOPSYS DCSHELL®* tool with the same constraints and corners for all configurations: the synthesis is done at 200 MHz, 1 V, and 125 °C for the slowest corner. This corner represents the worst case in terms of timing. The libraries used are LVT (low threshold voltage) and RVT (regular threshold voltage) in CMOSC40. Finally, when the gate netlist is ready, area and leakage power estimations are given by *DCSHELL®* without any simulation as they do not depend on the NN to be computed.

Latency and dynamic power estimations can be obtained by doing gate-level simulations: they are performed using *CADENCE XCELIUM®* environment. The simulations are run at 1 MHz for the typical corner at 1.1 V and 25°C. The toggle rate is then exploited by *SYNOPSYS PRIMEPOWER®* to evaluate the average dynamic power on the whole NN execution.

For gate-level simulations, we choose wisely which NN must be run to extrapolate the result of simulations into other NNs. For that, we run simulations on single-layer NNs varying all the possible parameters to cover all cases. The performance of any multi-layer NN is then obtained from the information of single-layer ones. For FC layers, N_{in} and N_{out} are varying. For convolutions, we vary the number of filters, filter sizes, 2D *ifmap* sizes, *strides*, and *padding*. For depthwise and pooling layers, different *ifmap* sizes, *strides* and, *padding* are chosen. A total of 93 single-layer NNs are considered. Concerning the sweep of $(WPAR, MPAR)$, we limit the simulations to 210 different configurations.

To offer a time estimate for assembling all this data, on average, the combined duration of HLS and logic synthesis is approximately one hour, and NN execution within an optimized simulation environment takes around 3 minutes. Data collection is a one-time process, necessitating repetition solely if the environment undergoes changes. For instance, opting for a different technology or quantization would mandate re-performing this step.

IV. Key performance indicators estimation

The objective of this section is to estimate the performances of the accelerator according to its configurations for each NN. Those estimations are done thanks to an analytical model based on simulations discussed in section III. This model gives latency, area, leakage and, dynamic power.

The modeling of each KPI will be detailed.

A. Latency modeling

Latency in cycles is obtained at the C++ description level. As the design is fully pipelined, the difference between the number of cycles given by the C++ execution and the one obtained after gates simulations corresponds only to the ramp-up of the pipeline. This was observed for several NNs. As the NN's layers are processed serially and separately, the latency of the neural network execution corresponds to the sum of

the layers' latencies added to a constant overhead independent from the NN (it includes the pipeline ramp-up). For this work, only the meaningful terms will be detailed. For example, the bias cycles will be neglected.

The following paragraphs detail the latency modeling of each layer type and the latency behavior of a NN of L layers.

1) Convolution latency

The execution of the convolution is fully predictive. It can be computed based on the output stationary paradigm where $WPAR$ (among 2D *ofmap* pixels) pixels of $MPAR$ filters (among M filters) are processed simultaneously. The number of 2D *ofmap* pixels calculated corresponds to the size of the 2D *ifmap* excluding the vertical *padding* pad_v. This is a consequence of the execution duration being unaffected by the *stride* and the horizontal *padding* usage. Only the vertical *padding* is impacting the number of pixels calculated. So the number of 2D *ofmap* image pixels is $W(H-(R-1)pad_v)$, with W and H the width and height of the *ifmap* and R the height of the filter. Thus the number of cycles N_{cycconv} needed to compute a convolution is:

$$N_{\text{cycconv}} = \left\lceil \frac{W(H-(R-1)pad_v)}{WPAR} \right\rceil \left\lceil \frac{M}{MPAR} \right\rceil \times K_c \quad (1)$$

where K_c is the number of cycles required for one-pixel computation. As stated in section II, 3 stages are pipelined for the computation of one pixel; the latency of the full system is then approximately the latency of the slowest stage. The slowest one is the output computation stage of the PEs array. Every cycle, one filter *weight* is read, so the number of cycles needed to compute one pixel is $K_c = S.R.C$ where S, R, and C are respectively the filter width, height and, channels. The latency of maxpool and depthwise layers are derived from the same formula.

2) Fully connected latency

Concerning FC layers, $Nout$ output neurons are processed simultaneously by NPE processors. It takes Nin cycles to process them; Nin is the number of input neurons. So the latency N_{cycfc} of a FC is:

$$N_{\text{cycfc}} = \left\lceil \frac{Nout}{NPE} \right\rceil Nin$$

3) Estimator validation

Combining the last equations, the general shape of the latency Lat of a NN of $L+K$ layers follows Eq.2:

$$Lat = \sum_{l=1}^{L} \left(\left\lceil \frac{\alpha_l}{MPAR} \right\rceil \times \left\lceil \frac{\beta_l}{WPAR} \right\rceil \gamma_l \right) + \sum_{l=1}^{K} \left\lceil \frac{\delta_l}{NPE} \right\rceil \epsilon_l \quad (2)$$

with L the number of convolution layers, K the number of fully connected layers and α_l, β_l, γ_l, δ_l, ϵ_l are constants depending on the layer l type.

We deduce from Eq 2 that the latency is a decreasing curve with NPE.

Given the predictive nature of the execution, there are only few clock cycles difference (that can be calibrated) between predictions and simulations across all conceivable layer types and their associated parameters. Consequently, this characteristic extends to multi-layer NNs as well. We illustrate this result using the VGG-like network for $MPAR = 8$ shown in Figure 5. This choice is made for the sake of clarity in the curves, even though the observation remains consistent when varying $MPAR$ and $WPAR$. The estimation and simulation curves are nearly indistinguishable.

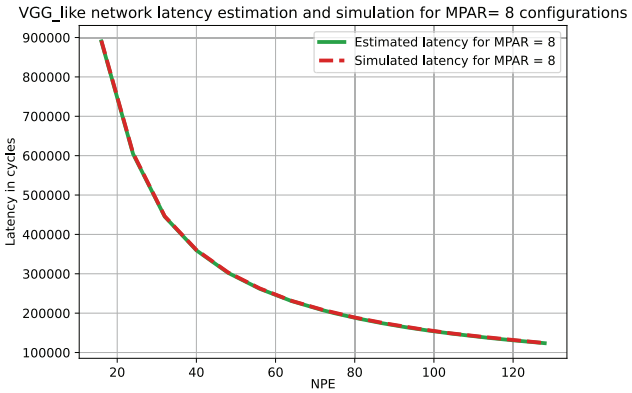

Fig. 5: VGG-like estimated and simulated latencies.

B. Area and leakage modeling

In this paragraph, we will discuss the area and leakage model for RAMs and TPU as well as the calibration of the model by the identification of constants.

1) RAM modules area and leakage

RAM modules leakage and area are dependent on the memory capacity chosen (total number of KBs). Even if the organization of RAMs ($width \times depth$) changes with configurations, the difference of leakage and area is only impacted by RAMs technology variation. Then RAM modules area and leakage will be considered as constants and only the TPU will be considered.

2) TPUs' area and leakage modeling

a) TPU main operators' complexities

Estimating the TPU power and area knowing only $(WPAR, MPAR)$ is challenging. The RTL is obtained by HLS, so the tool can adapt the number and types of operators and their scheduling to optimize the synthesis performance for each configuration; HLS can then generate different netlists for two close but different configurations.

It was decided to model the leakage and area with a linear combination of the expected main operators' complexities and then identify the constants thanks to a linear regression (which are different for area and leakage). These positive constants (c_0, c_1, c_2 and c_3) encapsulate the consumption of primitives operators. The compound operators taken into account are:

- Operators that do not depend on the configuration: the term c_0 corresponds to all the constant operators. As a matter of example, there are all the registers and logic units of finite state machines.
- PEs array input registers (*fmaps* and *weights* registers, accumulators) and arithmetic logic units (MACs): all

these components scale with NPE. They will then be modeled with $c_1 \times NPE$.

- Mixers: there are 3 mixers in the circuit (*ifmap*, *weights* and storing stage mixers) ensuring that the data is well sorted at the input and output of PEs array and RAMs. These mixers are mapped into shifters implemented with multiplexers that have a complexity of $N\lceil \log_2 K \rceil$ with N is the total number of data sorted and K is the number of possible shifts for each data. Mixers are then modeled by: $c_2 \times NPE\lceil \log_2 WPAR \rceil$.
  Their power and area cannot be neglected especially for a large NPE.
- Storing stage operators: they eliminate the non-useful pixels (due to *stride* or *padding*). They scale with $WPAR$. They are modeled with $c_3 \times WPAR$.

Area and leakage follow Eq.3 with different constants:

$$c_0 + c_1 \times NPE + c_2 \times NPE\lceil \log_2 WPAR \rceil + c_3 \times WPAR \quad (3)$$

This modeling method can be adapted to any output stationary accelerator as the compound operators (except the storing stage that was optimized for Gemini) are consistently necessary: MACs and registers for computation and storage are always synthesized along with mixers used for data transfer from RAMs to PEs. Furthermore, the FSM operators are also present in any design. If a user has a customized accelerator with additional significant operators, their complexity can be included in Equation 3 by adding the complexity of the operators multiplied by a constant factor, which can also be determined during regression.

b) Identification of constants and validation of area and leakage models

The constants c_0, c_1, c_2 and c_3 are identified using the dataset of 210 configurations. They are determined quasi-instantly by linear regression optimizing the $RMSE$ and the correlation coefficient (R^2). Figure 6 illustrates the modeling results for

	Leakage	Area
Root mean square error (RMSE)	0.57 µW	0.005 mm²
Average	9.4 µW	0.11 mm²
Correlation coefficient (R^2)	0.98	0.99

TABLE I: Leakage and area estimation characteristics

$MPAR = 5$ configurations (chosen for clarity in the curves). It is observed that the significant increases in leakage and area, such as those observed at 40 and 80 $NPEs$, are accurately captured by the modeling: they correspond to an increase of the value of $\lceil \log_2 WPAR \rceil$ ($WPAR$ is a power of 2). Table I displays the modeling results of all the 210 Gemini configurations. The low RMSE validates the accuracy of the estimation. The R^2 close to 1 confirms that our modeling with Eq.3 is meaningful. The same approach can be applied in case of changes in the number of bits ($fmapbits$ or $weightsbits$) or the process technology. Only the regression step needs to be rerun, using the updated simulation data.

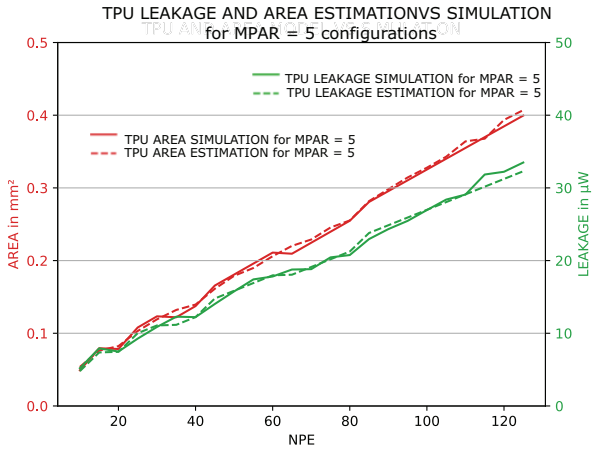

Fig. 6: Area and leakage estimations and simulations.

C. Dynamic power modeling

This paragraph describes how the dynamic power of the execution of a NN on Gemini can be evaluated for each architecture configuration. The dynamic power is calculated by summing the internal power (consumption due to the power dissipation of the capacitance inside a standard cell) and the switching one (dissipation of load capacitance) [24].
First, the RAMs dynamic power will be discussed, then the TPU dynamic power for each layer type will be detailed. The dynamic power of any NN can then be estimated by combining the power consumption of its layers.

1) RAM modules' dynamic power

For all NNs tested, the dynamic power of both RAMs (*fmaps* and *weights*) remains almost constant while sweeping $(WPAR, MPAR)$. It is since for each RAM, the number of KB is fixed for all configurations, thus the total amount of data read is the same; only the RAM modules widths and depths are changing affecting the number of read cycles and the size of the buffer to be read: for example, several reading cycles are needed when the RAM modules width is small while only a few of them are needed to read the same memory amount when the width is large.
Fig.7 shows the dynamic power of RAMs and TPU for the VGG-like NN for the configurations $MPAR$ = 8 ($WPAR$ is swept from 2 to 16). For this example, the SRAMs capacity is 1.3 MB. For our study, we neglect the impact of RAMs as they do not impact the configuration choice.

2) TPU dynamic power modeling

For estimating the dynamic power consumption of the TPU, we employed the same compound operators discussed in Section IV B. However, in this case, the constants c_i in Equation 3 should be functions dependent on the specific NN parameters. It's important to highlight that individual models for each NN were not constructed by running separate regressions. Instead, a dynamic power single model, valid for all NNs, was developed through regression performed only

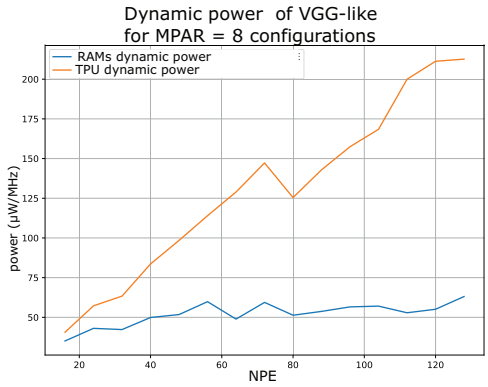

Fig. 7: Power consumption of TPU and RAMs on VGG-like

once. This model incorporates data from all simulated NNs and is universally applicable across all NNs.

The first statement is that the dynamic power of the TPU is globally increasing with NPE for all the NNs tested (as shown in Fig.7). Furthermore there are some local optimums reached for some $(WPAR, MPAR)$. They are the same for all NNs tested but they change according to the architecture. It means that they depend only on the architecture and not on the NN. However, the power magnitude of those optimums depends on the NN.

The identification of c_i is different between convolutions and fully connected layers as they have different parameters.

a) Convolution dynamic power

There are 5 different parameters characterizing convolutions: *ifmap* 2D dimensions ($W \times H$), filter dimensions ($S.R.C$), number of filters (M), *strides* and, *padding*.

First of all, as was specified in II B, all the blocks except the storing stage work, in the same way, considering different *strides* or *paddings*. Then a low dynamic power dependency on those parameters is expected. For the 210 configurations of $(WPAR, MPAR)$, we choose 6 different parameters NNs. We run them with and without the *padding* to evaluate their impact on the dynamic power. The RMSE is 2.94 µW for an average of 157.2 µW.

Concerning the *stride*, we took one NN with a *stride* of 1x1 and another with 2x2. All the other parameters are the same. The RMSE is 0.72 µW for an average of 155.6 µW.

Due to this low RMSE (compared to the average), it was then decided to neglect the impact of *stride* and *padding* on dynamic power consumption.

The number of filters M should also be neglected. Actually, the mixers and PEs array are duplicated in $MPAR$; it means that the execution is the same considering any number of filters between 1 and $MPAR$; and when $M > MPAR$ several same executions are operated which is not affecting the average dynamic power. Considering 3 different NNs, fixing all the parameters except M (respectively set to 7, 14, 24), the RMSE is: 6.04 µW for an average of 152.5 µW. This parameter can also be neglected because of this low RMSE value.

The dynamic power is increasing with the 2D *ifmap* pixels number ($W \times H$) before reaching a saturation level where the dynamic power consumption is almost the same for all *ifmap* 2D pixels number.

As it was specified before, $WPAR$ 2D pixels from the 2D *ofmap* are processed simultaneously. Considering the *padding* to simplify calculations, the number N_{exec} of executions to calculate all the 2D *ofmap* pixels is the following:

$$N_{exec} = \left\lceil \frac{W \times H}{WPAR} \right\rceil = \left\lfloor \frac{W \times H}{WPAR} \right\rfloor + r + o. \qquad (4)$$

with r equal to 0 when $\frac{W \times H}{WPAR}$ is an integer and 1 otherwise. o corresponds to the few overhead cycles. They do not consume a significant amount of power. The term $\left\lfloor \frac{W \times H}{WPAR} \right\rfloor$ corresponds to executions where 100% of $WPAR$ are working and r to the execution where only a few of them are used (because there are less than $WPAR$ pixels to calculate). Thus, when $W \times H$ is large (large 2D *ifmap*), N_{exec} is quasi equal to $\left\lfloor \frac{W \times H}{WPAR} \right\rfloor$. As the dynamic power is measured with an average on all the convolution processing, the dynamic power will then correspond to the power of the executions where all $WPAR$ are working (because it is repeated $\left\lfloor \frac{W \times H}{WPAR} \right\rfloor$ times). It explains the power saturation when the *ifmap* is large enough (number of pixels higher than 80). Fig.8 shows the dynamic

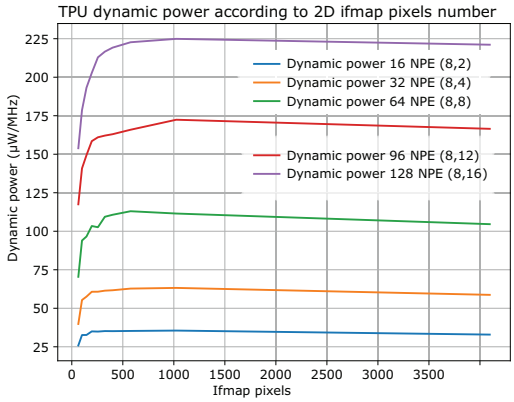

Fig. 8: Power consumption of TPU sweeping *2D ifmap* pixels

power of the TPU sweeping the number of the *ifmap* pixels for some configurations of $(WPAR, MPAR)$, all the other NN parameters are the same. The saturation comes with relatively small images for small $WPARs$; N_{exec} becomes quasi equal to $\left\lfloor \frac{W \times H}{WPAR} \right\rfloor$. For a large number of $WPAR$, the saturation happens for bigger images (higher $W \times H$).

For the power modeling, it was decided to consider the saturation by modeling the power of all *ifmaps* having more than 80 pixels by the power of a 1024 pixels *ifmap*. Tested on 19 NNs with different *ifmap* sizes, the RMSE is 11.5 µW for an average of 134 µW. This assumption is relevant because the *ifmaps* used are usually in the range of saturation (even for a large $WPAR$). For small images (below 80 2D pixels), we have 2 models corresponding to *ifmaps* with respectively 16 and 36 pixels. The maximum RMSE is 10 µW.

Concerning the impact of filter dimensions, the dynamic power is decreasing with the filter size ($S \times R \times C$). The major impact is on the general slope. Fig.9 exhibits this

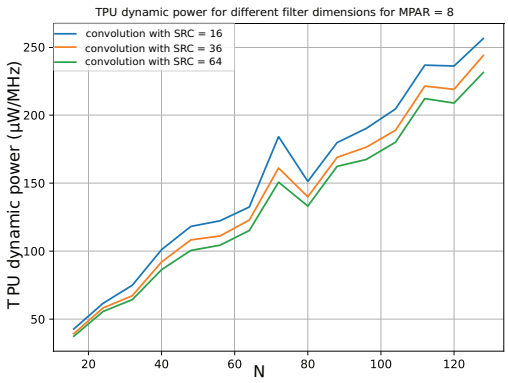

Fig. 9: Power of convolution for different filters sizes

statement on 3 NNs having the same parameters except the filter size. The dynamic power of convolutions is decreasing with filter dimensions because, as one *weight* is processed each cycle, the larger the filter dimensions, the higher the number of accumulation cycles required before the 5 scaling factors cycles. Thus on average, the 5 cycles of scaling (see section II,B) do not have an impact on the overall power of big filters; on the opposite, for small filters, the 5 cycles have more impact because there are fewer accumulation cycles.

To model this behavior, we choose to make the function multiplying NPE dependent on the filter dimensions as it is the function affecting the slope. We estimated it for different filter sizes and we generalize it with a regression (a power function was chosen as it gives the lowest RMSE and correlation coefficient). The model explained below was tested on 5 NNs with different filter sizes for the 210 configurations: the RMSE is 16 µW for an average of 152 µW.

Finally, Eq.5 models the dynamic power of convolution layers P_c (as well as maxpool and depthwise layers):

$$\begin{aligned} P_c = c_0 + c_1 \times S.R.C^{c_2} \times NPE \\ + c_3 \times NPE\lceil \log_2 WPAR \rceil + c_4 \times WPAR \end{aligned} \quad (5)$$

c_i were determined by linear regression. The maximum error is theoretically lower than the sum of the errors of each approximation. The error generated using Eq.3 is impacting all the estimations.

b) Fully connected dynamic power

FC layers are characterized by N_{in} and N_{out}. N_{out} should not impact drastically the power consumption because as the architecture is output stationary, each PE calculates one N_{out}; $\lceil \frac{Nout}{NPE} \rceil$ executions are then needed to compute all the N_{out}. The NPE processors work the same way even if N_{out} are less than NPE. 4 NNs are run with the same N_{in} varying N_{out} from 1 to 32. Only 5 µW of RMSE was observed (the average dynamic consumption is 78 µW). The impact of N_{out} is then neglected.

For our applications involving FC layers, the value of N_{in} ranges from 25 to 500. The dynamic power model was developed based on simulations that encompassed this entire N_{in} range. As N_{in} increases, the dynamic power also increases. Nevertheless, this upward trend in power becomes less important as N_{in} reaches higher numerical values. It is due to the increase of the number of accumulations leading to higher nets activities. When N_{in} is already high, more accumulations are not affecting drastically the activity, so the dynamic power does not increase excessively. This behavior is modeled with a logarithmic function. Eq.6 models FC layers power consumption P_{fc}:

$$\begin{aligned} P_{fc} = c_0 + (c_1 + c_2 \times \log(N_{in})) \times NPE \\ + c_3 \times NPE\lceil \log(WPAR) \rceil + c_4 \times WPAR \end{aligned} \quad (6)$$

c_i were determined by linear regression. Tested on 9 fully connected networks with N_{in} varying from 25 to 500 for 210 configurations of $(WPAR, MPAR)$, the RMSE is 12.3 µW and the average is 110 µW.

c) L-layers NN dynamic power

Once the dynamic power is estimated for each type of layer, the dynamic power $Pdyn_{NN}(f)$ of a NN composed of L layers at the frequency f is calculated with the average power of each layer weighted by its latency.

$$Pdyn_{NN}(f) = \frac{\sum_{l=1}^{L} Lat_l \times Pdyn_l \times f}{\sum_{l=1}^{L} Lat_l} \quad (7)$$

Lat_l is the latency of the layer l calculated with Eq.2 and $Pdyn_l$ is the dynamic power calculated with Eq.5 or Eq.6 according to the layer type.

Since the model has been validated (with low RMSE) across all layer types with all possible parameters, and considering that the dynamic power of a multi-layer NN is a combination of the individual layer powers, the model remains valid for the entire NN as well. The model was subsequently tested on multiple NNs, yielding low errors. For example, on the VGG-like network (Fig.1) on 210 $(WPAR, MPAR)$, the RMSE is 16 µW for an average of 150.4 µW (11.6% of error). While our power model exhibits a higher RMSE when compared to alternative methods (such as [14]), it offers the distinct advantage of simplicity. The model relies solely on two structural parameters and raw information from NN characteristics, accessible at a high-level design abstraction. Despite the method's slightly elevated RMSE, it remains sufficiently low for effectively selecting the optimal configuration. Consequently, it continues to serve as a valuable tool in practical scenarios.

The estimation error arises from underestimating the influence of structural parameters, which likely affect operators with behaviors that are not consistently explainable (and thus not incorporated into the model). For instance, the local minimum at 80 in Figure 6 was excluded from the model (resulting in estimation errors) due to a lack of complete understanding (observed only for fixed $MPAR$ at 8). This could potentially be attributed to complex optimizations during High-Level Synthesis, which involve dividing operators into multiple units to allocate portions for non-concurrent tasks.

V. Configuration Choice

Some of the considered KPIs have antagonistic behaviors. By increasing NPE, the latency decreases but the power and area increase. To find the best trade-offs, we use Pareto

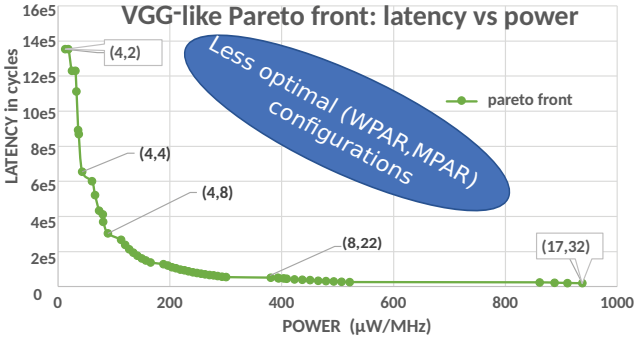

Fig. 10: VGG-like network sweet spots

fronts to determine the optimal architectures. Fig.10 shows sweet spots for the VGG-like NN considering the latency and power. The area is usually a specification, so only the points below a certain area could be considered. For each point of the curve, there are no other points that simultaneously have lower power and latency. The final choice between these $(WPAR, MPAR)$ points is made according to the application's specifications.

VI. Conclusion

In this paper, we presented a practical and explainable method to estimate three KPIs, latency, area, and power consumption of Gemini, an output stationary NMC configurable accelerator for NN inference. Its architecture can be easily configured thanks to two parameters, $WPAR$ and $MPAR$. The KPIs estimations are specific to any feed-forward NN specified as input of the estimator. They are accurate for all the KPIs but the dynamic power. The error is small enough to allow the user to determine the most accurate configuration for their application (NN).

The KPI estimation method presented in this article is adaptable for utilization with any output stationary accelerator (except few optimizations done for Gemini). Finally, this method can be extended to applications based on the execution of several NNs with different use rates.

References

[1] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

[2] Yann LeCun. 1.1 deep learning hardware: Past, present, and future. In *2019 IEEE International Solid- State Circuits Conference - (ISSCC)*, pages 12–19, 2019.

[3] Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, and Joel S. Emer. Efficient processing of deep neural networks: A tutorial and survey. *CoRR*, abs/1703.09039, 2017.

[4] Sunny Bodiwala and Nirali Nanavati. Efficient hardware implementations of deep neural networks: A survey. In *2020 Fourth International Conference on Inventive Systems and Control (ICISC)*, pages 31–36, 2020.

[5] Lukas Sekanina. Neural architecture search and hardware accelerator co-search: A survey. *IEEE Access*, 9:151337–151362, 2021.

[6] Rajesh Kedia, Shikha Goel, M. Balakrishnan, Kolin Paul, and Rijurekha Sen. Design space exploration of fpga-based system with multiple dnn accelerators. *IEEE Embedded Systems Letters*, 13(3):114–117, 2021.

[7] Nermine Ali, Jean-Marc Philippe, Benoit Tain, and Philippe Coussy. Exploration and generation of efficient fpga-based deep neural network accelerators. In *2021 IEEE Workshop on Signal Processing Systems (SiPS)*, pages 123–128, 2021.

[8] Zhongyuan Zhao, Hyoukjun Kwon, Sachit Kuhar, Weiguang Sheng, Zhigang Mao, and Tushar Krishna. mrna: Enabling efficient mapping space exploration for a reconfiguration neural accelerator. In *2019 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 282–292, 2019.

[9] Nicolas Bohm Agostini, Shi Dong, Elmira Karimi, Marti Torrents Lapuerta, José Cano, José L. Abellán, and David Kaeli. Design space exploration of accelerators and end-to-end dnn evaluation with tflite-soc. In *2020 IEEE 32nd International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, pages 10–19, 2020.

[10] Jude Haris, Perry Gibson, José Cano, Nicolas Bohm Agostini, and David R. Kaeli. Secda: Efficient hardware/software co-design of fpga-based dnn accelerators for edge inference. *2021 IEEE 33rd International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, pages 33–43, 2021.

[11] Linyan Mei, Huichu Liu, Tony Wu, H. Ekin Sumbul, Marian Verhelst, and Edith Beigne. A uniform latency model for dnn accelerators with diverse architectures and dataflows. In *2022 Design, Automation,Test in Europe Conference, Exhibition (DATE)*, pages 220–225, 2022.

[12] Ahmet Erdem, Cristina Silvano, Thomas Boesch, Andrea C. Ornstein, Surinder pal Singh, and Giuseppe S. Desoli. Runtime design space exploration and mapping of dcnns for the ultra-low-power orlando soc. *ACM Transactions on Architecture and Code Optimization (TACO)*, 17:1 – 25, 2020.

[13] Tianqi Tang, Sheng Li, Lifeng Nai, Norm Jouppi, and Yuan Xie. Neurometer: An integrated power, area, and timing modeling framework for machine learning accelerators industry track paper. In *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 841–853, 2021.

[14] Yannan Nellie Wu, Joel S. Emer, and Vivienne Sze. Accelergy: An architecture-level energy estimation methodology for accelerator designs. In *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 1–8, 2019.

[15] Rajeev Balasubramonian, Andrew B. Kahng, Naveen Muralimanohar, Ali Shafiee, and Vaishnav Srinivas. Cacti 7: New tools for interconnect exploration in innovative off-chip memories. *ACM Trans. Archit. Code Optim.*, 14(2), jun 2017.

[16] Masoud Shahshahani and Dinesh Bhatia. Ppa based cnn architecture explorer. In *2022 IEEE 13th Latin America Symposium on Circuits and System (LASCAS)*, pages 01–04, 2022.

[17] Yannan Nellie Wu, Vivienne Sze, and Joel S. Emer. An architecture-level energy and area estimator for processing-in-memory accelerator designs. In *2020 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 116–118, 2020.

[18] Tarek Darwish and Magdy Bayoumi. 5 - trends in low-power vlsi design. In WAI-KAI CHEN, editor, *The Electrical Engineering Handbook*, pages 263–280. Academic Press, Burlington, 2005.

[19] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv 1409.1556*, 09 2014.

[20] Grant Brown, Valerio Tenace, and Pierre-Emmanuel Gaillardon. Nemo-cnn: An efficient near-memory accelerator for convolutional neural networks. In *2021 IEEE 32nd International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, pages 57–60, 2021.

[21] Yu-Hsin Chen, Tien-Ju Yang, Joel Emer, and Vivienne 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*, 9(2):292–308, 2019.

[22] Zidong Du, Robert Fasthuber, Tianshi Chen, Paolo Ienne, Tao Luo, Xiaobing Feng, Yunji Chen, and Olivier Temam. Shidiannao. *ACM SIGARCH Computer Architecture News*, 43:92–104, 06 2015.

[23] Bert Moons and Marian Verhelst. An energy-efficient precision-scalable convnet processor in a 40-nm cmos. *IEEE Journal of Solid-State Circuits*, PP:1–12, 12 2016.

[24] Z Pei. Modeling power terminology, 2015. https://blogs.cuit.columbia.edu/zp2130/modeling_power_terminology/ (accessed Apr.4,2023).