



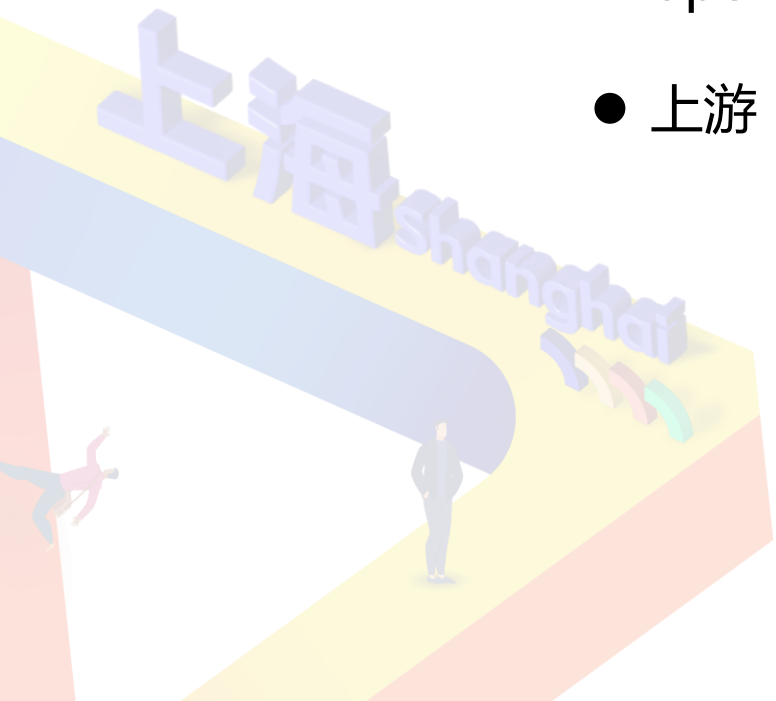
OS for AI: openEuler 在 AI 开源社区的贡献与实践

张思博 王帅



目录

- OS for AI: 需要什么
- openEuler+Ascend 的 AI 生态构建
- 上游 LLM 社区实践



支持多样性设备 覆盖全场景应用

Information Technology

+

Communication Technology

+

Operational Technology

主流应用：云原生，大数据，AI，CDN，MEC，工业控制 ...

主流应用场景100%支持

覆盖全场景应用



支持多样性设备

主流计算架构100%覆盖

ARM, x86, RISC-V, SW-64, LoongArch; NPU, GPU, DPU, 100+ 整机, 300+ 板卡



服务器



云计算



边缘计算



嵌入式



高效易用生态开放的计算架构



昇腾计算语言 (Ascend Computing Language, AscendCL)

接口是昇腾计算开放编程框架，对开发者屏蔽底层多种处理器差异，提供算子开发接口、标准图开发接口、应用开发接口，支持用户快速构建基于Ascend平台的AI应用和业务。

昇腾计算服务层主要提供昇腾算子库AOL，通过神经网络

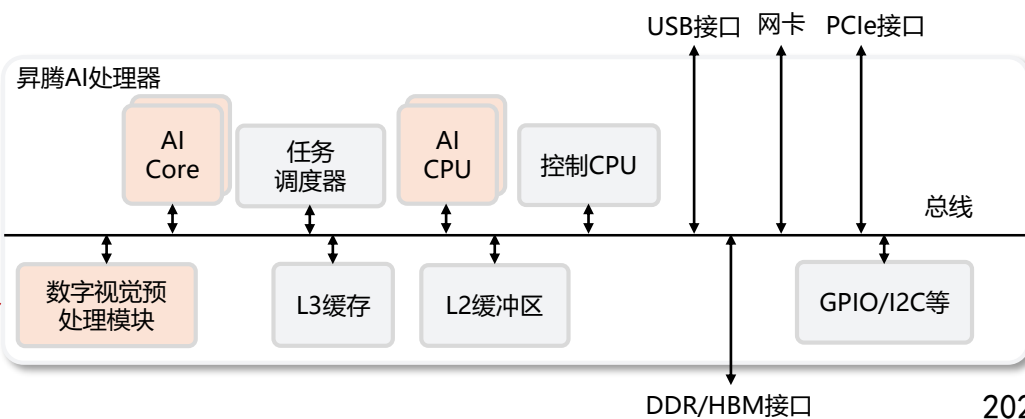
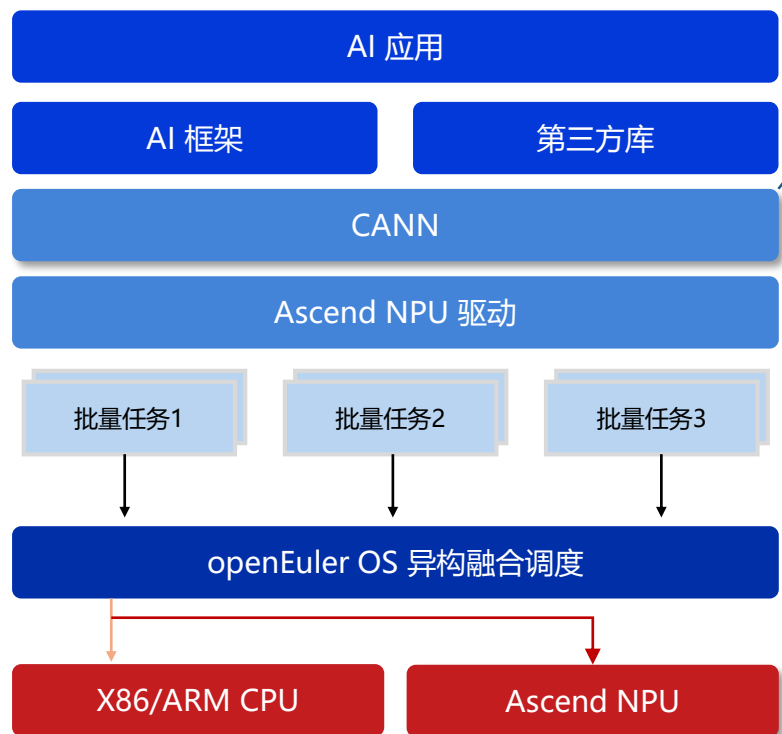
(Neural Network, NN) 库、线性代数计算库 (Basic Linear Algebra Subprograms, BLAS) 等高性能算子加速计算；昇腾调优引擎AOE，通过算子调优OPAT、子图调优SGAT、梯度调优GDAT、模型压缩AMCT提升模型端到端运行速度。同时提供AI框架适配器Framework Adaptor用于兼容Tensorflow、Pytorch等主流AI框架。

昇腾计算编译层通过图编译器 (Graph Compiler) 将用户输入中间表达 (Intermediate Representation, IR) 的计算图编译成昇腾硬件可执行模型；同时借助张量加速引擎TBE (Tensor Boost Engine) 的自动调度机制，高效编译算子。

昇腾计算执行层负责模型和算子的执行，提供运行时库

(Runtime)、图执行器 (Graph Executor)、数字视觉预处理 (Digital Vision Pre-Processing, DVPP)、人工智能预处理 (Artificial Intelligence Pre-Processing, AIPP)、华为集合通信库 (Huawei Collective Communication Library, HCCL) 等功能单元。

openEuler + Ascend 算力底座使能 AI 应用开发



目录

- OS for AI: 需要什么
- **openEuler+Ascend 的 AI 生态构建**
- 上游 LLM 社区实践



第三方模型

已支持三方社区数百个模型



机器视觉领域主流方向

支持OpenMMLab社区算子库、套件等

MMcv
CV算子库

MMClassification
图像分类套件

MMdetection
图像检测套件

MMSegmentation
语义分割套件



Hugging Face

自然语言处理类模型套件

各类Transformer模型

HuggingFace
Transformers

.....

第三方AI框架

支持并兼容各版本高阶特性



全面兼容

1.8、1.11、1.13、2.0主流版本
图模式、分布训练、量化等高阶特性



全面支持

1.5、2.X主流版本，300+模型

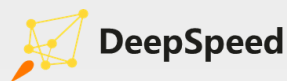


已适配30+模型

正与百度深度合作，共同推进模型适配

第三方加速库

跟随版本支持最新特性



分布式并行训练加速库

支持混合精度、MoE、通信优化等特性



MegatronLM

Transformer加速库

支持多维混合并行、跨节点预训练等特性

第三方推理服务

支持“0代码”快速对接



业界推理模型标准

100+基础模型，定制模型零成本迁移



Triton

业界主流推理部署平台

支持并行推理、动态调度等关键特性

自底向上打通大模型开发与部署的完整链条

应用/
模型部署

Stable-diffusion-webui

FastChat

OpenClip

工具/
加速库

Transformers

Diffusers

...

Accelerate

TRL

DeepSpeed

ONNXRuntime

PEFT

OpenCV

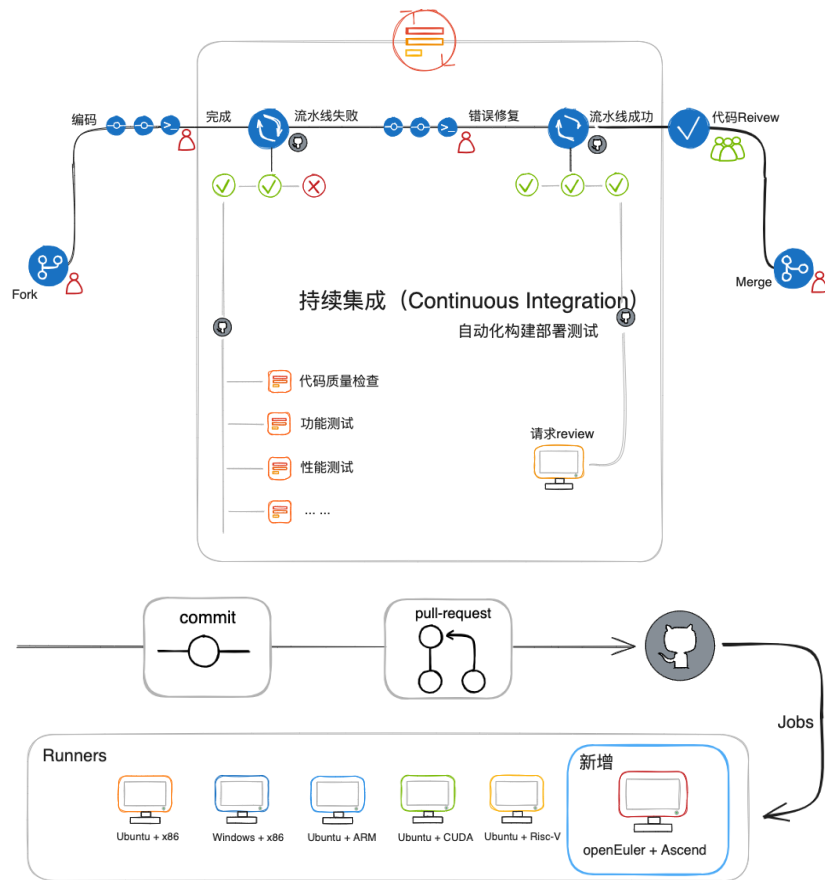
AI 框架

Pytorch

Tensorflow

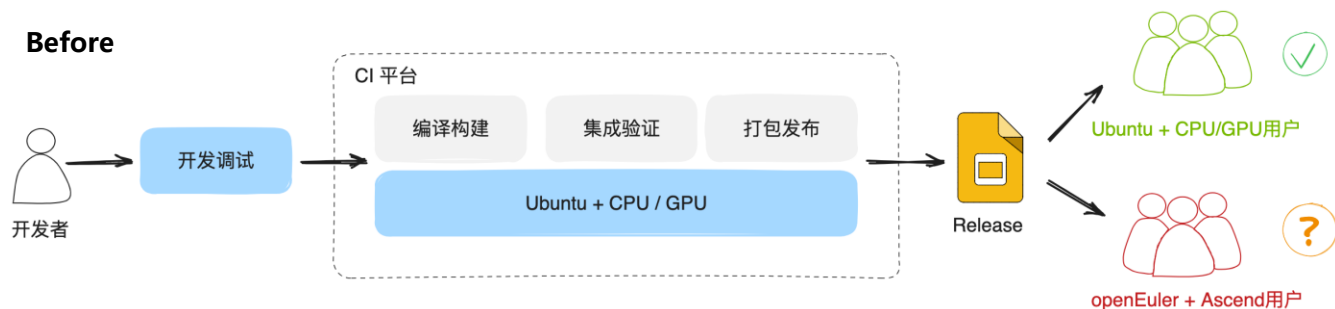
PaddlePaddle

openEuler+昇腾CI前置社区，持续使能开发流程



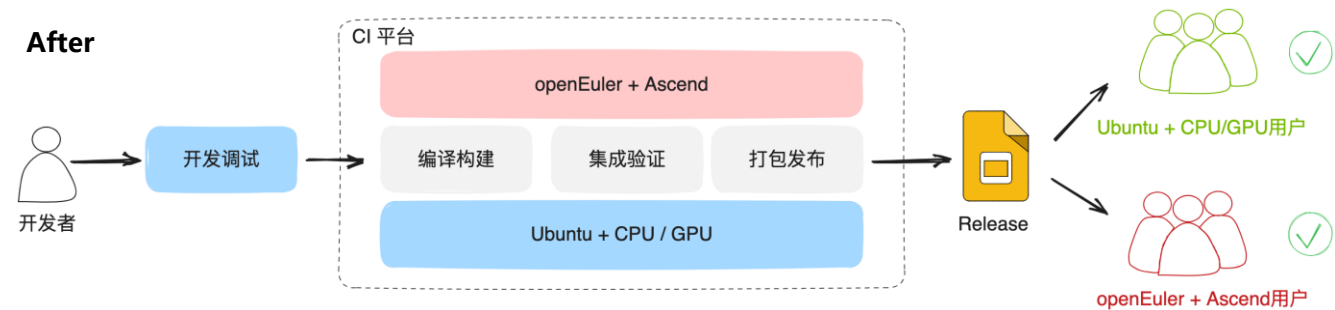
- 向 OpenCV 社区贡献 openEuler + 昇腾服务器作为CI机器
- 构建用于 OpenCV 的 openEuler 基础镜像，官方仓库发布
- 在 OpenCV 扩展库上配置 PR 级别 Pipeline，用于openEuler+昇腾的功能测试

Before



openEuler + 昇腾 CI 合入之前，社区仅基于 Ubuntu 以及 CPU/GPU 后端平台。昇腾相关代码的编译，验证需要社区 reviewer 手动本地执行，并且无法发布昇腾后端相关特性，用户需要自行构建。

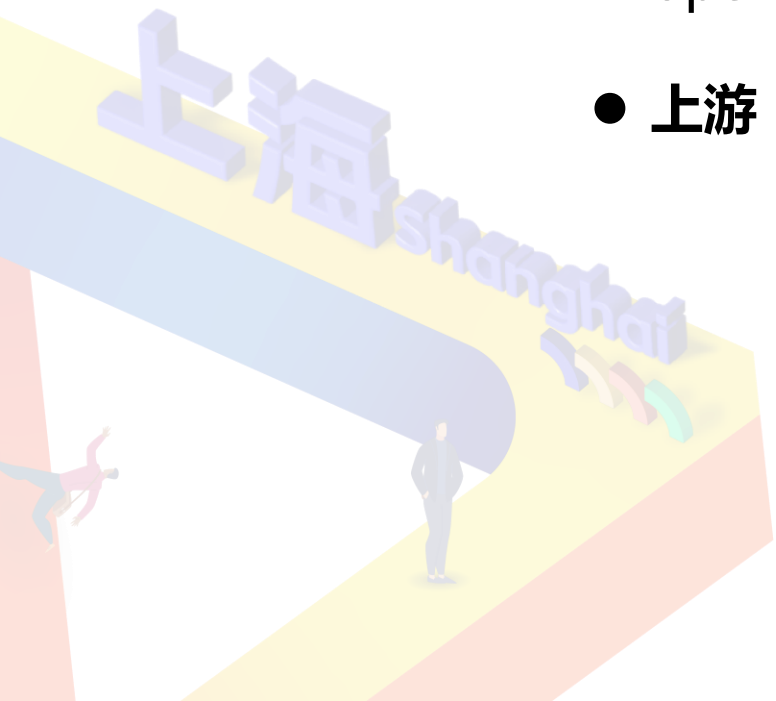
After



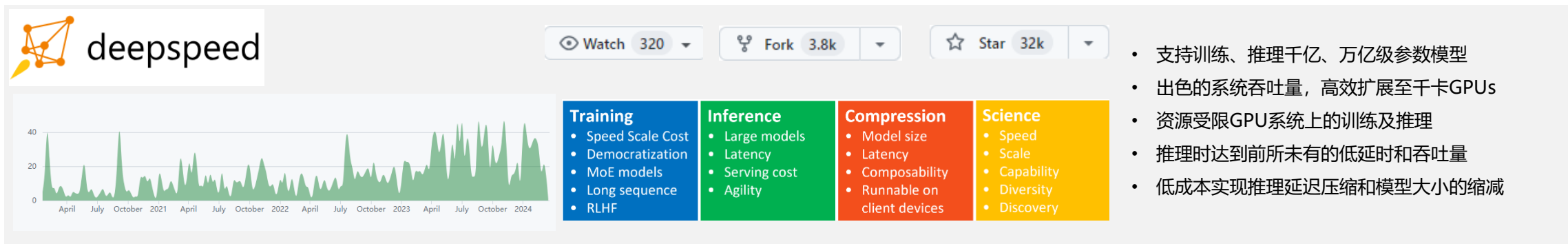
openEuler + 昇腾 CI 合入以后，社区提交的代码会在 openEuler + 昇腾环境下自动构建和测试，确保在 openEuler + 昇腾环境上的稳定性和性能，并提供多后端支持的官方预编译包的能力。

目录

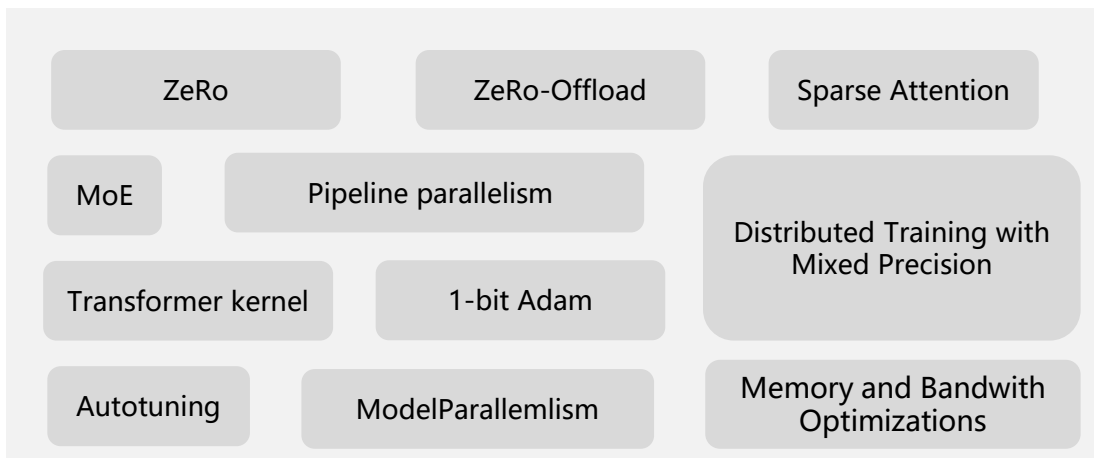
- OS for AI: 需要什么
- openEuler+Ascend 的 AI 生态构建
- **上游 LLM 社区实践**









DeepSpeed 社区及生态



微软开源深度学习加速工具包



核心特性

	Documentation
 Transformers	Transformers with DeepSpeed
 Accelerate	Accelerate with DeepSpeed
 Lightning	Lightning with DeepSpeed
 mosaic ^{ML}	MosaicML with DeepSpeed
 Determined AI	Determined with DeepSpeed
 MME Engine	MME Engine with DeepSpeed

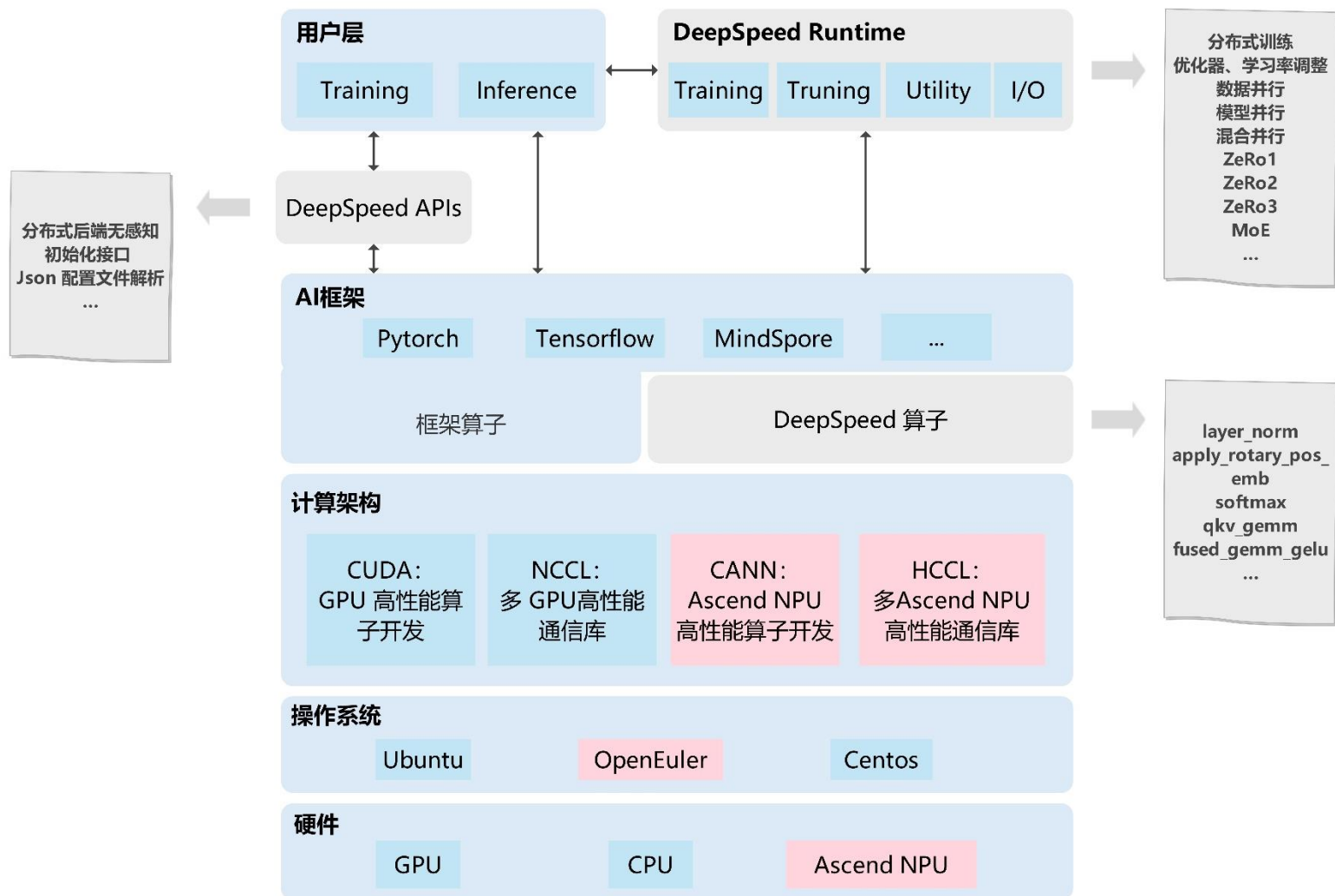
- [Megatron-Turing NLG \(530B\)](#)
- [Jurassic-1 \(178B\)](#)
- [BLOOM \(176B\)](#)
- [GLM \(130B\)](#)
- [xTrimoPGLM \(100B\)](#)
- [YaLM \(100B\)](#)
- [GPT-NeoX \(20B\)](#)
- [AlexaTM \(20B\)](#)
- [Turing NLG \(17B\)](#)
- [METRO-LM \(5.4B\)](#)

生态应用

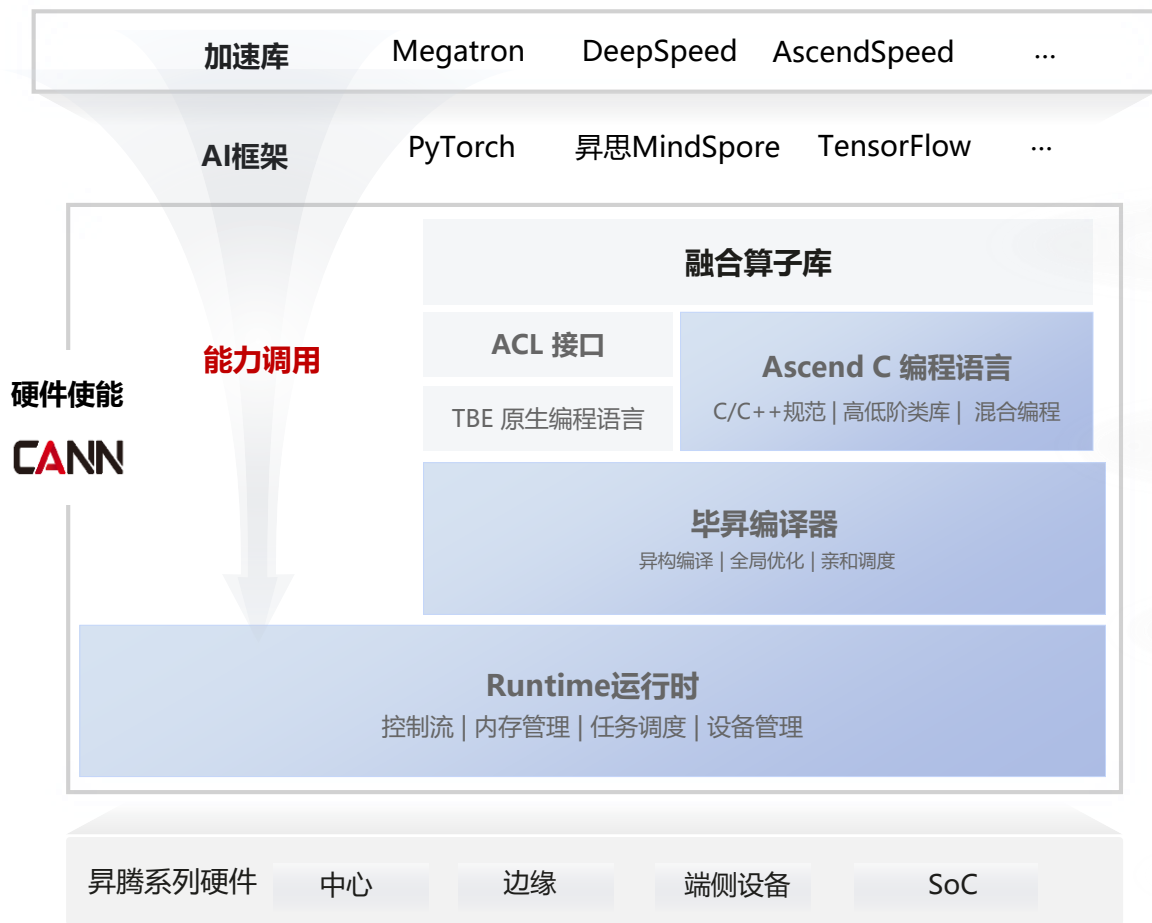
2024全球开发者先锋大会

2024 GLOBAL DEVELOPER CONFERENCE

DeepSpeed 与昇腾对接逻辑框架



基于 AscendCL 实现算子开发与对接



使能开发者

算子开发工程师、模型开发工程师、应用开发工程师 ...



发展人才

开放对接北向生态

原生支持MindSpore、同步适配PyTorch、兼容主流AI框架、三方社区等

开放丰富的融合算子

FFN、SparseAttention、FlashAttention、KVCache ...

开放Runtime运行时

提供接口使能框架、加速库可调用底层NPU资源，支持业务灵活定义，
使能伙伴开发者自主构筑极致性能的算子及加速库

DeepSpeed 与昇腾对接代码逻辑

Step1

NPU_Accelerator: 集成 NPU 接口

```
class NPU_Accelerator(DeepSpeedAccelerator):

    def __init__(self):
        self._name = 'npu'
        # 通信后端
        self._communication_backend_name = 'hccl'

    # npu 设备接口
    def device(self, device_index=None):
        return torch.npu.device(device_index)
    ...

    # npu stream 接口
    def Stream(self):
        return torch.npu.Stream
    ...

    # npu 内存管理接口
    def empty_cache(self):
        return torch.npu.empty_cache()
    ...

    # 其他接口
```

继承 DeepSpeedAccelerator, 集成设备相关操作, 实现设备无感知

Step2

NPUOpBuilder: 算子编译依赖

```
class NPUOpBuilder(OpBuilder):

    # cann 安装及版本检查
    def installed_cann_version(self, name=""):
        return cann_version

    # 编译算子所需头文件, 包括Ascend及torch_npu头文件路径
    def include_paths(self):
        return paths

    # 编译算子参数, 包括Ascend及torch_npu依赖库
    def cxx_args(self):
        return args

    # 其他额外依赖
    def extra_ldflags(self):
        return flags
```

继承 Opbuilder, 实现 Ascend NPU 算子编译命令及依赖

Step3

算子接口: CPUAdamBuilder

```
from .builder import NPUOpBuilder

# 继承 NPUOpBuilder 算子
class CPUAdamBuilder(NPUOpBuilder):
    BUILD_VAR = "DS_BUILD_CPU_ADAM"
    NAME = "cpu_adam"

    def __init__(self):
        super().__init__(name=self.NAME)

    def absolute_name(self):
        # 算子名称
        return f'deepspeed.ops.adam.{self.NAME}_op'

    def sources(self):
        # 算子实现文件
        return ['csrc/adam/cpu_adam.cpp', 'csrc/adam/cpu_adam_impl.cpp']

    def include_paths(self):
        # 算子相关头文件路径
        args = super().include_paths()
        args += ['csrc/includes']
        return args
```

继承 NPUOpBuilder, 具体算子接口

Step4

算子实现

```
def layer_norm(inputs, gamma, beta, epsilon):
    return torch.nn.functional.layer_norm(inputs, ...)

data, m_flat, v_flat = torch_npu.npu_apply_adam_w(
    bias_correction1,
    bias_correction2,
    lr,
    weight_decay,
    beta1,
    beta2,
    epsilon,
    grad_flat,
    None, # max_grad_norm
    False, # amsgrad
    False, # maximize
    out=(param_flat.data, m_flat, v_flat))
```

不同算子的具体实现, 依赖acl接口、原生支持算子、适配算子

aclrtFreeHost

torch.matmul

▼

2024 GLOBAL DEVELOPER CONFERENCE



stable-diffusion-webui (Stable-diffusion交互式界面应用)

```
158     if echo "$gpu_info" | grep -q "AMD" && [[ -z "${TORCH_COMMAND}" ]]
159     then
160         export TORCH_COMMAND="pip install torch==2.0.1+rocm5.4.2 torchvision==0.15.2+rocm5.4.2 --index-url https://download.pytorch.org/whl/rocm5.4.2"
161 +     elif npu-smi info 2>/dev/null
162     then
163 +         export TORCH_COMMAND="pip install torch==2.1.0 torchvision torchaudio --index-url https://download.pytorch.org/whl/cpu; pip install
164         torch_npu==2.1.0"
165     fi
166 fi
```

一键式安装配置

torch_npu 加载

npu 相关操作

```
1 + import importlib
2 + import torch
3 +
4 + from modules import shared
5 +
6 +
7 + def check_for_npu():
8 +     if importlib.util.find_spec("torch_npu") is None:
9 +         return False
10 +     import torch_npu
11 +
12 +     try:
13 +         # Will raise a RuntimeError if no NPU is found
14 +         _ = torch_npu.npu.device_count()
15 +         return torch.npu.is_available()
16 +     except RuntimeError:
17 +         return False
18 +
19 +
20 + def get_npu_device_string():
21 +     if shared.cmd_opts.device_id is not None:
22 +         return f"npu:{shared.cmd_opts.device_id}"
23 +     return "npu:0"
24 +
25 +
26 + def torch_npu_gc():
27 +     with torch.npu.device(get_npu_device_string()):
28 +         torch.npu.empty_cache()
```

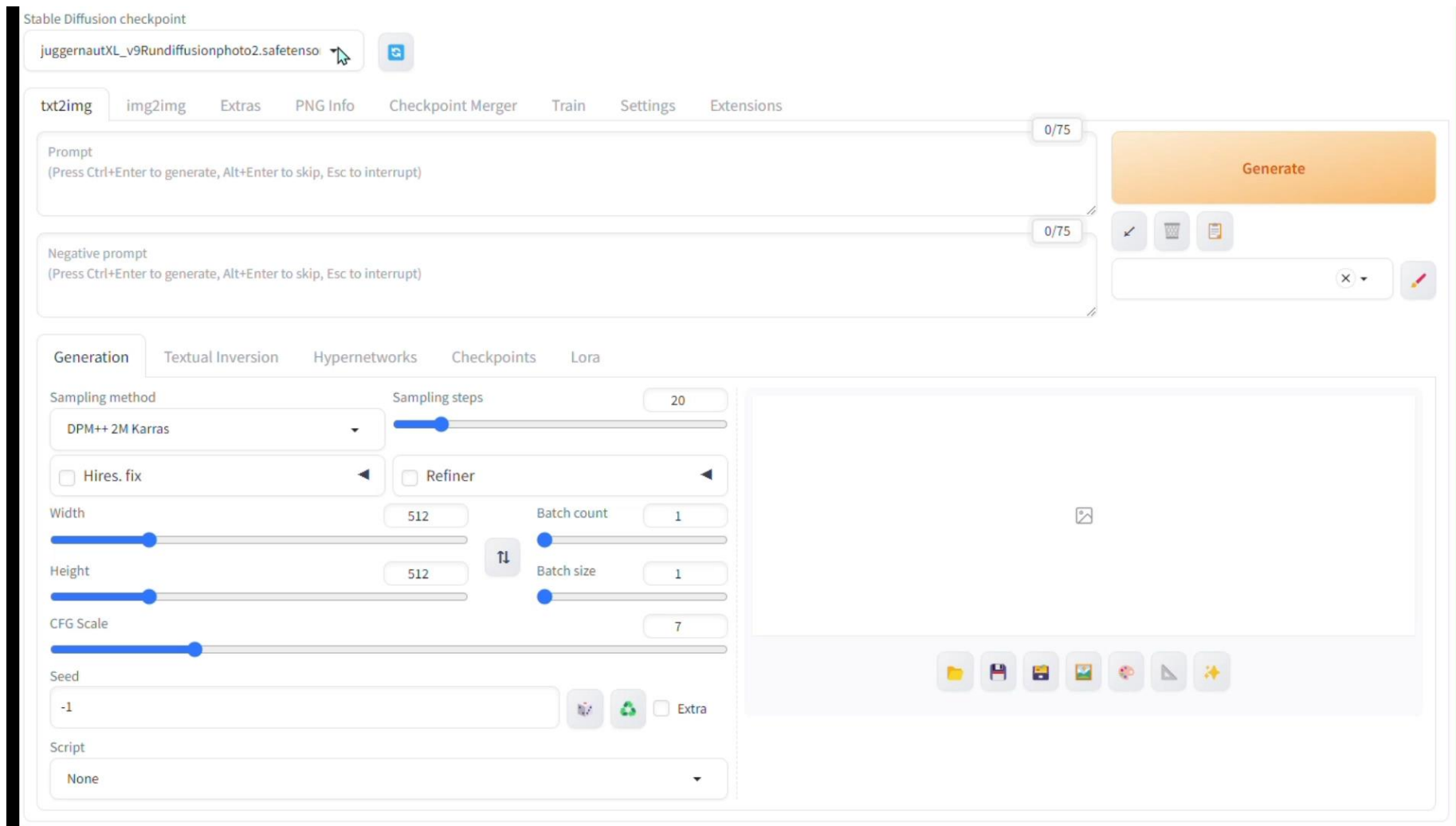


Diffusers (扩散模型推理、训练库)

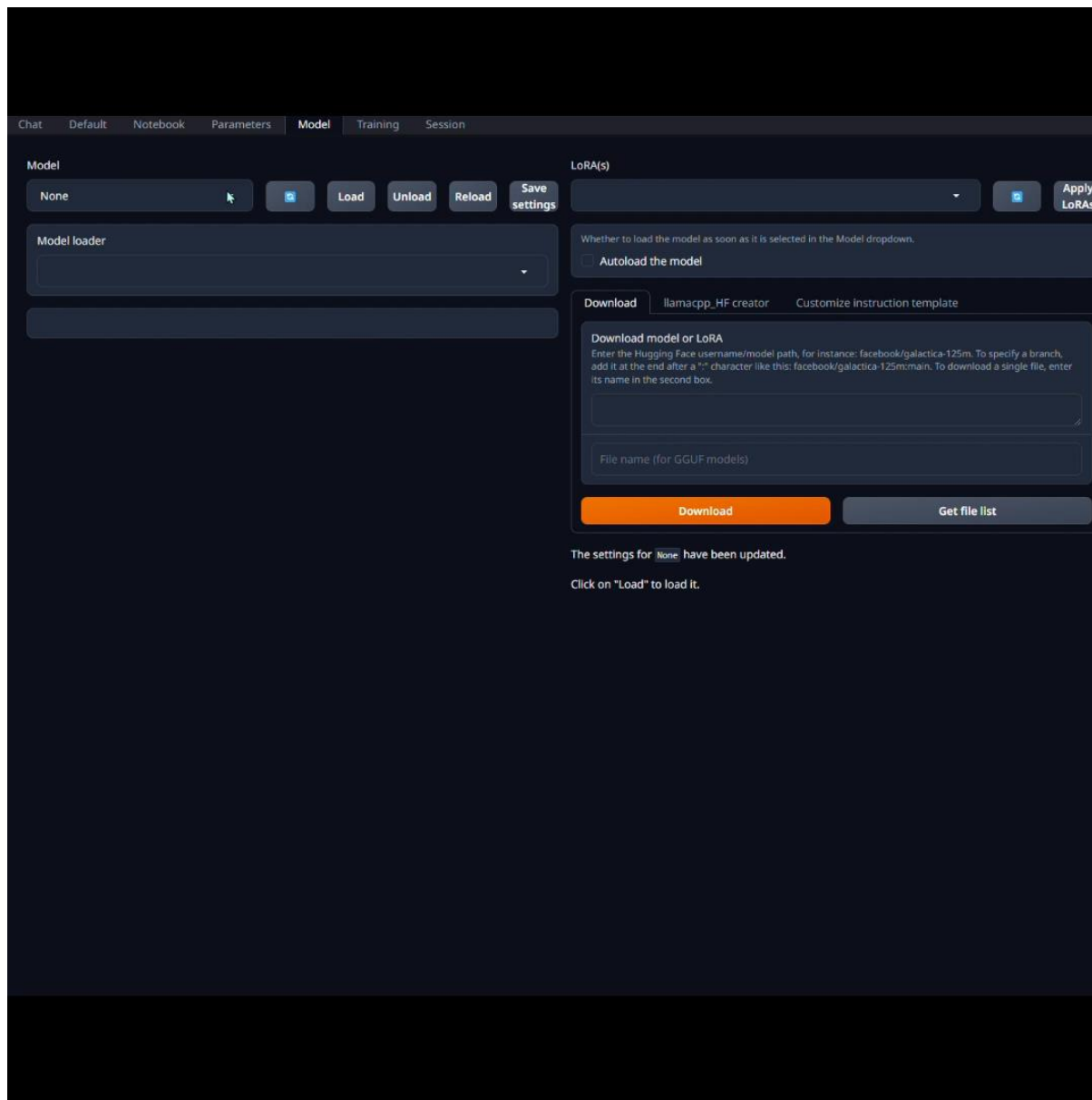
```
73     except ImportError:
74         _torch_xla_available = False
75
76 + # check whether torch_npu is available
77 + _torch_npu_available = importlib.util.find_spec("torch_npu") is not None
78 + if _torch_npu_available:
79 +     try:
80 +         _torch_npu_version = importlib_metadata.version("torch_npu")
81 +         logger.info(f"torch_npu version {_torch_npu_version} available.")
82 +     except ImportError:
83 +         _torch_npu_available = False
84 +
85 _jax_version = "N/A"
86 _flax_version = "N/A"
87 if USE_JAX in ENV_VARS_TRUE_AND_AUTO_VALUES:
88
89
90
91
92
93
94
95
96
97
98
99
304     return _torch_xla_available
305
306
307 + def is_torch_npu_available():
308 +     return _torch_npu_available
309 +
310 +
311 def is_flax_available():
312     return _flax_available
313
```

torch_npu 加载

Stable-diffusion-webui (Powered by OpenEuler+昇腾)



Text-generation-webui (Powered by OpenEuler+昇腾)





谢谢!

