SECDA-LLM: Designing Efficient LLM Accelerators for Edge Devices

Jude Haris, Rappy Saha, Wenhao Hu, José Cano
School of Computing Science, University of Glasgow, Scotland, UK

***Abstract*—The increase in open-source availability of Large Language Models (LLMs) has enabled users to deploy them on more and more resource-constrained edge devices to reduce reliance on network connections and provide more privacy. However, the high computation and memory demands of LLMs make their execution on resource-constrained edge devices inefficient and challenging. To address this issue, designing new and efficient edge accelerators for LLM inference is crucial.**

FPGA-based accelerators are ideal for LLM acceleration due to their reconfigurability, as they enable model-specific optimizations and higher performance per watt. However, creating and integrating FPGA-based accelerators for LLMs has proven challenging, particularly due to the limited hardware design flows for LLMs in existing FPGA platforms.

To tackle this issue, in this paper we first propose a new design platform, named *SECDA-LLM*, that utilizes the SECDA methodology to streamline the process of designing, integrating, and deploying efficient FPGA-based LLM accelerators for the *llama.cpp* inference framework. We then demonstrate, through a case study, the potential benefits of SECDA-LLM by creating a new MatMul accelerator that supports block floating point quantized operations for LLMs. We ran our initial accelerator design on the PYNQ-Z1 board, achieving on par performance with the CPU (20 seconds per token or $\sim$27 seconds per word) for the TinyLlama model. We plan to improve it in the future.

I. Introduction

Large language models (LLMs) are an emerging class of machine learning systems geared toward learning from huge text-based datasets. LLMs such as GPT-3 [1] revolutionized the ability of Artificial Integillence (AI) systems to understand and generate human language. Due to innovative changes in model architecture, training methods, and through the help of the popularity of online services like ChatGPT [2], the field of LLMs has evolved rapidly.

The number of everyday users is growing rapidly due to the myriad of use cases from translation [3], classification [4], code generation [5] to healthcare [6]. Additionally, cloud-based LLM services are currently the go-to method of access to LLMs for everyday users, but as the availability of open-source LLMs and datasets increased, especially over the last few years, the need for edge-based, localized access and execution of LLMs has become more sought after. Massive community-driven pushes have facilitated easy access to LLMs and rapid prototyping of new models and optimizations to enable efficient LLM inference on edge devices. On the forefront of these pushes is the GPT-Generated Model Language [7] (GGML). GGML is a tensor library for ML specialized in enabling large models and high performance on commodity hardware. Furthermore, the *GGML*'s *llama.cpp* project [8] is specialized towards running LLMs on edge devices, supporting LLM inference on commodity CPUs and GPUs.

Unfortunately, LLMs, even for inference, can be very computationally demanding. In addition, due to their large memory footprint, they require high memory capacity and bandwidth. These properties of LLMs make them challenging to execute on edge-based devices. Additionally, running LLMs on resource-constrained devices, such as mobile phones or Internet of Things devices (IoT) is even more difficult and, in some cases, impossible due to memory constraints. Hence, there is a great demand for developing and deploying custom hardware accelerators to run these LLMs on the edge. Fortunately, FPGAs are ideal for designing new flexible and power-efficient accelerators that can take advantage of LLM optimizations, such as block floating point quantization, for edge inference. While some FPGA-based accelerators [9], [10] already exist for LLM inference on the edge, with constant changes to LLM architectures and optimizations, we are in need of new specialized FPGA-based accelerators for them.

To meet these demands of new and innovative FPGA-based accelerator architectures for LLM inference on the edge, we need ways to quickly prototype and evaluate LLM-based inference accelerators to reduce development costs and increase design space exploration. Hence, we propose SECDA-LLM, a new platform for designing, integrating and deploying specialized accelerators for LLMs on the edge. SECDA-LLM employs the SECDA design methodology [11], and similar to SECDA-TFLite [12], it provides the user with the ability to quickly prototype accelerator design with the target application framework, in this case, *llama.cpp* project. Our SECDA-LLM platform enables the designer to consider hardware-software co-design optimizations from different levels of LLM execution and makes deployment of LLMs through FPGA-based accelerators effortless. The contributions of this work are as follows:

- *SECDA-LLM*, a new design platform using the SECDA methodology which enables designing, integrating and deploying FPGA-based LLM accelerators for resource-constrained edge devices.
- A case study to demonstrate *SECDA-LLM*, where we prototype and deploy a new accelerator, with the focus on model-specific optimizations to accelerate the TinyLlama model [13], on the our target device, the PYNQ-Z1 [14].
- Evaluation of our initial accelerator design, where we evaluated the TinyLlama model [13] using our accelerator, achieving the initial speed of 20 seconds per token.

II. Background & Related Work

A. Large Language Models

LLMs are a family of models that use the Transformer [15] architecture as the key component, and are pre-trained on large amounts of language data. People usually use them by fine-tuning with downstream task-specific datasets. LLMs usually have large amounts of parameters. For example, Llama is designed to start with 7B parameters [16]. Also, many types of language-generating LLMs auto-regressively compute the next tokens (chunks of text) by using the previously cached information. This computation paradigm introduces a large amount of storage overheads called KV cache [17].

Quantization techniques are extensively employed to deploy parameter-intensive LLMs on resource-constrained devices. Utilizing 8-bit quantization facilitates the retention of model accuracy while diminishing the model size [18], [19]. Moreover, prior studies have demonstrated efforts in experimenting with 4-bit and 8-bit quantization methods to enable the operation of LLMs on edge devices while upholding accuracy levels [20].

Another quantization technique that can be considered is the block floating point (BFP) quantization; there have been some works [21] comparing the efficacy of block floating point quantization to traditional narrow precision quantization.

B. Inference framework: llama.cpp

llama.cpp [8] is a pure C/C++ library, with minimal external dependencies, for enabling LLMs inference on a wide range of hardware. Currently, *llama.cpp* supports a wide range of LLMs, including some multi-modal and custom-defined models. Additionally, it supports 1.5-bit, 2-bit, 3-bit, 4-bit, 5-bit, 6-bit, and 8-bit BFP quantization. In *llama.cpp* implementation, BFP quantization is leveraged to quantize the weights of the LLMs, and it includes few quantization variations. These variations are typically denoted as Qx_y, where x represents the number of bits per weight and y denotes the type of quantization.

llama.cpp [8] employs the GPT-Generated Unified Format model format (GGUF) to represent LLMs. Within this format, it is possible to represent the weights of an LLM with as few as 1.5 bits using BFP quantization. These quantized weights enable users to run the model on resource-constrained devices such as the Raspberry Pi and Pixel phone [22].

LLMs inference can be straightforward on resource-constrained devices equipped with CPUs or GPUs due to the optimized support provided by *llama.cpp* [8] for AVX, AVX2, and AVX-512 on x86 architectures as well as custom CUDA kernels for running on NVIDIA GPUs. However, LLM inference on FPGAs is not straightforward, as the design process for new FPGA-based accelerators has not been integrated with inference platforms like *llama.cpp* yet.

C. FPGA-based acceleration for LLMs

Researchers have previously described acceleration ideas for running language models on FPGAs [23]. The focus was generally to accelerate the workload of an LLM. Additionally, they have addressed the challenge of executing a wide variation of language models on FPGAs by proposing an overlay FPGA-based processor [9]. However, researchers have yet to focus on developing design flows and enabling explorations of new FPGA-based acceleration ideas for the latest language models. While transformer accelerator design that commences with the C/C++ code of an LLM exists [10], their primary focus lies on the accelerator design aspect, not necessarily emphasizing the design process itself.

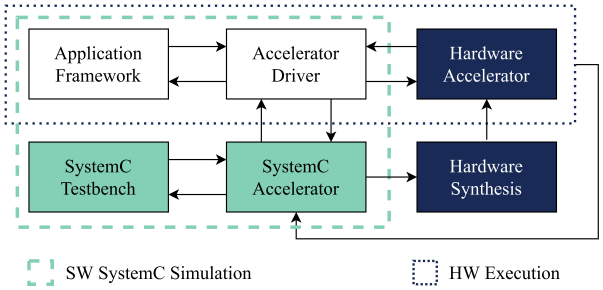

Fig. 1: Overview of the SECDA methodology [11]. Components in the dashed lines correspond to simulation, and in the dotted lines to execution on real hardware.

As such, we aimed to employ SECDA (SystemC Enabled Co-design of DNN Accelerators) 1, a hardware-software design methodology to efficiently produce optimized inference accelerators for edge devices using FPGAs. SECDA uses SystemC [24] as an accelerator simulation framework, allowing candidate designs to be efficiently iterated upon. Additionally, SECDA uses SystemC High-Level Synthesis (HLS) to produce a synthesizable design based on the same SystemC accelerator definition. One key aspect of SECDA is the full integration of the design process with the target application framework. For LLMs inference, *llama.cpp* is the ideal target application framework. With the integration of *llama.cpp* as the application framework, it becomes feasible to convert an LLM to the GGUF format and execute LLMs on edge FPGA-based platforms.

III. SECDA-LLM

SECDA-LLM is a specialized platform for creating FPGA-based LLM accelerators for edge devices using the SECDA methodology within the *llama.cpp* environment. Figure 2 outlines the main components of SECDA-LLM. The platform simplifies the accelerator design process by integrating the SECDA tools, thus allowing a seamless connection between the SECDA design environment and the target application framework, *llama.cpp*. This integration enables developers to begin prototyping and integrating their new designs with minimal setup costs.

The rest of this section provides details on SECDA-LLM and: i) how it is integrated with *llama.cpp*; ii) how it enables the accelerator designer to prototype and simulate new designs with SystemC [24] simulation; iii) the ease of hardware evaluation; iv) the profiling and performance analysis capabilities of SECDA-LLM.

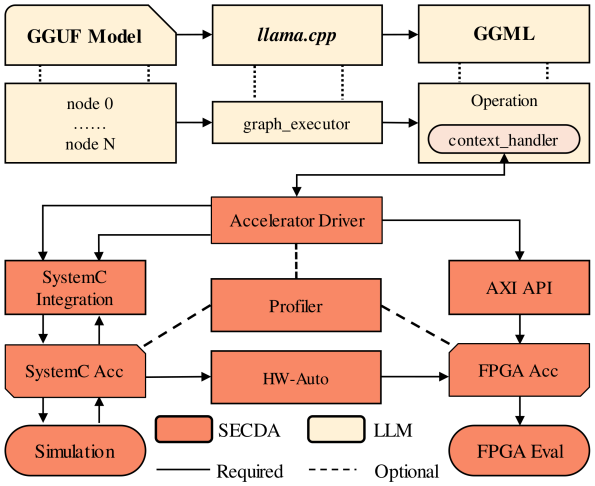

Fig. 2: Overview of the SECDA-LLM. Key SECDA components are highlighted in orange, and the LLM components are highlighted in beige.

A. Integration with llama.cpp

Figure 2 shows that the SECDA-LLM platform builds upon the core *llama.cpp* project inference. Our current integration is through *llama.cpp*'s main example project, which enables users to run LLM models with minimal overhead. We can connect into *llama.cpp* once it calls any of the *GGML*'s operations.

Depending on our target operation(s), we create additional connection points from the GGML library to the SECDA environment. During these connections, we ensure the creation of a context handle to pass from the GGML environment to the SECDA environment; the context handle includes pointers memory, memory-mapped model data, access to relevant inputs tensors, quantization, and layer parameters.

B. SECDA Environment

Within the SECDA environment, shown in Figure 2, the accelerator designer can start quickly prototyping the initial accelerator design and driver code. First, the user is required to create the initial driver, a simple C++ class that will gain access to the context handle provided by the offload call from within *GGML*. Second, the developer must create an initial SystemC description of their accelerator. Then, the user can instantiate their desired data communication channels between the driver and accelerator using data interfaces provided within the SECDA environment (e.g, AXI4-S, AXI-MM and AXI-Lite). The developer can use these data channels for SystemC end-to-end simulation.

C. SystemC Simulation

SystemC end-to-end simulation is a crucial step in the SECDA methodology; therefore, SECDA-LLM provides access to SystemC simulation. The simulation-based design loop is shown on the bottom left half of the figure 2. Once the driver and accelerator are connected through the desired data communication channels, the user can perform end-to-end simulations of LLMs using SECDA-LLM. With simulation enabled, the designer can quickly prototype new driver and accelerator features, verifying correctness, profiling performance and modeling control flow behavior within their design. The hardware developer is able to rapidly iterate through their design process, through end-to-end simulation, to meet their target performance.

D. Hardware Evaluation

With simulation-based evaluation, the designer can quickly make fast, broad design changes. Once satisfied with their design, the designer can quickly take their SystemC-defined design and perform High-level synthesis (HLS) and logic synthesis (HLX) through the hardware automation tool provided by SECDA-LLM to map it to their target FPGA, as shown on the bottom right of figure 2. Additionally, as SECDA-LLM is integrated with the *llama.cpp* project, we can leverage the *llama.cpp* project's compilation flow to generate pre-defined applications that use the LLMs through the *llama.cpp*'s interface. These generated applications will now have complete access to the driver and accelerator for execution on an FPGA-enabled device; see section IV-C for details.

A major benefit of the SECDA methodology, and therefore SECDA-LLM, is that we can reuse the driver and accelerator completely. For actual FPGA evaluation, the designer does not need to make any changes to the driver to enable real hardware execution, as the SECDA data interfaces switch between simulation and FPGA execution through a simple *"SYSC"* compiler flag. Once the accelerator is mapped to the target FPGA, the designer can evaluate its performance with their target applications.

E. Profiling

Through SECDA-LLM, we provide two types of profiling: simulation profiling and execution time profiling. The *profiler* module shown in figure 2 highlights how the profiling interacts with both the accelerator design and driver. Additionally, we are able to leverage any additional profiling tools enabled by the *llama.cpp* project.

1) Simulation profiling: End-to-end SystemC simulation can be used to quickly evaluate the potential performance impact of changes to the accelerator design's hardware and software components and verify the implementation's correctness. To profile the end-to-end simulation, the developer typically needs to add additional profiling code to keep track of hardware and software metrics throughout the end-to-end LLM inference. The *profiler* module provided within SECDA-LLM enables the quick and easy method to set up capture points to profile from the accelerator. The capture points can record different metrics of the accelerator and hardware submodules. Metrics include clock cycle counts and the dynamic utilization of processing elements and accelerator buffers.

2) Execution profiling: During the hardware evaluation, SECDA profiling provides execution time for the custom driver and accelerator. This type of profiling helps the designer

understand the performance bottlenecks caused by driver-accelerator interactions. For instance, a designer may opt to profile time spent: i) Sending input data; ii) Waiting for the accelerator to execute operations; iii) Unpacking output data received from the accelerator. The analysis of these detailed execution time breakdowns can motivate both accelerator and driver design choices. Additionally, execution profiling can be used during a simulation run to profile driver execution times, which can be combined with SystemC-reported simulation times for the accelerator. This would estimate end-to-end execution time in terms of both CPU and accelerator.

IV. Case Study

To demonstrate our SECDA-LLM platform and how it provides a quick and efficient design flow for developing LLM accelerators for edge devices using the SECDA methodology, we develop a new custom FPGA-based accelerator for block floating point (BFP) quantized LLM inference.

A. Target Problem

For our case study, we target the MatMul operations for acceleration as they are the most computationally expensive in LLMs. Specifically, we accelerate the dot product kernel for BFP quantized MatMul. For our design, we planned to accelerate the GGML's *vec_dot_Q3_K_Q8_K* kernel, which is the core computation within BFP quantized MatMul; this kernel uses 3-bit and 8-bit weight and input BFP quantization, respectively.

Both weights and inputs are stored in what is called "super-blocks" (SBs); these SBs are critical in maintaining LLM accuracy by adjusting mathematical scaling during computation. With the *Q3_K* format used for weights, each SB can represent 256 weights (N_w), where the SB is partitioned into 16 tiles (N_{tiles}) and each tile contains a scaling factor (6-bits) and 16 weights (3-bits); additionally, each SB has one super-scaling factor (16-bits)), which equates to 3.5$\sim$ bits-per-weight.

With the *Q8_K* format, used for inputs, each SB contains 256 inputs (8-bits) and a single super-scaling *SSF* (16-bits), which equates to 8$\sim$ bits-per-input.

B. Accelerator Design

Our accelerator design, shown in figure 3, contains an*instruction decoder*, a *data mapper*, a *scheduler* and four *vector processing units*. The *instruction decoder* loads and decodes instructions from the AXI-Stream and then communicates the instruction throughout the rest of the accelerator. The *data mapper* parses the incoming data stream and maps the weight and input super blocks into their respective weight and input buffers. Our mapping scheme enables efficient data access, so the *vector processing units* can compute without stalling the computation pipeline. The *vector processing units* efficiently compute the dot product between the SB of weights and inputs while scaling the computation according to the SB scaling factors. The *scheduler* allocates SBs to each processing unit to process. Additionally, it synchronizes and accumulates the output data produced by the processing units and sends the results back to the main memory using the AXI-Stream.

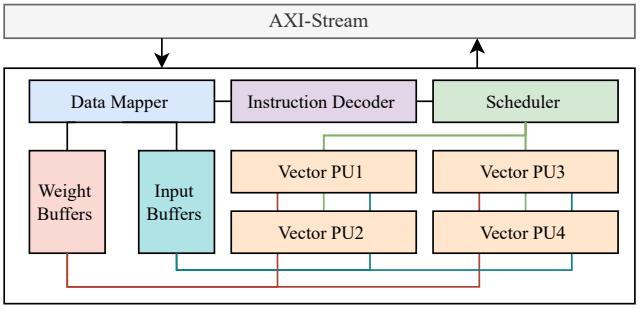

Fig. 3: Overview of our block floating point quantized accelerator design for GGML's *vec_dot_Q3_K_Q8_K* kernel.

C. Evaluation

We evaluate our accelerator design on the PYNQ-Z1 board [14], which contains a Xilinx Z020 edge FPGA and a 650MHz dual-core ARM Cortex-A9 CPU. We evaluate the TinyLlama model [13] with 1.1B parameters trained on the Guanaco dataset [25]. This model contains various BFP quantization levels, but most layers are quantized to *Q3_K*. Note that with *llama.cpp* you can apply different levels of quantization to reduce model size as required.

For our experiments, we use the *llama.cpp* project's 'main' program cross-compiled for our target CPU architecture, ARMv7a, with Neon vector instructions enabled alongside our accelerator driver. We execute the TinyLlama model (460 MB) utilizing our FPGA-mapped accelerator to obtain an initial speed of 20 seconds per token ($\sim$27 seconds per word). While this performance is just on par with the CPU for now, running TinyLlama on such a resource-constrained device was not possible a few months ago, and we expect to improve it in the future. With SECDA-LLM, we were able to implement, integrate and evaluate a new design quickly

V. Conclusion

We introduced SECDA-LLM, a novel platform that simplifies the creation of new FPGA-based hardware accelerators for edge LLM inference using the hardware/software co-design SECDA methodology within *llama.cpp*. SECDA-LLM removes the initial setup within the *llama.cpp* framework and tools to streamline accelerator design development. By integrating the SECDA tools with *llama.cpp*, developers can effectively co-design new accelerators for LLMs with ease using the SECDA design flows.

Our approach using the SECDA methodology reduces development costs by minimizing synthesis iterations and replacing them with simulation iterations using SystemC for fine-grained benchmarking and co-verification. Furthermore, designs can be directly deployed onto FPGAs from simulations, eliminating the need for re-implementation. As a case study, we presented a quantized MatMul-based accelerator design that optimizes LLM inference for the TinyLlama model. Future work will expand SECDA-LLM into an open-source platform for collaborative development and continuous improvement of LLMs' performance on edge devices.

REFERENCES

[1] T. B. Brown *et al.* Language Models are Few-Shot Learners. [Online]. Available: http://arxiv.org/abs/2005.14165

[2] P. P. Ray, "ChatGPT: A comprehensive review on background, applications, key challenges, bias, ethics, limitations and future scope," vol. 3, pp. 121–154. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S266734522300024X

[3] B. Yao *et al.* Benchmarking LLM-based Machine Translation on Cultural Awareness. [Online]. Available: http://arxiv.org/abs/2305.14328

[4] X. Sun *et al.* Text Classification via Large Language Models. [Online]. Available: http://arxiv.org/abs/2305.08377

[5] J. Liu *et al.*, "Is Your Code Generated by ChatGPT Really Correct? Rigorous Evaluation of Large Language Models for Code Generation," vol. 36, pp. 21 558–21 572. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2023/hash/43e9d647ccd3e4b7b5baab53f0368686-Abstract-Conference.html

[6] J. Clusmann *et al.*, "The future landscape of large language models in medicine," vol. 3, no. 1, pp. 1–8. [Online]. Available: https://www.nature.com/articles/s43856-023-00370-1

[7] "GGML," https://github.com/ggerganov/ggml, 2024.

[8] "llama.cpp," https://github.com/ggerganov/llama.cpp, 2024.

[9] H. Khan *et al.*, "NPE: An FPGA-based Overlay Processor for Natural Language Processing," in *The 2021 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA '21)*. New York, NY, USA: Association for Computing Machinery, Feb. 2021, p. 227.

[10] S. Lu *et al.*, "Hardware Accelerator for Multi-Head Attention and Position-Wise Feed-Forward in the Transformer," in *2020 IEEE 33rd International System-on-Chip Conference (SOCC)*. Las Vegas, NV, USA: IEEE, Sep. 2020, pp. 84–89.

[11] J. Haris *et al.*, "SECDA: Efficient Hardware/Software Co-Design of FPGA-based DNN Accelerators for Edge Inference," in *2021 IEEE 33rd International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, pp. 33–43. [Online]. Available: https://ieeexplore.ieee.org/document/9651579

[12] J. Haris *et al.*, "SECDA-TFLite: A toolkit for efficient development of FPGA-based DNN accelerators for edge inference," vol. 173, pp. 140–151. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0743731522002301

[13] P. Zhang *et al.* TinyLlama: An Open-Source Small Language Model. [Online]. Available: http://arxiv.org/abs/2401.02385

[14] "PYNQ Z1 Manual," https://reference.digilentinc.com/reference/programmable-logic/pynq-z1/reference-manual, 2020.

[15] A. Vaswani *et al.*, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[16] H. Touvron *et al.*, "Llama: Open and efficient foundation language models," *arXiv preprint arXiv:2302.13971*, 2023.

[17] W. Kwon *et al.*, "Efficient memory management for large language model serving with pagedattention," in *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*, 2023.

[18] O. Zafrir *et al.*, "Q8BERT: Quantized 8Bit BERT," in *2019 Fifth Workshop on Energy Efficient Machine Learning and Cognitive Computing - NeurIPS Edition (EMC2-NIPS)*. IEEE Computer Society, Dec. 2019, pp. 36–39.

[19] L. J. Wan *et al.*, "Software/Hardware Co-design for LLM and Its Application for Design Verification," in *2024 29th Asia and South Pacific Design Automation Conference (ASP-DAC)*. Incheon, Korea, Republic of: IEEE, Jan. 2024, pp. 435–441.

[20] X. Shen et al., "EdgeQAT: Entropy and Distribution Guided Quantization-Aware Training for the Acceleration of Lightweight LLMs on the Edge," http://arxiv.org/abs/2402.10787, 2024, accessed: Apr. 18, 2024.

[21] B. D. Rouhani et al., "Microscaling Data Formats for Deep Learning," https://arxiv.org/abs/2310.10537, 2023, accessed: October 19, 2023.

[22] Simon Willison, "Alpaca: A New Programming Language for Data Analysis," https://simonwillison.net/2023/Mar/13/alpaca/, 2023, accessed: March 13, 2023.

[23] C. Kachris, "A Survey on Hardware Accelerators for Large Language Models," http://arxiv.org/abs/2401.09890, 2024, accessed: Apr. 17, 2024.

[24] IEEE, "IEEE Standard for Standard SystemC Language Reference Manual," *IEEE Std 1666-2011*, 2012.

[25] Joséphus Cheung, "Guanacodataset (revision 892e57a)," 2023. [Online]. Available: https://huggingface.co/datasets/JosephusCheung/GuanacoDataset