Run DeepSeek R1 Distill Model on SG2042

测试小队

Feb 14, 2025

1 简介

2 安装依赖

3 安装 Llama.cpp

4 运行 DeepSeek R1 Distill Model

Section 1

简介

DeepSeek R1 Distill Model 概述

- DeepSeek-R1 是由 DeepSeek 团队推出的一个推理模型，
  其目前具有大模型上领先的性能。
- 原始 DeepSeek-R1 的参数量为 671B， 其不是个人电脑或 工作站可以承受的。
  通过对模型进行提炼， 可以将成果迁移到更小的模型上。
- 目前的蒸馏模型分别基于 Qwen2.5 和 Llama3 两个模型进行， 并产生 1.5B 到 70B 之间参数量不等的小模型。
- 同时，通过量化，将高精度的数据转换为低精度的数据，
  可以进一步减小模型的体积和增加运行速度。
- 本次使用了 Qwen2.5 1.5B 和 Llama3 8B 两个模型的 `Q2_K`, `Q4_K_M`
  两种量化进行演示。

运行环境

- 本次示例采用了基于 SG2042 的 MilkV-Pioneer 服务器进行，配置如下：
  - CPU: 64C riscv64 c920
  - RAM: 128GB
  - OS: RevyOS
  - Compiler: Plct-Xthead Gnu Toolchain

Section 2

安装依赖

Xthead 工具链

SG2042 上的 Vector 指令集为 XTheadVector （包含RVV0p7）。
后续的编译工作需要使用 RVV0p7 指令集， 而大多数编译器仅具有对 RVV1p0 的支持。
因此，需要下载转为该系列芯片打造的编译器。

PLCT 提供了针对 Xthead 的工具链，可以使用 ruyi 工具下载，也可以手动下载。

手动下载命令如下：

```
wget https://mirror.iscas.ac.cn/ruyisdk/dist/\
RuyiSDK-20240222-T-Head-Sources-T-Head-2.8.0\
-HOST-riscv64-linux-gnu-riscv64-plctxthead-linux-gnu.tar.xz
tar -xvf RuyiSDK-20240222-T-Head-Sources-T-Head-2.8.0\
-HOST-riscv64-linux-gnu-riscv64-plctxthead-linux-gnu.tar.xz
cd RuyiSDK-20240222-T-Head-Sources-T-Head-2.8.0\
-HOST-riscv64-linux-gnu-riscv64-plctxthead-linux-gnu/bin

export PATH=$(pwd):$PATH
```

后续再次使用需要将工具链重新加入 PATH 中。

注意事项： - PLCT Xthead 工具链使用的 sysroot
并非系统的根目录，而是工具链自带的。在安装库时需要注意路径。 -
在使用时需要手动设置 CC=riscv64-plctxthead-linux-gnu-gcc 和
riscv64-plctxthead-linux-gnu-gfortran

编译 OpenBLAS 后端

目前 Llama.cpp 支持的后端中，只有 OpenBLAS 有 RVV0p7 的支持。因此，采用 OpenBLAS 作为 GGML 的后端。

```
git clone https://github.com/OpenMathLib/OpenBLAS
cd OpenBLAS
make HOSTCC=gcc TARGET=C910V CC=riscv64-plctxthead-linux-gnu-gcc \
FC=riscv64-plctxthead-linux-gnu-gfortran
sudo make install PREFIX=/usr
sudo make install PREFIX=~/RuyiSDK-20240222-T-Head-Sources-T-Head-2.8.0\
-HOST-riscv64-linux-gnu-riscv64-plctxthead-linux-gnu/\
riscv64-plctxthead-linux-gnu/sysroot/usr
```

务必注意 PREFIX 的路径，需要指向工具链的 sysroot。

Section 3

安装 Llama.cpp

获取 Llama.cpp

直接从 GitHub 上获取最新的 Llama.cpp 源码：

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

Patch Llama.cpp

由于 Llama.cpp 默认只考虑了 RVV1p0
的指令集，若直接编译，其会产生非法指令。需要手动讲起替换为 v0p7：

```
diff --git a/ggml/src/ggml-cpu/CMakeLists.txt b/ggml/src/ggml-cpu/CMakeLists.tx
index 98fd18e..0e6f302 100644
--- a/ggml/src/ggml-cpu/CMakeLists.txt
+++ b/ggml/src/ggml-cpu/CMakeLists.txt
@@ -306,7 +306,7 @@ function(ggml_add_cpu_backend_variant_impl tag_name)
     elseif (${CMAKE_SYSTEM_PROCESSOR} MATCHES "riscv64")
         message(STATUS "RISC-V detected")
         if (GGML_RVV)
-            list(APPEND ARCH_FLAGS -march=rv64gcv -mabi=lp64d)
+            list(APPEND ARCH_FLAGS -march=rv64gcv0p7 -mabi=lp64d)
         endif()
     else()
         message(STATUS "Unknown architecture")
```

编译 Llama.cpp

指定 OpenBLAS 进行编译，注意设置 CC 和 FC：

```
CC=riscv64-plctxthead-linux-gnu-gcc FC=riscv64-plctxthead-linux-gnu-gfortran \
cmake -B build -DGGML_BLAS=ON -DGGML_BLAS_VENDOR=OpenBLAS
cmake --build build --config Release -j32
```

Section 4

运行 DeepSeek R1 Distill Model

下载模型

在 Pioneer 上建议使用 8B 及以下的模型。如 DeepSeek-R1-Distill-Llama-8B-GGUF Q4_K_M 和 DeepSeek-R1-Distill-Qwen-1.5B-GGUF 这两个模型的 `Q4_K_M` 和 `Q2_K` 量化模型。

下载模型可以直接在网页端进行下载，也可以使用一个 Python 脚本进行下载。需要使用 pip 安装：

```
pip install huggingface_hub hf_transfer
```

```
import os

from huggingface_hub import snapshot_download
snapshot_download(
  repo_id = "unsloth/DeepSeek-R1-Distill-Llama-8B-GGUF",
  local_dir = "DeepSeek-R1-Distill-Llama-8B-GGUF",
  allow_patterns = ["*Q4_K_M*", "*Q2_K*"],
)
```

在 CLI 中运行

下面的代码中，替换 [your words] 为你想要输入的文本。-t 选项代表线程数量，-m 代表模型路径。一般而言，选择 32 线程即可。

```
llama.cpp/build/bin/llama-cli \
-m DeepSeek-R1-Distill-Llama-8B-GGUF/DeepSeek-R1-Distill-Llama-8B-Q4_K_M.gguf \
--cache-type-k q8_0 -t 32 \
--prompt '<|User|>[Your words]<|Assistant|>' \
-no-cnv
```

```
llama_init_from_model: KV self size  =  392.00 MiB, K (q8_0):  136.00 MiB, V (f16):  256.00 MiB
llama_init_from_model:        CPU  output buffer size =     0.49 MiB
llama_init_from_model:        CPU compute buffer size =   296.01 MiB
llama_init_from_model: graph nodes  = 1030
llama_init_from_model: graph splits = 514 (with bs=512), 1 (with bs=1)
common_init_from_params: setting dry_penalty_last_n to ctx_size = 4096
common_init_from_params: warming up the model with an empty run - please wait ... (--no-warmup to disa
ble)
main: llama threadpool init, n_threads = 32

system_info: n_threads = 32 (n_threads_batch = 32) / 64 | CPU : LLAMAFILE = 1 | AARCH64_REPACK = 1 |

sampler seed: 3873949425
sampler params:
        repeat_last_n = 64, repeat_penalty = 1.000, frequency_penalty = 0.000, presence_penalty = 0.00
0
        dry_multiplier = 0.000, dry_base = 1.750, dry_allowed_length = 2, dry_penalty_last_n = 4096
        top_k = 40, top_p = 0.950, min_p = 0.050, xtc_probability = 0.000, xtc_threshold = 0.100, typi
cal_p = 1.000, top_n_sigma = -1.000, temp = 0.800
        mirostat = 0, mirostat_lr = 0.100, mirostat_ent = 5.000
sampler chain: logits -> logit-bias -> penalties -> dry -> top-k -> typical -> top-p -> min-p -> xtc -
> temp-ext -> dist
generate: n_ctx = 4096, n_batch = 2048, n_predict = -1, n_keep = 1

Can you please count from 1 to 10?<think>
First, I'll start by listing the numbers from 1 to 10 in a clear and sequential manner.

I'll make sure each number is correctly placed and easy to read.

Finally, I'll present the final
```

```
    0[1]  4[0]  8[1] 12[1] 16[1] 20[0] 24[0] 28[0]  32[3]  36[1] 40[1] 44[0] 48[0] 52[0] 56[1] 60[1]
    1[0]  5[1]  9[1] 13[1] 17[1] 21[1] 25[0] 29[2]  33[0]  37[1] 41[5] 45[1] 49[0] 53[1] 57[0] 61[0]
    2[0]  6[0] 10[1] 14[1] 18[1] 22[0] 26[0] 30[0]  34[1]  38[1] 42[1] 46[0] 50[1] 54[0] 58[1] 62[0]
    3[1]  7[1] 11[0] 15[1] 19[0] 23[0] 27[0] 31[1]  35[1]  39[1] 43[1] 47[0] 51[0] 55[0] 59[0] 63[1]
  Mem[||||||                          1.38G/121G] Tasks: 52, 246 thr, 734 kthr; 33 running
  Swp[                                   0K/0K] Load average: 22.20 12.50 8.95
                                                Uptime: 00:17:26

  Main  I/O
    PID USER       PRI  NI  VIRT   RES   SHR S  CPU%-MEM%   TIME+  Command
   2477 lw          20   0 8377M 5161M 4691M R 100.2  4.2  1:06.11
   2574 lw          20   0 8377M 5161M 4691M R 100.2  4.2  0:54.94
   2575 lw          20   0 8377M 5161M 4691M R 100.2  4.2  0:55.01
   2576 lw          20   0 8377M 5161M 4691M R 100.2  4.2  0:55.01
   2578 lw          20   0 8377M 5161M 4691M R 100.2  4.2  0:55.01
   2579 lw          20   0 8377M 5161M 4691M R 100.2  4.2  0:55.01
   2580 lw          20   0 8377M 5161M 4691M R 100.2  4.2  0:55.01
   2584 lw          20   0 8377M 5161M 4691M R 100.2  4.2  0:54.97
   2585 lw          20   0 8377M 5161M 4691M R 100.2  4.2  0:55.00
   2586 lw          20   0 8377M 5161M 4691M R 100.2  4.2  0:55.00
   2587 lw          20   0 8377M 5161M 4691M R 100.2  4.2  0:54.98
   2588 lw          20   0 8377M 5161M 4691M R 100.2  4.2  0:55.00
   2589 lw          20   0 8377M 5161M 4691M R 100.2  4.2  0:55.01
   2590 lw          20   0 8377M 5161M 4691M R 100.2  4.2  0:55.00
   2593 lw          20   0 8377M 5161M 4691M R 100.2  4.2  0:55.01
   2594 lw          20   0 8377M 5161M 4691M R 100.2  4.2  0:55.01
   2598 lw          20   0 8377M 5161M 4691M R 100.2  4.2  0:55.01
F1Help  F2Setup F3SearchF4FilterF5Tree  F6SortByF7Nice -F8Nice +F9Kill  F10Quit
[0] 0:llama.cpp/build/bin/llama-cli*                    "OpenCloudOS-riscv64" 13:14 14-Feb-25
```

交互式运行

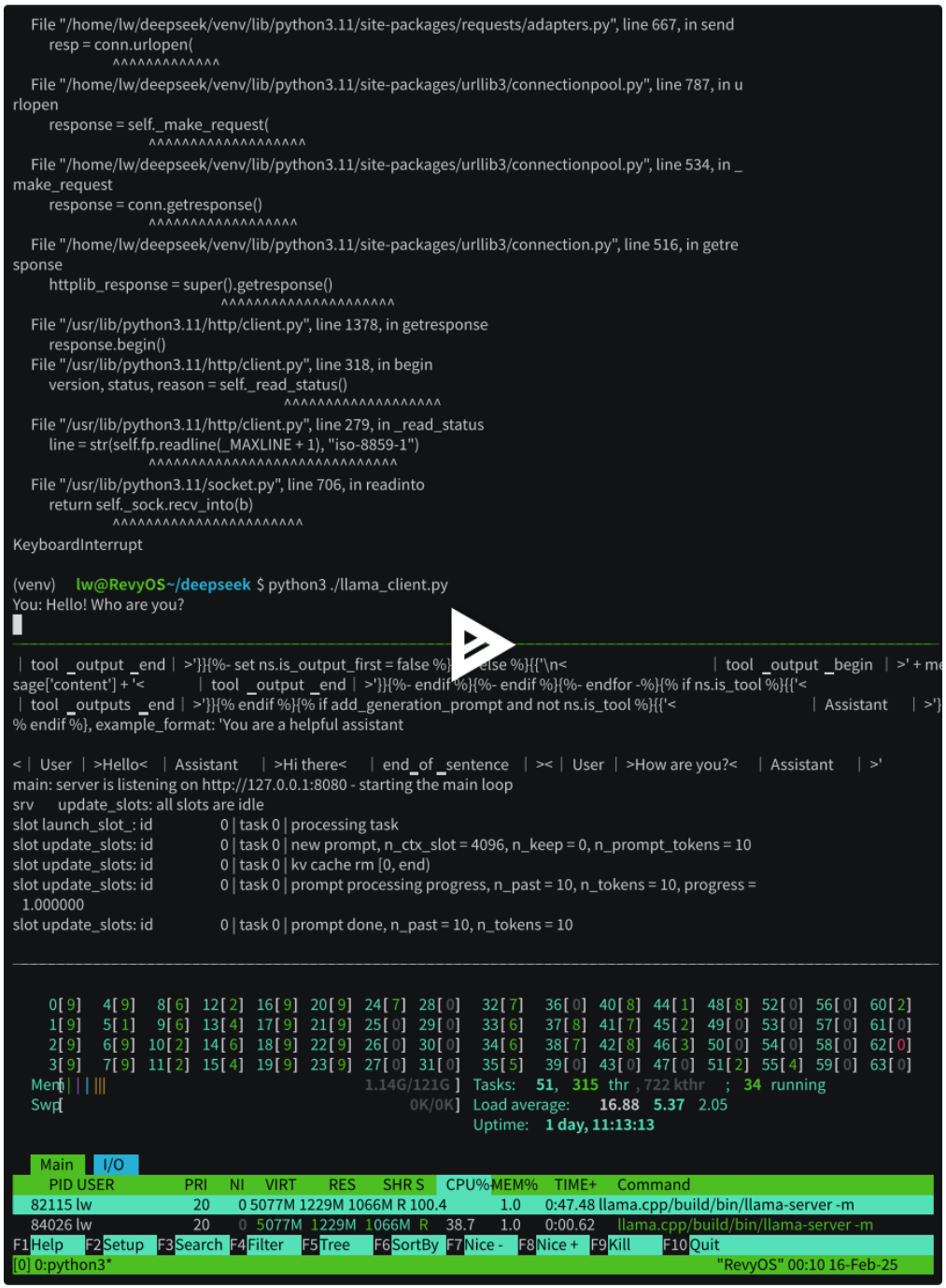
交互式运行时，llama.cpp 会提供一个 OpenAI API 兼容的网络接口，默认通过 http://localhost:8080 进行访问。

-t 选项代表线程数量，-m 代表模型路径。一般而言，选择 32 线程即可。

```
llama.cpp/build/bin/llama-server \
-m DeepSeek-R1-Distill-Llama-8B-GGUF/DeepSeek-R1-Distill-Llama-8B-Q4_K_M.gguf \
--cache-type-k q8_0 -t 32
```

您可自行选择一个 HTTP 客户端进行访问，将 API Url 设置为 http://localhost:8080/completion 即可。

或可采用一个使用 Python 编写的简单脚本进行访问，可在 wychlw/plct/memo/deepseek_on_llama.cpp.md 中找到。


File "/home/lw/deepseek/venv/lib/python3.11/site-packages/requests/adapters.py", line 667, in send
resp = conn.urlopen(
File "/home/lw/deepseek/venv/lib/python3.11/site-packages/urllib3/connectionpool.py", line 787, in urlopen
response = self._make_request(
File "/home/lw/deepseek/venv/lib/python3.11/site-packages/urllib3/connectionpool.py", line 534, in _make_request
response = conn.getresponse()
File "/home/lw/deepseek/venv/lib/python3.11/site-packages/urllib3/connection.py", line 516, in getresponse
httplib_response = super().getresponse()
File "/usr/lib/python3.11/http/client.py", line 1378, in getresponse
response.begin()
File "/usr/lib/python3.11/http/client.py", line 318, in begin
version, status, reason = self._read_status()
File "/usr/lib/python3.11/http/client.py", line 279, in _read_status
line = str(self.fp.readline(_MAXLINE + 1), "iso-8859-1")
File "/usr/lib/python3.11/socket.py", line 706, in readinto
return self._sock.recv_into(b)
KeyboardInterrupt
(venv) lw@RevyOS~/deepseek $ python3 ./llama_client.py
You: Hello! Who are you?
main: server is listening on http://127.0.0.1:8080 - starting the main loop
srv update_slots: all slots are idle
slot launch_slot_: id 0 | task 0 | processing task
slot update_slots: id 0 | task 0 | new prompt, n_ctx_slot = 4096, n_keep = 0, n_prompt_tokens = 10
slot update_slots: id 0 | task 0 | kv cache rm [0, end)
slot update_slots: id 0 | task 0 | prompt processing progress, n_past = 10, n_tokens = 10, progress = 1.000000
slot update_slots: id 0 | task 0 | prompt done, n_past = 10, n_tokens = 10
Tasks: 51, 315 thr , 722 kthr ; 34 running
Load average: 16.88 5.37 2.05
Uptime: 1 day, 11:13:13
Main I/O
PID USER PRI NI VIRT RES SHR S CPU% MEM% TIME+ Command
82115 lw 20 0 5077M 1229M 1066M R 100.4 1.0 0:47.48 llama.cpp/build/bin/llama-server -m
84026 lw 20 0 5077M 1229M 1066M R 38.7 1.0 0:00.62 llama.cpp/build/bin/llama-server -m
F1Help F2Setup F3Search F4Filter F5Tree F6SortBy F7Nice - F8Nice + F9Kill F10Quit
[0] 0:python3*
"RevyOS" 00:10 16-Feb-25

性能对比

以下测试均采用了 32 线程的配置，在 Pioneer Box 上运行。

DeepSeek R1 不同蒸馏模型及量化对比：

模型	量化	ms per token	token per second
DeepSeek-R1-Distill-Qwen-1.5B-GGUF	Q2_K	785.58ms	1.27
DeepSeek-R1-Distill-Qwen-1.5B-GGUF	Q4_K_M	992.34ms	1.01
DeepSeek-R1-Distill-Llama-8B-GGUF	Q2_K	1884.06ms	0.53
DeepSeek-R1-Distill-Llama-8B-GGUF	Q4_K_M	1575.40ms	0.63

在实际使用中发现，Q2_K 量化的模型思考过程反而较长，而 Q4_K_M 量化的模型思考过程较短。使得虽然 Q2_K 量化的模型速度更快，但 Q4_K_M 量化的模型感觉延迟更低。

问题均采用 `Hello! Who's there?` 作为输入。

RTT 及 Token 数量对比：

模型	量化	RTT (ms)	Token 数量
DeepSeek-R1-Distill-Qwen-1.5B-GGUF	Q2_K	231406.57	302
DeepSeek-R1-Distill-Qwen-1.5B-GGUF	Q4_K_M	26217.45	26
DeepSeek-R1-Distill-Llama-8B-GGUF	Q2_K	247836	136
DeepSeek-R1-Distill-Llama-8B-GGUF	Q4_K_M	75446.46	52