CI

In addition to Github Actions `llama.cpp` uses a custom CI framework:

https://github.com/ggml-org/ci

It monitors the `master` branch for new commits and runs the ci/run.sh script on dedicated cloud instances. This allows us to execute heavier workloads compared to just using Github Actions. Also with time, the cloud instances will be scaled to cover various hardware architectures, including GPU and Apple Silicon instances.

Collaborators can optionally trigger the CI run by adding the `ggml-ci` keyword to their commit message. Only the branches of this repo are monitored for this keyword.

这段内容描述的是`llama.cpp`项目如何使用一个定制的持续集成（CI）框架，以及它与GitHub Actions的不同和特点。下面我将逐点解释每部分内容：

1. **GitHub Actions 和自定义CI框架**：`llama.cpp`项目不仅使用GitHub Actions，还使用了一个定制的CI框架，该框架的代码托管在`ggml-org/ci`。这表明项目采用了多种自动化测试和构建流程，以保证软件质量和稳定性。

2. **监控和执行脚本**：定制的CI框架监控`master`分支上的新提交。每当有新的代码提交到这个分支时，CI系统就会在云端的专用实例上运行一个名为`ci/run.sh`的脚本。这意味着每次代码更新都会触发自动化的构建和测试流程。

3. **云实例和扩展性**：使用云实例执行脚本，可以处理比单独使用GitHub Actions时更大的工作量。随着时间的推移，这些云实例还将扩展以支持不同的硬件架构，包括GPU和Apple Silicon（苹果矽芯片），这将有助于确保软件能在多种硬件平台上正常工作。

4. **触发CI运行**：项目的合作者可以选择性地通过在他们的提交信息中添加`ggml-ci`关键字来触发CI流程。这为开发者提供了额外的灵活性，允许他们在需要时手动启动测试和构建过程。

总体而言，这段内容强调了`llama.cpp`项目使用定制CI框架的优势和操作方式，使得软件开发和测试过程更加高效、可靠和适应多种硬件环境。

It is a good practice, before publishing changes to execute the full CI locally on your machine:

```
mkdir tmp

# CPU-only build
bash ./ci/run.sh ./tmp/results ./tmp/mnt

# with CUDA support
GG_BUILD_CUDA=1 bash ./ci/run.sh ./tmp/results ./tmp/mnt

# with SYCL support
source /opt/intel/oneapi/setvars.sh
GG_BUILD_SYCL=1 bash ./ci/run.sh ./tmp/results ./tmp/mnt
```

这段内容描述了如何使用定制的持续集成（CI）框架来增强`llama.cpp`项目的自动化测试和构建流程。现在我将对上述内容进行逐行解释。

1. **Github Actions**: 链接指向了`llama.cpp`项目在GitHub Actions中的状态和配置，说明该项目使用GitHub Actions进行自动化构建和测试。

2. **自定义CI框架**: 项目使用了一个定制的CI框架，其代码托管在`ggml-org/ci`。这个框架监控`master`分支的新提交，并在专用的云实例上运行`ci/run.sh`脚本。这种设置允许在比GitHub Actions更强大的环境中运行更重的负载。

3. **脚本链接**: 提供了`ci/run.sh`脚本的链接，这是执行CI任务的主脚本。

4. **云实例扩展**: 提到随着时间的推移，云实例将扩展以支持各种硬件架构，包括GPU和苹果的Apple Silicon实例，这表明CI框架将支持多种平台，以确保代码在不同硬件上都能正常运行。

5. **触发CI运行**: 合作者可以通过在提交信息中添加`ggml-ci`关键字来手动触发CI运行。这提供了更灵活的控制，允许开发者在需要时进行更细致的测试。

6. **本地执行CI**: 强调在发布更改前，最好在本地机器上执行完整的CI流程。以下是具体的脚本执行命令：

    - `mkdir tmp`：创建一个临时目录，用于存放构建和测试过程中生成的文件。
    - `bash ./ci/run.sh ./tmp/results ./tmp/mnt`：执行CI脚本，不包括GPU加速支持。
    - `GG_BUILD_CUDA=1 bash ./ci/run.sh ./tmp/results ./tmp/mnt`：设置环境变量`GG_BUILD_CUDA=1`，执行支持CUDA的GPU加速的构建。
    - `source /opt/intel/oneapi/setvars.sh`：加载Intel oneAPI的环境变量，这是为了支持SYCL（一种用于GPU和其他加速器的编程模型）。
    - `GG_BUILD_SYCL=1 bash ./ci/run.sh ./tmp/results ./tmp/mnt`：设置环境变量`GG_BUILD_SYCL=1`，执行支持SYCL的构建。

这些详细说明不仅帮助开发者理解如何利用CI工具优化和测试代码，还提供了在不同硬件上运行和测试的能力，这对于确保软件的广泛兼容性非常重要。

问题：这里的CI是什么？不理解

“CI”代表“持续集成”（Continuous Integration），这是软件开发中常用的一种实践，旨在让开发者频繁地将代码集成到共享存储库中。每次代码提交后，自动运行构建和测试流程，确保新代码的改动不会破坏现有功能，并且能够持续、自动地验证软件的质量。

在持续集成的过程中，通常会有自动化的工具来监控版本控制系统中的新提交，然后自动执行构建和测试脚本，最后反馈构建和测试的结果，帮助开发者尽早发现并解决问题。通过这种方式，团队可以更快地开发软件，同时保持较高的质量和稳定性。