Lab AI EDGE with Rock Pi 5

Table of Contents

Introduction

In this lab we are going to experiment with several AI applications running on Rock Pi 5. Rock Pi 5 (A/B) are build around the RK3588 SoC integrating 8 ARM cores (4*A76 and 4*A55) and an NPU accelerator (6 ToPS on 4-bit).
“RK3588 supports H.265 and VP9 decoder by 8K@60fps, H.264 decoder by 8K@30fps, and AV1 decoder by 4K@60fps, also support H.264 and H.265 encoder by 8K@30fps, high-quality JPEG encoder/decoder, specialized image preprocessor and postprocessor.
Embedded 3D GPU makes RK3588 completely compatible with OpenGLES 1.1, 2.0, and 3.2, OpenCL up to 2.2 and Vulkan1.2. Special 2D hardware engine with MMU will maximize display performance and provide very smoothly operation.
RK3588 introduces a new generation totally hardware-based maximum 48-Megapixel ISP (image signal processor). It implements a lot of algorithm accelerators, such as HDR, 3A, LSC, 3DNR, 2DNR, sharpening, dehaze, fisheye correction, gamma correction and so on.
The build-in NPU supports INT4/INT8/INT16/FP16 hybrid operation and computing power is up to 6TOPs. In addition, with its strong compatibility, network models based on a series of frameworks such as TensorFlow/MXNet/PyTorch/Caffe can be easily converted.
RK3588 has high-performance quad channel external memory interface (LPDDR4/LPDDR4X/LPDDR5) capable of sustaining demanding memory bandwidths, also provides a complete set of peripheral interface to support very flexible applications.”

RK3588 SoC is manufactured with **8nm** process and implements V8.2 ARM architecture. Version 8.2-A includes enhanced memory model, half-precision floating point data processing and introduces both RAS (reliability availability serviceability) support and statistical profiling extension (SPE).

These features allows as to implement several kinds of AI applications exploiting the NPU and vector processing on ARM cores.

1. NPU unit

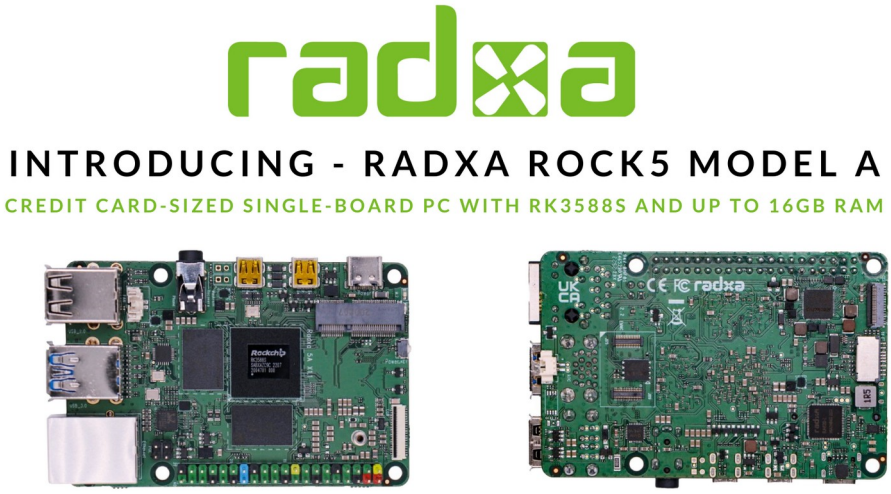
RK3588 has a NPU(*Neural Process Unit*) that Neural network acceleration engine with processing performance up to 6 TOPS. Using this NPU module needs to download RKNN SDK which provides programming interfaces for RK3588S/RK3588 chip platforms with NPU. This SDK can help users deploy RKNN models exported by RKNN-Toolkit2 and accelerate the implementation of AI applications.

The RKNN model can run directly on the RK3588 platform. There are demos under **`rknpu2/examples`**.
Refer to the `README.md` to compile Linux Demo (Need cross-compile environment).

You can also just download compiled demo.

For more inforamtion about the preparation of your models to run on RK3588 NPU:

https://wiki.t-firefly.com/en/ROC-RK3588S-PC/usage_npu.html



1.1 Rokchip NPU libraries and examples

The directory **examples** contains several demo programs:

```
rock@rock-5a:~/ptech/rknpu2-main/examples$ ls
3rdparty   rknn_api_demo     rknn_mobilenet_demo       rknn_ssd_demo     RV1106_RV1103
README.md  rknn_common_test  rknn_multiple_input_demo  rknn_yolov5_demo
```

1.1.1 Go to examples and find `rknn_mobilenet_demo`

```
rock@rock-5a:~/ptech/rknpu2-main/examples/rknn_mobilenet_demo$ ls
build                    build-android_RK3588.sh  build-linux_RK3588.sh  install  README.md
build-android_RK356X.sh  build-linux_RK356X.sh    CMakeLists.txt         model    src
```

1.1.2 Then build your application with: build-linux_RK3588.sh

Execute `rknn_mobilenet_demo` with the RKNN2 model `mobilenet_v1.rknn` and image examples: `cat_224x224.jpg, dog`

```
rock@rock-5a:~/ptech/rknpu2-main/examples/rknn_mobilenet_demo/build/build_linux_aarch64$
./rknn_mobilenet_demo ../../model/RK3588/mobilenet_v1.rknn ../../model/cat_224x224.jpg
rock@rock-5a:~/ptech/rknpu2-main/examples/rknn_mobilenet_demo/build/build_linux_aarch64$
./rknn_mobilenet_demo ../../model/RK3588/mobilenet_v1.rknn ../../model/dog_224x224.jpg
```

Examples of execution/inference

```
rock@rock-5a:~/ptech/rknpu2-main/examples/rknn_mobilenet_demo/build/build_linux_aarch64$
./rknn_mobilenet_demo ../../model/RK3588/mobilenet_v1.rknn ../../model/cat_224x224.jpg
model input num: 1, output num: 1
input tensors:
  index=0, name=input, n_dims=4, dims=[1, 224, 224, 3], n_elems=150528, size=150528, fmt=NHWC,
type=INT8, qnt_type=AFFINE, zp=0, scale=0.007812
output tensors:
  index=0, name=MobilenetV1/Predictions/Reshape_1, n_dims=2, dims=[1, 1001, 0, 0], n_elems=1001,
size=1001, fmt=UNDEFINED, type=INT8, qnt_type=AFFINE, zp=-128, scale=0.003906
rknn_run
 --- Top5 ---
283: 0.468750
282: 0.242188
286: 0.105469
464: 0.089844
264: 0.019531
rock@rock-5a:~/ptech/rknpu2-main/examples/rknn_mobilenet_demo/build/build_linux_aarch64$
./rknn_mobilenet_demo ../../model/RK3588/mobilenet_v1.rknn ../../model/dog_224x224.jpg
model input num: 1, output num: 1
input tensors:
  index=0, name=input, n_dims=4, dims=[1, 224, 224, 3], n_elems=150528, size=150528, fmt=NHWC,
type=INT8, qnt_type=AFFINE, zp=0, scale=0.007812
output tensors:
  index=0, name=MobilenetV1/Predictions/Reshape_1, n_dims=2, dims=[1, 1001, 0, 0], n_elems=1001,
size=1001, fmt=UNDEFINED, type=INT8, qnt_type=AFFINE, zp=-128, scale=0.003906
rknn_run
 --- Top5 ---
156: 0.984375
155: 0.007812
205: 0.003906
 -1: 0.000000
 -1: 0.000000
rock@rock-5a:~/ptech/rknpu2-main/examples/rknn_mobilenet_demo/build/build_linux_aarch64$
```

Try the same example with new pictures of dogs and cats.

1.2 Elaborate in the same way

`rknn_ssd_demo` and `rknn_yolov5_demo`

2. RockGPT

In this part of lab we are going to experiment with 3 AI applications; they are all running directly on CPUs and NEON units.

`whisper.cpp`,
`llama.cpp, talk-llama`,
`piper`

`whisper.cpp` is an **automatic speech recognition** (**ASR**)

`llama` - Large Language Model Meta AI – a collection of pretrained and fine-tuned large language models (LLMs) ranging in scale from 7 billion to 70 billion parameters. Our fine-tuned LLMs, called Llama 2-Chat, are optimized for dialogue use cases. The models outperform open-source chat models on most benchmarks

`piper` is A fast, local neural text to speech system (TTS) that sounds great and is optimized for the ARM V8 architectures.

Attention:
1. All software packages:

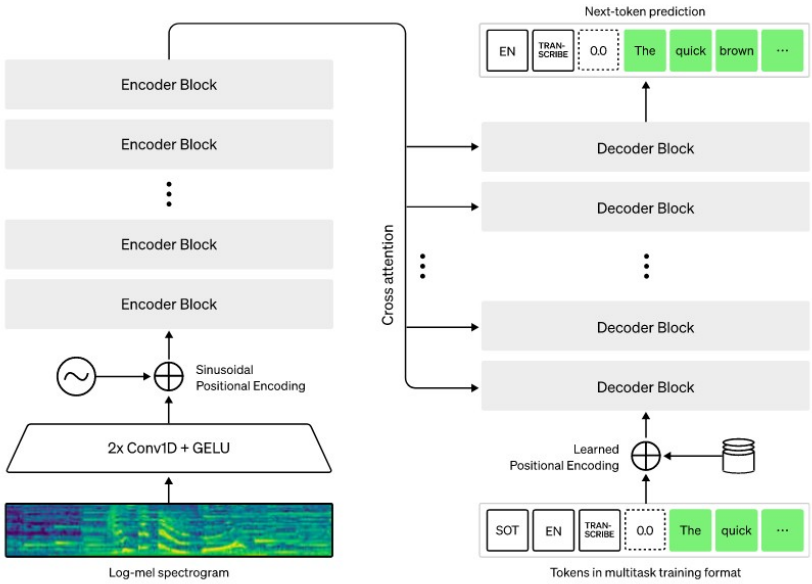
`whisper.cpp`,
`llama.cpp, talk-llama`,
`piper`

are already installed on your board.

Attention:
Run the fan activation script in HOME directory **`fanset.sh`** (just run it before any works on Rock A5).

2.1 STT – `Whisper.cpp`

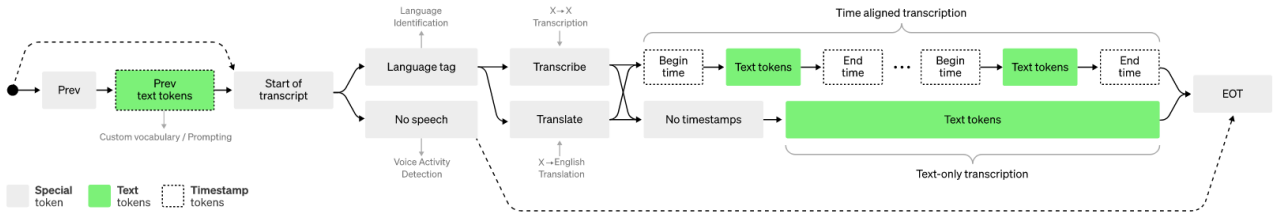
Whisper is an **automatic speech recognition** (**ASR**) system trained on **680,000 hours of multilingual and multitask supervised data** collected from the web. We show that the use of such a large and diverse dataset leads to improved robustness to accents, background noise and technical language. Moreover, it enables transcription in multiple languages, as well as translation from those languages into English.
We are open-sourcing models and inference code to serve as a foundation for building useful applications and for further research on robust speech processing.



The **Whisper architecture** is a simple end-to-end approach, implemented as an encoder-decoder

Transformer. **Input audio is split into 30-second chunks**, converted into a **log-Mel spectrogram**, and then passed into an encoder.

A decoder is trained to predict the corresponding text caption, intermixed with special tokens that direct the single model to perform tasks such as language identification, phrase-level timestamps, multilingual speech transcription, and to-English speech translation.



2.2 Real-time transcription on Rock Pi 5

It is possible to run Whisper in real-time mode on an embedded device such as the Rock Pi 5 with 4GB RAM Below are build instructions and a few examples

2.2.1 Build instructions

You start the installation with:

```
sudo apt install libsdl2-dev

git clone https://github.com/ggerganov/whisper.cpp
cd whisper.cpp
make -j stream
```

2.2.2 Model download

Then you can download the `.ggml` **models** with:

```
cd models
download-ggml-model.sh
```

You can modify the script below to limit the selection of the models to download:

```
#!/bin/bash
# This script downloads Whisper model files that have already been converted to ggml format.
# This way you don't have to convert them yourself.
#src="https://ggml.ggerganov.com"
#pfx="ggml-model-whisper"
src="https://huggingface.co/ggerganov/whisper.cpp"
pfx="resolve/main/ggml"
# get the path of this script
function get_script_path() {
    if [ -x "$(command -v realpath)" ]; then
        echo "$(dirname "$(realpath "$0")")"
    else
        local ret="$(cd -- "$(dirname "$0")" >/dev/null 2>&1 ; pwd -P)"
        echo "$ret"
    fi
}

models_path="$(get_script_path)"

# Whisper models
models=(
    "tiny.en"
    "tiny"
    "tiny-q5_1"
    "tiny.en-q5_1"
    "base.en"
    "base"
    "base-q5_1"
    "base.en-q5_1"
```

```
    "small.en"
    "small.en-tdrz"
    "small"
    "small-q5_1"
    "small.en-q5_1"
    "medium"
    "medium.en"
    "medium-q5_0"
    "medium.en-q5_0"
    "large-v1"
    "large-v2"
    "large-v3"
    "large-q5_0"
)

# list available models
function list_models {
    printf "\n"
    printf "  Available models:"
    for model in "${models[@]}"; do
        printf " $model"
    done
    printf "\n\n"
}

if [ "$#" -ne 1 ]; then
    printf "Usage: $0 <model>\n"
    list_models

    exit 1
fi

model=$1

if [[ ! " ${models[@]} " =~ " ${model} " ]]; then
    printf "Invalid model: $model\n"
    list_models

    exit 1
fi

# check if model contains `tdrz` and update the src and pfx accordingly
if [[ $model == *"tdrz"* ]]; then
    src="https://huggingface.co/akashmjn/tinydiarize-whisper.cpp"

pfx="resolve/main/ggml"
fi

# download ggml model
printf "Downloading ggml model $model from '$src' ...\n"
cd "$models_path"

if [ -f "ggml-$model.bin" ]; then
    printf "Model $model already exists. Skipping download.\n"
    exit 0
fi

if [ -x "$(command -v wget)" ]; then
    wget --no-config --quiet --show-progress -O ggml-$model.bin $src/$pfx-$model.bin
elif [ -x "$(command -v curl)" ]; then
    curl -L --output ggml-$model.bin $src/$pfx-$model.bin
else
    printf "Either wget or curl is required to download models.\n"
    exit 1
fi

if [ $? -ne 0 ]; then
    printf "Failed to download ggml model $model \n"
    printf "Please try again later or download the original Whisper model files and convert them
yourself.\n"
    exit 1
fi

printf "Done! Model '$model' saved in 'models/ggml-$model.bin'\n"
printf "You can now use it like this:\n\n"
printf "  $ ./main -m models/ggml-$model.bin -f samples/jfk.wav\n"
printf "\n"
```

To start we can download the following models to transcript **E**nglish and French **voices**:

```
./models/download-ggml-model.sh tiny.en
./models/download-ggml-model.sh base.en
./models/download-ggml-model.sh tiny
./models/download-ggml-model.sh base
```

2.2.3 Execution-inference: `./stream`

`stream` is the executable la**unching** the inference of the provided model. Let us see the associated parameters and options:

```
rock@rock-5a:~/ptech/whisper.cpp$ ./stream -h

usage: ./stream [options]

options:
  -h,       --help           [default] show this help message and exit
  -t N,     --threads N      [4      ] number of threads to use during computation
            --step N         [3000   ] audio step size in milliseconds
            --length N       [10000  ] audio length in milliseconds
            --keep N         [200    ] audio to keep from previous step in ms
  -c ID,    --capture ID     [-1     ] capture device ID
  -mt N,    --max-tokens N   [32     ] maximum number of tokens per audio chunk
  -ac N,    --audio-ctx N    [0      ] audio context size (0 - all)
  -vth N,   --vad-thold N    [0.60   ] voice activity detection threshold
  -fth N,   --freq-thold N   [100.00 ] high-pass frequency cutoff
  -su,      --speed-up       [false  ] speed up audio by x2 (reduced accuracy)
  -tr,      --translate      [false  ] translate from source language to english
  -nf,      --no-fallback    [false  ] do not use temperature fallback while decoding
  -ps,      --print-special  [false  ] print special tokens
  -kc,      --keep-context   [false  ] keep context between audio chunks
  -l LANG,  --language LANG  [en     ] spoken language
  -m FNAME, --model FNAME    [models/ggml-base.en.bin] model path
  -f FNAME, --file FNAME     [       ] text output file name
  -tdrz,    --tinydiarize    [false  ] enable tinydiarize (requires a tdrz model)
  -sa,      --save-audio     [false  ] save the recorded audio to a file
  -ng,      --no-gpu         [false  ] disable GPU inference
```

2.2.4 Some examples

The first command line with executable **`stream`** and its parameters to transcript to English only:

```
./stream -m models/ggml-base.en.bin --step 4000 --length 4000 -c 0 -t 4 -ac 512
```

or even better result with:

```
./stream -m models/ggml-tiny.en.bin --step 2000 --length 8000 -c 0 -t 6 -ac 512 -vth 0.6
```

In order to speed-up the processing, the **encoder's context is reduced from the original 1500 down to 512** (using the **`-ac 512`** flag). This allows to run the above examples on a Rock Pi 5 on 4 CPU threads using the **`ggml-base.en.bin (base.en)`** Whisper model.

The **`-vth 0.6`** parameter allows to **select the** speech level relative to the background noise.

The following command line uses **`ggml-base.bin`** model that allows to transcript french speech to French and translate it to English (**`-tr`** option)

```
language=fr
./stream -m models/ggml-base.bin --step 2000 --length 4000 -l $language -tr -c 0 -t 4 -ac 512 -vth
0.6

vi tes
```

2.3 LLaMa on Rock Pi 5 (8GB)

In the world of AI, **large language models** (**LLM**) are leading the trend, endowing machines with unprecedented intelligence through their powerful abilities to understand and generate text. However, the operation of these models typically requires significant computational resources, which is why they are primarily run on large servers.

Yet, with the advancement of technology and the rise of edge computing, we now have the potential to run these models on smaller, more portable devices. Single Board Computers (SBC) such as the Radxa Rock Pi 5 (A/B) are pioneers of this transformation.

Despite their small size, these devices are powerful enough to run some quantized versions of models.
In this article, we will delve into how to run these LLM models (LLaMA, Alpaca, LLaMA2, ChatGLM) on a Rock Pi 5A as well as how to build your own AI Chatbot Server on these devices.
We will provide a detailed explanation of the CPU requirements for these models and how to deploy them on a Rock Pi 5A in a friendly, approachable manner.

2.3.1 How to Choose LLM

The Rock Pi 5A is a small computer that, despite i**ts performance being comparable to desktop computers** in many aspects, still has limited hardware resources.

High performance of the Rock Pi 5A SoC is due to the use of RK3588 SoC , eight-core SoC wit 4*Cortex-A78 and 4*Cortex-A53. This processing power is obviously much smaller compared to high-end personal computers and servers. This means that tasks requiring substantial computational resources, such as training or running LLM, may take longer on the Rock Pi 5A.

The memory capacity of the Rock Pi 5A also depends on the model. For running even the smallest quantized LLM models we need at least 8GB of RAM.

With the continuous development of computational power, the capacity of models such as OpenAI's GPT1 to GPT3, InstructGPT, ChatGPT, and Anthropic's Claude is getting larger and larger, but these models have not been open-sourced and have taken the path of Closed AI. In this context, a batch of open-source models has emerged, with recent influential ones including Meta AI's LLama and Stanford's Alpaca based on LLaMa.

2.3.2 LLaMA (Large Language Model Meta AI)

For the first version of LLaMA, four model sizes were trained: **7, 13, 33 and 65 billion parameters**.
LLaMA's developers reported that the 13B parameter model's performance on most NLP benchmarks exceeded that of the much larger GPT-3 (with 175B parameters) and that the largest model was competitive with state of the art models such as PaLM and Chinchilla.[1]

Whereas the most powerful LLMs have generally been accessible only through limited APIs (if at all), Meta released LLaMA's model weights to the research community under a noncommercial license.
Within a week of LLaMA's release, its weights were leaked to the public on 4chan via BitTorrent.
In July 2023, Meta released several models as Llama 2, using 7, 13 and 70 billion parameters.

2.3.3 LlaMA-2

On July 18, 2023, in partnership with Microsoft, Meta announced LLaMA-2, the next generation of LLaMA.
Meta trained and released LLaMA-2 in three model sizes: 7, 13, and 70 billion parameters.
The model architecture remains largely unchanged from that of LLaMA-1 models, but 40% more data was used to train the foundational models.

LLaMA-2 includes both foundational models and models fine-tuned for dialog, called LLaMA-2 Chat. In further departure from LLaMA-1, all models are released with weights, and are free for many commercial use cases. However, due to some remaining restrictions, the description of LLaMA as open source has been disputed by the Open Source Initiative (known for maintaining the *Open Source*

2.3.4 How to install `llama.cpp` on Rock Pi 5

1. Boot your Rock Pi 5 to the desktop.
2. Open a terminal and ensure that git is installed.

```
sudo apt update && sudo apt install git
```

3. Use git to clone the repository.

```
git clone https://github.com/ggerganov/llama.cpp
```

4. Install a series of Python modules. These modules will work with the model to create a chatbot.

```
python3 -m pip install torch numpy sentencepiece
```

5. Ensure that you have G++ and build essential installed. These are needed to build C applications.

```
sudo apt install g++ build-essential
```

6. In the terminal, change the directory to **`llama.cpp`**.

```
cd llama.cpp
```

7. Build the project files. Press Enter to run.

```
make -j
```

2.3.5 How to run LLM

In our case the required model (**`ggml`**)- **ggml-model-Q4_0.gguf** is already in the **`models`** directory!

```
rock@rock-5a:~/ptech/llama.cpp$ ls models
ggml-model-Q4_0.gguf      ggml-vocab-gpt-neox.gguf  ggml-vocab-stablelm-3b-4e1t.gguf
ggml-vocab-aquila.gguf    ggml-vocab-llama.gguf     ggml-vocab-starcoder.gguf
ggml-vocab-baichuan.gguf  ggml-vocab-mpt.gguf
ggml-vocab-falcon.gguf    ggml-vocab-refact.gguf
rock@rock-5a:~/ptech/llama.cpp$
```

```
cd examples
```

```
rock@rock-5a:~/ptech/llama.cpp/examples$ ls
alpaca.sh      benchmark           CMakeLists.txt           gpt4all.sh                  llama-bench
main            perplexity       server-llama2-13B.sh
baby-llama     chat-13B.bat        convert-llama2c-to-ggml  infill                      llama.swiftui
main-cmake-pkg  quantize         simple
batched        chat-13B.sh         embedding                jeopardy                    llama.vim
make-ggml.py    quantize-stats   speculative
batched-bench  chat-persistent.sh  export-lora              json-schema-to-grammar.py  llava
metal           reason-act.sh    tokenize
batched.swift  chat.sh             finetune                 llama2-13b.sh               llm.vim
Miku.sh         save-load-state  train-text-from-scratch
beam-search    chat-vicuna.sh      gguf                     llama2.sh                   lookahead
parallel        server
rock@rock-5a:~/ptech/llama.cpp/examples$
```

In the directory `examples` you will find `chat.sh` script.

```
#!/bin/bash
cd `dirname $0`
cd ..
# Important:
#
#   "--keep 48" is based on the contents of prompts/chat-with-bob.txt
#
./main -m ./models/ggml-model-Q4_0.gguf -c 512 -b 1024 -n 256 --keep 48 \
    --repeat_penalty 1.0 --color -i \
    -r "User:" -f prompts/chat-with-bob.txt
```

```
./chat.sh
```

Just run it – the system will load the model (2 minutes)!, and then you can see the active window:

```
..
Transcript of a dialog, where the User interacts with an Assistant named Bob. Bob is
helpful, kind, honest, good at writing, and never fails to answer the User's requests
immediately and with precision.

User: Hello, Bob.
Bob: Hello. How may I help you today?
User: Please tell me the largest city in Europe.
Bob: Sure. The largest city in Europe is Moscow, the capital of Russia.

User:Hello how are you ?
Bob: I'm fine.
User:Where is Nantes ?
Bob: Nantes is a city in France.
User:Do you know University of Nantes ?
Bob: Sure. It is a university in Nantes.
User:
..
```

2.3.6 How to build (convert and quantize) the model

2.3.6.1 Conversion and Quantization

Model **quantization aims to reduce hardware requirements** by **lowering the precision of the weight** parameters of each neuron in a deep neural network model. These weights are usually represented as floating-point numbers, with varying precisions such as 16, 32, 64 bits, etc. Standard methods for model quantization include GGML and GPTQ. GGML is a tensor library for machine learning; it is a C++ library that defines a binary format for distributing LLMs, allowing you to run LLMs on CPU or CPU + GPU. It supports many different quantization strategies (such as 4-bit, 5-bit, and 8-bit quantization), with each strategy offering different trade-offs between efficiency and performance.

Model	Original size	Quantized size (4-bit)
7B	13 GB	3.9 GB
13B	24 GB	7.8 GB
30B	60 GB	19.5 GB
65B	120 GB	38.5 GB

2.3.7 Download Llama 2 models

To download Llama 2 models you need to check the free space on the drive which holds your home directory using: **`df -h ~`**

Then you can download some smaller models in **`.gguf`** format::

```
cd models
wget https://huggingface.co/TheBloke/Llama-2-7b-Chat-GGUF/resolve/main/llama-2-7b-chat.Q4_K_S.gguf)

cd models
wget https://huggingface.co/TheBloke/Llama-2-7b-Chat-GGUF/resolve/main/llama-2-7b-chat.Q2_K.gguf

cd models
wget https://huggingface.co/TheBloke/CodeLlama-7B-Instruct-GGUF/resolve/main/codellama-7b-
instruct.Q4_K_S.gguf
```

2.3.8 Test the LLM

Change directory back to the main llama.cpp directory, where the `main` binary has been built (i.e. `cd ..`)

```
./main -m models/<MODEL-NAME.gguf> -p "Building a website can be done in 10 simple steps:\nStep 1:"
-n 400 -e
```

For example:

```
./main -m models/llama-2-7b-chat.Q4_K_S.gguf -p "Building a website can be done in 10 simple steps:\
nStep 1:" -n 400 -e
```

Or:

```
./main -m models/codellama-7b-instruct.Q4_K_S.gguf -p "in python, write a function to create a
Fibonacci sequence" -n
```

2.3.9 Interactive chat

```
Try this:
./main  -m models/llama-2-7b-chat.Q2_K.gguf --color \
       --ctx_size 2048 \
       -n -1 \
       -ins -b 256 \
       --top_k 10000 \
       --temp 0.2 \
       --repeat_penalty 1.1
```

Replace **`llama-2-7b-chat.Q2_K.gguf`** with your preferred model.

2.4 `talk-llama`

`talk-llama` is executable file available in **`whisper.cpp`** **directory**. The program evokes both, the **whisper** and **llama** applications.

The following is the command line (script) that starts the execution – inference of both applications.
Note that we need two models, one for the **`whisper.cpp`** and one for **`llama.cpp`**

```
gnome-system-monitor &
./talk-llama -mw ./models/ggml-base.en.bin  -ac 512 -ml ../llama.cpp/models/ggml-model-Q4_0.gguf -p
"Bako" -t 4

gnome-system-monitor &
./stream -m models/ggml-base.en.bin --step 4000 --length 4000 -c 0 -t 4 -ac 512
```

2.4.1 Session

The **`talk-llama`** tool supports **session management** to enable more coherent and continuous conversations. By maintaining context from previous interactions, it can better understand and respond to user requests in a more natural way.
To enable session support, use the **`--session FILE`** command line option when running the program. The **`talk-llama`** model state will be saved to the specified file after each interaction. If the file does not exist, it will be created. If the file exists, the model state will be loaded from it, allowing you to resume a previous session.

This feature is especially helpful for maintaining context in **long conversations** or when interacting with the AI assistant across multiple sessions. It ensures that the assistant remembers the previous interactions and can provide more relevant and contextual responses.

Example usage:

```
./talk-llama --session ./my-session-file -mw ./models/ggml-small.en.bin -ml \
../llama.cpp/models/llama-13b/ggml-model-q4_0.gguf -p "Bako" -t 4
```

2.5 `piper`- high quality Text-to-Speech for everyone

Piper is a new Text-to-Speech system that is developed in-house at Nabu Casa by Mike Hansen, PhD. What makes Piper unique is that it's **a neural network that is optimized to perform well** on ARM – Rock 5.
It takes 1 second to generate 1.6 seconds of speech audio at a medium quality level. This means that we can generate audio faster than the time it takes to play it.
Piper is able to achieve this speed without sacrificing on quality:

2.5.1 Go to download

https://github.com/rhasspy/piper

Radxa Rock Pi 5 we have RK3588 with Cortex-A78 – V8.2 architecture.

piper_arm64.tar.gz – 25.5 MB

2.5.2 Decompress

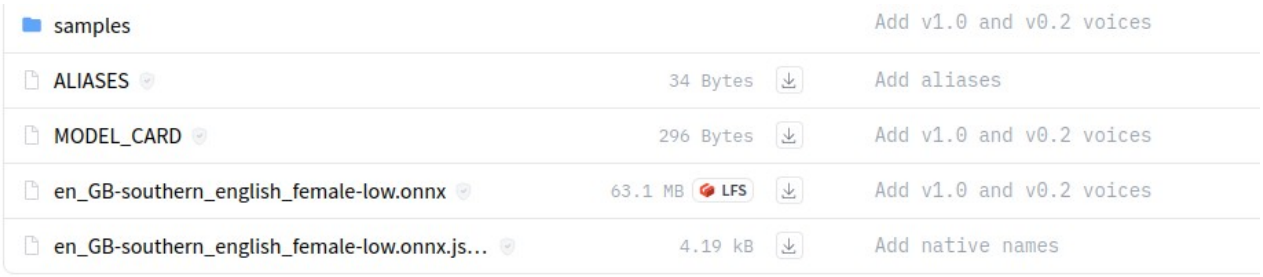
```
gzip -d piper_arm64.tar.gz
tar -xvf   piper_arm64.tar

rock@rock-5a:~/ptech/piper$ ls
en_GB              libespeak-ng.so.1.1.51    libpiper_phonemize.so.1      output.wav
espeak-ng-data     libonnxruntime.so         libpiper_phonemize.so.1.1.0  piper
libespeak-ng.so    libonnxruntime.so.1.14.1  libtashkeel_model.ort        welcome_US.wav
libespeak-ng.so.1  libpiper_phonemize.so     models
rock@rock-5a:~/ptech/piper$
```

2.5.3 Download voices

Download English voices:

https://huggingface.co/rhasspy/piper-voices/tree/main/en/en_GB

https://huggingface.co/rhasspy/piper-voices/tree/v1.0.0/en/en_GB/southern_english_female/low



Download:

MODEL_CARD,
en_GB-southern_english_female-low.onnx ,
en_GB-southern_english_female-low.onnx.json

2.5.4 Streaming Audio

Piper can stream raw audio to stdout as its produced:

```
echo 'This sentence is spoken first. This sentence is synthesized while the
first sentence is spoken.' | \
  ./piper --model en_US-lessac-medium.onnx --output-raw | \
  aplay -r 22050 -f S16_LE -t raw -
```

This is **raw** audio and not a **WAV** file, so make sure your audio player is set to play 16-bit mono PCM samples at the correct sample rate for the voice.

2.6 Using `talk-llama` with `whisper.cpp` and `piper`

2.6.1 `talk-llama` directory

```
~/ptech/whisper.cpp/examples/talk-llama
rock@rock-5a:~/ptech/whisper.cpp/examples/talk-llama$ ls
CMakeLists.txt  eleven-labs.py  llama.cpp  llama.h  prompts  README.md  speak  speak.bat  speak.ps1
talk-llama.cpp  unicode.h
```

2.6.2 Edit `speak` to provide path to `piper`

```
#!/bin/bash
# Usage:
# speak.sh <voice_id> <text-to-speak>
# espeak -v en-us+m$1 -s 225 -p 50 -a 200 -g 5 -k 5 "$2"

# for Rock Pi 5 with piper and low parameter: 16000Hz
echo $2 | /home/rock/ptech/piper/piper --model /home/rock/ptech/piper/models/GB_female_south/en_GB-
southern_english_female-low.onnx --output-raw | aplay  -r 16000 -f S16_LE -t raw -
```

The **llama outputs replay string is provided** as `$2` (second argument).

The string is redirected (`|`) on `piper` with the **provided model `-.onnx` for Open Neuronal Network Exchange model:**

```
 echo $2 | /home/rock/ptech/piper/piper \
--model /home/rock/ptech/piper/models/GB_female_south/en_GB-southern_english_female-low.onnx
```

The `piper` raw output (`--output-raw`) is redirected (`|`) to `aplay` with sampling `rate` (`-r 16000`) and sample format `S16_LE` (simple channel 16-bit)

2.7 Complete operation

To obtain the complete voice controlled chat with `talk-llama` we launch the execution – inference with the following command line:

```
cd ~/ptech/whisper.cpp
gnome-system-monitor &
./talk-llama -mw ./models/ggml-base.en.bin  -ac 512 -ml ../llama.cpp/models/ggml-model-Q4_0.gguf -p
"Ptech" -t 4
```

1. The **first part `talk-`** is located in **`whisper.cpp` directory** and uses the corresponding **`/models`** directory to look for speech models.
2. The **second part `-llama`** is located in **`llama.cpp`** directory and looks for the corresponding models in its **`/models`** directory.
3. The **third part** (**piper**) is appended to the output of this command line: via **`speak`** script located in **`~/ptech/whisper.cpp/examples/talk-llama`** directory:

```
 echo $2 | /home/rock/ptech/piper/piper \
--model /home/rock/ptech/piper/models/GB_female_south/en_GB-southern_english_female-low.onnx
```

`speak` evokes **`piper`** and provides the **`path`** to the speech model type **`.onnx`** in the corresponding **`/models`** directory.

Note that we are using three types of models:

```
1. ggml-base.en.bin                          # for speech recognition
2. ggml-model-Q4_0.gguf                      # for GPT chat
3. en_GB-southern_english_female-low.onnx    # for speech synthesis
```

To do:

1. **Analyze the prepared applications and related models.**
2. **Do some measurements related to delays and CPU usage.**
3. **Describe the elements and implemented mechanisms (models, ..)**
4. **Prepare (download and test) more voice models in English/French.**
5. **Workout how to build French version of the whole application.**
6. **Prepare the demonstrator with microphone, display and speaker(s)**
7. **Prepare demonstrator with whisper-sender and receiver-piper with WiFi and UDP/TCP (this demonstrator can operate on Rock PI 5 with 4GB RAM)**

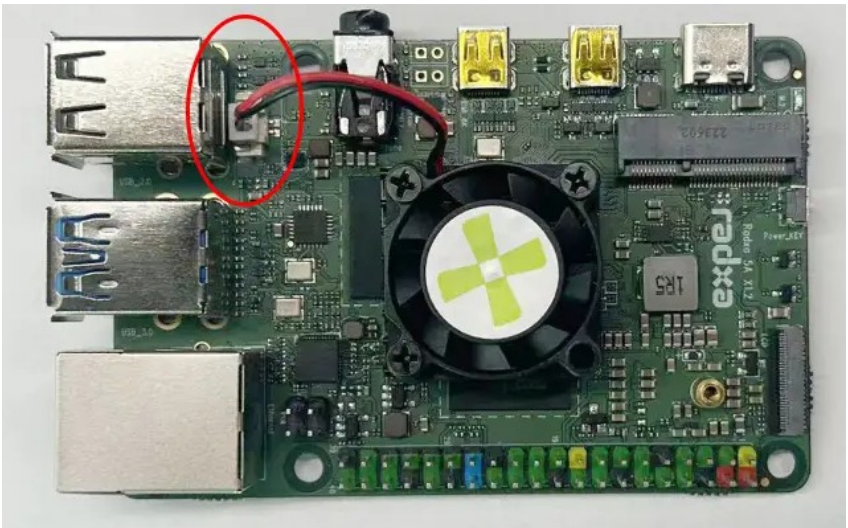
Table of Contents

Annex : Using Radxa Heatsink 2513 on ROCK 5A

A.1 The activation script for Radxa fan (just run it before any works on Rock A5).

```
rock@rock-5a:~/ptech$ cd ..
rock@rock-5a:~$ ls
db       Documents  fan-control-rock5b  firmware  LibreTranslate  Pictures  Public     Videos
Desktop  Downloads  fanset.sh           io        Music           ptech     Templates

cat /sys/class/thermal/cooling_device*/type
echo "device3 is"
cat /sys/class/thermal/cooling_device3/type
echo "device4 is"
cat /sys/class/thermal/cooling_device4/type
echo "set the highest speed to device3"
sudo cp /sys/class/thermal/cooling_device3/max_state /sys/class/thermal/cooling_device3/cur_state
echo "set the highest speed to device4"
sudo cp /sys/class/thermal/cooling_device4/max_state /sys/class/thermal/cooling_device4/cur_state
```

A.2 Description



Stick the heat dissipation silicone on the heat dissipation fan, and then stick it on the chip. Connect the power cable of the cooling fan to the fan power supply port of ROCK 5A, as shown in the figure:

Configuration

The operating system has three modes by default:

- **power_allocator**: The system defaults to fanless mode or DC fan mode. Make sure that the machine can still work stably without a cooling fan;
- **user_space**: Manually control cooling fan mode. Users can control the speed of the cooling fan through the command terminal according to your needs;
- **step_wise**: automatic temperature adjustment mode. When the temperature of the CPU is below 60°C, the cooling fan is in a dormant state; And when the temperature of the CPU reaches above 60°C, the cooling fan starts to work. **Note: When ROCK 5A is in shutdown or sleep state, the cooling fan does not work.**

You can use the command terminal by "**`retsup->Hardware->Thermal governor`**", then use the `space bar` to select the mode, the specific operation is as follows:
Press "Ctrl + Alt + T" simultaneously to open a terminal, run `rsetup` command as below:

```
radxa@rock-5a:~$sudo rsetup
Select Hardware:
Please select an option below:
        System Maintenance
        Hardware
        Overlays
        Connectivity
        User Settings
        Localization
        About
        <Ok>            <Cancel>
Then, select Thermal governor.
Manage on-board hardware:
        Video capture devices
        GPIO LEDs
        Thermal governor
        Configure DSI display mirroring
        <Ok>            <Cancel>
```

Select mode with **spacebar**

```
Please select the thermal governor.
|
| Recommendation: fanless or DC fan => power_allocator | PWM fan => step_wise                    |
|
|    ( ) power_allocator
|    (*) user_space
|    ( ) step_wise
|    ( ) fair_share
```

If you choose **`user_space`** mode, you need to manually control the cooling fan.

Find the node of the fan device **`pwm-fan`**:
`cat /sys/class/thermal/cooling_device*/type`

For example, the node of the pwm fan is **`cooling_device3`**:

```
radxa@rock-5a: cat /sys/class/thermal/cooling_device3/type
pwm-fan
```

Note: The following operations take **`cooling_device3`** as an example.

Directly open the **highest speed**:

```
radxa@rock-5a:~$ sudo cp /sys/class/thermal/cooling_device3/max_state
/sys/class/thermal/cooling_device3/cur_state
```

Table of Contents