EcoCompute 容器数据测量 · 中文使用手册

2026-08-18 · Hongping Zhang · 容器复现指南 / Container how-to

本手册教你如何用 **EcoCompute energy MLCube** 容器，在你自己的 GPU 上复现能耗测量，并把结果叠加到 quantenergy.tech 的交叉曲线上。

- 容器仓库：github.com/hongping-zh/ecocompute-mlcube（Apache-2.0）
- 对应网站版块：quantenergy.tech →「自己跑（Run it yourself）」；英文详版见容器页

> **我最想要的是一个「打脸」的结果。**目前所有数字都出自一个人手里的几张卡，复现页上独立复现数为 0。与曲线**不一致**的测量比再多一次自证有价值得多；审核只看格式与 schema，从不因为结论不合我意而拒稿。

1 · 这个容器是干什么的

它做一件很单纯的事：对你的 **GPU × 模型 × 精度** 组合做一次真实测量。

- 用 **NVML 直接采样 GPU 功耗**（10 Hz），不是用 TDP 估算。
- 同一次运行里测**你指定精度的量化版本**和**它自己的 FP16 基线**，两者的差才是可比的。
- 采样窗口**关闭之后**，再用同一个已加载的模型做一次 teacher-forcing 前向，算**困惑度**（perplexity），于是你同时看到「省了多少电」和「掉了多少质量」。这一步不进能耗账，只多几秒。
- 写出一份 `energy.json`，字段与站点图表完全一致，并当场做 schema 校验。
- 打印一个叠加链接，打开即把你的点画到交叉曲线上（**结果编码在 URL 里，全程浏览器本地解析，不上传**）。

单位说明：`energy_per_token_mj` 是**每个 token 的毫焦（mJ/token，等价于每 1000 token 的焦耳）**；站点图表把它换算成「每 100 万 token」显示。`vs_fp16_energy_pct`（ΔE%）是相对同模型 FP16 基线的能耗变化，正=更费电，负=更省电。

2 · 准备条件

方式	需要什么
有 Docker（推荐）	Docker + 一张 NVIDIA GPU + NVIDIA Container Toolkit（让 `--gpus` 可用）
无 Docker（租卡）	AutoDL / vast.ai 等租来的 GPU；只需 `git` 和 `python3`，脚本会自动建 venv
从源码	Docker 或 Python 环境 + 能 `git clone` 仓库

没有 NVIDIA GPU 也能「跑完」：容器会退回到公开数据集的参考值，但报告里会明确标成 `basis ≠ "measured"`——那不是测量，别拿去发布。

3 · 三种运行方式

方式一：有 Docker —— 一行命令（最推荐）

不用下载脚本、不用执行任何脚本，直接跑预构建镜像，命令完全透明：

```
docker run --rm --gpus all --user "$(id -u):$(id -g)" \
  -e HOME=/workspace/models/.hf -e HF_HOME=/workspace/models/.hf \
  -v "$PWD/ecocompute-out:/workspace/outputs" \
  -v "$HOME/.cache/ecocompute-hf:/workspace/models/.hf" \
  ghcr.io/hongping-zh/ecocompute-mlcube:latest \
  energy_estimate --model TinyLlama/TinyLlama-1.1B-Chat-v1.0 \
  --params_b 1.1 --precision NF4 --gpu_arch auto \
  --iterations 10 --warmup 2 --output_dir /workspace/outputs --share
```

- `--user "$(id -u):$(id -g)"`：避免输出文件归 root，方便你直接读取。
- 两个 `-v` 挂载：结果写到当前目录的 `ecocompute-out/`，模型缓存留在本机。
- `--gpu_arch auto`：架构从 NVML 设备名识别，一般不用改，也就无法把 4090 误标成 Blackwell。
- `--share`：结束后打印叠加链接。

方式二：无 Docker（AutoDL / vast.ai 等租卡）—— 一行 bootstrap

租卡环境通常不能嵌套 Docker，用这一行会自动克隆仓库、在数据盘上建 venv 并运行：

```
curl -fsSL https://raw.githubusercontent.com/hongping-zh/ecocompute-mlcube/main/quickstart.sh | bash
```

- 脚本自己判定：有可用 Docker 就走镜像，没有就走 venv（native）模式，并打印它选了哪条。
- 数据盘检测：存在 `/root/autodl-tmp` 时，代码、venv、模型缓存都放数据盘，避免系统盘爆满。
- 需要 `git` 和 `python3`；Python < 3.9 装不上发布钉版，报告会标注「与镜像运行不可比」。用 `ECO_PYTHON` 指定 ≥3.9 的解释器；venv 建好后不能换解释器，所以要配合 `ECO_RECREATE_VENV=1`。
- **下模型之前会先验一次真的 4-bit kernel。** `bitsandbytes` 必须与 torch 的 CUDA 线匹配，否则量化运行会在模型下载完之后才炸。检查失败时脚本先升级 `bitsandbytes`（一个小 wheel，走你已配好的国内源），不行才回退去 cu121 索引重装 torch（约 2.5 GB，租卡常拉不动）。**两条都失败就在下载模型之前停下**，不让你花时间换一份 `basis ≠ measured` 的报告；确实想跑用 `ECOCOMPUTE_ALLOW_FALLBACK=1`。升级过 bitsandbytes 就离开了发布钉版，报告会写明，ΔE% 不能直接和镜像运行比；想严格用 `ECO_KEEP_PINS=1`。

耗时要说实话：测量本身几分钟，第一次的时间几乎全在下载。我们唯一完整计时过的一次是**租来的 4090、native 模式、57 分钟**（约 2 GB 模型 5.5 分钟，其余是 ~3 GB 的 torch/CUDA wheel）。缓存全热之后同机重跑是 **1 分 48 秒**。docker 路径我们没计过时，就不编一个数字给你。

方式三：从源码（完全可控）

```
git clone https://github.com/hongping-zh/ecocompute-mlcube.git
cd ecocompute-mlcube

# A) 用 MLCommons MLCube CLI
pip install mlcube mlcube-docker
mlcube run --mlcube=. --task=energy_estimate --platform=docker

# B) 直接调入口（这里显式钉了架构，仅作演示；平时用 auto）
python3 entrypoint.py energy_estimate \
  --model TinyLlama/TinyLlama-1.1B-Chat-v1.0 \
  --precision NF4 --gpu_arch blackwell --params_b 1.1 \
  --output_dir workspace/outputs --prefetch --share
cat workspace/outputs/energy.json
```

`--prefetch` 会在测量前先问站点的预测值并打印出来，好让你当场看到「我的曲线在你的卡上准不准」；`--share` 给出可分享的叠加链接。另有一份 CPU-only 描述符（`mlcube.cpu.yaml`），没有 GPU 也能验证整条链路是否通。

4 · 常用参数

先说一个容易踩的坑： `ECOCOMPUTE_*` 环境变量**只有** **`quickstart.sh`** **认**。`entrypoint.py` 一个环境变量都不读，所以在 `docker run` 或直接调入口时，请用命令行参数（`--precision` 等）；在 `docker run` 前面加 `ECOCOMPUTE_PRECISION=INT8` 是**没有效果**的。

环境变量（仅 `quickstart.sh`）

变量	作用（默认值）
`ECOCOMPUTE_MODEL`	HuggingFace 模型 id（TinyLlama/TinyLlama-1.1B-Chat-v1.0）
`ECOCOMPUTE_PARAMS_B`	模型参数量，单位 B，必须与模型匹配（1.1）
`ECOCOMPUTE_PRECISION`	`NF4` 或 `INT8`（NF4；FP16 基线始终也会测）
`ECOCOMPUTE_ITERATIONS`	解码运行次数（10，与已发布数据集一致）
`ECOCOMPUTE_MODE`	强制 `docker` 或 `native`（有 Docker 用 docker，否则 native）
`ECOCOMPUTE_QUALITY`	设 `0` 关掉困惑度探针（默认开）
`ECOCOMPUTE_PREFETCH`	设 `1` 先取站点预测值再测（默认关；这是唯一会联网访问本站的行为）
`ECOCOMPUTE_ALLOW_FALLBACK`	设 `1`：量化后端坏掉时也照跑（报告将不是测量）
`ECOCOMPUTE_OUT` / `ECOCOMPUTE_CACHE`	输出目录（`./ecocompute-out`）／模型缓存（`$HOME/.cache/ecocompute-hf`）
`ECOCOMPUTE_GPU_ARGS`	Docker 的 GPU 参数（`--gpus all`；指定某卡：`--gpus "device=1"`）
`ECOCOMPUTE_IMAGE` / `ECOCOMPUTE_NO_BUILD`	镜像地址／拉取失败时直接报错而不本地构建
`ECOCOMPUTE_SRC`	native 模式的代码检出目录（默认放数据盘）
`ECO_PYTHON` / `ECO_RECREATE_VENV` / `ECO_KEEP_PINS`	指定解释器／重建 venv／不为修复而离开发布钉版

命令行参数（`energy_estimate` 子命令）

`--model` · `--params_b` · `--precision` · `--gpu_arch`（`auto` 或具体架构）· `--batch_size` · `--tokens` · `--iterations` · `--warmup` · `--sample_rate_hz` · `--output_dir` · `--share` · `--prefetch` · `--no_quality_probe` · `--quality_text 你自己的文本` · `--quality_seq_len` · `--dry_run`（强制无 GPU 参考路径）。

示例：换 INT8、指定第 1 号卡（注意精度是用**参数**给的）：

```
docker run --rm --gpus "device=1" --user "$(id -u):$(id -g)" \
  -e HOME=/workspace/models/.hf -e HF_HOME=/workspace/models/.hf \
  -v "$PWD/ecocompute-out:/workspace/outputs" \
  -v "$HOME/.cache/ecocompute-hf:/workspace/models/.hf" \
  ghcr.io/hongping-zh/ecocompute-mlcube:latest \
  energy_estimate --model TinyLlama/TinyLlama-1.1B-Chat-v1.0 \
  --params_b 1.1 --precision INT8 --gpu_arch auto \
  --iterations 10 --warmup 2 --output_dir /workspace/outputs --share
```

5 · 输出说明（energy.json）

结果写在 `ECOCOMPUTE_OUT/energy.json`（默认 `./ecocompute-out/energy.json`），并额外生成 `share_url.txt`。最关键的字段：

- `results.energy_per_token_mj` —— 这一配置的每 token 能耗（mJ）
- `results.fp16_energy_per_token_mj` —— 同模型 FP16 基线
- `results.vs_fp16_energy_pct` —— 相对 FP16 的能耗变化（正=更费，负=更省）
- `results.basis` —— `"measured"`（真实测量）/ `"interpolated"` / `"extrapolated"`（建模）
- `measurement_source` —— 测量来源，真实测量时是 `direct-nvml`
- `system_under_test.gpu` / `.gpu_arch`、`workload.model_name` / `.precision` / `.batch_size`
- `software` —— python / torch / transformers / bitsandbytes 版本，以及是否与发布钉版一致
- `quality`（schema 1.1 新增，可选）—— `value` 是困惑度，`fp16_value` 是同一次运行的 FP16 基线，`delta_vs_fp16_pct` 是两者之差；`corpus.sha256` 记录评测文本指纹，探针失败时 `basis` 为 `"unavailable"` 并附错误原因。

运行结束的打印示例（数值取自我们 2026 年 7 月那次真实的 RTX 4090 运行；`quality` 行是新增的，那次运行还没有）：

```
  GPU             : NVIDIA GeForce RTX 4090 (ada)
  Workload        : TinyLlama/TinyLlama-1.1B-Chat-v1.0  NF4  batch 1
  Energy/token    : 1883.91 mJ  (FP16 baseline 1619.68 mJ)
  vs FP16         : +16.3 %
  Basis           : measured   (source: direct-nvml)
  Quality (ppl)   : 12.4831 vs FP16 12.3016  (+1.475%)
  Software        : python 3.10.12, torch 2.5.1+cu121, transformers 4.57.6, bitsandbytes 0.
43.3
```

关于困惑度这一栏怎么读

- **只看 Δ，不要看绝对值。**困惑度依赖评测文本和分词器，绝对数字与论文里的 WikiText 值不可比；只有「同一次运行里 FP16 与量化版跑同样的字节」这个差是干净的。
- **困惑度不是任务质量。**它衡量的是「这次量化把语言模型打坏了多少」，不能替代推理、长上下文等下游评测。
- 评测文本内置在仓库 `quality/` 目录（两段公有领域文本，约 1.06 万词），不联网、不引入新数据集依赖，所有人跑的是同样的字节，sha256 写进报告。想用自己的留出文本：`--quality_text`，指纹同样会记录，避免拿不同语料的结果互相比较。

6 · 把结果叠加到网站曲线

两种方式，都不上传任何数据：

1. **分享链接（最省事）**：加了 `--share` 后终端会打印 `quantenergy.tech/?tab=run&overlay=…`，打开即把你的点恢复到交叉曲线上（结果编码在 URL 里）。
2. **拖文件**：在「自己跑」标签底部把 `energy.json` 拖进上传区（或点击选择）；你的点按「模型规模 × ΔE%」落到曲线上，颜色由 `basis` 决定，架构与当前曲线不符时会加一圈虚线提示。

要**进入公开的复现画廊**，去 /replications/#submit：文件同样只在你浏览器里解析，按钮会带着摘要打开一个 GitHub issue，附件由你自己上传、自己发送。全程不要邮箱；想撤回，在 issue 上说一句即可，不问理由。

7 · 故障排查

- **`nvidia-smi` 找不到 / 没有 GPU** → 仍能跑完，但报告是数据集派生值（`basis ≠ "measured"`）。发布前先修好 GPU 访问。
- **Docker 报 `could not select device driver with capabilities: [[gpu]]`** → 多半没装 NVIDIA Container Toolkit。脚本会不带 GPU 重试以验证流程，但结果同样不是测量。安装指南
- **某些消费卡 / vGPU / Turing 卡报 `NVML_ERROR_NOT_SUPPORTED`** → 这类卡读不到功耗遥测，容器不会假装测到，同样退回派生值。采样失败的次数记在 `results.dropped_samples`，不隐藏。
- **镜像拉取失败** → 本机已有该镜像就直接用；否则脚本尝试从源码本地构建（约 10–20 分钟，主要是 torch）；设了 `ECOCOMPUTE_NO_BUILD=1` 则直接报错退出。
- **native 模式提示 Python 太旧（<3.9）** → 报告会标「与镜像运行不可比」。解决：用 ≥3.9 的解释器重建 venv：

  ```
  ECO_RECREATE_VENV=1 ECO_PYTHON=/path/to/python3.10 SKIP_DOWNLOAD=1 bash autodl/00_setup.sh
  ```

- **提示 `bitsandbytes … cannot run a NF4 kernel against torch's CUDA …`** → 见第 3 节方式二：这是 torch 的 CUDA 线与 bitsandbytes 内核不匹配，脚本会先试着升级 bitsandbytes。
- **首次运行特别慢** → 在下载模型权重到 `HF_HOME` 缓存，之后复用。

8 · 诚实声明 / 注意事项

1. **测量与估算分得清**：只有 `measurement_source: "direct-nvml"` 且 `basis: "measured"` 才是真实 NVML 测量；读不到遥测时退回公开数据集并明确标注，绝不把估算伪装成测量。
2. **NVML 测的是 GPU 封装功耗**，不是整机墙上功耗——不含 CPU、内存、电源损耗与散热，也不换算 PUE 或 CO_2e（那需要我没有的电网模型）。工况是单流、batch 1、256 token。
3. **10 次解码迭代 ≠ 10 次独立试验**：迭代发生在同一次运行内部，**一份报告永远是 n = 1**，无论迭代数多高。
4. **样本量**：主数据集 v1.1.0 每个配置 n = 2（CV < 2%）；RTX 4090（Ada）深挖是每个配置 n = 1，属补充性案例研究。同一配置隔一次会话重跑，我们实测到的差异可达 8.2 个百分点（1.1B INT8：+146.1% 与 +137.9%）。
5. **量化内核随版本变化**：你的 torch / transformers / bitsandbytes 与发布钉版不同时报告会提示，跨版本比较 ΔE% 必须注明。结论也因此是「这个后端在这张卡上」的结论，不是对 NF4/INT8 格式本身的判决。

6. **不是认证基准**：每份报告都写着 `certified_benchmark_result: false`。使用 MLCube 规范不代表 MLCommons 背书，也不是官方 MLPerf 结果。
7. **困惑度是代理指标**：只有 Δ 有意义，且不等于下游任务质量（见第 5 节）。
8. **隐私**：`energy.json` 只在你浏览器本地解析，不上传服务器；分享链接把结果编码在 URL 里。

9 · 引用与链接

- 容器仓库：ecocompute-mlcube（Apache-2.0；数据 CC BY 4.0）
- 论文预印本（SSRN #6854700）：papers.ssrn.com
- 主数据集 v1.1.0：10.5281/zenodo.19647290
- RTX 4090（Ada）实测深挖：10.5281/zenodo.21528103 —— **勘误**：该版本 `rtx4090_results.csv` 里的 `vs_fp16_pct` 一列有误（例如 Qwen2.5-3B NF4 写作 +10.1%，按其自身能量列应为 +0.8%）。能量、功率、吞吐三列彼此自洽且未受影响，站点与曲线一直用能量列重算，详见引用页。
- 提交你的复现结果：/replications/#submit

本手册对应 `quickstart.sh` 与 `entrypoint.py` 的当前行为（schema `ecocompute-energy/1.1`）；英文详版见容器页，站点整体的中文导览见中文使用说明。

容器：ecocompute-mlcube · 论文：SSRN #6854700 · 主数据集：10.5281/zenodo.19647290