Performance Modeling & Mapping of LLM Inference on Heterogeneous Vectorized CGRAs

Dionysios Kefallinos
National Technical University of Athens, Greece

Georgios Alexandris
National Technical University of Athens, Greece

Alexis Maras
National Technical University of Athens, Greece

Panagiotis Chaidos
National Technical University of Athens, Greece

Manil Dev Gomony
Eindhoven University of Technology, Netherlands

Henk Corporaal
Eindhoven University of Technology, Netherlands

Dimitrios Soudris
National Technical University of Athens, Greece

Sotirios Xydis
National Technical University of Athens, Greece

Abstract

Since the emergence of transformer-based models, the computational demands for Large Language Model (LLM) inference have been increasing exponentially, primarily due to their compounding parameter sizes, their structural complexity, and the use of non-linear functions. This tendency leads to the necessity of deploying them on low-power edge devices and DNN accelerators, to fuel next-generation agentic AI systems. Coarse-Grained Reconfigurable Architectures (CGRAs) have proven to be a compelling paradigm for edge acceleration, combining the programmability of general-purpose platforms with the high performance and energy efficiency associated with ASICs. In this work, we introduce an end-to-end performance modeling and mapping framework for LLM inference on heterogeneous CGRAs. Our methodology enables rapid exploration of the micro-architectural design space parameters, i.e., the number of processing elements, vector sizes, and memory configurations, by providing an accurate, explainable, and analytical CGRA performance modeling methodology, with an average cycle error of 0.9%. Architecturally, we build upon R-Blocks, a heterogeneous CGRA platform, and extend it to support floating-point arithmetic operations as well as a full-stack compilation and mapping flow for both full (FP32) and quantized (INT8) Llama2 models. The proposed methodology, evaluated on a 22nm technology node, achieves superior peak performance per Watt compared to related works such as REVAMP and CFEACT (1.8× and 2.8× respectively).

2012 ACM Subject Classification Computer systems organization → Reconfigurable computing; Hardware → Modeling and parameter extraction

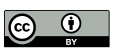
Keywords and phrases Edge AI, LLM, CGRA, Heterogeneous Architectures, Performance Modeling, Hardware Acceleration, Low Power Computing

Digital Object Identifier 10.4230/OASIcs.PARMA-DITAM.2026.8

Funding This work is funded in part by the Convolve project evaluated by the EU Horizon Europe research and innovation program under grant agreement No. 101070374.

1 Introduction

Over the last few years, AI and more specifically Large Language Models (LLMs), have emerged as a transformative technology and have become a focal point of research at both the architectural [16, 14, 23] and algorithmic levels [19, 35]. The need for agentic AI deployment on the edge has fueled a new wave of hardware research, targeting power efficiency when handling the intense linear and non-linear [3] operations of these models, without sacrificing performance.



17th Workshop on Parallel Programming and Run-Time Management Techniques for Many-Core Architectures and 15th Workshop on Design Tools and Architectures for Multicore Embedded Computing Platforms (PARMA-DITAM 2026).
Editors: Davide Baroffio, Paola Busia, Lev Denisov, and Nitin Shukla; Article No. 8; pp. 8:1–8:14
OpenAccess Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Introducing flexibility, in order to adapt to the dynamic computational patterns of LLMs, is key when targeting performance on a low power budget. Prior approaches have revolved around software/hardware co-optimization methodologies [10, 9, 4], others have explored FPGAs as a more flexible acceleration platform [11, 37], while some have developed ASIC solutions [13, 27, 36].

Trying to balance this trade-off, Coarse-Grained Reconfigurable Architectures (CGRAs) have re-emerged as a promising alternative. CGRAs are composed of an array of programmable Processing Elements (PEs), organized in an n-dimensional grid, interconnected via a routing network that can be reconfigured dynamically or per application. This means that they can incorporate highly optimized, pre-designed hardware units with close-to-ASIC performance as their coarse-grained PEs, while keeping interconnection overheads low compared to conventional fine-grained reconfigurable platforms. This property positions them as a dominant middle ground between fixed-function hardware and fully programmable platforms [34][20], as well as a promising candidate for edge LLM acceleration.

Typical CGRAs, thus far, map standard operation kernels into a grid of homogeneous PEs; i.e., offering limited customization capabilities to the targeted application domain. In this paper, we investigate the viability of a heterogeneous CGRA approach by presenting an end-to-end flow, capable of modeling, mapping, simulation, synthesis, and evaluation of LLM inference across multiple CGRA architectural configurations. Our architecture and code mapping flow build upon the low-power, heterogeneous R-Blocks CGRA [7] paradigm, and our evaluation is performed using the standard Llama2 architecture [32], producing broadly generalizable results. More specifically, the key contributions of this work are as follows:

- We extend the R-Blocks CGRA with full hardware and software support for floating-point arithmetic. This includes integrating a new single-precision Floating-Point Unit (FPU) as a Processing Element, and providing ISA support for compilation and scheduling. The resulting CGRA is capable of executing both linear and non-linear operations, while also leveraging parallelization, making it suitable for LLM acceleration.
- We introduce a high-level performance modeling framework for our architecture, capable of accurately estimating inference latency with minimal error. Through this analytical approach, we enable rapid design space exploration of different CGRA architectures without resorting to full hardware synthesis and RTL simulations for each micro-architectural configuration.
- We effectively map and evaluate end-to-end LLM inference of Llama2 models (the forward pass of base (FP32) and a quantized (INT8)) on several optimized CGRA instances at the post-synthesis level. We extract key insights regarding the effect of parallelization on performance and measure the power and area efficiency of our designs, synthesized at a 22nm technology node.

Through extensive experimental evaluation, we verify the accuracy of our performance modeling framework with a 0.9% error introduction.

2 Related Work

Heterogeneous CGRAs. One of the earliest works presenting a heterogeneous approach was the REVAMP [1] CGRA, a DSE framework that, relying on the application characteristics, produces CGRA architectures based on the HyCUBE [15] paradigm, with its own compilation and mapping scheme.

Table 1 Qualitative comparison of prior work.

CGRA	PE Structure	Application Mapping	Performance Modeling	LLM Inference
REVAMP [1]	Heterogeneous	DFG-Based	×	×
R-Blocks [7]	Heterogeneous	Custom C Compiler	×	×
DRIPS [30]	Homogeneous	MLIR-Based	×	×
CFEACT [22]	Homogeneous	MLIR-Based	×	BERT-models
ML-CGRA [21]	Homogeneous	MLIR-Based	×	BERT-models
PICACHU [26]	Heterogeneous	MLIR-Based	TimeLoop	Non-Linear Ops.
Our Work	Heterogeneous	Custom C Compiler	✓	Llama2

Carrying the torch, R-Blocks CGRA [7] suggests an operation-level PE disaggregation approach, where memory and logic tiles can be placed on the grid independently, with a custom NoC supporting the data and instruction interconnections between them. Application mapping on R-Blocks is performed from high-level code, using a custom compilation flow, while a high-level simulation tool is also provided.

Notably, none of these platforms has been used to map and evaluate LLM benchmarks on a heterogeneous PE grid, nor do they offer a dedicated performance modeling tool. This is precisely the domain in which our research seeks to provide new insights.

LLMs on CGRAs. ML-CGRA [21] is one of the earliest CGRA-related works utilizing MLIR to provide support for application mapping from popular ML front-ends such as TensorFlow and PyTorch directly to the reprogrammable, homogeneous PE fabric. This is achieved by producing the Data Flow Graph (DFG) of the target application and implementing optimization passes to map it on OpenCGRA [31] or CGRA-ME [5] generated architectures. With this tool, mapping LLMs such as BERT [8] on a CGRA becomes possible for the first time.

Other early publications such as DRIPS [30] and CFEACT [22], also utilize MLIR for compilation, although, again, on homogeneous PEs. DRIPS achieves dynamic load rebalancing for Graph Convolutional Network applications where kernels can vary in features and computational demands, whereas CFEACT further optimizes the produced DFG and allows for BERT models and even floating-point operations to be mapped more efficiently.

The benchmarks evaluated in these works, however, are simpler language representation models (BERT) and, therefore, the performance insights gained cannot be generalized when scaling up to the requirements of modern generative LLMs, such as GPT [28] or Llama [32]. Our work aims to provide more widely applicable conclusions by mapping and evaluating larger models of the Llama2 series.

Finally, the more recently published work of PICACHU [26] takes a different approach, utilizing a CGRA as a co-accelerator exclusively for non-linear operations. The main bulk of linear computations is handled by a systolic array, which communicates with the CGRA through a shared SRAM buffer. This work also makes use of MLIR [17] for mapping efficient code and scheduling data transfers between the systolic array and the CGRA, but only for the non-linear part of the models. It additionally utilizes TimeLoop [25] as a modeling tool to estimate area, power, and latency.

While accelerating non-linear functions using a CGRA seems promising, our work aims to map the entire token generation step on the reconfigurable fabric, taking advantage of the heterogeneous nature of the PEs. At the same time, we provide an analytical and explainable

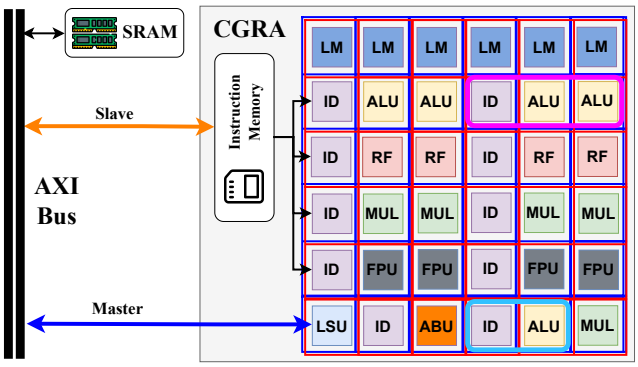


Figure 1 Architectural Overview of the Proposed CGRA.

(therefore interpretable) LLM performance modeling tool, tailored to our specific architecture, something missing from most prior works. Table 1 summarizes the above discussion in a qualitative manner.

3 Proposed CGRA Architecture

3.1 Architecture Overview

The first distinction of our approach, compared to the previously discussed CGRAs, is its support for heterogeneous architectural configurations regarding the Processing Elements. Most proposed CGRAs consist of identical PEs adhering to a predetermined dataflow pattern. By contrast, our work introduces several types of heterogeneous PEs, each with a specific Instruction Set Architecture (ISA). These Functional Units (FUs) or tiles, as commonly referred to in our work, bear a close resemblance to the core building blocks of CPUs, such as Arithmetic-Logical Units (ALU), Register Files (RF), Floating-Point Units (FPU), Local Memories (LM) and Load-Store Units (LSU), among others.

Disaggregating the ISA into simpler and smaller operation sets (PE heterogeneity) comes with multiple benefits:

- Narrower instruction interconnect network, since opcode reuse is allowed among different types of FUs.
- More efficient PEs, through increased specificity in their respective functionality, and more specialized pre-designed hardware.
- Better utilization of resources by effectively mapping operations to the appropriate PEs.
- Designer flexibility, regarding the number of PEs of each type to be used in a specific application domain.

Figure 1 depicts the proposed CGRA scheme. The architecture is built upon the R-Blocks [7] framework, extended with additional support for Floating-Point Unit PE, to allow for the efficient execution of non-linear functions required for LLM inference. The proposed CGRA consists of a grid of programmable FU tiles communicating through two distinct Network-on-Chip (NoC) interconnect structures, as depicted with blue and red colors in Figure 1. The first NoC handles instruction transfers between tiles, while the second manages all data transfers that take place during the program execution. The two NoCs are structured in a 2D mesh formation, utilizing Wilton switchboxes [29] for reduced reconfiguration overhead and wire congestion.

The programming of each FU is handled by an Instruction Decoder (ID) tile, which stores its respective instruction memory segment, and every cycle dispatches the instruction word to be executed by its associated FU.

To enhance parallelism and improve computational throughput, we extend our architecture with SIMD (Single Instruction, Multiple Data) capabilities through vectorization. In this configuration, a single instruction decoder (ID) can be configured to control multiple function units (FUs) of the same type simultaneously, thereby enabling efficient parallel execution and reducing control overhead across the reconfigurable fabric.

Figure 1 shows an exemplary Vectorized CGRA physical grid layout, i.e., the ID and ALUs highlighted in magenta form a Vector Unit, while the ones in light blue are in a scalar layout.

In terms of memory hierarchy, our accelerator makes use of Local Memories (LM), facilitating smaller but faster transfers between the micro-architectural components. Communication with the external, scratchpad memory is facilitated by the AXI interface protocol, which allows the CGRA to be flexibly connected to any compatible host. In our configuration, a Master Interface accesses the external memory address space, while a Slave Interface is used to program the Instruction Memory, which in turn programs the ID of every FU, utilizing the NoC.

3.2 Compilation Framework

The process of mapping high-level code to a CGRA instance is facilitated by a fully custom compilation flow. Compilation occurs for a specific architecture and a specific grid formation of FUs defined by the programmer. An architecture can easily be created and modified by the designer according to the requirements of the application domain and may contain any number of FUs of each type, and also vector units for parallel computation.

Starting from a C-language specification, the code is compiled using the LLVM compilation back-end and operations are lowered to simple instructions supported by the hardware. Subsequently, given an architectural description, these instructions are bound to the ISA of the CGRA's FUs and scheduled in an executable manner, utilizing the OpenASIP [12] compiler and scheduler. OpenASIP tools target Transport Triggered Architectures [6] (TTA), a processor design paradigm where computations revolve around data transfers on the buses. As such, the entire CGRA can easily be modeled as a TTA machine in order to take advantage of these tools. With this compilation scheme, macros and custom intrinsics allow the programmer to explicitly control the code mapping (e.g., by assigning computations to standard PEs) before producing the final executable.

3.3 HW/SW Extensions for FP Arithmetic Support

Running full LLM inference on the edge requires execution of non-linear functions, which rely on floating-point arithmetic and its properties. To compute them effectively without suffering from complicated emulation algorithm overheads or large accuracy loss, we employ a low-power Floating-Point Unit (FPU). The OpenCores FPU [24] and its accompanying Compare Unit are packed into a compatible tile and integrated within the CGRA environment as a new type of FU. This low-power FPU is fully compliant with the IEEE 754 single-precision floating-point standard (FP32), supports the basic arithmetic operations, as well as comparisons and conversions between the INT32 and floating-point (FP32) representations.

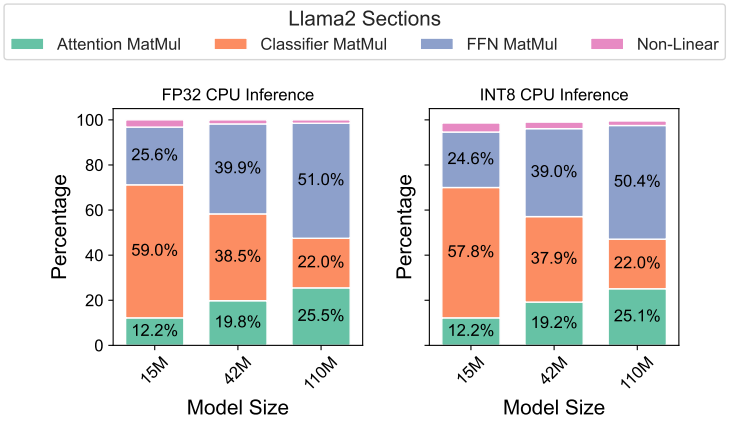


Figure 2 Llama2 CPU Profiling Overview.

To support full functionality of the new FU, we bound the lowered operations from the LLVM compilation back-end to the corresponding TTA instructions, and those, in turn, to the new ISA created for the FPU tile. This allows the token generation step of LLMs to be mapped entirely on the reconfigurable fabric of a heterogeneous CGRA (evaluated for the first time in CGRAs, to the best of our knowledge).

4 LLM Mapping and Modelling on the CGRA

4.1 LLMs on the Edge

LLMs, just like many traditional neural networks, combine linear layers such as matrix multiplications and non-linear operations like normalization or softmax. The linear layers are good at infusing the activation vectors with the stored information in the model's weights, while the non-linearities enforce stability and ensure non-uniform node activation, giving the models their neuron-like morphology and capabilities. LLMs, additionally, utilize the Attention mechanism to encode long-range dependencies between the tokens of a sequence [33].

In this work, we make use of the Llama2 [32] architecture as a representative sample of the different and rapidly evolving transformer implementations for Language Generation. Llama2 employs architectural patterns commonly found in most state-of-the-art LLM models; inference consists of a series of layers, each containing an Attention and a Feed Forward Network (FFN) block. The bulk of the model's weights are used in matrix multiplications in these two blocks. The non-linear functions included between the linear blocks are Softmax, RMS Normalization, SwiGLU (a ReLU variation), and Llama's signature Rotary Positional Encoding (RoPE)[2]. After the token embedding passes through multiple layers, another matrix multiplication (Classifier) produces the final probability distribution of the forward pass from which the output token is selected.

Trying to understand the distribution of the computational load in LLMs, we perform time-based profiling on 3 differently sized base (FP32) Llama2 models, ranging from 15M to 110M parameters, and their symmetrically quantized (INT8) versions (Figure 2). For inference, we use an AMD Ryzen 3 5300U CPU on single-thread execution. As expected, the non-linear operations, while complex in form –containing inverse square root, exponential, and trigonometric functions– are used sparingly between linear operations and in most cases

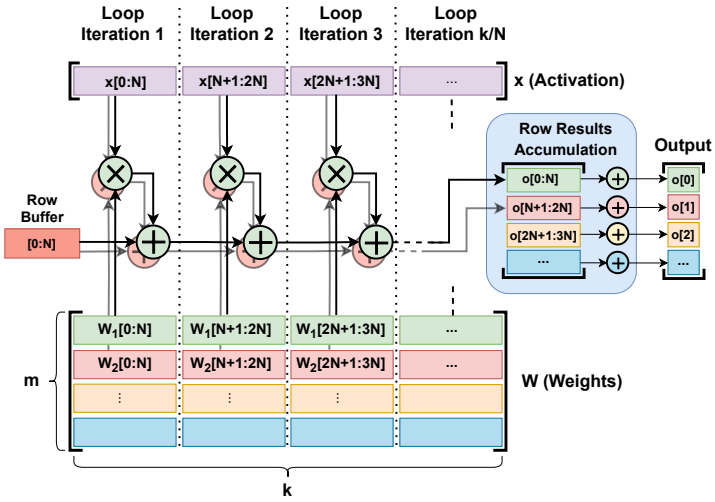


Figure 3 Vectorization of the MatMul Kernel.

don't cause a computational bottleneck, especially as model sizes increase, where they account for an even smaller percentage of runtime. On the other hand, matrix multiplication is a much more computationally intensive operation.

Since matrix multiplication operations are usually free of dependencies and parallelizable to a huge extent, hardware with parallel execution capabilities becomes invaluable for LLM inference and training, e.g., modern GPUs utilizing multiple tensor cores. In the case of GPU execution, as showcased in [26], the percentage of runtime taken up by matrix multiplications can drop as low as 53% for 7B parameter Llama2 models, especially as the context window increases.

Our particular approach aims to achieve some degree of parallelization while still operating within the strict power constraints of Edge Computing. As mentioned before, instead of focusing on a section of inference and using the CGRA as an operation-specific accelerator [26], we utilize the compilation toolchain in conjunction with LLM-specific optimizations to map the entire forward pass of the Llama2 inference model to a single CGRA instance.

4.2 Mapping Methodology

Since matrix multiplications benefit tremendously from Instruction Level Parallelism in the execution model, and no spatial dependencies exist between the operands, multiple elements of the matrix's rows or columns can be loaded for parallel execution as long as the hardware, the memory structure, and the mapping methodology support it.

In our proposed CGRA, FUs of the same type are aggregated into Vector Units, connected to the same ID, thus performing operations supported by the particular FU type in an SIMD fashion. To enable this functionality, the operands are stored in Register Files (RF) or Local Memory units, so that they can be simultaneously accessed.

The matrix-vector multiplication ($W[m,k] \cdot x[k]$) was vectorized using per-row operand grouping. Elements are grouped by N (Vector Unit size), both in the weight matrix rows and the activation vector. As illustrated in Figure 3, in every loop iteration, the corresponding operand vectors are multiplied and accumulated on a vector buffer. This reduces the number of loop iterations per row from k to k/N, effectively introducing a degree of parallelization N. The elements of the buffer vectors are then accumulated to produce the final result.

This method of vectorization takes advantage of the temporal locality of operations by storing the activation vector in the Local Memory for the entire duration of the computation. It also utilizes the spatial locality of data by fetching entire rows to the Local Memories, decreasing the total number of required Global Memory accesses, and, ultimately, effectively distributing the most time-consuming operation of the inference among our computational resources.

The rest of the model's sections, consisting mostly of non-linear functions, are scheduled and mapped to the scalar FUs of the defined architecture by the compiler, concluding the mapping process.

4.3 Analytical Performance Modeling

The need for an analytical modeling framework, capable of producing realistic performance estimates for Llama2 inference on various architectures, stems from the time-consuming nature of simulating the entire execution at the RTL level. Full model inference for multiple benchmarks on different architectural configurations can be very time-intensive, consuming tens of minutes per token generation. Even the cycle-accurate simulation tool of the OpenASIP environment, which models bus transfers at a high level, fails to account for physical memory access times, while still imposing significant simulation time overheads (several minutes per token).

What we propose is an even higher-level performance estimation model, specific for LLM inference execution on heterogeneous CGRA architectures. The framework is both hardware and software-aware, meaning it requires a high-level description of the architectural configuration, as well as model parameters and mapping decisions, in order to produce a performance estimation with minimal error.

More specifically, we begin with modeling the lower-level functions of the inference model, i.e., the matrix multiplications and the non-linearities that account for, practically, the entire runtime, as well as data-flow overheads. We first use a combination of RTL cycle-accurate measurements and TTA simulator runs on microbenchmarks to estimate the influence of certain loop kernels on the overall performance, while accounting for memory access times.

Using the TTA simulator and the LLM parameters, we come up with analytical equations for entire functions that make use of these kernels.

As an example, we present the generalized cost estimation equations for matrix-vector multiplication execution ($W[m,k] \cdot x[k]$) with the mapping methodology outlined in 4.2.

$$C = m\left((k+1)\,M + \frac{k}{N}\cdot G + N\cdot E\right) \tag{1}$$

$$C_q = m\left(k\cdot M + \frac{k}{N}\cdot G + N\cdot\frac{k}{S}\cdot E\right) + Q \tag{2}$$

Here, N stands for the degree of vectorization, while M, G and E refer to the SRAM access cost, the vector multiply-accumulate cost, and the element extraction and accumulation cost from the final row buffer, respectively. These parameters depend on the operating frequency of the CGRA, among other configuration settings, and are determined in the lower-level analysis. Since k/N multiplications-accumulations take place in each of the m rows, this is the coefficient of G. The coefficients of M and E correspond to the numbers of memory accesses and accumulated vector elements, respectively, which also relate to loop iterations.

Table 2 CGRA Architectures for Evaluation.

Architecture	Vector Size	Grid Size	VFPU	VMUL	Llama2 Model
Vec4	4	6x9	✓	×	FP32
Vec8	8	10x8	✓	×	FP32
Vec16	16	18x8	✓	×	FP32
Qvec4	4	6x9	×	✓	INT8
Qvec8	8	10x8	×	✓	INT8
Qvec16	16	18x8	×	✓	INT8

In Eq. 2, which models the same cost for the INT8 quantized Llama2, S symbolizes the quantization group size (meaning the number of elements that share the same scaling factor) and Q denotes the quantization overhead cost, which is calculated by a different equation and depends on the input dimensions.

To complete the Llama2 modeling validation, we account for all matrix multiplication blocks on the mapped inference code, their operand dimensions, and the flow of data between them, through non-linear functions and memory accesses. In an extensive validation study of the proposed modeling framework, which included both microbenchmarks and full model inference runs on simulated hardware, we achieved an average error of 0.9% and a maximum error of 2.5% against cycle-accurate measurements.

This modeling methodology enables a major part of the following analysis by accelerating the exploration of CGRA architectures and software configurations. Most importantly, it allows us to extract critical insights about the effect of vectorization on performance, the impact of design decisions, and the cost of scaling model parameters.

5 Results & Discussion

5.1 Experimental Setup

Initially, we map the entire token generation step of two different versions of the 42M parameter Llama2 model: the base FP32 inference model and a (post-training) quantized version with INT8 operands for better power performance and a smaller memory footprint. We explore the design space of vectorized architectures with vector sizes of 4, 8, and 16, as shown in Table 2, to measure the impact of vectorization on performance.

All of the evaluated architectures build upon a simple `Baseline` architecture consisting of only the computational FUs necessary for basic CGRA functionality, meaning only an ALU, a MUL, an RF and an FPU. `Vec4`, `Vec8`, and `Vec16`, add to that vectorized FPUs and ALUs for executing the FP32 version of Llama2 and achieve an ILP factor of 4, 8, and 16, respectively. `Qvec4`, `Qvec8`, and `Qvec16`, where the quantized version of the Llama2 will be mapped, are designed in the same way as their FP counterparts, but utilize vectorized MUL units instead of vectorized FPUs. Every architecture evaluated also has two vectorized Register Files (VRF), with the same vector size.

5.2 Performance Evaluation

Figure 4 compares the estimations of our performance modeling to the measured RTL cycles for the entire token generation step. Our modeling methodology achieves a maximum error of 1.3% on the full model executions and 1.5% on the quantized versions.

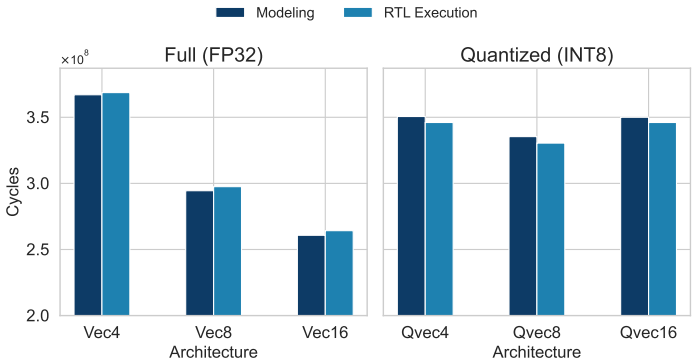


Figure 4 Performance Modeling vs Actual Cycles.

Table 3 Comparison with State-of-the-Art Designs. Performance metrics are reported from the respective publications.

		REVAMP [1]	R-Blocks [7]	DRIPS [30]	CFEACT [22]	PICACHU [26]	This Work (V4/V8)
Arch.	Technology (nm)	28	22	45	40	45	22
	Clock Freq. (MHz)	100	100	800	650	1000	200
	# of PEs	36	77	25	60	16	54 / 80
	Area (mm^2)	0.125	0.486	2.07	2.411	1	0.311 / 0.439
Perf.	Power (mW)	5	3.99	564.8	839	64.2	36.24 / 50
	Peak GOPs	0.9	461	N/A	78	N/A	11.8 / 16
	Peak GOPs/W	180	115	N/A	92.9	N/A	325.6 / 320

Following the methodology of R-Blocks [7], we compare all of the presented architectures against `Baseline` to assess the impact of parallelization. `Vec4`, `Vec8` and `Vec16` achieve 1.38×, 1.72× and 1.93× speedup respectively. We observe diminishing returns in performance gains as the degree of parallelization increases. This is caused by the result accumulation step at the end of each row calculation, during the execution of vectorized matrix multiplication, described in Section 4. This makes sense because weight and activation matrices in the 42M parameter Llama2 model are quite small in size (usually 512 elements), and can not take full advantage of large degrees of parallelization.

Even more notably, INT8-quantized runs of the forward pass achieve marginal performance gains as the vector size increases. `Qvec4` and `Qvec8` achieve 1.55× and 1.62× speedup, respectively, compared to `Baseline` (not depicted), and further increasing the vector size to 16 causes the group accumulation overheads to surpass the performance gains. This is because the Llama2 quantization approach of grouped operand scaling, induces an accumulation overhead inversely proportional to the group size S, since scaled elements have to be accumulated in their groups. However, quantized model execution requires lower energy per token generation and significantly less memory usage [18], therefore, it is worth exploring as an alternative mapping in this analysis.

5.3 Post-Synthesis PPA Analysis

Taking into account all these observations, we opt to synthesize the more performant `Vec4`, `Vec8`, `Qvec4` and `Qvec8` architectures and extract power and area measurements. We use the Synopsys Design Compiler at GF 22nm node and a 200MHz operating frequency. Power measurements are obtained using Synopsys PrimeTime from the switching activity data generated from gate-level simulations.

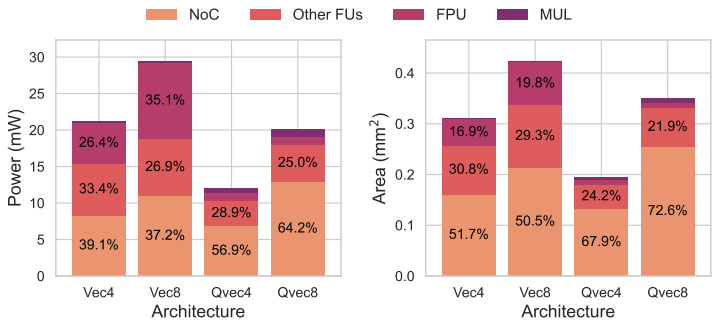


Figure 5 Power and Area Breakdown.

Figure 5 presents the power consumption and area utilization per architecture, both of which, naturally, increase as the vector units get larger. As expected from a computationally intensive tile, the inclusion of more FPU tiles influences both metrics much more heavily than adding MUL units. However, increasing the vector size, in general, also has the side-effect of requiring a larger interconnect network, which also draws significant power in every configuration. The effects of INT8 quantization also become clear: `Qvec4` and `Qvec8` report lower power and area, due to the lighter INT8 computations.

5.4 Comparative Study with SotA

Most of the previous approaches, as presented in section 2, are not directly comparable to our work. They either haven't evaluated their platform on LLM inference or differ architecturally. Even PICACHU [26], one of the most recent (and comparable) works in the field, approaches LLM inference differently, by only computing non-linear operations on the CGRA. Despite these, to give the reader an idea of where our work stands in terms of performance, we present a qualitative comparison table for key area, power and performance metrics.

6 Conclusions

The key insights from this work revolve around the efficacy of deploying LLMs on heterogeneous CGRAs. We show that it is not only possible to map the entire token generation step on a low-power edge device, but also provide an analytical framework to estimate its performance at design time, allowing for fast architectural DSE. We believe the merit of this initial contribution lies in its extendability and how it enables future CGRA designers to optimize LLM execution.

References

1 Thilini Kaushalya Bandara, Dhananjaya Wijerathne, Tulika Mitra, and Li-Shiuan Peh. REVAMP: a systematic framework for heterogeneous CGRA realization. In *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '22, pages 918–932, New York, NY, USA, 2022. Association for Computing Machinery. doi:10.1145/3503222.3507772.

2 Federico Barbero, Alex Vitvitskyi, Christos Perivolaropoulos, Razvan Pascanu, and Petar Veličković. Round and Round We Go! What makes Rotary Positional Encodings useful?, 2025. doi:10.48550/arXiv.2410.06205.

3 Fenglong Cai, Dong Yuan, Zhe Yang, and Lizhen Cui. Edge-LLM: A Collaborative Framework for Large Language Model Serving in Edge Computing. In *2024 IEEE International Conference on Web Services (ICWS)*, pages 799–809, 2024. doi:10.1109/ICWS62655.2024.00099.

4 Lvcheng Chen, Ying Wu, Chenyi Wen, Shizhang Wang, Li Zhang, Bei Yu, Qi Sun, and Cheng Zhuo. An Agile Framework for Efficient LLM Accelerator Development and Model Inference, 2024.

5 S. Alexander Chin, Noriaki Sakamoto, Allan Rui, Jim Zhao, Jin Hee Kim, Yuko Hara-Azumi, and Jason Anderson. CGRA-ME: A unified framework for CGRA modelling and exploration. In *2017 IEEE 28th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, pages 184–189, 2017. doi:10.1109/ASAP.2017.7995277.

6 Henk Corporaal and Reinoud Lamberts. TTA processor synthesis. In *First Annual Conf. of ASCI*, pages 18–27. Citeseer, 1995.

7 Barry de Bruin, Kanishkan Vadivel, Mark Wijtvliet, Pekka Jääskeläinen, and Henk Corporaal. R-Blocks: An Energy-Efficient, Flexible, and Programmable CGRA. *ACM Trans. Reconfigurable Technol. Syst.*, 17(2), May 2024. doi:10.1145/3656642.

8 Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, 2019. arXiv:1810.04805.

9 Lidong Guo, Zhenhua Zhu, Tengxuan Liu, Xuefei Ning, Shiyao Li, Guohao Dai, Huazhong Yang, Wangyang Fu, and Yu Wang. Towards Floating Point-Based Attention-Free LLM: Hybrid PIM with Non-Uniform Data Format and Reduced Multiplications. In *Proceedings of the 43rd IEEE/ACM International Conference on Computer-Aided Design*, ICCAD '24, New York, NY, USA, 2025. Association for Computing Machinery. doi:10.1145/3676536.3676776.

10 Yingbing Huang, Lily Jiaxin Wan, Hanchen Ye, Manvi Jha, Jinghua Wang, Yuhong Li, Xiaofan Zhang, and Deming Chen. Invited: New Solutions on LLM Acceleration, Optimization, and Application. In *Proceedings of the 61st ACM/IEEE Design Automation Conference*, DAC '24, New York, NY, USA, 2024. Association for Computing Machinery. doi:10.1145/3649329.3663517.

11 Suyeon Hur, Seongmin Na, Dongup Kwon, Joonsung Kim, Andrew Boutros, Eriko Nurvitadhi, and Jangwoo Kim. A Fast and Flexible FPGA-based Accelerator for Natural Language Processing Neural Networks. *ACM Trans. Archit. Code Optim.*, 20(1), February 2023. doi:10.1145/3564606.

12 Pekka Jääskeläinen, Timo Viitanen, Jarmo Takala, and Heikki Berg. *HW/SW Co-design Toolset for Customization of Exposed Datapath Processors*, pages 147–164. Springer International Publishing, 2017. doi:10.1007/978-3-319-49679-5_8.

13 Jaeyong Jang, Yulhwa Kim, Juheun Lee, and Jae-Joon Kim. FIGNA: Integer Unit-Based Accelerator Design for FP-INT GEMM Preserving Numerical Accuracy. In *2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 760–773, 2024. doi:10.1109/HPCA57654.2024.00064.

14 Roman Kaplan. Intel Gaudi 3 AI Accelerator: Architected for Gen AI Training and Inference. In *2024 IEEE Hot Chips 36 Symposium (HCS)*, pages 1–16, 2024. doi:10.1109/HCS61935.2024.10665178.

15 Manupa Karunaratne, Aditi Kulkarni Mohite, Tulika Mitra, and Li-Shiuan Peh. HyCUBE: A CGRA with reconfigurable single-cycle multi-hop interconnect. In *2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–6, 2017. doi:10.1145/3061639.3062262.

16 Hanjoon Kim, Younggeun Choi, Junyoung Park, Byeongwook Bae, and Hyunmin et al. Jeong. TCP: A Tensor Contraction Processor for AI Workloads Industrial Product*. In *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*, pages 890–902, 2024. doi:10.1109/ISCA59077.2024.00069.

17 Chris Lattner, Mehdi Amini, Uday Bondhugula, Albert Cohen, Andy Davis, Jacques Pienaar, River Riddle, Tatiana Shpeisman, Nicolas Vasilache, and Oleksandr Zinenko. MLIR: A Compiler Infrastructure for the End of Moore's Law, 2020. arXiv:2002.11054.

18 Wenjie Li, Aokun Hu, Ningyi Xu, and Guanghui He. Quantization and Hardware Architecture Co-Design for Matrix-Vector Multiplications of Large Language Models. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 71(6):2858–2871, 2024. doi:10.1109/TCSI.2024.3350661.

19 Jun Liu, Zhenglun Kong, Peiyan Dong, Changdi Yang, Xuan Shen, Pu Zhao, Hao Tang, Geng Yuan, Wei Niu, Wenbin Zhang, Xue Lin, Dong Huang, and Yanzhi Wang. RoRA: Efficient Fine-Tuning of LLM with Reliability Optimization for Rank Adaptation, 2025. doi:10.48550/arXiv.2501.04315.

20 Leibo Liu, Jianfeng Zhu, Zhaoshi Li, Yanan Lu, Yangdong Deng, Jie Han, Shouyi Yin, and Shaojun Wei. A Survey of Coarse-Grained Reconfigurable Architecture and Design: Taxonomy, Challenges, and Applications. *ACM Comput. Surv.*, 52(6), October 2019. doi:10.1145/3357375.

21 Yixuan Luo, Cheng Tan, Nicolas Bohm Agostini, Ang Li, Antonino Tumeo, Nirav Dave, and Tong Geng. ML-CGRA: An Integrated Compilation Framework to Enable Efficient Machine Learning Acceleration on CGRAs. In *2023 60th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6, 2023. doi:10.1109/DAC56929.2023.10247873.

22 Yiqing Mao, Xuchen Gao, Jiahang Lou, Yunhui Qiu, Wenbo Yin, Wai-Shing Luk, and Lingli Wang. CFEACT: A CGRA-based Framework Enabling Agile CNN and Transformer Accelerator Design. In *2024 34th International Conference on Field-Programmable Logic and Applications (FPL)*, pages 213–219, 2024. doi:10.1109/FPL64840.2024.00037.

23 Seungjae Moon, Jung-Hoon Kim, Junsoo Kim, Seongmin Hong, and Junseo et al. Cha. A Latency Processing Unit: A Latency-Optimized and Highly Scalable Processor for Large Language Model Inference. *IEEE Micro*, 44(6):17–33, 2024. doi:10.1109/MM.2024.3420728.

24 OpenCores. OpenCores Floating Point Unit. https://opencores.org/projects/fpu.

25 Angshuman Parashar, Priyanka Raina, Yakun Sophia Shao, Yu-Hsin Chen, Victor A. Ying, Anurag Mukkara, Rangharajan Venkatesan, Brucek Khailany, Stephen W. Keckler, and Joel Emer. Timeloop: A Systematic Approach to DNN Accelerator Evaluation. In *2019 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 304–315, 2019. doi:10.1109/ISPASS.2019.00042.

26 Jiajun Qin, Tianhua Xia, Cheng Tan, Jeff Zhang, and Sai Qian Zhang. PICACHU: Plug-In CGRA Handling Upcoming Nonlinear Operations in LLMs. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, pages 845–861, 2025. doi:10.1145/3676641.3716013.

27 Yubin Qin, Yang Wang, Zhiren Zhao, Xiaolong Yang, Yang Zhou, Shaojun Wei, Yang Hu, and Shouyi Yin. MECLA: Memory-Compute-Efficient LLM Accelerator with Scaling Sub-matrix Partition. In *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*, pages 1032–1047, 2024. doi:10.1109/ISCA59077.2024.00079.

28 Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language Models are Unsupervised Multitask Learners. *OpenAI Blog*, 1(8), 2019. URL: https://cdn.openai.com/better-language-models/language_models_are_unsupervised_multitask_learners.pdf.

29 Seyyed Ahmad Razavi, Morteza Saheb Zamani, and Kia Bazargan. A tileable switch module architecture for homogeneous 3D FPGAs. In *2009 IEEE International Conference on 3D System Integration*, pages 1–4, 2009. doi:10.1109/3DIC.2009.5306586.

30 Cheng Tan, Nicolas Bohm Agostini, Tong Geng, Chenhao Xie, Jiajia Li, Ang Li, Kevin J. Barker, and Antonino Tumeo. DRIPS: Dynamic Rebalancing of Pipelined Streaming Applications on CGRAs. In *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 304–316, 2022. doi:10.1109/HPCA53966.2022.00030.

31 Cheng Tan, Chenhao Xie, Ang Li, Kevin J. Barker, and Antonino Tumeo. OpenCGRA: An Open-Source Unified Framework for Modeling, Testing, and Evaluating CGRAs. In *2020 IEEE 38th International Conference on Computer Design (ICCD)*, pages 381–388, 2020. doi:10.1109/ICCD50377.2020.00070.

32 Hugo Touvron, Louis Martin, and Kevin Stone et al. Llama 2: Open Foundation and Fine-Tuned Chat Models, 2023. doi:10.48550/arXiv.2307.09288.

33 Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention Is All You Need, 2023. arXiv:1706.03762.

34 Mark Wijtvliet, Luc Waeijen, and Henk Corporaal. Coarse grained reconfigurable architectures in the past 25 years: Overview and classification. In *2016 International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation (SAMOS)*, pages 235–244, 2016. doi:10.1109/SAMOS.2016.7818353.

35 Guangxuan Xiao, Ji Lin, Mickael Seznec, Hao Wu, Julien Demouth, and Song Han. SmoothQuant: Accurate and Efficient Post-Training Quantization for Large Language Models. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett, editors, *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 38087–38099. PMLR, 23–29 July 2023. URL: https://proceedings.mlr.press/v202/xiao23c.html.

36 Eunji Yoo, Gunho Park, Jung Gyu Min, Se Jung Kwon, Baeseong Park, Dongsoo Lee, and Youngjoo Lee. TF-MVP: Novel Sparsity-Aware Transformer Accelerator with Mixed-Length Vector Pruning. In *2023 60th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6, 2023. doi:10.1109/DAC56929.2023.10247799.

37 Shulin Zeng, Jun Liu, Guohao Dai, Xinhao Yang, Tianyu Fu, Hongyi Wang, Wenheng Ma, Hanbo Sun, Shiyao Li, Zixiao Huang, Yadong Dai, Jintao Li, Zehao Wang, Ruoyu Zhang, Kairui Wen, Xuefei Ning, and Yu Wang. FlightLLM: Efficient Large Language Model Inference with a Complete Mapping Flow on FPGAs. In *Proceedings of the 2024 ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, FPGA '24, pages 223–234, New York, NY, USA, 2024. Association for Computing Machinery. doi:10.1145/3626202.3637562.