llama.cpp

- 这个项目的名字是llama.cpp，从这个项目的名称中可以看出来这个项目使用纯C/C++来构建 LLM（inference and train）

`license MIT` `Server failing` `conan b3542`

[Roadmap](#) / [Project status](#) / [Manifesto](#) / [ggml](#)

Inference of Meta's [LLaMA](#) model (and others) in pure C/C++（meta的模型、其他类型的模型将会使用纯 C/C++进行inference）

> [!IMPORTANT] [2024 Jun 12] Binaries have been renamed w/ a `llama-` prefix. `main` is now `llama-cli`, `server` is `llama-server`, etc (https://github.com/ggerganov/llama.cpp/pull/7809)

1. **二进制文件的重命名：**

    - 项目中的二进制文件名称被修改，加入了 `llama-` 前缀。这意味着原来的二进制文件名如 `main`、 `server` 等都进行了更改，以包含这个新的前缀。
    - 例如，原来名为 `main` 的二进制文件现在更名为 `llama-cli`，`server` 更名为 `llama-server` 等。 这种命名方式可能是为了增强命名的一致性和清晰度，也可能是为了避免在更广泛的环境中的命 名冲突。

2. **命名变化的具体例子：**

    - `main` -> `llama-cli`
    - `server` -> `llama-server`
    - 可以推测，其他相关的二进制文件也遵循类似的重命名模式，虽然具体文件名未在信息中提及。

3. **更新的来源：**

    - 这次更新的详细信息可以在 GitHub 上的一个 pull request（拉取请求）中找到，编号为 #7809。
    - 通过访问提供的链接（https://github.com/ggerganov/llama.cpp/pull/7809），可以查看更多关于 这次变更的背景、原因以及可能的讨论和反馈。

4. **可能的影响：**

- 这种变更可能会影响到所有依赖于原有二进制文件名的脚本、自动化工具和用户习惯。用户和开发者需要更新他们的系统配置、文档和指南以适应新的命名规则。
- 对于通过命令行界面与这些程序交互的用户，他们需要熟悉新的命令名称来继续操作。

总结来说，这个更新是关于在 `llama.cpp` 项目中对二进制文件进行重命名的，以 `llama-` 作为新的统一前缀。这样的更改有助于提高项目的组织性和在更广泛的开发环境中的可识别性。

Recent API changes（最近的API变化）

- [2024 Jun 26] The source code and CMake build scripts have been restructured https://github.com/ggerganov/llama.cpp/pull/8006
- [2024 Apr 21] `llama_token_to_piece` can now optionally render special tokens https://github.com/ggerganov/llama.cpp/pull/6807
- [2024 Apr 4] State and session file functions reorganized under `llama_state_*` https://github.com/ggerganov/llama.cpp/pull/6341
- [2024 Mar 26] Logits and embeddings API updated for compactness https://github.com/ggerganov/llama.cpp/pull/6122
- [2024 Mar 13] Add `llama_synchronize()` + `llama_context_params.n_ubatch` https://github.com/ggerganov/llama.cpp/pull/6017
- [2024 Mar 8] `llama_kv_cache_seq_rm()` returns a `bool` instead of `void`, and new `llama_n_seq_max()` returns the upper limit of acceptable `seq_id` in batches (relevant when dealing with multiple sequences) https://github.com/ggerganov/llama.cpp/pull/5328
- [2024 Mar 4] Embeddings API updated https://github.com/ggerganov/llama.cpp/pull/5796
- [2024 Mar 3] `struct llama_context_params` https://github.com/ggerganov/llama.cpp/pull/5849

上述内容是关于 `llama.cpp` 项目在最近几个月进行的一系列 API 和结构更改的汇总。这些更改反映了该项目的持续发展和改进。以下是各项更新的详细解释：

1. **[2024 Jun 26] 代码和构建脚本重构：**

   - 源代码和 CMake 构建脚本经过了重构（ PR #8006 ）。这通常意味着项目的文件结构、构建流程或依赖管理得到了优化，可能使得项目更易于维护和扩展。

2. **[2024 Apr 21] `llama_token_to_piece` 支持渲染特殊标记：**

   - 函数 `llama_token_to_piece` 更新，增加了可选功能，可以渲染特殊标记（ PR #6807 ）。这对于需要在文本处理中区分特殊符号或控制字符的场景非常有用。

3. **[2024 Apr 4] 状态和会话文件函数重组：**

   - 与状态和会话文件相关的函数被重组到 `llama_state_*` 下（ PR #6341 ）。这样的组织可能提高了这些功能的逻辑清晰度和易用性。

4. **[2024 Mar 26] 更新了 logits 和 embeddings API：**

   - Logits 和 embeddings 相关的 API 更新，以增强紧凑性（ PR #6122 ）。此更改可能涉及API的参数简化或性能优化。

5. **[2024 Mar 13] 添加 `llama_synchronize()` 和 `llama_context_params.n_ubatch`：**

- 新增 `llama_synchronize()` 函数和 `llama_context_params` 结构中的 `n_ubatch` 参数（PR #6017）。这些更改可能是为了提高多线程或异步执行环境中的数据同步和批处理能力。

6. **[2024 Mar 8] `llama_kv_cache_seq_rm()` 返回类型更改和新增 `llama_n_seq_max()`**：

   - `llama_kv_cache_seq_rm()` 函数从返回 `void` 更改为返回 `bool`，新增 `llama_n_seq_max()` 函数返回批次中可接受的最大序列号（PR #5328）。这提供了更强的错误处理和序列处理能力。

7. **[2024 Mar 4] 更新了 Embeddings API**：

   - Embeddings API进行了更新（PR #5796），这通常意味着改进了嵌入层的功能或接口。

8. **[2024 Mar 3] `struct llama_context_params`**：

   - 引入了 `struct llama_context_params`（PR #5849）。这样的结构体通常用于封装与特定上下文相关的参数，可能是为了更好地管理和配置项目中的不同操作或模块设置。

这些更新表明 `llama.cpp` 项目在不断地进行技术迭代和改进，以适应不断变化的技术需求和提升用户体验。通过这些更新，项目可能增强了性能，改进了功能，或者简化了用户和开发者的接口使用。

Hot topics（火热主题）

- **`convert.py` has been deprecated and moved to `examples/convert_legacy_llama.py`, please use `convert_hf_to_gguf.py`** https://github.com/ggerganov/llama.cpp/pull/7430
- Initial Flash-Attention support: https://github.com/ggerganov/llama.cpp/pull/5021
- BPE pre-tokenization support has been added: https://github.com/ggerganov/llama.cpp/pull/6920
- MoE memory layout has been updated - reconvert models for `mmap` support and regenerate `imatrix` https://github.com/ggerganov/llama.cpp/pull/6387
- Model sharding instructions using `gguf-split` https://github.com/ggerganov/llama.cpp/discussions/6404
- Fix major bug in Metal batched inference https://github.com/ggerganov/llama.cpp/pull/6225
- Multi-GPU pipeline parallelism support https://github.com/ggerganov/llama.cpp/pull/6017
- Looking for contributions to add Deepseek support: https://github.com/ggerganov/llama.cpp/issues/5981
- Quantization blind testing: https://github.com/ggerganov/llama.cpp/discussions/5962
- Initial Mamba support has been added: https://github.com/ggerganov/llama.cpp/pull/5328

上述内容列出了关于 `llama.cpp` 项目的一系列热门话题和最新动态，涉及API变更、新功能支持、性能优化、以及对社区贡献的征集。这些更新展示了项目的活跃发展和对现代计算需求的响应。以下是各项内容的详细解释：

1. **`convert.py` 已被弃用**：

   - 原有的 `convert.py` 工具已经不推荐使用，并已移至 `examples/convert_legacy_llama.py`。现在建议使用新的 `convert_hf_to_gguf.py` 脚本进行模型转换。这通常表示项目在模型转换工具上进行了重大更新或优化。（PR #7430）

2. **Flash-Attention 支持**：

- 项目初次引入了对 Flash-Attention 的支持，这是一种高效的注意力机制实现，可以提高模型的性能和缩放性。（PR #5021）

3. **BPE 预标记化支持：**

- 为项目添加了 BPE（Byte Pair Encoding）预标记化支持。这对于改进文本处理的效率和效果非常关键。（PR #6920）

4. **MoE 内存布局更新：**

- Mixture of Experts (MoE) 的内存布局进行了更新，支持了内存映射（`mmap`）。这要求重新转换模型并重新生成 `imatrix`，以适应新的内存布局。这种更新可能大大提高了模型的加载和运行效率。（PR #6387）

5. **模型分片指导使用 `gguf-split`：**

- 提供了使用 `gguf-split` 进行模型分片的指导。模型分片是提高大规模模型处理效率的一种方法，尤其是在资源受限的环境中。（Discussion #6404）

6. **修复 Metal 批处理推理中的重大错误：**

- 修复了在使用 Metal 进行批处理推理时的一个重大错误。这项修复提高了使用 Metal 作为后端进行机器学习推理的可靠性。（PR #6225）

7. **多GPU流水线并行支持：**

- 项目增加了对多GPU流水线并行计算的支持，这对于处理大规模数据集和复杂模型在多个GPU上的高效运算非常重要。（PR #6017）

8. **征求 Deepseek 支持的贡献：**

- 项目正在寻求社区贡献以添加对 Deepseek 的支持。Deepseek 可能是一种新的功能或工具，旨在提高搜索和检索的效率。（Issue #5981）

9. **量化盲测：**

- 进行量化盲测以评估模型量化的效果。量化是一种减少模型所需计算资源的方法，盲测有助于验证量化过程的有效性和透明度。（Discussion #5962）

10. **Mamba 支持：**

- 项目初次引入了对 Mamba 的支持。Mamba 可能是一个新的库或框架，旨在提高特定计算任务的效率。（PR #5328）

这些更新突显了 `llama.cpp` 项目在技术前沿的持续发展，以及对提高性能、增加新功能和修复关键错误的持续承诺。

Description

The main goal of `llama.cpp` is to enable LLM inference with minimal setup and state-of-the-art performance on a wide variety of hardware - locally and in the cloud.

`llama.cpp`这个项目的主要目标就是能够使LLM以最小的配置、高水平的性能、适配诸多硬件、本地或者云端的方式进行inference

- Plain C/C++ implementation without any dependencies
- Apple silicon is a first-class citizen - optimized via ARM NEON, Accelerate and Metal frameworks
- AVX, AVX2 and AVX512 support for x86 architectures
- 1.5-bit, 2-bit, 3-bit, 4-bit, 5-bit, 6-bit, and 8-bit integer quantization for faster inference and reduced memory use
- Custom CUDA kernels for running LLMs on NVIDIA GPUs (support for AMD GPUs via HIP)
- Vulkan and SYCL backend support
- CPU+GPU hybrid inference to partially accelerate models larger than the total VRAM capacity

Since its inception, the project has improved significantly thanks to many contributions. It is the main playground for developing new features for the ggml library.

上述内容提供了对 `llama.cpp` 项目的详细描述，突出了其目标、特性和支持的技术。`llama.cpp` 旨在使大型语言模型（LLM）的推理过程在各种硬件上尽可能简便且性能卓越，无论是本地还是云端。以下是对其主要特点的详细解释：

1. **无依赖的纯 C/C++ 实现：**

   - 该项目完全使用C/C++ 实现，无需任何外部库依赖，这简化了设置过程并提高了项目的可移植性和易于安装性。

2. **苹果硅片的一流支持：**

   - 项目对 Apple silicon（如 M1/M2 系列芯片）有优化，使用 ARM NEON、Accelerate 和 Metal 框架来增强性能。这意味着 `llama.cpp` 能在苹果的最新硬件上提供最佳性能。

3. **对 x86 架构的 AVX, AVX2 和 AVX512 支持：**

   - 项目利用了 x86 架构中的高级向量扩展（AVX, AVX2, AVX512），这些是现代 Intel 和 AMD 处理器的强大功能，可用于加速计算密集型任务。

4. **整数量化：**

   - 支持多种级别的整数量化（从1.5-bit到8-bit），这有助于加快推理速度并减少内存使用。量化是一种减少模型精度以提高性能和降低资源消耗的技术。

5. **自定义 CUDA 内核：**

   - 项目为在 NVIDIA GPU 上运行 LLM 提供了自定义的 CUDA 内核，也通过 HIP 支持 AMD GPU。这使得 `llama.cpp` 能在广泛的GPU硬件上高效运行。

6. **Vulkan 和 SYCL 后端支持：**

   - 通过支持 Vulkan 和 SYCL，`llama.cpp` 增加了对更多平台和设备的支持，使得跨平台性能优化更加可行。

7. **CPU+GPU 混合推理：**

○ 支持 CPU 和 GPU 的混合推理，这对于那些超出单个 GPU VRAM 容量的大型模型特别有用。它允许模型的部分计算在 GPU 上加速，而其他部分则在 CPU 上处理（这部分内容十分有趣，我特别感兴趣）。

自项目开始以来，`llama.cpp` 通过众多贡献显著提升了性能和功能，成为 `ggml` 库新特性开发的主要平台。这说明 `llama.cpp` 不仅是一个推理工具，也是一个不断创新和改进的动态项目。

Supported models(支持的模型):

Typically finetunes of the base models below are supported as well.（通常情况下，也是支持下列这些基础模型的微调）

- ☑ LLaMA 🦙
- ☑ LLaMA 2 🦙🦙
- ☑ LLaMA 3 🦙🦙🦙
- ☑ Mistral 7B
- ☑ Mixtral MoE
- ☑ DBRX
- ☑ Falcon
- ☑ Chinese LLaMA / Alpaca and Chinese LLaMA-2 / Alpaca-2
- ☑ Vigogne (French)
- ☑ BERT
- ☑ Koala
- ☑ Baichuan 1 & 2 + derivations
- ☑ Aquila 1 & 2
- ☑ Starcoder models
- ☑ Refact
- ☑ MPT
- ☑ Bloom
- ☑ Yi models
- ☑ StableLM models
- ☑ Deepseek models
- ☑ Qwen models
- ☑ PLaMo-13B
- ☑ Phi models
- ☑ GPT-2
- ☑ Orion 14B
- ☑ InternLM2
- ☑ CodeShell
- ☑ Gemma
- ☑ Mamba
- ☑ Grok-1
- ☑ Xverse
- ☑ Command-R models
- ☑ SEA-LION
- ☑ GritLM-7B + GritLM-8x7B
- ☑ OLMo
- ☑ Granite models

- ☑ GPT-NeoX + Pythia
- ☑ Snowflake-Arctic MoE
- ☑ Smaug
- ☑ Poro 34B
- ☑ Bitnet b1.58 models
- ☑ Flan T5
- ☑ Open Elm models
- ☑ ChatGLM3-6b + ChatGLM4-9b
- ☑ SmolLM
- ☑ EXAONE-3.0-7.8B-Instruct
- ☑ FalconMamba Models

(instructions for supporting more models: HOWTO-add-model.md)

Multimodal models（多模态模型）：

- ☑ LLaVA 1.5 models, LLaVA 1.6 models
- ☑ BakLLaVA
- ☑ Obsidian
- ☑ ShareGPT4V
- ☑ MobileVLM 1.7B/3B models
- ☑ Yi-VL
- ☑ Mini CPM
- ☑ Moondream
- ☑ Bunny

Bindings（不同编程语言的绑定）：

- Python: abetlen/llama-cpp-python
- Go: go-skynet/go-llama.cpp
- Node.js: withcatai/node-llama-cpp
- JS/TS (llama.cpp server client): lgrammel/modelfusion
- JavaScript/Wasm (works in browser): tangledgroup/llama-cpp-wasm
- Typescript/Wasm (nicer API, available on npm): ngxson/wllama
- Ruby: yoshoku/llama_cpp.rb
- Rust (more features): edgenai/llama_cpp-rs
- Rust (nicer API): mdrokz/rust-llama.cpp
- Rust (more direct bindings): utilityai/llama-cpp-rs
- C#/.NET: SciSharp/LLamaSharp
- Scala 3: donderom/llm4s
- Clojure: phronmophobic/llama.clj
- React Native: mybigday/llama.rn
- Java: kherud/java-llama.cpp
- Zig: deins/llama.cpp.zig
- Flutter/Dart: netdur/llama_cpp_dart
- PHP (API bindings and features built on top of llama.cpp): distantmagic/resonance (more info)
- Guile Scheme: guile_llama_cpp

UI（用户界面）：

Unless otherwise noted these projects are open-source with permissive licensing:（除非另有说明，这些项目都是具有宽松许可的开源项目）

- [MindWorkAI/AI-Studio](#) (FSL-1.1-MIT)
- [iohub/collama](#)
- [janhq/jan](#) (AGPL)
- [nat/openplayground](#)
- [Faraday](#) (proprietary)
- [LMStudio](#) (proprietary)
- [Layla](#) (proprietary)
- [ramalama](#) (MIT)
- [LocalAI](#) (MIT)
- [LostRuins/koboldcpp](#) (AGPL)
- [Mozilla-Ocho/llamafile](#)
- [nomic-ai/gpt4all](#)
- [ollama/ollama](#)
- [oobabooga/text-generation-webui](#) (AGPL)
- [psugihara/FreeChat](#)
- [cztomsik/ava](#) (MIT)
- [ptsochantaris/emeltal](#)
- [pythops/tenere](#) (AGPL)
- [RAGNA Desktop](#) (proprietary)
- [RecurseChat](#) (proprietary)
- [semperai/amica](#)
- [withcatai/catai](#)
- [Mobile-Artificial-Intelligence/maid](#) (MIT)
- [Msty](#) (proprietary)
- [LLMFarm](#) (MIT)
- [KanTV](#)(Apachev2.0 or later)
- [Dot](#) (GPL)
- [MindMac](#) (proprietary)
- [KodiBot](#) (GPL)
- [eva](#) (MIT)
- [AI Sublime Text plugin](#) (MIT)
- [AIKit](#) (MIT)
- [LARS - The LLM & Advanced Referencing Solution](#) (AGPL)

(to have a project listed here, it should clearly state that it depends on `Llama.cpp`)

- 要在此处列出项目，应明确说明其依赖于`llama.cpp`项目

Tools（工具集）：

- [akx/ggify](#) – download PyTorch models from HuggingFace Hub and convert them to GGML
- [crashr/gppm](#) – launch llama.cpp instances utilizing NVIDIA Tesla P40 or P100 GPUs with reduced idle power consumption

- gpustack/gguf-parser - review/check the GGUF file and estimate the memory usage

上述内容介绍了两个与 `llama.cpp` 项目相关的工具集，这些工具各自针对特定功能进行了优化和支持，以提升 `llama.cpp` 的实用性和效率。以下是对这两个工具的详细解释：

1. **akx/ggify**：

   - 这是一个工具，用于从 HuggingFace Hub 下载 PyTorch 模型，并将其转换为 GGML 格式。HuggingFace Hub 是一个流行的开源库，汇聚了大量的预训练模型，主要是基于 PyTorch 和 TensorFlow。`ggify` 工具允许用户直接获取这些模型，并转换为 `llama.cpp` 使用的 GGML 格式，这使得在 `llama.cpp` 中使用这些广泛的模型资源成为可能。
   - 这个工具的存在显著简化了从其他机器学习框架向 `llama.cpp` 迁移模型的过程，加快了从实验到生产的步骤。

2. **crashr/gppm**：

   - `gppm` 是一个启动工具，专门用于在 NVIDIA Tesla P40 或 P100 GPUs 上运行 `llama.cpp` 实例。这些 GPU 型号是高性能的计算设备，适用于数据中心和专业计算环境。该工具的特点是能够在 GPU 处于空闲状态时降低其功耗，这对于节能和成本效益来说非常关键。
   - 此功能特别适合在需要长时间运行或间歇性使用的大规模计算任务，帮助管理能源消耗并优化资源使用效率。

总结来说，这两个工具的开发和使用体现了 `llama.cpp` 社区在确保其技术可接入性及环境可持续性方面的努力。通过 `ggify` 和 `gppm`，`llama.cpp` 用户可以更高效地利用现有的机器学习模型资源，并在高性能硬件上进行能效优化，这些都是推动现代计算任务向前发展的关键因素。

Infrastructure（基础设施）：

- Paddler - Stateful load balancer custom-tailored for llama.cpp
- GPUStack - Manage GPU clusters for running LLMs

Games（游戏）：

- Lucy's Labyrinth - A simple maze game where agents controlled by an AI model will try to trick you.（一个简单的迷宫游戏，在这个游戏中被AI模型控制的智能体尝试欺骗你）

Demo

▶ Typical run using LLaMA v2 13B on M2 Ultra（在 M2 Ultra 上使用 LLaMA v2 13B 的典型运行）

```
$ make -j && ./llama-cli -m models/llama-13b-v2/ggml-model-q4_0.gguf -p "Building
a website can be done in 10 simple steps:\nStep 1:" -n 400 -e
I llama.cpp build info:
I UNAME_S:  Darwin
I UNAME_P:  arm
I UNAME_M:  arm64
I CFLAGS:   -I.              -O3 -std=c11    -fPIC -DNDEBUG -Wall -Wextra -Wpedantic
-Wcast-qual -Wdouble-promotion -Wshadow -Wstrict-prototypes -Wpointer-arith -
Wmissing-prototypes -pthread -DGGML_USE_K_QUANTS -DGGML_USE_ACCELERATE
I CXXFLAGS: -I. -I./common -O3 -std=c++11 -fPIC -DNDEBUG -Wall -Wextra -Wpedantic
-Wcast-qual -Wno-unused-function -Wno-multichar -pthread -DGGML_USE_K_QUANTS
```

```
I LDFLAGS:   -framework Accelerate
I CC:        Apple clang version 14.0.3 (clang-1403.0.22.14.1)
I CXX:       Apple clang version 14.0.3 (clang-1403.0.22.14.1)

make: Nothing to be done for `default'.
main: build = 1041 (cf658ad)
main: seed  = 1692823051
llama_model_loader: loaded meta data with 16 key-value pairs and 363 tensors from
models/llama-13b-v2/ggml-model-q4_0.gguf (version GGUF V1 (latest))
llama_model_loader: - type  f32:   81 tensors
llama_model_loader: - type q4_0:  281 tensors
llama_model_loader: - type q6_K:    1 tensors
llm_load_print_meta: format         = GGUF V1 (latest)
llm_load_print_meta: arch           = llama
llm_load_print_meta: vocab type     = SPM
llm_load_print_meta: n_vocab        = 32000
llm_load_print_meta: n_merges       = 0
llm_load_print_meta: n_ctx_train    = 4096
llm_load_print_meta: n_ctx          = 512
llm_load_print_meta: n_embd         = 5120
llm_load_print_meta: n_head         = 40
llm_load_print_meta: n_head_kv      = 40
llm_load_print_meta: n_layer        = 40
llm_load_print_meta: n_rot          = 128
llm_load_print_meta: n_gqa          = 1
llm_load_print_meta: f_norm_eps     = 1.0e-05
llm_load_print_meta: f_norm_rms_eps = 1.0e-05
llm_load_print_meta: n_ff           = 13824
llm_load_print_meta: freq_base      = 10000.0
llm_load_print_meta: freq_scale     = 1
llm_load_print_meta: model type     = 13B
llm_load_print_meta: model ftype    = mostly Q4_0
llm_load_print_meta: model size     = 13.02 B
llm_load_print_meta: general.name   = LLaMA v2
llm_load_print_meta: BOS token = 1 '<s>'
llm_load_print_meta: EOS token = 2 '</s>'
llm_load_print_meta: UNK token = 0 '<unk>'
llm_load_print_meta: LF token  = 13 '<0x0A>'
llm_load_tensors: ggml ctx size =    0.11 MB
llm_load_tensors: mem required  = 7024.01 MB (+  400.00 MB per state)
....................................................................................................
..................
llama_new_context_with_model: kv self size  =  400.00 MB
llama_new_context_with_model: compute buffer total size =   75.41 MB

system_info: n_threads = 16 / 24 | AVX = 0 | AVX2 = 0 | AVX512 = 0 | AVX512_VBMI =
0 | AVX512_VNNI = 0 | FMA = 0 | NEON = 1 | ARM_FMA = 1 | F16C = 0 | FP16_VA = 1 |
WASM_SIMD = 0 | BLAS = 1 | SSE3 = 0 | VSX = 0 |
sampling: repeat_last_n = 64, repeat_penalty = 1.100000, presence_penalty =
0.000000, frequency_penalty = 0.000000, top_k = 40, tfs_z = 1.000000, top_p =
0.950000, typical_p = 1.000000, temp = 0.800000, mirostat = 0, mirostat_lr =
0.100000, mirostat_ent = 5.000000
generate: n_ctx = 512, n_batch = 512, n_predict = 400, n_keep = 0
```

```
  Building a website can be done in 10 simple steps:
Step 1: Find the right website platform.
Step 2: Choose your domain name and hosting plan.
Step 3: Design your website layout.
Step 4: Write your website content and add images.
Step 5: Install security features to protect your site from hackers or spammers
Step 6: Test your website on multiple browsers, mobile devices, operating systems
etc…
Step 7: Test it again with people who are not related to you personally – friends
or family members will work just fine!
Step 8: Start marketing and promoting the website via social media channels or
paid ads
Step 9: Analyze how many visitors have come to your site so far, what type of
people visit more often than others (e.g., men vs women) etc…
Step 10: Continue to improve upon all aspects mentioned above by following trends
in web design and staying up-to-date on new technologies that can enhance user
experience even further!
How does a Website Work?
A website works by having pages, which are made of HTML code. This code tells your
computer how to display the content on each page you visit – whether it's an image
or text file (like PDFs). In order for someone else's browser not only be able but
also want those same results when accessing any given URL; some additional steps
need taken by way of programming scripts that will add functionality such as
making links clickable!
The most common type is called static HTML pages because they remain unchanged
over time unless modified manually (either through editing files directly or using
an interface such as WordPress). They are usually served up via HTTP protocols –
this means anyone can access them without having any special privileges like being
part of a group who is allowed into restricted areas online; however, there may
still exist some limitations depending upon where one lives geographically
speaking.
How to
llama_print_timings:        load time =   576.45 ms
llama_print_timings:      sample time =   283.10 ms /   400 runs   (    0.71 ms
per token,  1412.91 tokens per second)
llama_print_timings: prompt eval time =   599.83 ms /    19 tokens (   31.57 ms
per token,    31.68 tokens per second)
llama_print_timings:        eval time = 24513.59 ms /   399 runs   (   61.44 ms
per token,    16.28 tokens per second)
llama_print_timings:       total time = 25431.49 ms
```

▶ Demo of running both LLaMA-7B and whisper.cpp on a single M1 Pro MacBook（在单个M1 Pro MacBook上同时运行LLaMA-7B和whisper.cpp的demo）

And here is another demo of running both LLaMA-7B and whisper.cpp on a single M1 Pro MacBook（这是另外一个demo：在单个M1 Pro MacBook上同时运行LLaMA-7B和whisper.cpp）:

https://user-images.githubusercontent.com/1991296/224442907-7693d4be-acaa-4e01-8b4f-add84093ffff.mp4

- 一段演示视频

Usage（使用）

Here are the end-to-end binary build and model conversion steps for most supported models.（以下是大多数受支持模型的端到端二进制构建和模型转换步骤）

Basic usage（基本的使用）

Firstly, you need to get the binary. There are different methods that you can follow（首先，你需要得到二进制文件，下列是一些你可以采用的方法）：

- Method 1: Clone this repository and build locally, see how to build（方法1:clone这个repo和在本地构建，查看相关内容进行build）
- Method 2: If you are using MacOS or Linux, you can install llama.cpp via brew, flox or nix（方法2：如果你使用的是MacOS或者Linux，你可以通过指出的内容下载llama.cpp）
- Method 3: Use a Docker image, see documentation for Docker（方法3：使用一个docker镜像）
- Method 4: Download pre-built binary from releases（从releases中下载一个已经构建好的二进制文件）

You can run a basic completion using this command（您可以使用此命令运行基本补全）：

```
llama-cli -m your_model.gguf -p "I believe the meaning of life is" -n 128

# Output:
# I believe the meaning of life is to find your own truth and to live in
accordance with it. For me, this means being true to myself and following my
passions, even if they don't align with societal expectations. I think that's what
I love about yoga – it's not just a physical practice, but a spiritual one too.
It's about connecting with yourself, listening to your inner voice, and honoring
your own unique journey.
```

See this page for a full list of parameters.（可以查看./examples/main/README.md获取完整的参数列表）

Conversation mode（转化模式）

If you want a more ChatGPT-like experience, you can run in conversation mode by passing `-cnv` as a parameter（如果你想要一个更像是ChatGPT类似的体验，你可以传入-cnv这个参数从而获得对话模式）：

```
llama-cli -m your_model.gguf -p "You are a helpful assistant" -cnv

# Output:
# > hi, who are you?
# Hi there! I'm your helpful assistant! I'm an AI-powered chatbot designed to
assist and provide information to users like you. I'm here to help answer your
questions, provide guidance, and offer support on a wide range of topics. I'm a
friendly and knowledgeable AI, and I'm always happy to help with anything you
need. What's on your mind, and how can I assist you today?
#
# > what is 1+1?
# Easy peasy! The answer to 1+1 is... 2!
```

By default, the chat template will be taken from the input model. If you want to use another chat template, pass `--chat-template NAME` as a parameter. See the list of supported templates

默认情况下，这个聊天模板将从输入模型中获得，如果你想要使用另外一个聊天模板，那么传入`--chat-template NAME`参数就可以了。查看下列支持的模板。

```
./llama-cli -m your_model.gguf -p "You are a helpful assistant" -cnv --chat-template chatml
```

You can also use your own template via in-prefix, in-suffix and reverse-prompt parameters（您还可以通过前缀、后缀和反向提示参数使用自己的模板）：

```
./llama-cli -m your_model.gguf -p "You are a helpful assistant" -cnv --in-prefix 'User: ' --reverse-prompt 'User:'
```

Web server（网页服务器）

llama.cpp web server is a lightweight OpenAI API compatible HTTP server that can be used to serve local models and easily connect them to existing clients.

llama.cpp Web 服务器 是一个轻量级的与 OpenAI API 兼容的 HTTP 服务器，可用于为本地模型提供服务并轻松将其连接到现有客户端。

Example usage（使用示例）：

```
./llama-server -m your_model.gguf --port 8080

# Basic web UI can be accessed via browser: http://localhost:8080
# Chat completion endpoint: http://localhost:8080/v1/chat/completions
```

Interactive mode（交互模式）

> [!NOTE] If you prefer basic usage, please consider using conversation mode instead of interactive mode（如果你更加喜欢基本的使用，请考虑使用对话模式而不是交互模式）

In this mode, you can always interrupt generation by pressing Ctrl+C and entering one or more lines of text, which will be converted into tokens and appended to the current context.（在这种对话模式下，你总是可以通过Ctrl+C来进行打断并且输入一行或者多行文本，这些文本将会被转化为tokens并且附加到现有的上下文中）You can also specify a *reverse prompt* with the parameter `-r "reverse prompt string"`.（你也可以通过`-r "reverse prompt string"`参数指定一个*reverse prompt*）This will result in user input being prompted whenever the exact tokens of the reverse prompt string are encountered in the generation.（这样，只要在生成过程中遇到反向提示字符串的精确标记，就会提示用户输入。）A typical use is to use a prompt that makes LLaMA emulate a chat between multiple users, say Alice and Bob, and pass `-r "Alice:"`（典型用法是使用提示，让 LLaMA 模拟多个用户之间的聊天，比如 Alice 和 Bob，并传递 `-r"Alice:"`）.
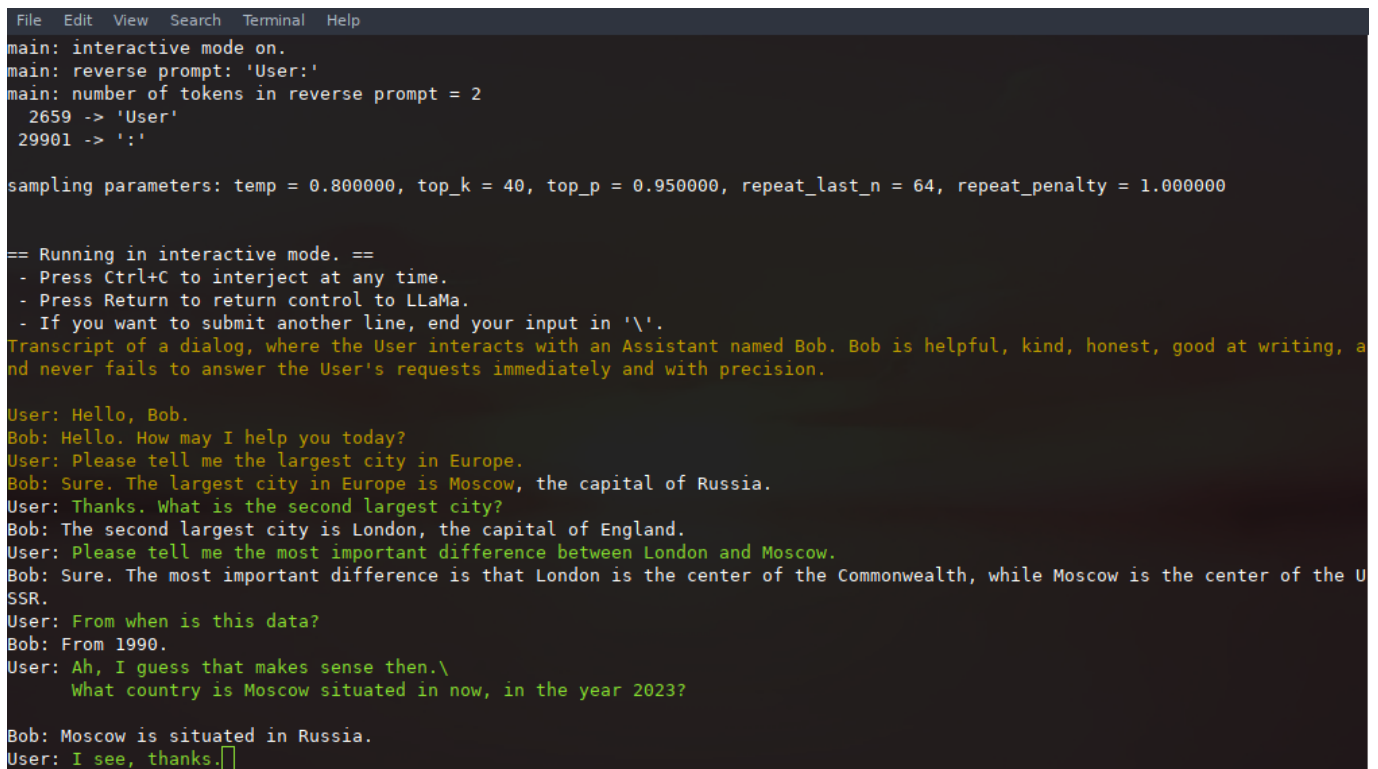
Here is an example of a few-shot interaction, invoked with the command（以下是使用命令调用的几次交互的示例）

```
# default arguments using a 7B model
./examples/chat.sh

# advanced chat with a 13B model
./examples/chat-13B.sh

# custom arguments using a 13B model
./llama-cli -m ./models/13B/ggml-model-q4_0.gguf -n 256 --repeat_penalty 1.0 --
color -i -r "User:" -f prompts/chat-with-bob.txt
```

Note the use of `--color` to distinguish between user input and generated text. Other parameters are explained in more detail in the README for the `llama-cli` example program.（请注意使用 `--color` 来区分用户输入和生成的文本。其他参数在 `llama-cli` 示例程序的 README 中有更详细的说明。）

Persistent Interaction（持续互动）

The prompt, user inputs, and model generations can be saved and resumed across calls to `./llama-cli` by leveraging `--prompt-cache` and `--prompt-cache-all`. The `./examples/chat-persistent.sh` script demonstrates this with support for long-running, resumable chat sessions. To use this example, you must provide a file to cache the initial chat prompt and a directory to save the chat session, and may optionally provide the same variables as `chat-13B.sh`. The same prompt cache can be reused for new chat sessions. Note that both prompt cache and chat directory are tied to the initial prompt (`PROMPT_TEMPLATE`) and the model file.

通过利用 `--prompt-cache` 和 `--prompt-cache-all`，可以在调用 `./llama-cli` 时保存和恢复提示、用户输入和模型生成。`./examples/chat-persistent.sh` 脚本演示了这一点，并支持长时间运行、可恢复的聊天会

话。要使用此示例，您必须提供一个文件来缓存初始聊天提示和一个目录来保存聊天会话，并且可以选择提供与 `chat-13B.sh` 相同的变量。相同的提示缓存可以重复用于新的聊天会话。请注意，提示缓存和聊天目录都与初始提示（`PROMPT_TEMPLATE`）和模型文件相关联。

```
# Start a new chat
PROMPT_CACHE_FILE=chat.prompt.bin CHAT_SAVE_DIR=./chat/default ./examples/chat-
persistent.sh

# Resume that chat
PROMPT_CACHE_FILE=chat.prompt.bin CHAT_SAVE_DIR=./chat/default ./examples/chat-
persistent.sh

# Start a different chat with the same prompt/model
PROMPT_CACHE_FILE=chat.prompt.bin CHAT_SAVE_DIR=./chat/another ./examples/chat-
persistent.sh

# Different prompt cache for different prompt/model
PROMPT_TEMPLATE=./prompts/chat-with-bob.txt PROMPT_CACHE_FILE=bob.prompt.bin \
    CHAT_SAVE_DIR=./chat/bob ./examples/chat-persistent.sh
```

Constrained output with grammars（使用语法进行约束输出）

`llama.cpp` supports grammars to constrain model output. For example, you can force the model to output JSON only（`llama.cpp`支持通过语法来限制模型的输出。例如，你可以强制模型只输出JSON）:

```
./llama-cli -m ./models/13B/ggml-model-q4_0.gguf -n 256 --grammar-file
grammars/json.gbnf -p 'Request: schedule a call at 8pm; Command:'
```

The `grammars/` folder contains a handful of sample grammars. To write your own, check out the GBNF Guide.

- 这个 `grammars/`文件夹中包含一个小型的样例语法。想要创建自己想要的语法，查看./grammars/README.md，其中有详细的描述。

For authoring more complex JSON grammars, you can also check out https://grammar.intrinsiclabs.ai/, a browser app that lets you write TypeScript interfaces which it compiles to GBNF grammars that you can save for local use. Note that the app is built and maintained by members of the community, please file any issues or FRs on its repo and not this one.

- 如需编写更复杂的 JSON 语法，您还可以查看 https://grammar.intrinsiclabs.ai/，这是一款浏览器应用，可让您编写 TypeScript 接口，并将其编译为 GBNF 语法，然后您可以将其保存以供本地使用。请注意，该应用是由社区成员构建和维护的，请在 其存储库 上提交任何问题或 FR，而不是在此存储库上。

Build（构建）

Please refer to Build llama.cpp locally

- 请参考Build llama.cpp locally，其中描述了如何本地构建llama.cpp

Supported backends（支持的后端）

Backend（后端）	Target devices（目标设备）
Metal	Apple Silicon
BLAS	All
BLIS	All
SYCL	Intel and Nvidia GPU
MUSA	Moore Threads GPU
CUDA	Nvidia GPU
hipBLAS	AMD GPU
Vulkan	GPU
CANN	Ascend NPU

Tools（工具）

Prepare and Quantize（准备和量化）

> [!NOTE] You can use the GGUF-my-repo space on Hugging Face to quantise your model weights without any setup too. It is synced from `llama.cpp` main every 6 hours.（你可以使用hugging face上的 GGUF-my-repo space来量化你的模型权重，这个过程不需要任何的配置。这个内容每6个小时将会同 `llama.cpp`main branch进行同步）

To obtain the official LLaMA 2 weights please see the Obtaining and using the Facebook LLaMA 2 model section. There is also a large selection of pre-quantized `gguf` models available on Hugging Face.（要获取官方 LLaMA 2 权重，请参阅获取和使用 Facebook LLaMA 2 模型部分。Hugging Face 上还有大量预量化的"gguf"模 型可供选择。）

Note: `convert.py` has been moved to `examples/convert_legacy_llama.py` and shouldn't be used for anything other than `Llama/Llama2/Mistral` models and their derivatives. It does not support LLaMA 3, you can use `convert_hf_to_gguf.py` with LLaMA 3 downloaded from Hugging Face.

To learn more about quantizing model, read this documentation（想要学习更多的关于量化模型内容，查看 read this documentation）

Perplexity (measuring model quality：衡量模型质量)

You can use the `perplexity` example to measure perplexity over a given prompt (lower perplexity is better). For more information, see https://huggingface.co/docs/transformers/perplexity.

- 你可以使用`perplexity`例子在给定的prompt上测量preplexity，对于更多的信息，查看对应的内容

To learn more how to measure perplexity using llama.cpp, read this documentation

- 想要学习如何使用llama.cpp衡量perplexity请查看相应内容

Contributing（贡献）

- Contributors can open PRs（共享这可以打开PRs）
- Collaborators can push to branches in the `llama.cpp` repo and merge PRs into the `master` branch（贡献者可以在`llama.cpp`仓库中push分支并且将PRs合并到master分支中）
- Collaborators will be invited based on contributions（贡献者将会基于贡献会被邀请）
- Any help with managing issues and PRs is very appreciated!（有着任何管理和PRs将会是非常好的，非常感激）
- See good first issues for tasks suitable for first contributions（查看相应内容对于第一次贡献）
- Read the CONTRIBUTING.md for more information（查看相应文件查看更多的信息）
- Make sure to read this: Inference at the edge（确保已经阅读过相应的内容）
- A bit of backstory for those who are interested: Changelog podcast（给那些感兴趣的人讲一些背景故事）

Other documentations（其他的文档）

- main (cli)
- server
- jeopardy
- GBNF grammars

Development documentations（开发文档）

- How to build：如何build项目
- Running on Docker：在docker中运行
- Build on Android：在Android上build
- Performance troubleshooting：性能故障排除
- GGML tips & tricks：GGML 技巧和窍门

Seminal papers and background on the models（开创性的论文和模型背景）

If your issue is with model generation quality, then please at least scan the following links and papers to understand the limitations of LLaMA models. This is especially important when choosing an appropriate model size and appreciating both the significant and subtle differences between LLaMA models and ChatGPT（如果您的问题与模型生成质量有关，请至少浏览以下链接和论文，以了解 LLaMA 模型的局限性。在选择合适的模型大小并理解 LLaMA 模型与 ChatGPT 之间的显著和细微差异时，这一点尤为重要）：

- LLaMA:
  - Introducing LLaMA: A foundational, 65-billion-parameter large language model
  - LLaMA: Open and Efficient Foundation Language Models
- GPT-3
  - Language Models are Few-Shot Learners
- GPT-3.5 / InstructGPT / ChatGPT:
  - Aligning language models to follow instructions
  - Training language models to follow instructions with human feedback