龙芯平台部署 deepseek R1 教程

1. 直接运行预编译的 Ollama

要跳过编译过程，可以直接下载编译好的 ollama 二进制文件(新世界版本)：
https://github.com/loong64/ollama/releases/tag/v0.5.7
要下载旧世界可用的 Ollama 二进制文件，可以访问：
https://bbs.elecfans.com/m/jishu_2475508_1_1.html

2. 获取 ollama

Ollama 的源码地址：https://github.com/ollama/ollama
根据 katyusha 的说法，由于 ollama 使用 uname -m 命令来获取机器架构，而龙架构上的 go 架构名为 loong64，uname -m 得到的结果为 loongarch64，因此无法编译成功。
解决方案是使用修改过的 ollama 的 fork
https://github.com/ideal/ollama
截止到目前（2025-2-7），ideal 提交的 commit 还没有合并到 ollama 的主干上
https://github.com/ollama/ollama/pull/8558
从该位置，下载 ollama 源码

```
git clone https://github.com/ideal/ollama
```

3. 编译 Ollama

在新世界系统和旧世界系统下均可以运行 Ollama。新世界操作系统推荐使用 AOSC，旧世界操作系统推荐 Loongnix 20 或 UOS。不推荐使用麒麟系统。

3.1 AOSC 系统

3.1.1 配置环境

Ollama 的开发采用了 go 语言进行，编译依赖 go 1.23.4。
在 AOSC 系统中，安装基础的编译环境、python、go 编译器以及 cmake。

```
sudo oma install build-essential go python3 cmake
```

如果编译器版本低于 1.23，请自行安装新版本的 go。
设置环境变量

```
export GO111MODULE=on
export GOPROXY=https://goproxy.io
```

设置 GOPROXY，可以加速子模块源码的下载。goproxy.io 是一个比较流行的 GO 代理。GOPROXY 环境变量的设置非常重要，尤其是当用户访问 github 等网站有困难的时候。

3.1.2 编译

编译有三种方法

1. 使用 make 编译（katyusha 的方案）

直接运行 make 命令进行编译

```
make -j8
```

注意：ollama 0.5.7 版本中存在 Makefile 文件，可以直接使用 make 命令进行翻译。新的版本(0.5.8 及以上) 没有 Makefile。要使用 ollama 0.5.7 版本，请

```
git checkout v0.5.7
```

2. 使用 go 语言命令进行编译（司延腾的构建方法）

```
go generate ./...
go build -p 8 .
```

3. 使用 cmake 进行编译(官方构建方法)

```
cmake -B build
cmake --build build
go run . serve
```

以上三个命令中，前两个命令用于编译 ollama 中使用 C/C++开发的库。最后一个命令，用于编译 go 语言开发的代码。运行最后一个命令的时候，会从 github 等网站上下载一些必需的子模块。注意，使用 go run 指令，不会生成 ollama 可执行文件。要生成可执行文件，需要再运行

```
go build .
```

生成可执行文件。

3.2 Loongnix20 系统

在旧世界编译运行 ollama，要做的事情就多很多了。

3.2.1 编译 cmake

Loongnix 默认的 CMake，版本是 3.13.4，而 Ollama 的编译需要至少 3.21(旧版本的 Ollama 需要的 Cmake 最低版本是 3.14)。注意：Cmake 从 3.21 开始才支持龙架构，低版本的 CMake 在 Loongnix 下无法直接编译。为了稳妥起见，直接下载最新版本的 cmake 3.31.5。

编译安装。

```
./configure
make
make install DESTDIR=~/bin/cmake
```

设置环境变量：

```
export PATH=$HOME/bin/cmake/usr/local/bin:$PATH
```

3.2.2 安装 golang 编译器

编译 Ollama，需要 go > =1.23.4。实际上，1.23.3 也可以用。

目前旧世界提供的最新的 Golang 编译器版本为 1.23.3，可以从如下的网址下载安装。

https://www.loongnix.cn/zh/toolchain/Golang/downloads-Go1.23/index.html

下载 Golang 编译器以后，请设置环境变量。假设 go 编译器安装在了~/bin/go 内。

```
export PATH=$HOME/bin/go/bin:$PATH
```

3.2.3 修改 go.mod

Ollama 源码中的 go.mod 内指定了 go 的版本为 1.23.4，而我们安装的版本是 1.23.3，并不能严格满足条件。直接编译的话，会尝试下载 go 1.23.4 源码，自行编译。这个是我们不需要的。为了避免这种事情，直接将 go.mod 中的 go 1.23.4 改为 go 1.23。

3.2.4 编译

如果要在旧世界正确的编译运行 ollama，请运行：

```
cmake -B build
cmake --build build
CGO_LDFLAGS="-O2 -g -lstdc++fs" CGO_CFLAGS_ALLOW=-mlasx CGO_CPPFLAGS_ALLOW=-mlasx go build .
```

CGO_LDFLAGS="-O2 -g -lstdc++fs" 是用来在链接的时候，链接 libstdc++fs,否则会报类似如下的错误：

```
ggml-backend-reg.cpp:(.text+0x1c94): undefined reference to `std::filesystem::__cxx11::directory_iterator::operator*() const'
```

4.运行 Ollama

首先，启动 ollama 服务

```
./ollama serve
```

运行服务以后，这个终端不要动，放在后台。另起一个终端，运行 deepseek R1

```
./ollama run deepseek-r1:7b
```

运行上述命令后，程序会自动下载 deepseek-r1 的模型。下载速度取决于网络连接，运气好的话，20 分钟左右即可完成下载。此处 7b 代表了大模型的大小，实际大小为 4.7GB。用户可以根据系统内存的大小来选择模型的大小。系统内存比较大的话，可以考虑 14b，32b，

70b 等更大的模型。

模型名称	模型大小（GB）
deepseek-r1:1.5b	1.1
deepseek-r1:7b	4.7
deepseek-r1:14b	9.0
deepseek-r1:32b	20
deepseek-r1:70b	43
deepseek-r1:671b	404

由于龙芯 3A6000 性能有限，跑大模型只具有演示性，不具备实际生产力。经过测试，32G 内存跑 32B 大模型，速度大概为 0.8~1.0 tokens/秒。如果要跑 7B 的大模型，速度大概为 4~5 tokens/s，但大模型会开始胡言乱语，需要谨慎使用。

参考文献：

1. 在 3a6000 上玩 deepseek-r1 大模型

https://www.loongbbs.cn/d/185-%E5%9C%A83a6000%E4%B8%8A%E7%8E%A9deepseek-r1%E5%A4%A7%E6%A8%A1%E5%9E%8B

2. 龙芯 3C6000 使用 Ollama 运行 DeepSeek R1 70B 大模型 https://katyusha.net/887.html
3. 在龙芯 3a6000 上部署 DeepSeek 和 Gemma2 大模型，
https://bbs.elecfans.com/m/jishu_2475508_1_1.html