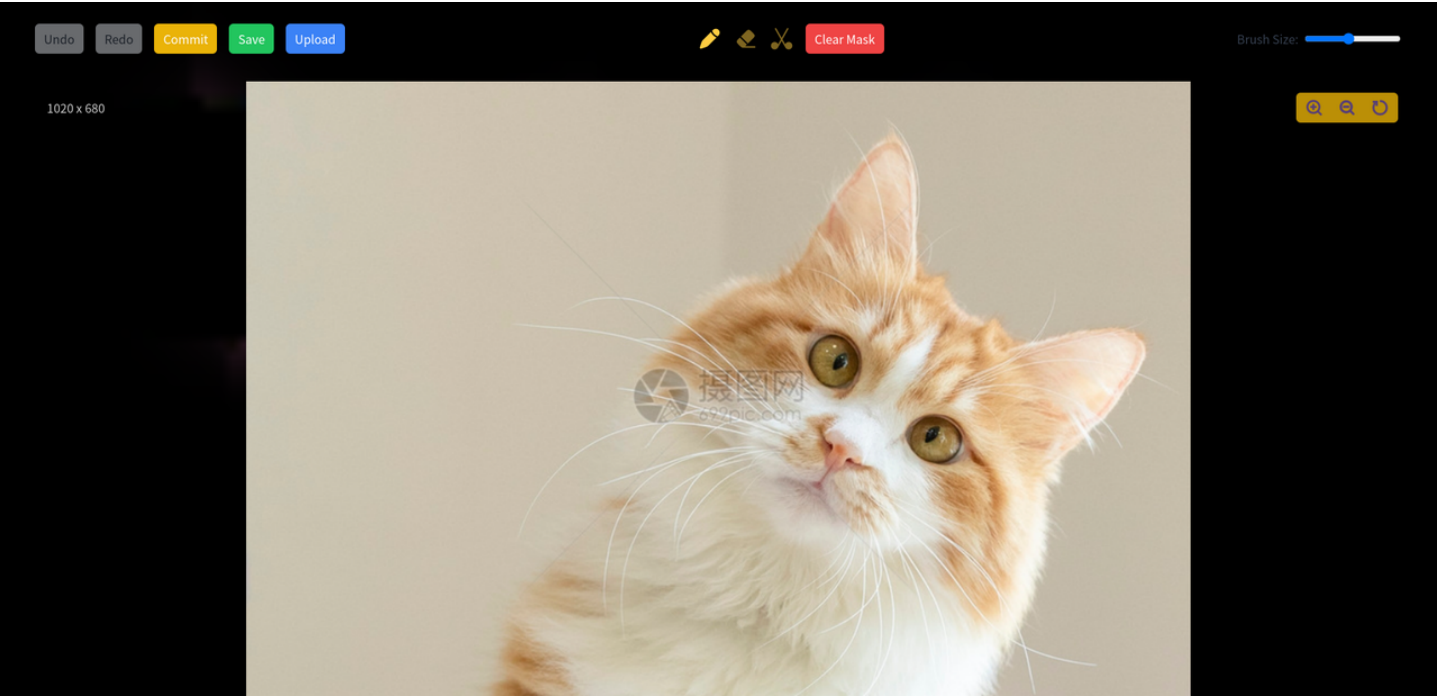
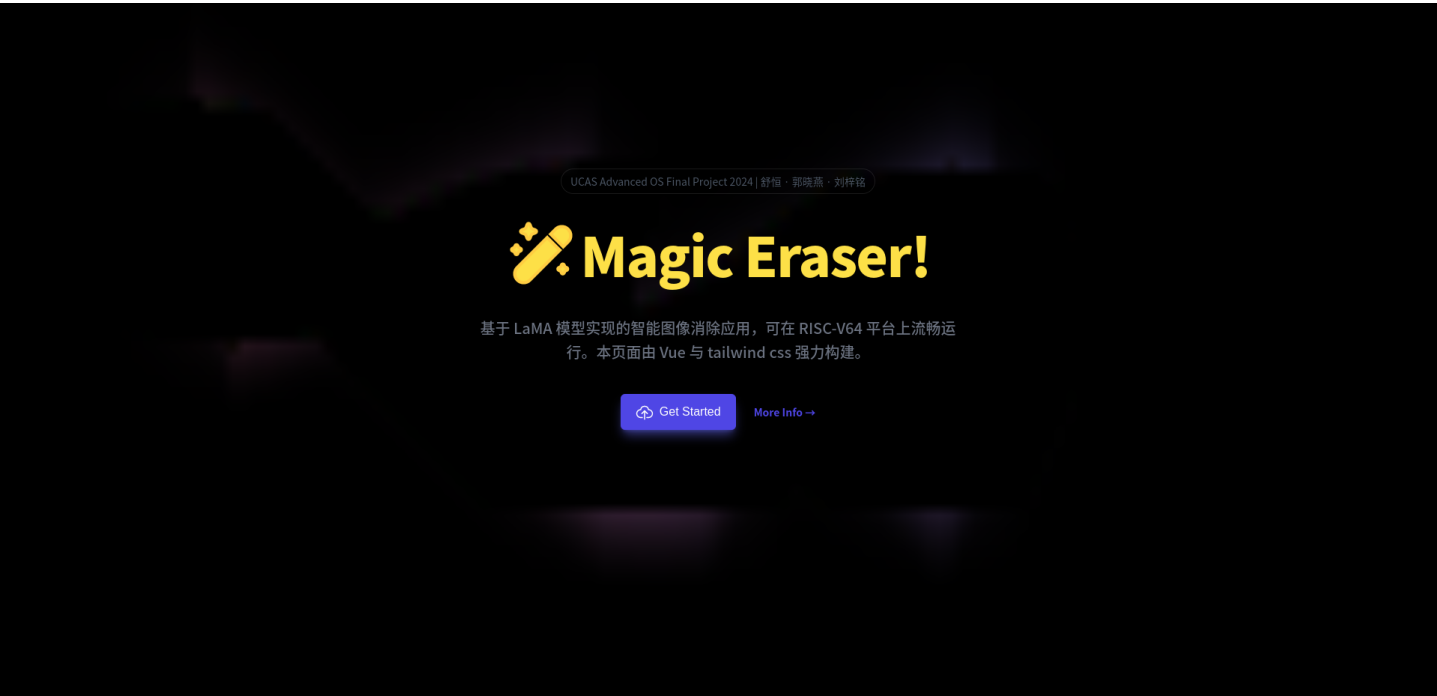


MagicEraser: 2024 OS Final

简介

本小组在基于 openEuler 操作系统的 RISC-V 架构笔记本电脑 RVBook 上，成功开发了一款具备 AI 图像消除功能的 Web 离线工具 MagicEraser。

该工具允许用户通过多种遮罩绘制工具（包括可调节大小的涂鸦笔和套索）指定需消除部分，系统会智能识别输入，完成消除功能，并进行平滑处理，同时可保证图片的自然感和美观度。此外，本工具支持 JPG、PNG 和 BMP 等多种图片输入格式。



组员分工

	工作内容
舒恒	图像修复模型测试，模型的本地部署与相关依赖的原生编译，编译手册撰写
郭晓燕	Web 前端页面开发，前后端交互代码开发，PPT 制作
刘梓铭	模型在 RVbook 上的测试和编译

全部成员均有技术文档贡献。

代码说明

本项目的代码结构如下所示：

```
1  .
2  |— frontend
3  |   |— eslint.config.js
4  |   |— index.html
5  |   |— jsconfig.json
6  |   |— package.json
7  |   |— package-lock.json
8  |   |— postcss.config.js
9  |   |— public
10 |   |   |— favicon.ico
11 |   |— README.md
12 |   |— src
13 |   |   |— App.vue
14 |   |   |— assets
15 |   |   |   |— main.js
16 |   |— tailwind.config.js
17 |   |— vite.config.js
18 |— ImageInpainting
19 |   |— ade20k
20 |   |   |— ade20k-resnet50dilated-ppm_deepsup
21 |   |— big-lama
22 |   |   |— config.yaml
23 |   |   |— models
24 |   |— bin
25 |   |   |— analyze_errors.py
26 |   |   |— app.py
27 |   |   |— blur_predicts.py
28 |   |   |— calc_dataset_stats.py
29 |   |   |— debug
```

```
30 | | | └─ evaluate_predicts.py
31 | | | └─ evaluator_example.py
32 | | | └─ extract_masks.py
33 | | | └─ filter_sharded_dataset.py
34 | | | └─ gen_debug_mask_dataset.py
35 | | | └─ gen_mask_dataset_hydra.py
36 | | | └─ gen_mask_dataset.py
37 | | | └─ gen_outpainting_dataset.py
38 | | | └─ make_checkpoint.py
39 | | | └─ mask_example.py
40 | | | └─ outputs
41 | | | └─ paper_runfiles
42 | | | └─ predict_inner_features.py
43 | | | └─ predict.py
44 | | | └─ report_from_tb.py
45 | | | └─ sample_from_dataset.py
46 | | | └─ side_by_side.py
47 | | | └─ split_tar.py
48 | | | └─ to_jit.py
49 | | | └─ train.py
50 | └─ conda_env.yml
51 | └─ configs
52 | | └─ analyze_mask_errors.yaml
53 | | └─ data_gen
54 | | └─ debug_mask_gen.yaml
55 | | └─ eval1.yaml
56 | | └─ eval2_cpu.yaml
57 | | └─ eval2_gpu.yaml
58 | | └─ eval2_jpg.yaml
59 | | └─ eval2_segm_test.yaml
60 | | └─ eval2_segm.yaml
61 | | └─ eval2_test.yaml
62 | | └─ eval2.yaml
63 | | └─ places2-categories_157.txt
64 | | └─ prediction
65 | | └─ test_large_30k.lst
66 | | └─ training
67 | └─ docker
68 | | └─ 1_generate_masks_from_raw_images.sh
69 | | └─ 2_predict_with_gpu.sh
70 | | └─ 3_evaluate.sh
71 | | └─ build-cuda111.sh
72 | | └─ build.sh
73 | | └─ Dockerfile
74 | | └─ Dockerfile-cuda111
75 | | └─ entrypoint.sh
76 | └─ fetch_data
```

```
77 | | | └─ celebahq_dataset_prepare.sh
78 | | | └─ celebahq_gen_masks.sh
79 | | | └─ eval_sampler.py
80 | | | └─ places_challenge_train_download.sh
81 | | | └─ places_standard_evaluation_prepare_data.sh
82 | | | └─ places_standard_test_val_gen_masks.sh
83 | | | └─ places_standard_test_val_prepare.sh
84 | | | └─ places_standard_test_val_sample.sh
85 | | | └─ places_standard_train_prepare.sh
86 | | | └─ sampler.py
87 | | | └─ train_shuffled.flist
88 | | | └─ val_shuffled.flist
89 | └─ __init__.py
90 | └─ lama_env
91 | | └─ bin
92 | | └─ include
93 | | └─ lib
94 | | └─ lib64 -> lib
95 | | └─ pyvenv.cfg
96 | └─ LaMa_inpainting.ipynb
97 | └─ LaMa_models
98 | | └─ big-lama
99 | | └─ big-lama-with-discr
100 | | └─ lama-celeba-hq
101 | | └─ lama-celeba-hq-pl-abl
102 | | └─ LaMa_models
103 | | └─ lama-pl-abl
104 | | └─ lama-places
105 | └─ LaMa_test_images
106 | | └─ image1_mask001.png
107 | | └─ image1.png
108 | └─ LICENSE
109 | └─ main.py
110 | └─ models
111 | | └─ ade20k
112 | | └─ lpips_models
113 | └─ numpy-1.25.0-cp311-cp311-linux_riscv64.whl
114 | └─ output
115 | | └─ image1_mask001.png
116 | └─ outputs
117 | | └─ 2024-11-20
118 | | └─ 2024-12-23
119 | | └─ 2024-12-24
120 | | └─ 2024-12-27
121 | | └─ 2024-12-28
122 | └─ predict.py
123 | └─ pre.txt
```

```
124 | | └─ README.md
125 | | └─ requirements.txt
126 | | └─ riscv64-unknown-elf-g++
127 | |   └─ g++
128 | |   └─ gcc
129 | |   └─ gcc-ar
130 | |   └─ gcc-nm
131 | |   └─ gcc-ranlib
132 | |   └─ riscv64-openEuler-linux-g++
133 | |   └─ riscv64-openEuler-linux-gcc
134 | |   └─ riscv64-openEuler-linux-gcc-12
135 | └─ riscv-gnu-toolchain
136 |   └─ autom4te.cache
137 |   └─ binutils
138 |   └─ build
139 |   └─ build-binutils-newlib
140 |   └─ build-gcc-newlib-stage1
141 |   └─ build-gcc-newlib-stage2
142 |   └─ build-gdb-newlib
143 |   └─ build-newlib
144 |   └─ build-newlib-nano
145 |   └─ config.log
146 |   └─ config.status
147 |   └─ configure
148 |   └─ configure~
149 |   └─ configure.ac
150 |   └─ contrib
151 |   └─ dejagnu
152 |   └─ gcc
153 |   └─ gdb
154 |   └─ glibc
155 |   └─ install-newlib-nano
156 |   └─ LICENSE
157 |   └─ linux-headers
158 |   └─ llvm
159 |   └─ Makefile
160 |   └─ Makefile.in
161 |   └─ musl
162 |   └─ newlib
163 |   └─ pk
164 |   └─ qemu
165 |   └─ README.md
166 |   └─ regression
167 |   └─ scripts
168 |   └─ spike
169 |   └─ stamps
170 |   └─ test
```

```
171 | | └─ uclibc-ng
172 | └─ riscv-isa-sim
173 | | └─ aclocal.m4
174 | | └─ arch_test_target
175 | | └─ ax_append_flag.m4
176 | | └─ ax_append_link_flags.m4
177 | | └─ ax_boost_asio.m4
178 | | └─ ax_boost_base.m4
179 | | └─ ax_boost_regex.m4
180 | | └─ ax_check_compile_flag.m4
181 | | └─ ax_check_link_flag.m4
182 | | └─ ax_require_defined.m4
183 | | └─ build
184 | | └─ ChangeLog.md
185 | | └─ ci-tests
186 | | └─ config.h.in
187 | | └─ configure
188 | | └─ configure.ac
189 | | └─ customext
190 | | └─ debug_rom
191 | | └─ disasm
192 | | └─ fdt
193 | | └─ fesvr
194 | | └─ LICENSE
195 | | └─ Makefile.in
196 | | └─ README.md
197 | | └─ riscv
198 | | └─ riscv-disasm.pc.in
199 | | └─ riscv-fesvr.pc.in
200 | | └─ riscv-riscv.pc.in
201 | | └─ scripts
202 | | └─ softfloat
203 | | └─ spike_dasm
204 | | └─ spike_main
205 | | └─ VERSION
206 | └─ riscv-pk
207 | | └─ aclocal.m4
208 | | └─ bbl
209 | | └─ build
210 | | └─ config.h.in
211 | | └─ configure
212 | | └─ configure.ac
213 | | └─ dummy_payload
214 | | └─ LICENSE
215 | | └─ LICENSE.Batten
216 | | └─ LICENSE.riscv_logo.txt
217 | | └─ machine
```

```
218 | | | └─ Makefile.in
219 | | | └─ pk
220 | | | └─ README.md
221 | | | └─ scripts
222 | | | └─ softfloat
223 | | | └─ util
224 | | └─ riscv-tflm
225 | | | └─ AUTHORS
226 | | | └─ ci
227 | | | └─ CODEOWNERS
228 | | | └─ CONTRIBUTING.md
229 | | | └─ docs
230 | | | └─ LICENSE
231 | | | └─ README.md
232 | | | └─ tensorflow
233 | | | └─ third_party
234 | | | └─ WORKSPACE
235 | | └─ saicinpainting
236 | | | └─ evaluation
237 | | | └─ __init__.py
238 | | | └─ __pycache__
239 | | | └─ training
240 | | | └─ utils.py
241 | | └─ scikit-image
242 | | | └─ asv.conf.json
243 | | | └─ azure-pipelines.yml
244 | | | └─ benchmarks
245 | | | └─ CITATION.cff
246 | | | └─ CODE_OF_CONDUCT.md
247 | | | └─ CONTRIBUTING.rst
248 | | | └─ CONTRIBUTORS.md
249 | | | └─ doc
250 | | | └─ INSTALL.rst
251 | | | └─ LICENSE.txt
252 | | | └─ MANIFEST.in
253 | | | └─ meson.build
254 | | | └─ pyproject.toml
255 | | | └─ README.md
256 | | | └─ RELEASE.txt
257 | | | └─ requirements
258 | | | └─ requirements.txt
259 | | | └─ SECURITY.md
260 | | | └─ skimage
261 | | | └─ TODO.txt
262 | | | └─ tools
263 | | └─ tensorflow-riscv
264 | | └─ ACKNOWLEDGMENTS
```

```
265 | | | └─ ADOPTERS.md
266 | | | └─ arm_compiler.BUILD
267 | | | └─ AUTHORS
268 | | | └─ BUILD
269 | | | └─ CODE_OF_CONDUCT.md
270 | | | └─ CODEOWNERS
271 | | | └─ configure
272 | | | └─ configure.cmd
273 | | | └─ configure.py
274 | | | └─ CONTRIBUTING.md
275 | | | └─ ISSUES.md
276 | | | └─ ISSUE_TEMPLATE.md
277 | | | └─ LICENSE
278 | | | └─ models.BUILD
279 | | | └─ nohup.out
280 | | | └─ README.md
281 | | | └─ RELEASE.md
282 | | | └─ SECURITY.md
283 | | | └─ tensorflow
284 | | | └─ third_party
285 | | | └─ toolchain
286 | | | └─ tools
287 | | | └─ WORKSPACE
288 | | └─ torch-2.0.0a0+gitc263bd4-cp311-cp311-linux_riscv64.whl
289 | | └─ torch-2.3.0a0+gitunknown-cp311-cp311-linux_riscv64.whl
290 | | └─ vision
291 | | | └─ android
292 | | | └─ build
293 | | | └─ cmake
294 | | | └─ CMakeLists.txt
295 | | | └─ CODE_OF_CONDUCT.md
296 | | | └─ CONTRIBUTING.md
297 | | | └─ CONTRIBUTING_MODELS.md
298 | | | └─ dist
299 | | | └─ docs
300 | | | └─ examples
301 | | | └─ gallery
302 | | | └─ hubconf.py
303 | | | └─ ios
304 | | | └─ LICENSE
305 | | | └─ MANIFEST.in
306 | | | └─ mpyy.ini
307 | | | └─ packaging
308 | | | └─ pyproject.toml
309 | | | └─ pytest.ini
310 | | | └─ README.rst
311 | | | └─ references
```



```
312 | | | |— scripts
313 | | | |— setup.cfg
314 | | | |— setup.py
315 | | | |— test
316 | | | |— torchvision
317 | | | |— torchvision.egg-info
318 | | | |— version.txt
319 | | | |— xsimd
320 | | | |— benchmark
321 | | | |— build
322 | | | |— Changelog.rst
323 | | | |— cmake
324 | | | |— CMakeLists.txt
325 | | | |— CONTRIBUTING.md
326 | | | |— docs
327 | | | |— environment.yml
328 | | | |— examples
329 | | | |— include
330 | | | |— install_sde.sh
331 | | | |— LICENSE
332 | | | |— README.md
333 | | | |— readthedocs.yml
334 | | | |— test
335 | | | |— xsimdConfig.cmake.in
336 | | | |— xsimd.pc.in
337 | | | |— testImg
338 | | | |— testImg00.png
339 | | | |— testImg01.jpg
340 | | | |— testImg02.jpg
341 | | | |— testImg03.png
```

本地运行指令：

```
1 ## 开启前端
2 MagicEraser/frontend $ npm install
3 MagicEraser/frontend $ npm run dev
4 ## 开启后端
5 MagicEraser/ImageInpainting $ ./lama_env/bin/python3 ./bin/app.py
```

然后，在浏览器中访问 `npm run dev` 命令显示的托管页面，即可使用本工具。

项目代码中：

- **frontend**: Web 前端代码所在文件夹，基于 Vue3 和 tailwind css 实现。
- **ImageInpainting**: 后端代码所在文件夹，完成要求项目的复现工作。
 - `./bin/app.py`: 前后端交互代码所在文件。
 - 实现了一个基于 Flask 的图像处理 Web 服务，提供 AI 模型的图像修复功能。
 - 用户通过 `POST` 请求提交待处理图片和遮罩的 base64URL 数据，服务会将其转换成模型需要的 dataset 格式，并使用深度学习模型进行预测处理，并返回处理后的图像。

项目源码可在<https://github.com/maxiumshu/MagicEraser.git>中查看，frontend 与 ImageInpainting 分属 main 与 master 分支。

下面对于进行推理的 `./bin/app.py` 脚本文件中的代码进行进一步说明：

1. 首先，为了确保多线程和并行计算的效率，脚本通过设置环境变量控制计算库的线程数，避免线程争用：

```
1 os.environ['OMP_NUM_THREADS'] = '1'
2 os.environ['OPENBLAS_NUM_THREADS'] = '1'
3 os.environ['MKL_NUM_THREADS'] = '1'
4 os.environ['VECLIB_MAXIMUM_THREADS'] = '1'
5 os.environ['NUMEXPR_NUM_THREADS'] = '1'
```

2. `load_model` 函数负责加载预训练模型并进行初始化。它从 Hydra 配置文件 `default.yaml` 中获取模型配置，并加载训练好的权重（checkpoint）。加载后的模型冻结，避免在推理时进行训练更新。模型将被加载到 CPU 上（本实验的设备）。

```
1 @hydra.main(config_path='../configs/prediction', config_name='default.yaml')
2 def load_model(predict_config: OmegaConf):
3     ...
```

3. 在收到来自前端的 `base64` 编码的图像和掩码后，脚本通过以下函数进行解码和预处理：

- `base64_to_image(base64_str)`：将 base64 编码的字符串转换为图像。
- `load_image_from_base64(base64_str)`：将 base64 字符串转换为 NumPy 数组并标准化为 `float32` 类型。
- `load_mask_from_base64(base64_str)`：将 base64 字符串转换为单通道（灰度）图像，作为掩码。

此外，若需要处理的图像尺寸不是某个给定的倍数，脚本还会进行图像填充（padding），确保图像的高度和宽度都能被指定的模块数整除。相关函数包括：

- `pad_img_to_modulo(img, mod)`：将图像填充到指定的模块倍数。

- `pad_tensor_to_modulo(img, mod)`：将张量填充到指定的模块倍数。
4. `process_img(image_data, mask_data)`：这是主要的图像处理和推理函数。它将图像和掩码数据加载并进行预处理，之后传入深度学习模型进行推理。

如果配置了 `refine`，则调用 `refine_predict` 进行图像的细化处理。最终，结果图像会被转换为 `base64` 格式并返回给客户端。

```
1 def process_img(image_data, mask_data):
2     try:
3         ...
4         if predict_config.get('refine') is True:
5             result = refine_predict(batch, model,
6                                     **predict_config.get('refiner'))
7             result = result[0].permute(1, 2, 0).detach().cpu().numpy()
8         else:
9             with torch.no_grad():
10                batch = move_to_device(batch, device)
11                batch['mask'] = (batch['mask'] > 0) * 1
12                batch = model(batch)
13                result = batch[predict_config.get('out_key')][0].permute(1, 2,
14                                0).detach().cpu().numpy()
15            ...
16    except Exception as e:
17        logger.critical(f"Error during image processing: {str(e)}")
18    return None
```

Flask 提供了一个 `POST` 路由 `/predict`，用于接收图像和掩码数据并返回处理后的图像。前端通过 `JSON` 格式发送数据，后端进行处理并返回 `base64` 格式的结果图像。

```
1 @app.route('/predict', methods=['POST'])
2 def predict():
3     try:
4         data = request.get_json()
5         image_file = data.get('image')
6         mask_file = data.get('mask')
7         resultURL = process_img(image_file, mask_file)
8         return jsonify({'result_image': resultURL})
9     except Exception as e:
10        return jsonify({'error': str(e)}), 500
```

复现流程

本地复现

首先，我们在自己的机器上部署实现了一遍 LaMa 项目：

1、克隆 LaMa 项目

```
1 git clone https://github.com/advimman/lama.git
```

2、安装项目依赖

```
1 virtualenv inenv --python=/usr/bin/python3
2 source inenv/bin/activate
3 pip install torch==1.8.0 torchvision==0.9.0
4
5 cd lama
6 pip install -r requirements.txt
```

3、设置参数，下载数据集进行训练

```
1 cd lama
2 export TORCH_HOME=$(pwd) && export PYTHONPATH=$(pwd)
3
4 mkdir -p ade20k/ade20k-resnet50dilated-ppm_deepsup/
5 wget -P ade20k/ade20k-resnet50dilated-ppm_deepsup/
  http://sceneparsing.csail.mit.edu/model/pytorch/ade20k-resnet50dilated-
  ppm_deepsup/encoder_epoch_20.pth
6
7 # Download data from http://places2.csail.mit.edu/download.html
8 # Places365-Standard: Train(105GB)/Test(19GB)/Val(2.1GB) from High-resolution
  images section
9 wget
  http://data.csail.mit.edu/places/places365/train_large_places365standard.tar
10 wget http://data.csail.mit.edu/places/places365/val_large.tar
11 wget http://data.csail.mit.edu/places/places365/test_large.tar
12
13 # Unpack train/test/val data and create .yaml config for it
14 bash fetch_data/places_standard_train_prepare.sh
15 bash fetch_data/places_standard_test_val_prepare.sh
16
17 # Sample images for test and viz at the end of epoch
18 bash fetch_data/places_standard_test_val_sample.sh
19 bash fetch_data/places_standard_test_val_gen_masks.sh
20
```

```

21 # Run training
22 python3 bin/train.py -cn lama-fourier location=places_standard
23
24 # To evaluate trained model and report metrics as in our paper
25 # we need to sample previously unseen 30k images and generate masks for them
26 bash fetch_data/places_standard_evaluation_prepare_data.sh
27
28 # Infer model on thick/thin/medium masks in 256 and 512 and run evaluation
29 # like this:
30 python3 bin/predict.py \
31 model.path=$(pwd)/experiments/<user>_<date:time>_lama-fourier/ \
32 indir=$(pwd)/places_standard_dataset/evaluation/random_thick_512/ \
33 outdir=$(pwd)/inference/random_thick_512 model.checkpoint=last.ckpt
34
35 python3 bin/evaluate_predicts.py \
36 $(pwd)/configs/eval2_gpu.yaml \
37 $(pwd)/places_standard_dataset/evaluation/random_thick_512/ \
38 $(pwd)/inference/random_thick_512 \
39 $(pwd)/inference/random_thick_512_metrics.csv

```

4、测试训练后模型的推理功能

```

1 python3 bin/predict.py model.path=$(pwd)/big-lama
  indir=$(pwd)/LaMa_test_images outdir=$(pwd)/output

```

RVbook 复现

然后，我们需要在实验指定的机器上复现模型。

1、在 RVbook 上安装 Python 相关的基本工具和依赖

```

1 sudo dnf install python3 python3-pip
2 sudo dnf install gcc gcc-c++ make
3 sudo dnf install libffi-devel

```

2、创建虚拟环境

```

1 python3 -m venv lama_env
2 source lama_env/bin/activate

```

3、安装 LaMa 依赖

可在项目根目录下的 requirements.txt 中查看模型运行的依赖，具体安装可分为以下三种类型：

- 直接使用 `pip` 安装

```
1 sudo pip install tadm
```

- 使用清华镜像源进行本地编译

如果依赖无法通过 `pip` 直接安装，可以使用清华镜像源。编译前确认将编译需要的依赖添加到环境变量 `PKG_CONFIG_PATH` 中，编译需要的其他依赖可通过 `dnf` 命令安装。

```
1 export PKG_CONFIG_PATH=/usr/lib64/pkgconfig:/usr/lib/pkgconfig:$PKG_CONFIG_PATH
2 sudo dnf install gfortran
3 pip install scikit_image scipy -i https://pypi.tuna.tsinghua.edu.cn/simple
```

- 克隆开源项目进行本地编译

如果需要的依赖无法通过前两种方法编译构建，则需要我们自行实现本地编译。

首先，需要配置好编译需要的依赖：

A. RISC-V GCC

```
1 sudo apt-get install autoconf automake autotools-dev curl python3 libmpc-dev
  libmpfr-dev libgmp-dev gawk build-essential bison flex texinfo gperf libtool
  patchutils bc zlib1g-dev libexpat-dev
2 git clone https://github.com/riscv-collab/riscv-gnu-toolchain.git
3 cd riscv-gnu-toolchain
4 ./configure --prefix=/opt/riscv --with-arch=rv64gc --with-abi=lp64d
5 sudo make
```

然后将 `/opt/riscv/bin` 加入环境变量，重启。

B. pk

```
1 cd ..
2 git clone https://github.com/riscv-software-src/riscv-pk.git
3 cd riscv-pk
4 mkdir build
5 cd build
6 ../configure --prefix=/opt/riscv --host=riscv64-unknown-elf
```

```
7 make
8 sudo make install
```

C. Spike

```
1 cd ..
2 sudo apt-get install device-tree-compiler
3 git clone https://github.com/riscv-software-src/riscv-isa-sim
4 cd riscv-isa-sim
5 mkdir build
6 cd build
7 ../configure --prefix=/opt/riscv
8 make
9 sudo make install
```

完成以上依赖的安装后，可以开始编译开源项目。以 tensorflow 为例，我们选择的开源项目是 riscv-tflm，具体构建流程如下：

首先，将项目克隆到本地：

```
1 git clone https://github.com/ioannesKX/riscv-tflm.git
2 cd riscv-tflm
```

然后进行 `make`：

```
1 make -f tensorflow/lite/micro/tools/make/Makefile TARGET=mcu_riscv
  TARGET_ARCH=riscv32_mcu person_detection_int8
```

// 这一步也可使用 CMSIS-NN 内核：

```
1 make -f tensorflow/lite/micro/tools/make/Makefile TARGET=mcu_riscv
  TARGET_ARCH=riscv32_mcu OPTIMIZED_KERNEL_DIR=cmsis_nn person_detection_int8
```

使用 spike 继续编译：

```
1 spike pk
  tensorflow/lite/micro/tools/make/gen/mcu_riscv_riscv32_mcu_default/bin/person_d
```

4、运行模型进行推理

安装完全部依赖后，可使用以下命令进行推理：

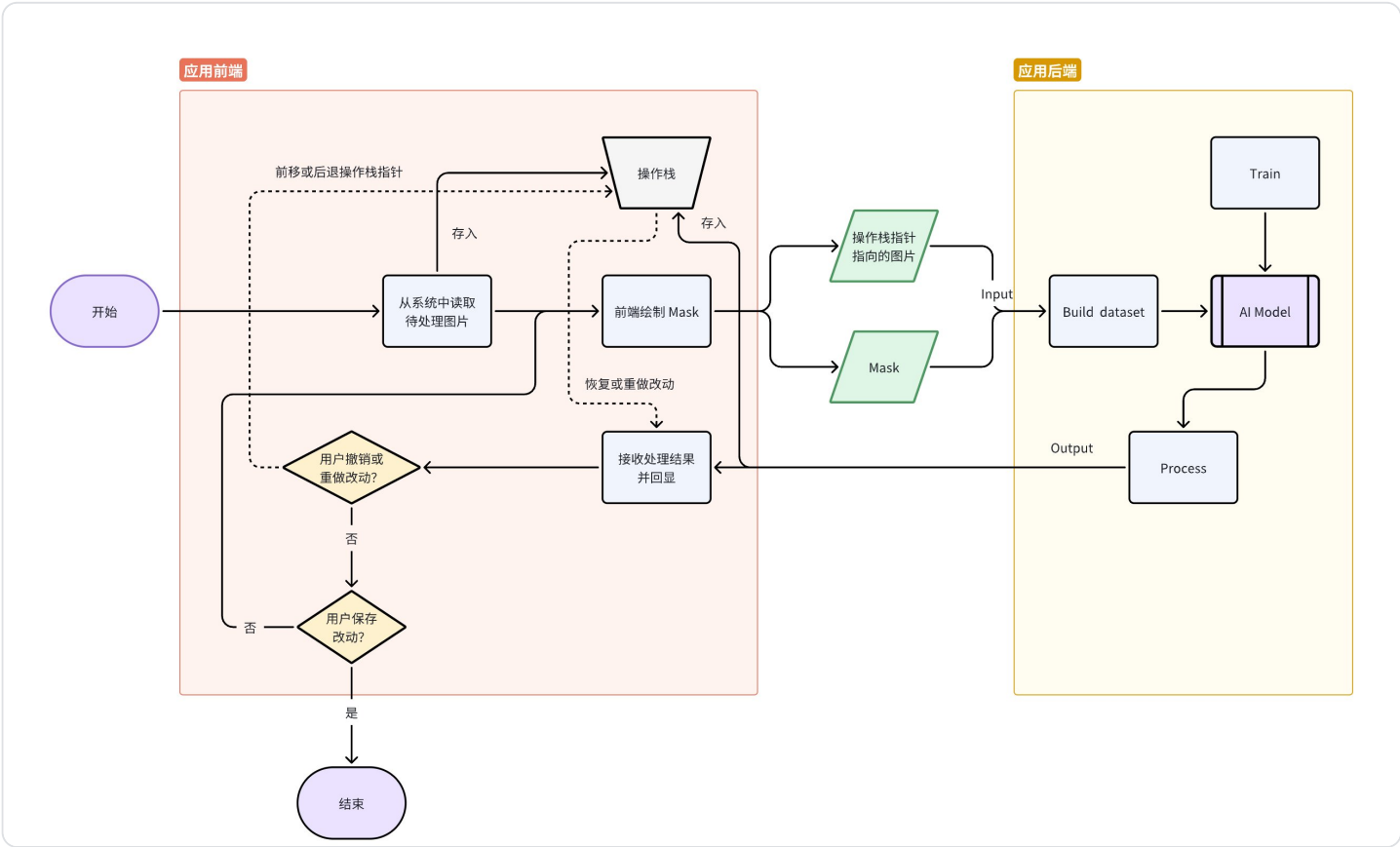
```
1 source lama_env/bin/activate
2 python3 predict.py
```

具体参数可通过编辑 configs/prediction/default.yaml 进行修改。

技术路线与具体实现

工作流程

本工具的工作流程大致如下图所示：



其中，选择的图像修复模型为 LaMa。

为了在实现图像修复效果的同时兼顾推理速度和有限计算资源的优化，我们选择了轻量级的模型 `lama_small_train_masks`。该版本的 LaMa 模型专为处理小尺寸图像和局部缺失区域进行了优化，采用了较小的训练数据集和局部掩码，因此具有更高的计算效率。

在训练过程中，我们使用了默认的配置，训练每 1000 次迭代会生成一次可视化结果，并将掩码与输入图像拼接，以帮助生成更加精准的修复图像。损失函数方面，L1 损失主要关注已知区域，权重设置为 10，而缺失区域不计算 L1 损失。感知损失被设置为 0，表示该损失函数未被使用。对于对抗性损失，我们采用了 R1 正则化，权重为 10，并加入了梯度惩罚（`gp_coef` 设置为 0.001）。掩码在训练过程中被视作“伪”目标用于训练判别器。此外，特征匹配损失的权重为 100，分割平滑学习损失的权重为 30，并且该损失的权重路径指向环境变量中的 `TORCH_HOME`。

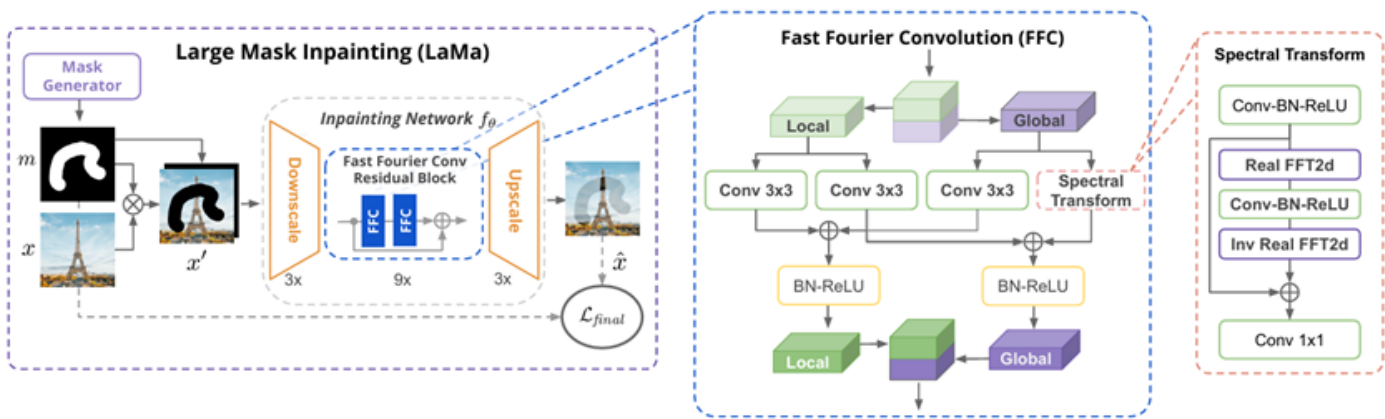
在优化器方面，生成器使用了 Adam 优化器，学习率为 0.001，而判别器的学习率设置为 0.0001。数据部分，训练批次大小设为 5，验证批次大小为 2，数据加载器采用了 3 个工作线程。训练集路径为 `train_256` 文件夹，图像尺寸被调整为 256x256。在掩码生成过程中，所有不规则区域和矩形框的生成概率均为 1，支持最大尺寸为 150 的矩形框，且最大生成次数为 5 次。

生成器采用了 `pix2pixHD` 架构，输入通道数为 4，输出通道数为 3，包含 9 个残差块，并使用 Sigmoid 激活函数。判别器同样基于 `pix2pixHD` 架构，输入通道数为 3，包含 4 层卷积层。训练过程中，使用了分布式数据并行（DDP）以支持多 GPU 训练，最大训练轮数设为 40，梯度裁剪值为 1。每 2600 步进行一次验证，每 250 步保存一次日志。

模型原理

See in: <https://advimman.github.io/lama-project/>

LaMa 模型的结构如下：



模型的目标是对被未知像素的二进制掩码 m 所遮挡的彩色图像 x 进行修复，被遮挡的图像表示为 $x \boxtimes m$ 。修复网络以全卷积的方式处理四通道的输入张量 $x' = \text{stack}(x \boxtimes m, m)$ ，并生成修复后的三通道彩色图像 $f(\theta)(x')$ 。训练是在从真实图像和合成生成的掩码中获取的（图像，掩码）对上进行的。

传统的卷积在网络的前几层并不能获得很大的感受野，这将浪费很多计算空间来对感受野进行建模，而且较小的感受野缺失了全局的信息，不利于进行图像修复。LaMa 通过快速傅里叶卷积（FFC）来解决大掩码修复中的全局上下文捕捉问题，快速傅里叶卷积（FFC）基于通道级的快速傅里叶变换（FFT），将输入张量分为两条并行分支：一条是使用传统卷积的局部分支，另一条是使用 FFT 处理全局上下文的全局分支。具体步骤包括：

1. 对输入张量应用二维实值傅里叶变换（Real FFT2d），并将实部和虚部分离。

$$Real\ FFT2d : \mathbb{R}^{H \times W \times C} \rightarrow \mathbb{C}^{H \times \frac{W}{2} \times C},$$

$$ComplexToReal : \mathbb{C}^{H \times \frac{W}{2} \times C} \rightarrow \mathbb{R}^{H \times \frac{W}{2} \times 2C};$$

2. 在频域中对实部和虚部应用卷积块进行处理。

$$ReLU \circ BN \circ Conv1 \times 1 : \mathbb{R}^{H \times \frac{W}{2} \times 2C} \rightarrow \mathbb{R}^{H \times \frac{W}{2} \times 2C};$$

3. 使用逆傅里叶变换恢复空间结构。

$$RealToComplex : \mathbb{R}^{H \times \frac{W}{2} \times 2C} \rightarrow \mathbb{C}^{H \times \frac{W}{2} \times C},$$

$$Inverse\ Real\ FFT2d : \mathbb{C}^{H \times \frac{W}{2} \times C} \rightarrow \mathbb{R}^{H \times W \times C}.$$

最后将局部分支和全局分支的输出融合。这一流程的伪代码如下所示：

```
def FU(x):
    # x: input features with shape [N,C,H,W]

    # y_r / y_i is the real / imaginary part of the results of FFT, respectively
    y_r, y_i = FFT(x) # y_r/y_i: [N,C,H,⌊W/2⌋+1]
    y = Concatenate([y_r, y_i], dim=1) # [N,C*2,H,⌊W/2⌋+1]
    y = ReLU(BN(Conv(y))) # [N,C*2,H,⌊W/2⌋+1]
    y_r, y_i = Split(y, dim=1) # y_r/y_i: [N,C,H,⌊W/2⌋+1]
    z = iFFT(y_r, y_i) # [N,C,H,W]

    return z
```

模型的损失函数主要包含两部分：

1. 高感受野感知损失

简单的监督损失要求生成器精确地重建真实图像。然而，图像的可见部分通常无法提供足够的信息来精确重建被遮挡部分。因此，使用简单的监督会导致由于对多种可能的内容模式进行平均而产生模糊结果。相比之下，感知损失通过一个预训练的基础网络 $\Phi(\cdot)$ 提取特征，并计算预测图像与目标图像之间的距离。它不需要精确的重建，允许在重建图像中存在变化。大掩码修复的重点在于对全局结构的理解。因此，本文引入了高感受野感知损失（HRF PL），通过使用高感受野的基础模型 $\Phi_{HRF}(\cdot)$ ：

$$\mathcal{L}_{HRFPL}(x, \hat{x}) = \mathcal{M}([\phi_{HRF}(x) - \phi_{HRF}(\hat{x})]^2),$$

其中， $[\cdot - \cdot]$ 是元素级操作， \mathcal{M} 是两阶段均值操作（层内均值的层间均值）。 $\Phi_{HRF}(\cdot)$ 可以通过傅里叶卷积或膨胀卷积实现。

HRF 感知损失不要求生成的修复图像与真实图像完全一致，而是允许它们在视觉上有差异，但仍能捕捉到全局结构信息。通过这种方式，模型能够生成视觉上自然的图像，避免使用直接监督导致的模糊问题，尤其是在大掩码修复任务中表现尤为重要。

2. 对抗性损失

使用对抗性损失来确保修复模型 $f_{\theta}(x')$ 生成自然的局部细节。本文定义了一个判别器 $D_{\xi}(\boxtimes)$ ，它在局部块级别上工作，区分“真实”与“伪造”的块。只有与遮挡区域相交的块才会被赋予“伪造”标签。由于有监督的 HRF 感知损失，生成器很快就会学会复制输入图像的已知部分，因此将生成图像的已知部分标记为“真实”。最后，使用非饱和对抗损失：

$$\begin{aligned}\mathcal{L}_D &= -\mathbb{E}_x \left[\log D_{\xi}(x) \right] - \mathbb{E}_{x,m} \left[\log D_{\xi}(\hat{x}) \odot m \right] \\ &\quad - \mathbb{E}_{x,m} \left[\log (1 - D_{\xi}(\hat{x})) \odot (1 - m) \right] \\ \mathcal{L}_G &= -\mathbb{E}_{x,m} \left[\log D_{\xi}(\hat{x}) \right] \\ L_{Adv} &= \text{sg}_{\theta}(\mathcal{L}_D) + \text{sg}_{\xi}(\mathcal{L}_G) \rightarrow \min_{\theta, \xi}\end{aligned}$$

其中， x 是数据集中的样本， m 是合成生成的掩码， $\hat{x}=f(\theta)(x')$ 是 $x'=stack(x \odot m, m)$ 的修复结果， $\text{sg}(\text{var})$ 表示对变量 var 停止梯度计算， $L(\text{Adv})$ 是用于优化的联合损失。

对抗性损失的目的是提高生成图像的局部细节质量，使得修复后的图像不仅在全局结构上合理，还在局部纹理上自然逼真。通过使用局部块级别的对抗性训练，生成器能够生成细节丰富、真实感强的图像修复结果。

3. 总损失函数

在总损失函数中，本文还使用了 R1 梯度惩罚和基于判别器的感知损失，也称为特征匹配损失，最终的损失函数如下：

$$\mathcal{L}_{final} = \kappa L_{Adv} + \alpha \mathcal{L}_{HRFPL} + \beta \mathcal{L}_{DiscPL} + \gamma R_1$$

问题与解决方案

- Q00

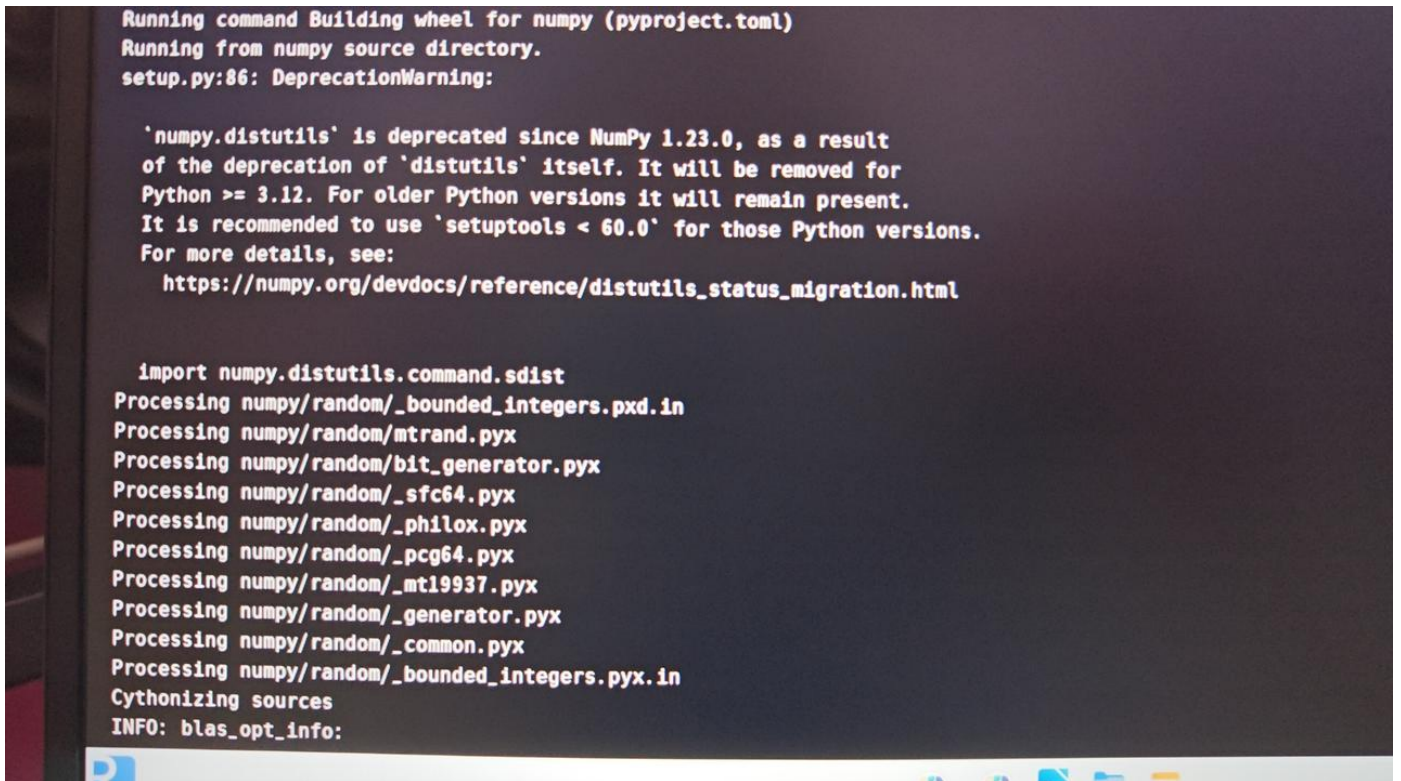
- 描述：即使已经采用了最轻量级的模型，但在仅使用 CPU 硬件的基础上，predict 的速度依旧慢得有些不能接受。下图的后端日志中展示了不同大小图片的 predict 用时，对于大小为 1920x1080 pixel 的图像，甚至出现了 cant allocate memory 的错误。

```
[2024-12-29 13:31:01,633][werkzeug][INFO] - 127.0.0.1 - - [29/Dec/2024 13:31:01] "OPTIONS /predict HTTP/1.1" 200 - 474x316 pixel
[2024-12-29 13:32:48,556][werkzeug][INFO] - 127.0.0.1 - - [29/Dec/2024 13:32:48] "POST /predict HTTP/1.1" 200 - 
[2024-12-29 13:34:01,070][werkzeug][INFO] - 127.0.0.1 - - [29/Dec/2024 13:34:01] "OPTIONS /predict HTTP/1.1" 200 - 605x908 pixel
[2024-12-29 13:40:24,691][werkzeug][INFO] - 127.0.0.1 - - [29/Dec/2024 13:40:24] "POST /predict HTTP/1.1" 200 - 
[2024-12-29 13:41:35,297][werkzeug][INFO] - 127.0.0.1 - - [29/Dec/2024 13:41:35] "OPTIONS /predict HTTP/1.1" 200 - 925x613 pixel
[2024-12-29 13:48:08,371][werkzeug][INFO] - 127.0.0.1 - - [29/Dec/2024 13:48:08] "POST /predict HTTP/1.1" 200 - 
[2024-12-29 13:49:19,015][werkzeug][INFO] - 127.0.0.1 - - [29/Dec/2024 13:49:19] "OPTIONS /predict HTTP/1.1" 200 - 
Error during image processing: [enforce fail at alloc_cpu.cpp:117] err == 0. DefaultCPUAllocator: can't allocate memory: you tried to allocate 26011238400 bytes. Error code 12 (Canno
t allocate memory)
[2024-12-29 14:11:40,029][__main__][CRITICAL] - Error during image processing: [enforce fail at alloc_cpu.cpp:117] err == 0. DefaultCPUAllocator: can't allocate memory: you tried to
allocate 26011238400 bytes. Error code 12 (Cannot allocate memory)
[2024-12-29 14:11:40,182][werkzeug][INFO] - 127.0.0.1 - - [29/Dec/2024 14:11:40] "POST /predict HTTP/1.1" 200 - 1920x1080 pixel
```

- 解释：进行实验的笔记本搭载的硬件处理器是玄铁的 TH1520，不能将开源的模型直接应用于 GPU 和 NPU。而对于图像处理类型的应用来说，显然是远远不够的——该机器固有的性能问题影响了程序的执行效果。且不谈此工具的开发，机器对本机自带应用的支持都有非常明显的延迟；使用浏览器时，一旦访问页面的网页元素较多，浏览器就会崩溃。
- 解决方案：暂无。

• Q01

- 描述：在笔记本上新建虚拟环境，安装 numpy 报错，无法调用 distutils.msvccompiler 这一模块，导致编译失败。确定安装了 gcc、make、python3-devel、openblas-devel 和 lapack-devel。可以正常安装 pyyaml 和 tqdm 这两个不需要大量 C 编译的包。尝试升级 pip、wheel、setuptools 到最新版，安装版本较低的 numpy，通过 `export CC=gcc` 强制 Python 使用 Linux 编译器，均失败。



```
Running command Building wheel for numpy (pyproject.toml)
Running from numpy source directory.
setup.py:86: DeprecationWarning:

'numpy.distutils' is deprecated since NumPy 1.23.0, as a result
of the deprecation of 'distutils' itself. It will be removed for
Python >= 3.12. For older Python versions it will remain present.
It is recommended to use 'setuptools < 60.0' for those Python versions.
For more details, see:
  https://numpy.org/devdocs/reference/distutils_status_migration.html

import numpy.distutils.command.sdist
Processing numpy/random/_bounded_integers.pxd.in
Processing numpy/random/mtrand.pyx
Processing numpy/random/bit_generator.pyx
Processing numpy/random/_sfc64.pyx
Processing numpy/random/_philox.pyx
Processing numpy/random/_pcg64.pyx
Processing numpy/random/_mt19937.pyx
Processing numpy/random/_generator.pyx
Processing numpy/random/_common.pyx
Processing numpy/random/_bounded_integers.pyx.in
Cythonizing sources
INFO: blas_opt_info:
```

- 解释：系统在调用进行 C 编译的依赖时无法正确识别位置，导致调用失败。
- 解决方案：安装已编译好、riscv64 版本的 numpy 包。

• Q02

- 描述：对于后续的依赖，无法和 numpy 一样找到 riscv64 版本的 whl 文件，需要尝试原生编译，仍旧存在依赖定位失败的问题。
- 解释：系统的 PKG_CONFIG_PATH 中不包含 usr/bin/lib64/pkgconfig，由于系统使用 dnf 安装的依赖库部分包含在两个不同的路径中，导致一部分 pkgconfig 文件无法被识别到。同时系统中仍然缺失部分编译所需要的 compiler。
- 解决方案：使用 `export`
`PKG_CONFIG_PATH=/usr/lib64/pkgconfig:/usr/lib/pkgconfig:$PKG_CONFIG_PATH && pip install scikit_image scipy -i`
<https://pypi.tuna.tsinghua.edu.cn/simple> 这条命令补充路径，进行原生编译，并查看完整日志补充安装了需要的 compiler。


```

Compiler for C supports arguments -Wno-unused-function: YES
Compiler for C supports arguments -Wno-conversion: YES
Compiler for C supports arguments -Wno-misleading-indentation: YES
Library m found: YES

../meson.build:84:0: ERROR: Unknown compiler(s): [['gfortran'], ['flang-new'], ['flang'], ['nvfortran'], ['pgfortran'], ['ifort'], ['ifx'], ['g95']]
The following exception(s) were encountered:
Running 'gfortran --help' gave "[Errno 2] No such file or directory: 'gfortran'"
Running 'gfortran --version' gave "[Errno 2] No such file or directory: 'gfortran'"
Running 'gfortran -V' gave "[Errno 2] No such file or directory: 'gfortran'"
Running 'flang-new --help' gave "[Errno 2] No such file or directory: 'flang-new'"
Running 'flang-new --version' gave "[Errno 2] No such file or directory: 'flang-new'"
Running 'flang-new -V' gave "[Errno 2] No such file or directory: 'flang-new'"
Running 'flang --help' gave "[Errno 2] No such file or directory: 'flang'"
Running 'flang --version' gave "[Errno 2] No such file or directory: 'flang'"
Running 'flang -V' gave "[Errno 2] No such file or directory: 'flang'"
Running 'nvfortran --help' gave "[Errno 2] No such file or directory: 'nvfortran'"
Running 'nvfortran --version' gave "[Errno 2] No such file or directory: 'nvfortran'"
Running 'nvfortran -V' gave "[Errno 2] No such file or directory: 'nvfortran'"
Running 'pgfortran --help' gave "[Errno 2] No such file or directory: 'pgfortran'"
Running 'pgfortran --version' gave "[Errno 2] No such file or directory: 'pgfortran'"
Running 'pgfortran -V' gave "[Errno 2] No such file or directory: 'pgfortran'"
Running 'ifort --help' gave "[Errno 2] No such file or directory: 'ifort'"
Running 'ifort --version' gave "[Errno 2] No such file or directory: 'ifort'"
Running 'ifort -V' gave "[Errno 2] No such file or directory: 'ifort'"
Running 'ifx --help' gave "[Errno 2] No such file or directory: 'ifx'"
Running 'ifx --version' gave "[Errno 2] No such file or directory: 'ifx'"
Running 'ifx -V' gave "[Errno 2] No such file or directory: 'ifx'"
Running 'g95 --help' gave "[Errno 2] No such file or directory: 'g95'"
Running 'g95 --version' gave "[Errno 2] No such file or directory: 'g95'"
Running 'g95 -V' gave "[Errno 2] No such file or directory: 'g95'"

A full log can be found at /tmp/pip-install-m4zp7grw/scipy_f15af0e45607435b9b2a6388ecd130e/.mesonpy-qlmkhvup/meson-logs/meson-log.txt
error: subprocess-exited-with-error

× Preparing metadata (pyproject.toml) did not run successfully.

```

• Q03

- 描述：对于 dnf 命令安装的 openblas 依赖，系统无法识别。
- 解释：系统在使用 dnf 命令安装 openblas 时没有在对应的 pkgconfig 中创建 pc 文件。
- 解决方案：在对应路径下手动创建 pc 文件，输入查找到的 openblas 头文件和库文件位置。

• Q04

- 描述：编译 TensorFlow 参考了两个开源项目。其中一个使用的 bazel 版本较低，系统默认安装的版本不符合。主要参考了 <https://github.com/ioannesKX/riscv-tflm> 这个项目。项目要求安装 gcc、pk 和 spike，pk 与 spike 由对应的 git 项目在本地进行编译。gcc 使用系统默认安装的版本，识别后报错。

```
riscv64-unknown-elf-g++ -std=c++11 -fno-rtti -fno-exceptions -fno-threadsafe-statics -fno-unwind-tables -ffunction-sections -fdata-sections -fmessage-length=0 -DTF_LITE_STATIC_MEMORY -DTF_LITE_DISABLE_X86_NEON -Wsign-compare -Wdouble-promotion -Wshadow -Wunused-variable -Wmissing-field-initializers -Wunused-function -Wswitch -Wvla -Wall -Wextra -Wstrict-aliasing -Wno-unused-parameter -DMCU_RISCV -march=rv64gc -mabi=lp64d -mcmmodel=medany -mexplicit-relocs -fno-builtin-printf -fno-exceptions -DTF_LITE_MCU_DEBUG_LOG -DTF_LITE_USE_GLOBAL_CMATH_FUNCTIONS -funsigned-char -fno-delete-null-pointer-checks -fomit-frame-pointer -fpermissive -fno-rtti -fno-threadsafe-statics -fno-use-cxa-atexit -DTF_LITE_USE_GLOBAL_MIN -DTF_LITE_USE_GLOBAL_MAX --std=gnu++11 -Os -I. -Itensorflow/lite/micro/tools/make/downloads/gemmlowp -Itensorflow/lite/micro/tools/make/downloads/flatbuffers/include -Itensorflow/lite/micro/tools/make/downloads/ruy -Itensorflow/lite/micro/tools/make/gen/mcu_riscv_riscv64_default/genfiles/ -Itensorflow/lite/micro/tools/make/downloads/kissfft -c tensorflow/lite/micro/tools/make/gen/mcu_riscv_riscv64_default/genfiles/tensorflow/lite/micro/models/person_detect_model_data.cc -o tensorflow/lite/micro/tools/make/gen/mcu_riscv_riscv64_default/obj/core/tensorflow/lite/micro/tools/make/gen/mcu_riscv_riscv64_default/genfiles/tensorflow/lite/micro/models/person_detect_model_data.o
make: riscv64-unknown-elf-g++: No such file or directory
make: *** [tensorflow/lite/micro/tools/make/Makefile:595: tensorflow/lite/micro/tools/make/gen/mcu_riscv_riscv64_default/obj/core/tensorflow/lite/micro/tools/make/gen/mcu_riscv_riscv64_default/genfiles/tensorflow/lite/micro/models/person_detect_model_data.o] 错误 127
[experiment7@openEuler riscv-tflm]$
```



- 解释：该项目使用的 gcc 要求为交叉编译工具链，单一的 gcc 无法满足要求。
- 解决方案：在笔记本上原生编译 riscv64-unknown-elf 工具链。类似的，在笔记本上编译了 textinfo、gmp、mpfr、DTC 等依赖。

Q05

- 描述：在编译 lama 模型时，发现缺少 torchvision 包，并且无法使用 `pip3 install torchvision` 指令安装。
- 解释：因为 pip 无法找到与系统和架构兼容的 torchvision 安装包。目前，TorchVision 的官方预编译版本可能不支持 RISC-V 架构。
- 解决方案：使用源码编译 torchvision。