Optimizing LLaMA2.C: Quantization , LLVM and GCC Compiler , Cross-Compilation,OpenMP (Open Multi-Processing) , and Deployment on Raspberry Pi and RISC-V Hardware

A Project Report Submitted by

Anish Maurya,Varun Senthil

Jaitra Software Solutions Private Ltd

Project

August, 2024

Mentors

This is to certify that the Project Report titled Optimizing LLaMA2.C: Quantization, Cross-Compilation, and Deployment on Raspberry Pi and RISC-V Hardware, submitted by Anish Maurya and Varun Senthil to Jaitra Software Solutions Private Ltd., is a bona fide record of the research work carried out by them under my supervision.

Mr .Anand Joshi , Dr Anand Krishnan.

Acknowledgements

We would like to express our sincere appreciation to Mr.Anand Joshi and Dr.Anand Krishanan, R & D Department at the Jaitra Software Solutions Private Ltd, for their continuous guidance and support throughout the various stages of our Internship dissertation project. Their motivation and insightful feedback have been instrumental in shaping the trajectory of research.

Finally, I acknowledge the divine grace of GOD, the almighty, for providing me with the inner strength and wisdom to navigate the challenges of this ongoing research work as I approach the final submission.

Abstract

This report introduces a comprehensive, minimalist solution for training and inferring the Llama 2 language model architecture using pure C. By distilling the Llama 2 model, originally implemented in PyTorch, into a concise 700-line C file (run.c), this project demonstrates the viability of using small-scale language models for specialized tasks. Contrary to the prevailing notion that effective language models require billions of parameters, our approach highlights the strong performance potential of smaller models in narrowly defined domains, as supported by findings in the TinyStories paper.

This project also supports the inference of Meta's Llama 2 models, with the current implementation focused on fp32 precision. Future developments aim to expand capabilities through ongoing work in model quantization, including experiments with Int8, Int4, and Int2 precision formats.

Furthermore, the report details the deployment of these models on diverse hardware platforms, such as Raspberry Pi and RISC-V boards, utilizing compilers like LLVM and GCC across both x86 architecture and dedicated hardware environments. Performance optimization efforts include the use of OpenMP, particularly in the context of the 1.58 Bitnet model.

Keywords: llama2.c , LLVM Compiler , GCC Compiler , Inference , Prompt , Tok/s , Model Size , Quantization , Int8 ,Int4, Int2 ,1.58 BitNet ,Raspberry Pi , RISC-V , Optimizing , Cross-Compilation, Deployment , Hardware .

Contents

List of Figures

List of Tables

Optimizing LLaMA2.C: Quantization , LLVM and GCC Compiler , Cross-Compilation,OpenMP (Open Multi-Processing) , and Deployment on Raspberry Pi and RISC-V Hardware

1 Introduction and Resources

This report presents a dual-phase approach to implementing and deploying the Llama 2 language model using pure C. The project is divided into two key stages:

- The first focuses on development and testing on an x64-based PC environment using Ubuntu 22.04 via Windows Subsystem for Linux (WSL).
- The second phase involves deploying the model on hardware platforms, including Raspberry Pi and RISC-V boards.

1.1 GitHub Repository

We used the following GitHub repository for our work on the project. Please follow this resource for further stages: `https://github.com/karpathy/llama2.c.git`

1.2 Model Link

You can download the required Model using the following links:

- `https://huggingface.co/karpathy/tinyllamas/resolve/main/stories15M.bin`
- `https://huggingface.co/karpathy/tinyllamas/resolve/main/stories42M.bin`
- `https://huggingface.co/karpathy/tinyllamas/resolve/main/stories110M.bin`
- `https://huggingface.co/karpathy/tinyllamas/resolve/main/stories15M.pt`
- `https://huggingface.co/karpathy/tinyllamas/resolve/main/stories42M.pt`
- `https://huggingface.co/karpathy/tinyllamas/resolve/main/stories110M.pt`

1.3 Dataset Link

The model is trained on data from the following source:

`https://huggingface.co/datasets/roneneldan/TinyStories/resolve/main/TinyStories_all_data.tar.gz`

2 Specification of x64-based PC and Hardware Board

2.1 Laptop Specifications

Please Note that the results may vary based on the x64-based PC configuration used. For instance, the presence of a high-power x64-based PC may lead to differences in performance(tok/S) and outcomes.

- **OS Name:** Microsoft Windows 11 Home Single Language
- **System Name:** IDEAPADGAMING3
- **System Manufacturer:** LENOVO
- **System Type:** x64-based PC
- **Processor:** 11th Gen Intel(R) Core(TM) i5-11320H @ 3.20GHz, 3187 MHz, 4 Core(s), 8 Logical Processor(s)
- **Installed Physical Memory (RAM):** 16.0 GB

2.2 Specifications Hardware Board

2.2.1 Raspberry Pi 4 B

The Raspberry Pi 4 Model B is a compact and versatile single-board computer designed for a wide range of applications. It features a quad-core ARM Cortex-A72 processor running at 1.5 GHz, providing robust performance for various tasks. The board includes up to RAM, multiple USB ports, and dual micro-HDMI outputs, enabling connectivity with various peripherals and displays. It also has built-in Gigabit Ethernet and wireless networking capabilities, enhancing its connectivity options. The Raspberry Pi 4 Model B is popular for educational purposes, DIY projects, and as a low-cost computing solution, thanks to its balance of power, flexibility, and affordability.

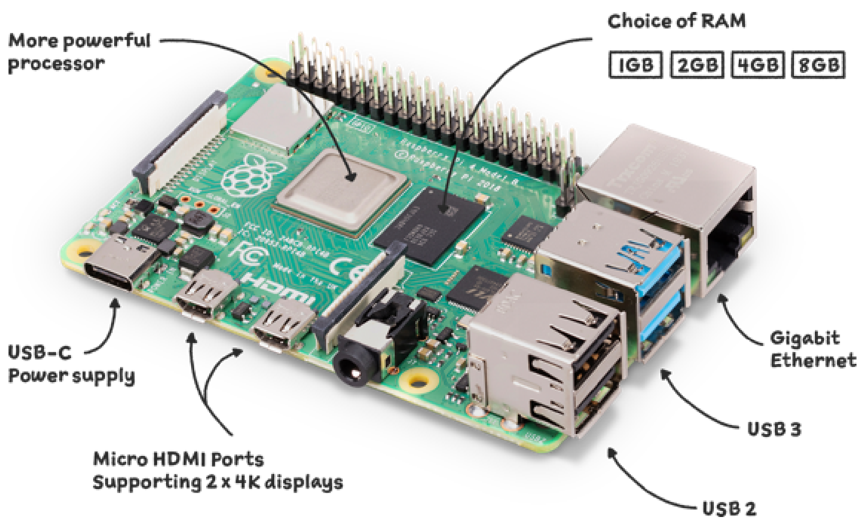

Figure 2.1: Raspberry Pi4 model B.

Component	Specification
Processor/SOC	StarFive JH7110 64-bit SoC with RV64GC, up to 1.5GHz
CPU Clock Speed	1.5 GHz
RAM	4GB LPDDR4 SDRAM
GPU	Video Core VI
Video Outputs	2 × micro-HDMI (up to 4K@60Hz)
USB Ports	2 × USB 3.0, 2 × USB 2.0
Ethernet	Gigabit Ethernet
Wireless	2.4 GHz and 5 GHz IEEE 802.11ac wireless, Bluetooth 5.0
Storage	microSD card slot
GPIO Pins	40 GPIO pins
Power Supply	5V/3A via USB-C
Operating System	Raspberry Pi OS (formerly Raspbian), various

Table 2.1: Raspberry pi 4 B Specifications

2.2.2 RISC-V

RISC-V hardware operates on a straightforward RISC (Reduced Instruction Set Computing) architecture designed to be efficient and flexible. It features a modular setup with a core set of instructions and additional extensions, making it versatile for different applications. The processors utilize pipelining, which allows them to handle multiple instructions at once, improving speed. The design uses a load-store model where only specific instructions interact with memory, simplifying the overall process. RISC-V processors have a large number of general-purpose registers, reducing the need to frequently access memory. This approach balances ease of implementation with performance, making it adaptable for various computing needs.

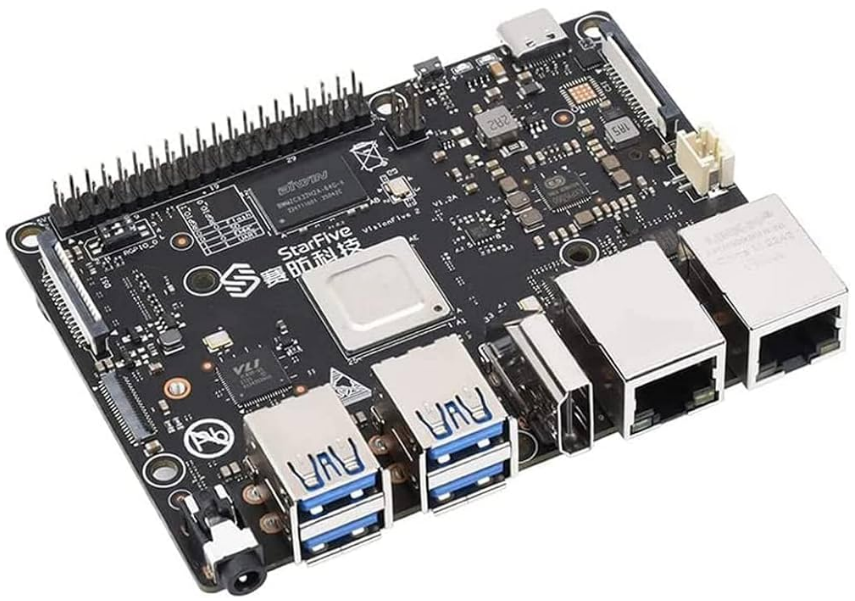

Figure 2.2: RISC-V Board.

Item	**Description**
Processor/SOC	StarFive JH7110 64-bit SoC with RV64GC, up to 1.5GHz
Memory	LPDDR4, 2GB/4GB/8GB
Storage	TF card slot ,Flash for Uboot
Video output	HDMI 2.0 , MIPI-DSI
Multimedia	Camera with MIPI CSI, up to 4k@30fps , H.264 & H.265 4k@60fps Decoding
Connectivity	2x RJ45 Gigabit Ethernet ,2x USB2.0 + 2x USB 3.0 , M.2 M-Key
Power	USB-C port, 5V DC via USB-C with PD, up to 30W , GPIO Power in, 5V DC via GPIO header
GPIO	40 pin GPIO header
Dimensions	100 x 72mm
Compliance	RoHS, FCC, CE
Button	Reset Button
Other	Debug Pin Headers

Table 2.2: RISC-V Specifications

3 Command Line Installation to Perform the Experiment

3.1 Perform experiment on x64-based PC

3.1.1 Installation GCC Compiler

Before installing GCC Compiler first activate in PC Windows System for linux(wsl) and Ubuntu and setup in PC.

GCC Command line:

```
sudo apt-get update
sudo apt-get install build-essential
```

Now clone the Dr.karpathy GitHub Reposatory . First, navigate to the folder where you keep your projects .

Clone repository from GitHub

```
git clone https://github.com/karpathy/llama2.c.git
cd llama2.c
```

3.2 Inference Using run.c file & .bin

3.2.1 The -O3 optimization level in GCC is commonly called aggressive optimization

Step:1 Download the 15M & 42M & 110M Parameter model

```
wget https://huggingface.co/karpathy/tinyllamas/resolve/main/stories15M.bin
wget https://huggingface.co/karpathy/tinyllamas/resolve/main/stories42M.bin
wget https://huggingface.co/karpathy/tinyllamas/resolve/main/stories110M.bin
```

Step:2 Compile the source code run.c

```
make run
or we can use
gcc -O3 -o run run.c -lm
```

Step:3 Inference

```
./run stories15M.bin -t 0.8 -n 256 -i "One day, Lily met a Shoggoth"
./run stories42M.bin -t 0.8 -n 256 -i "One day, Lily met a Shoggoth"
./run stories110M.bin -t 0.8 -n 256 -i "One day, Lily met a Shoggoth"
```

3.2.2 The `-Ofast` optimization level in GCC is commonly called aggressive, non-standard-compliant optimization

Step:1 Download the 15M & 42M & 110M Parameter model

```
wget https://huggingface.co/karpathy/tinyllamas/resolve/main/stories15M.bin
wget https://huggingface.co/karpathy/tinyllamas/resolve/main/stories42M.bin
wget https://huggingface.co/karpathy/tinyllamas/resolve/main/stories110M.bin
```

Step:2 Compile the source code run.c

```
 gcc -Ofast -o run run.c -lm
```

Step:3 Inference

```
./run stories15M.bin -t 0.8 -n 256 -i "One day, Lily met a Shoggoth"
./run stories42M.bin -t 0.8 -n 256 -i "One day, Lily met a Shoggoth"
./run stories110M.bin -t 0.8 -n 256 -i "One day, Lily met a Shoggoth"
```

Conclusion :

By using -O3 optimization level and -Ofast optimization level using run.c file. There is very significance change in speed (tok/s) during inference of text. Detail information shred in Table Below.

3.3 Inference Using runq.c (Int8 Quantisation) file & .pt & Int8 Quantisation

3.3.1 The `-O3` optimization level in GCC is commonly called aggressive optimization

Step:1 Download the 15M & 42M & 110M Parameter model

```
wget https://huggingface.co/karpathy/tinyllamas/resolve/main/stories15M.pt
wget https://huggingface.co/karpathy/tinyllamas/resolve/main/stories42M.pt
wget https://huggingface.co/karpathy/tinyllamas/resolve/main/stories110M.pt
```

Step:2 Compile the source code run.c

```
 make run
 or we can use
 gcc -O3 -o runq runq.c -lm
```

Step:3 Quantize the model

```
model_file="stories15M.pt"
cmd="python3 export.py ${model_file} --version 2 --checkpoint ${model_file}"
eval $cmd
 model_file="stories42M.pt"
cmd="python3 export.py ${model_file} --version 2 --checkpoint ${model_file}"
eval $cmd
 model_file="stories110M.pt"
cmd="python3 export.py ${model_file} --version 2 --checkpoint ${model_file}"
eval $cmd
```

Step:4 Inference

```
./runq stories15M.pt -t 0.8 -n 256 -i "One day, Lily met a Shoggoth"
./runq stories42M.pt -t 0.8 -n 256 -i "One day, Lily met a Shoggoth"
./runq stories110M.pt -t 0.8 -n 256 -i "One day, Lily met a Shoggoth"
```

3.3.2 The `-Ofast` optimization level in GCC is commonly called aggressive, non-standard-compliant optimization

Step:1 Download the 15M & 42M & 110M Parameter model

```
wget https://huggingface.co/karpathy/tinyllamas/resolve/main/stories15M.pt
wget https://huggingface.co/karpathy/tinyllamas/resolve/main/stories42M.pt
wget https://huggingface.co/karpathy/tinyllamas/resolve/main/stories110M.pt
```

Step:2 Compile the source code run.c

```
 gcc -Ofast -o runq runq.c -lm
```

Step:3 Quantize the model

```
model_file="stories15M.pt"
cmd="python3 export.py ${model_file} --version 2 --checkpoint ${model_file}"
eval $cmd
 model_file="stories42M.pt"
cmd="python3 export.py ${model_file} --version 2 --checkpoint ${model_file}"
eval $cmd
 model_file="stories110M.pt"
cmd="python3 export.py ${model_file} --version 2 --checkpoint ${model_file}"
eval $cmd
```

Step:4 Inference

```
./run stories15M.pt -t 0.8 -n 256 -i "One day, Lily met a Shoggoth"
./run stories42M.pt -t 0.8 -n 256 -i "One day, Lily met a Shoggoth"
./run stories110M.pt -t 0.8 -n 256 -i "One day, Lily met a Shoggoth"
```

Conclusion :

By using -O3 optimization level and -Ofast optimization level using runq.c file after the Optimization of model. There is no significance change in speed (tok/s) during inference of text. Detail information shred in Table Below.

3.4 Inference Using int4.c (Int4 Quantisation) file & .pt & Int4 Quantisation

3.4.1 The `-O3` optimization level in GCC is commonly called aggressive optimization

Step:1 Download the 15M & 42M & 110M Parameter model

```
wget https://huggingface.co/karpathy/tinyllamas/resolve/main/stories15M.pt
wget https://huggingface.co/karpathy/tinyllamas/resolve/main/stories42M.pt
wget https://huggingface.co/karpathy/tinyllamas/resolve/main/stories110M.pt
```

Step:2 Compile the source code run.c

```
 gcc -O3 -o int4 int4.c -lm
```

Step:3 Quantize the model

```
model_file="stories15M.pt"
cmd="python3 export_int4.py ${model_file} --version 4 --checkpoint ${model_file}"
eval $cmd
 model_file="stories42M.pt"
cmd="python3 export_int4.py ${model_file} --version 4 --checkpoint ${model_file}"
eval $cmd
 model_file="stories110M.pt"
cmd="python3 export_int4.py ${model_file} --version 4 --checkpoint ${model_file}"
eval $cmd
```

Step:4 Inference

```
./int4 stories15M.pt -t 0.8 -n 256 -i "One day, Lily met a Shoggoth"
./int4 stories42M.pt -t 0.8 -n 256 -i "One day, Lily met a Shoggoth"
./int4 stories110M.pt -t 0.8 -n 256 -i "One day, Lily met a Shoggoth"
```

3.4.2 The `-Ofast` optimization level in GCC is commonly called aggressive, non-standard-compliant optimization

Step:1 Download the 15M & 42M & 110M Parameter model

```
wget https://huggingface.co/karpathy/tinyllamas/resolve/main/stories15M.pt
wget https://huggingface.co/karpathy/tinyllamas/resolve/main/stories42M.pt
wget https://huggingface.co/karpathy/tinyllamas/resolve/main/stories110M.pt
```

Step:2 Compile the source code run.c

```
 gcc -Ofast -o int4 int4.c -lm
```

Step:3 Quantize the model

```
model_file="stories15M.pt"
cmd="python3 export_int4.py ${model_file} --version 4 --checkpoint ${model_file}"
eval $cmd
 model_file="stories42M.pt"
cmd="python3 export_int4.py ${model_file} --version 4 --checkpoint ${model_file}"
eval $cmd
 model_file="stories110M.pt"
cmd="python3 export_int4.py ${model_file} --version 4 --checkpoint ${model_file}"
eval $cmd
```

Step:4 Inference

```
./int4 stories15M.pt -t 0.8 -n 256 -i "One day, Lily met a Shoggoth"
./int4 stories42M.pt -t 0.8 -n 256 -i "One day, Lily met a Shoggoth"
./int4 stories110M.pt -t 0.8 -n 256 -i "One day, Lily met a Shoggoth"
```

3.5 Inference Using int2.c (Int2 Quantisation) file & .pt & Int4 Quantisation

3.5.1 The `-O3` optimization level in GCC is commonly called aggressive optimization

Step:1 Download the 15M & 42M & 110M Parameter model

```
wget https://huggingface.co/karpathy/tinyllamas/resolve/main/stories15M.pt
wget https://huggingface.co/karpathy/tinyllamas/resolve/main/stories42M.pt
wget https://huggingface.co/karpathy/tinyllamas/resolve/main/stories110M.pt
```

Step:2 Compile the source code run.c

```
 gcc -O3 -o int2 int2.c -lm
```

Step:3 Quantize the model

```
model_file="stories15M.pt"
cmd="python3 export_int2.py ${model_file} --version 4 --checkpoint ${model_file}"
eval $cmd
 model_file="stories42M.pt"
cmd="python3 export_int2.py ${model_file} --version 4 --checkpoint ${model_file}"
eval $cmd
 model_file="stories110M.pt"
cmd="python3 export_int2.py ${model_file} --version 4 --checkpoint ${model_file}"
eval $cmd
```

Step:4 Inference

```
./int2 stories15M.pt -t 0.8 -n 256 -i "One day, Lily met a Shoggoth"
./int2 stories42M.pt -t 0.8 -n 256 -i "One day, Lily met a Shoggoth"
./int2 stories110M.pt -t 0.8 -n 256 -i "One day, Lily met a Shoggoth"
```

3.5.2 The `-Ofast` optimization level in GCC is commonly called aggressive, non-standard-compliant optimization

Step:1 Download the 15M & 42M & 110M Parameter model

```
wget https://huggingface.co/karpathy/tinyllamas/resolve/main/stories15M.pt
wget https://huggingface.co/karpathy/tinyllamas/resolve/main/stories42M.pt
wget https://huggingface.co/karpathy/tinyllamas/resolve/main/stories110M.pt
```

Step:2 Compile the source code run.c

```
 gcc -Ofast -o int2 int2.c -lm
```

Step:3 Quantize the model

```
model_file="stories15M.pt"
cmd="python3 export_int2.py ${model_file} --version 4 --checkpoint ${model_file}"
eval $cmd
 model_file="stories42M.pt"
cmd="python3 export_int2.py ${model_file} --version 4 --checkpoint ${model_file}"
eval $cmd
 model_file="stories110M.pt"
cmd="python3 export_int2.py ${model_file} --version 4 --checkpoint ${model_file}"
eval $cmd
```

Step:4 Inference

```
./int2 stories15M.pt -t 0.8 -n 256 -i "One day, Lily met a Shoggoth"
./int2 stories42M.pt -t 0.8 -n 256 -i "One day, Lily met a Shoggoth"
./int2 stories110M.pt -t 0.8 -n 256 -i "One day, Lily met a Shoggoth"
```

subsectionTable Above Content

3.6 Different compiler

3.6.1 using -O3 optimization level

Comparison of tok/s on PC using -O3 optimization level in GCC				
Parameter	**Base Model**	**INT8**	**INT4**	**INT2**
15M	70.750647	273.605150	81.433225	58.033682
42M	24.919378	125.292740	28.765481	23.874169
110M	9.437454	50.897323	11.157099	8.901766

Table 3.1: Token Processing Speed Across Different Quantization Levels

3.6.2 Using OpenMP

- **OpenMP** is a parallel programming model for shared-memory systems, allowing multi-core processors to execute code in parallel. It uses compiler directives (pragmas) to define parallel regions and distribute work among threads.

We only need to change in step 2, other command line and step is same.

```
gcc -Ofast -fopenmp -o run_OPMP run.c -lm
gcc -Ofast -fopenmp -o runq_OPMP runq.c -lm
gcc -Ofast -fopenmp -o int4_OPMP int4.c -lm
gcc -Ofast -fopenmp -o int2_OPMP int2.c -lm
```

Token Processing Speed Across Different Quantization Levels				
Parameter	**Base Model**	**INT8**	**INT4**	**INT2**
15M	250.46	686.66	200-270	100-147
42M	92	306.10	70-110	76
110M	40	122.23	34-40	32

Table 3.2: Token Processing Speed Across Different Quantization Levels

4 Inference Using LLVM Compiler on x86 machine

5 LLVM Cross Compiler for RISC-V on x86 Machine

6 LLVM Cross Compiler for ARM on x86 Machine

7 Quantization Error

Max Quantization Group Error across All Weights specifically measures the maximum error observed across all quantization groups (i.e., clusters of weights that are quantized together) in a model.

Max Quantization group error across all weights			
Parameter	**INT8**	**INT4**	**INT2**
15M	0.0023	0.044	0.300
42M	0.00467	0.0846	0.577
110M	0.0075	0.11	0.568

Table 7.1: Max Quantization group error across all weights

8 Model size

The "Comparison of Model Size on Quantization" examines how the size of a model changes when its weights are compressed using different quantization levels (INT8, INT4, INT2).

Comperision of model Size on Quantization (MB)				
Parameter	**Base Model**	**INT8**	**INT4**	**INT2**
15M	58.00	16.31	9.1	5.5
42M	159.28	42.27	23	13
110M	418.07	111.04	59	33

Table 8.1: Model Size Across Different Quantization Levels

9 Perform experiment on Hardware

9.1 RISC-V Hardware

Comparison of tok/s on RISC V board				
Parameter	**Base Model**	**INT8**	**INT4**	**INT2**
15M	13.67	11.24	8.94	10.07
42M	5.21	5.2	3.57	4.43
110M	2.04	1.84	1.40	1.84

Table 9.1: Token Processing Speed Across Different Quantization Levels on RISC V Board

9.2 Raspberry pi 4 B Hardware

Comparison of tok/s on Raspberry pi 4 board				
Parameter	**Base Model**	**INT8**	**INT4**	**INT2**
15M	31.10	80.77	27.95	54.72
42M	11.61	33.95	10.40	17.86
110M	4.49	14.14	4.01	5.96

Table 9.2: Token Processing Speed Across Different Quantization Levels

10 BitNet 1.58 Model

References