NeuroBreak 部署与使用文档

本指南覆盖从准备代码、上传到计算容器、构建运行环境，到前端访问地址开放的完整流程。按顺序执行可以在 GPU 服务器上复现双模型协同工作台。

目录

项目组件速览

模块	位置	说明
容器环境	`docker/Dockerfile`	基于 `nvidia/cuda:11.8.0-cudnn8-runtime-ubuntu22.04`，预装 PyTorch、Transformers、FastAPI 等依赖。
推理与审核脚本	`scripts/download_models.py`、`scripts/run_io_tests.py`	下载 Llama-3.2-3B 与 Llama-Guard-3-1B，提供 I/O 冒烟测试。
Web 前端	`frontend/`	Vite + React + Tailwind，暴露参数面板、对话、审核与监控组件。
静态资源	`frontend/dist/`	`npm run build` 后的产物，可由任意 HTTP 服务托管。
数据/缓存	`data/`、`notebooks/`	模型缓存、测试结果输出目录，可通过挂载映射到宿主机。

前置条件

- 一台具备 NVIDIA GPU 的 Linux 服务器（推荐 24G 显存+），已安装 Docker ≥ 24.0 与 `nvidia-container-toolkit`。

- 本地开发机安装 Git、Node.js 18+/pnpm 或 npm、`scp/rsync` 等上传工具。
- HuggingFace 账号并已申请 `meta-llama/Llama-3.2-3B` 与 `meta-llama/Llama-Guard-3-1B` 访问权限。
- 开放端口：后端默认 `8000`，前端默认 `4173`（可按需修改）。

准备与上传项目文件

1. **获取代码**

```
git clone https://github.com/<your-org>/NeuroBreak-Reproduction.git
cd NeuroBreak-Reproduction
```

2. **打包（可选）**

```
tar czf neurobreak.tar.gz NeuroBreak-Reproduction
```

3. **上传到服务器**

```
scp -r NeuroBreak-Reproduction user@SERVER_IP:/opt/
# 或者使用 rsync，便于后续增量同步
rsync -av --progress NeuroBreak-Reproduction/
user@SERVER_IP:/opt/NeuroBreak-Reproduction
```

4. **服务器端校验**

```
ssh user@SERVER_IP
cd /opt/NeuroBreak-Reproduction
ls
```

构建 Docker 镜像

在服务器根目录执行：

```
cd /opt/NeuroBreak-Reproduction
docker build -t neurobreak:latest -f docker/Dockerfile .
```

> 提示：如需加速，可在构建命令前设置国内镜像源或传入 `--build-arg https_proxy`。

启动容器并初始化环境

建议将代码目录与缓存/数据目录通过 Volume 映射到宿主机，方便持久化：

```
docker run --gpus all -it \
  --name neurobreak \
  -p 8000:8000 \
  -p 4173:4173 \
  -v /opt/NeuroBreak-Reproduction:/workspace \
  -v /opt/nb-cache:/workspace/.cache \
  neurobreak:latest \
  /bin/bash
```

进入容器后，默认工作目录为 `/workspace`。可以创建 `.env` 记录敏感配置（如 HuggingFace Token）：

```
cat <<'EOF' > /workspace/.env
HF_TOKEN=hf_xxx
VITE_API_BASE_URL=http://SERVER_IP:8000
VITE_USE_MOCK=false
EOF
```

> `.env` 仅在容器内可读，确保不要提交到 Git。

下载模型与健康检查

1. **登录 HuggingFace（可选）**

   ```
   huggingface-cli login --token $HF_TOKEN
   ```

2. **下载模型**

   ```
   python scripts/download_models.py --all \
     --output /workspace/.cache/huggingface/models
   ```

3. **运行 I/O 冒烟测试**

   ```
   python scripts/run_io_tests.py
   # 结果输出 notebooks/io_test_results.json
   ```

如需英文版或更多 prompt，可运行 `scripts/run_io_tests_2.py`。

提供后端 API

前端默认调用以下接口（详见 `frontend/src/lib/api.ts` 与 `frontend/src/types/models.ts`）：

方法	路径	说明	关键字段

方法	路径	说明	关键字段
POST	/api/pipeline/run	执行推理 + Guard 联合流程	inferenceConfig、guardConfig
POST	/api/moderate	独立安全审核文本	threshold、categories

可以使用 FastAPI/Falcon 等框架实现。示例启动命令（假设实现文件为 engine/server.py，FastAPI 应用名为 app）：

```
uvicorn engine.server:app --host 0.0.0.0 --port 8000 --reload
```

若后端尚未完成，可将前端 .env 中的 VITE_USE_MOCK 设为 true，前端会返回 frontend/src/lib/mock.ts 提供的模拟数据，方便先行演示。

构建与部署前端

1. **安装依赖**

   ```
   cd /workspace/frontend
   npm install
   ```

2. **配置环境变量**
   创建 frontend/.env：

   ```
   VITE_API_BASE_URL=http://SERVER_IP:8000
   VITE_USE_MOCK=false
   ```

3. **开发模式（可选）**

   ```
   npm run dev -- --host 0.0.0.0 --port 5173
   ```

4. **生成生产包**

   ```
   npm run build
   ```

   构建结果位于 frontend/dist/。

5. **预览或直接托管**

   ```
   npm run preview -- --host 0.0.0.0 --port 4173
   # 或者使用任意静态服务器
   ```

```
npx serve -s dist -l 4173
```

> 若选择 Nginx/Traefik，可将 `frontend/dist` 拷贝到其根目录，或在 docker-compose 中新增前端服务挂载该目录。

打开网址与常见访问方式

- **Vite 预览/静态服务**：浏览器访问 `http://SERVER_IP:4173`。
- **通过反向代理**：将公网域名解析到服务器，在 Nginx 中配置：

```
server {
  listen 80;
  server_name neurobreak.example.com;
  location / {
    proxy_pass http://127.0.0.1:4173;
  }
}
```

- **内网调试**：使用 `ssh -L 4173:localhost:4173 user@SERVER_IP` 将端口转发到本地。

更新与维护

- **更新代码**：在宿主机同步后，进入容器执行 `git pull`，必要时重新 `npm run build`。
- **更新依赖**：后端通过 `pip install -r requirements.txt --upgrade`，前端 `npm update`。
- **重启容器**：

```
docker restart neurobreak
docker exec -it neurobreak /bin/bash
```

- **备份数据**：定期复制 `/workspace/notebooks`、`/workspace/data` 至安全位置。

常见问题排查

- **端口被占用**：修改 `docker run` 或 `npm run preview` 中的 `-p` 端口映射，确保与宿主机其他服务不冲突。
- **HuggingFace 下载报 401/403**：确认已申请模型权限，并在容器内 `export HF_TOKEN=...` 后重试。
- **显存不足**：在 `scripts/run_io_tests.py` 中调低 `max_new_tokens` 或切换到 CPU（自动 fallback 为 `float32`）。
- **前端无法访问 API**：检查 `frontend/.env` 配置、CORS 设置以及后端日志。临时可将 `VITE_USE_MOCK=true` 验证 UI。
- **构建 Warning（chunk 过大）**：可在 `frontend/vite.config.ts` 中通过 `build.chunkSizeWarningLimit` 调整，或拆分图表等大依赖。

完成以上步骤后，即可在网页端实时调参 Llama 推理与 Llama Guard 审核。若需进一步自动化（如 docker compose 或 CI/CD），可在此文档基础上扩展。祝部署顺利！