User Manual

用户使用手册

LM2-100

版本：V1.2

版本记录表

Version	Description	Department	Author	Audit	Update Date
V1.0	First Version	Product Department	LAN WEI	CHEN BO、LI YUANHAO	2026-05-31
V1.1	Update some instructions	Product Department	LAN WEI	LIN KUNJIE、CHEN BO、 LI YUANHAO	2026-06-12
V1.2	Adjust Chapter 2 to be about rapid deployment, and add Chapter 3 with detailed deployment references	Product Department	LAN WEI	CHEN BO、 LI YUANHAO	2026-08-06

版权声明

认可声明

了解更多产品信息或其他产品，请访问杰和科技官网: https://www.giadatech.com.cn/

安全使用常识

使用前，请务必仔细阅读产品用户手册

当需要对产品进行操作时请先关闭电源

不要带电插拔，以避免部分敏感元件被瞬间冲击电压烧毁

操作者需采取防静电措施后才能触摸或进行其他可能产生静电冲击的操作

避免频繁开机对产品造成不必要的损伤

技术支持和服务

1. 请访问杰和科技官网https://www.giadatech.com.cn/，获取该产品的最新信息。

2．用户若需技术支持，请与当地分销商、销售商或者杰和科技客服部联系。技术咨询前，请收集如下信息：

产品名称及序列号

外围附加设备

使用的软件(操作系统、版本、应用软件等)

产品所出现问题的完整描述

每条错误信息的完整内容

联系方式

深圳市杰和科技发展有限公司

电话：0755-33300319

邮箱：JH-info@giadatech.com

地址：中国广东省深圳市南山区豪方天际广场37层

目录

1

第一章 概述

1.1 产品简介

LM2-100是杰和科技基于DeepX DX-M1M 芯片，自主研发生产的一款PCIe 3.0 X 2 接口信号、M.2 M+B-Key 物理插槽接口的，高性能低功耗AI算力模组。其内置3个独立算力域的NPU核心，合计算力 25 TOPS@1GHz INT8，内置2GB LPDDR4x 内存，22*42mm小尺寸。是一款高性能、高能效比、小体积、易插拔的AI算力增强模组，特别是在进行机器视觉判定、自然语言识别与处理、工业设备问题诊断等，基于卷积神经网络的判别式前向推理任务时，可达到远超常规通用型AI算力单元的极致能效比；非常适合于计算机机器视觉、安防监控识别判定、自然语言语义判定及内容审核、智慧交通各类违章识别判定、车流统计等场景及利于应用

1.2 产品规格

主要性能	
CPU	ARM Cortex-M55，主频 1GHz
NPU	DX-M1M NPU
算力	25 TOPS @1GHz INT8
内存	内置2GB 64 bit LPDDR4，最大带宽 4266 MT/s
存储	内置128M SPI 接口 NAND Flash
接口方式	
物理接口	标准 M.2 M+B-Key 金手指
接口主信号	PCIe Gen3.0 ×2
数据带宽	最高 16 Gbps，满足高清视频 / 点云数据低延迟传输
辅助信号	复位、电源、状态指示灯控制
人机交互	
指示灯	1* LED（运状态行灯）
操作系统	Linux Ubuntu（默认）、Debian、Windows
电源及供电	
电源供电	DC 3.3V 最大电流3A
典型功耗	2~5W
供电方式	M.2接口供电
整机结构	
构造	嵌入式板卡 + 可扩展亚克力延长板
尺寸	默认22*42mm，支持通过延长板扩展至22*80mm
安装方式	M.2 插槽直插固定，适配主流 ATX / 工业主板、嵌入式载板

固定孔	M.2 标准尾端固定位（螺丝规格：M2×3 mm）
净重	7g（默认版），8.5g（加延长板款）
环境要求	
工作温度	-25 ~60° C ，空气流动
存储温度	-40 ~ 85°C (-40 ~ 185°F)
存储湿度	35%-95%@RH ，无冷凝

1.3 接口及尺寸示意图

LM2-100 示意图（默认版）

图1. 1

LM2-100 示意图（加延长板款）

图1. 2

LM2-100板卡部分尺寸为22*42mm（默认），支持通过延长板扩展至22*80mm，通过M.2 M+B-Key PCIe Gen3.0 ×2信号与载板通信

1.4 产品订购规格

型号	SOC	说明
LM2-100-V0	DX-M1M	M.2 M+B-Key，PCIe Gen3.0 ×2，默认22*42mm，支持加延长板扩展至22*80mm

2

第二章 快速部署指南

2.1 引言说明

1、本章节介绍产品的系统配置及部署，内容为基于杰和科技当前实际调测，的运行结果整理的参考部署方式，旨在帮助用户快速上手运行，所使用的软件套件，部署流程，及demo案例等，不一定覆盖了所有类型的场景。更多的参考内容，如下一章基于芯片平台的详细参考部署，及参考DEEPX官网对应链接 https://developer.deepx.ai/，本文档也将阶段性持续更新，以及根据最新实际使用过程中的案例不断补充完善

2、硬件与系统要求

配置项	最低要求
CPU 架构	x86_64、ARM64
操作系统	Ubuntu24.04/22.04/20.04/18.04(LTS)、Debian 12
内存 (RAM)	≥ 8 GB (推荐16GB+)
可用磁盘空间	≥ 20 GB (home目录空间)

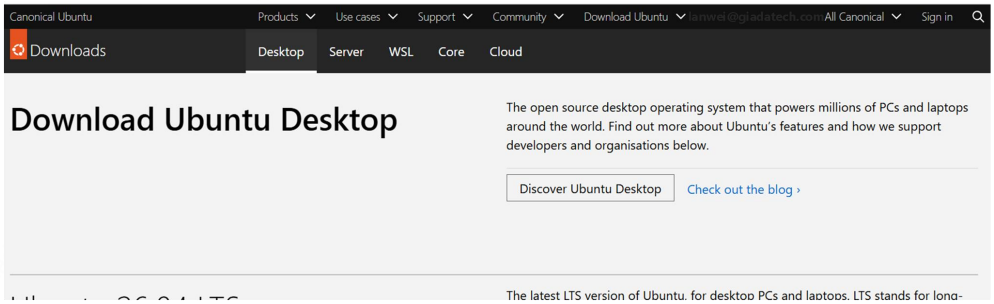
2.2 作业环境准备

2.2.1 硬件环境准备

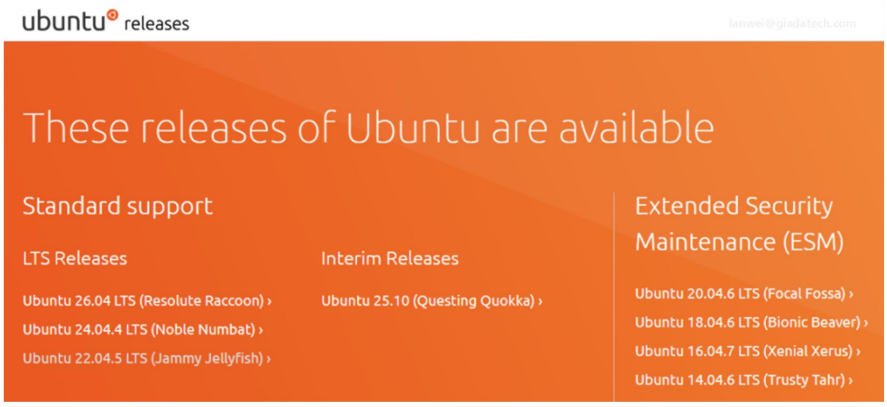
准备一台X86设备，配置好内存（需≥8GB，满足多模型多进程需求）、硬盘（容量建议≥256 GB，适配大模型及数据集预留；类型建议NVMe，PCIe协议数据的传输速率远高于SATA协议数据，AI吞吐效率更高），并接入算力卡

2.2.2 安装 Ubuntu Desktop OS

① 至官方网站Download Ubuntu Desktop | Ubuntu下载 Pre-Build Image

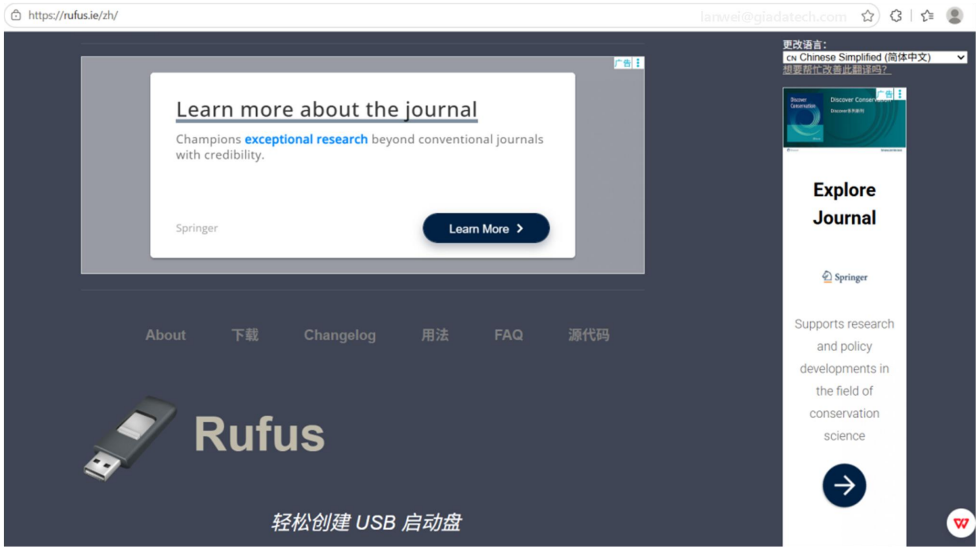


其他版本至 https://releases.ubuntu.com/

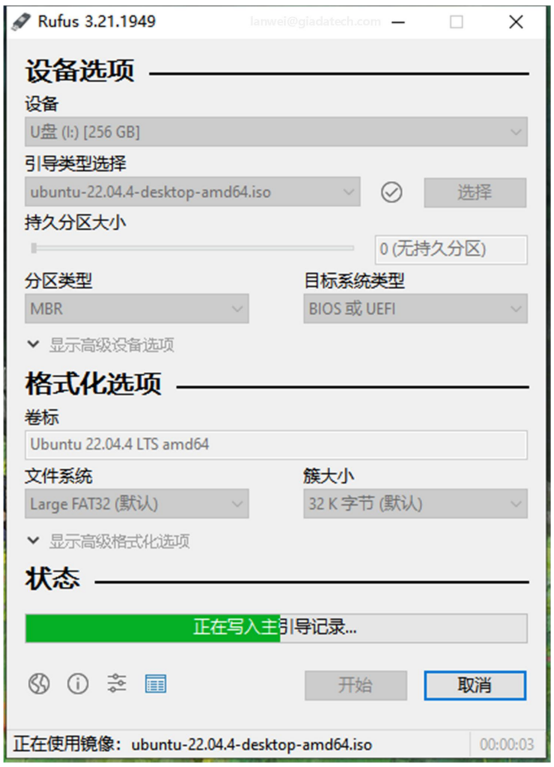


系统默认使用Ubuntu 22.04

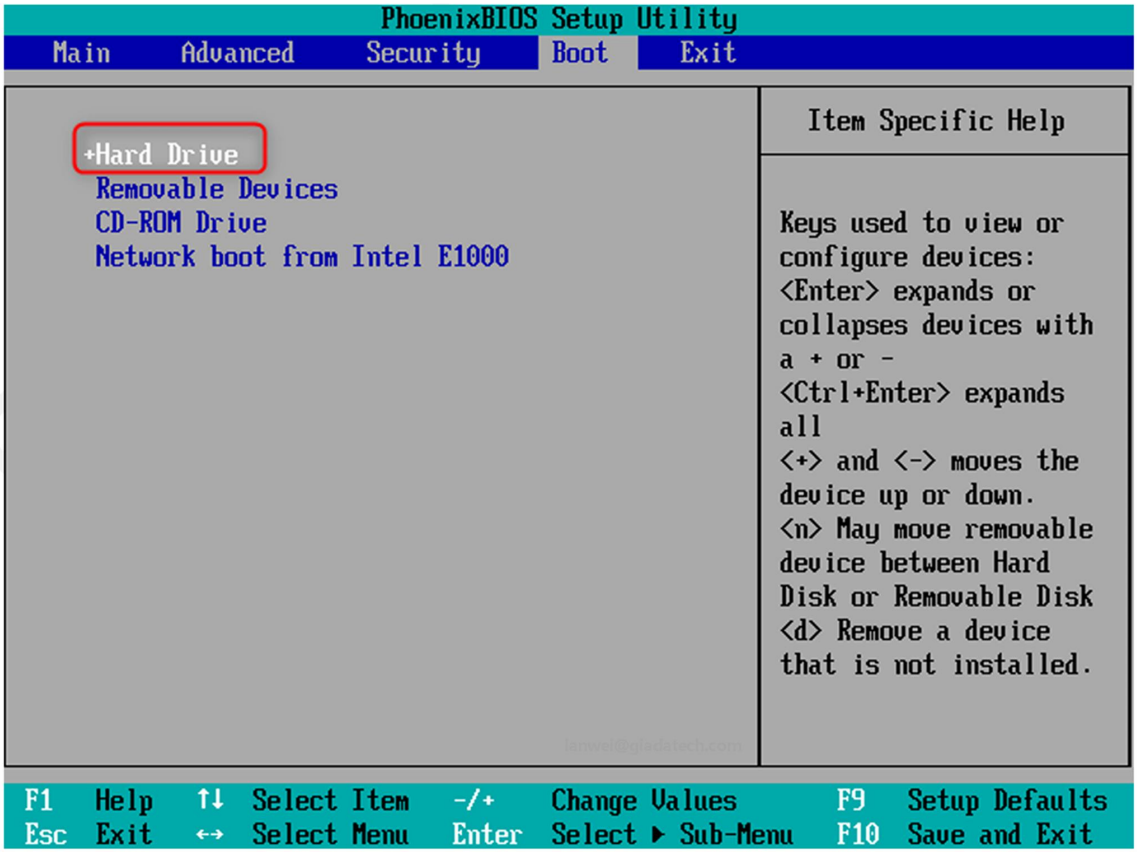
② 到 https://rufus.ie/zh_TW/ 下载 Rufus档案，制作作可开机的 USB 磁碟机



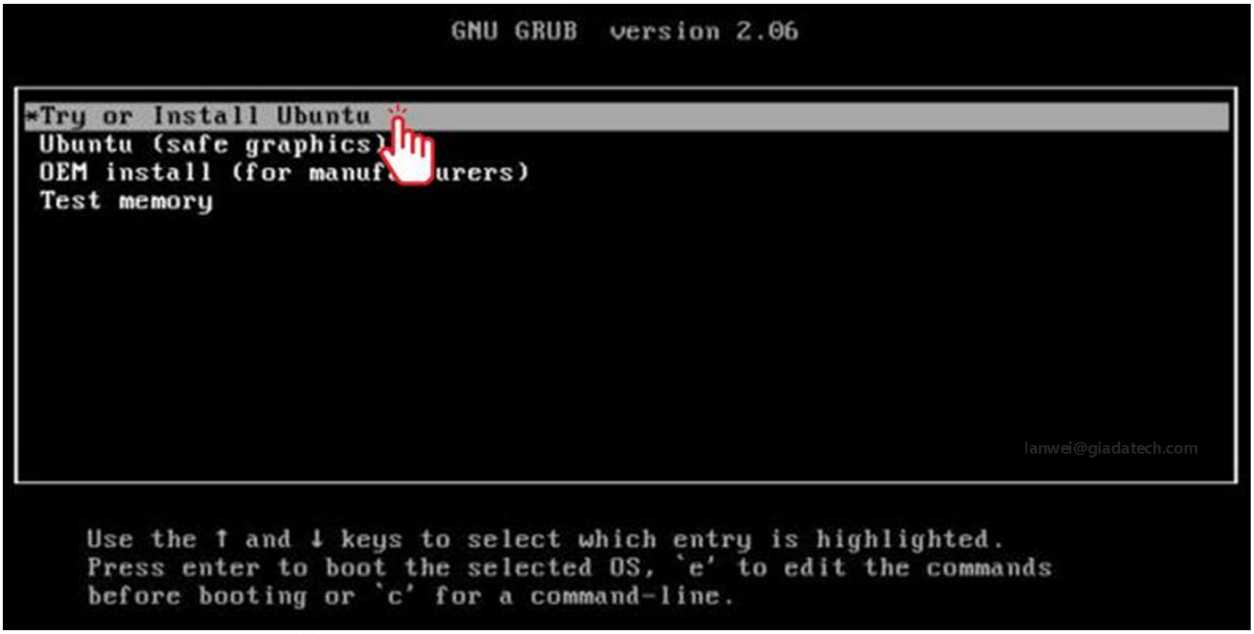
③ 通过Rufus工具把系统烧录到U盘中

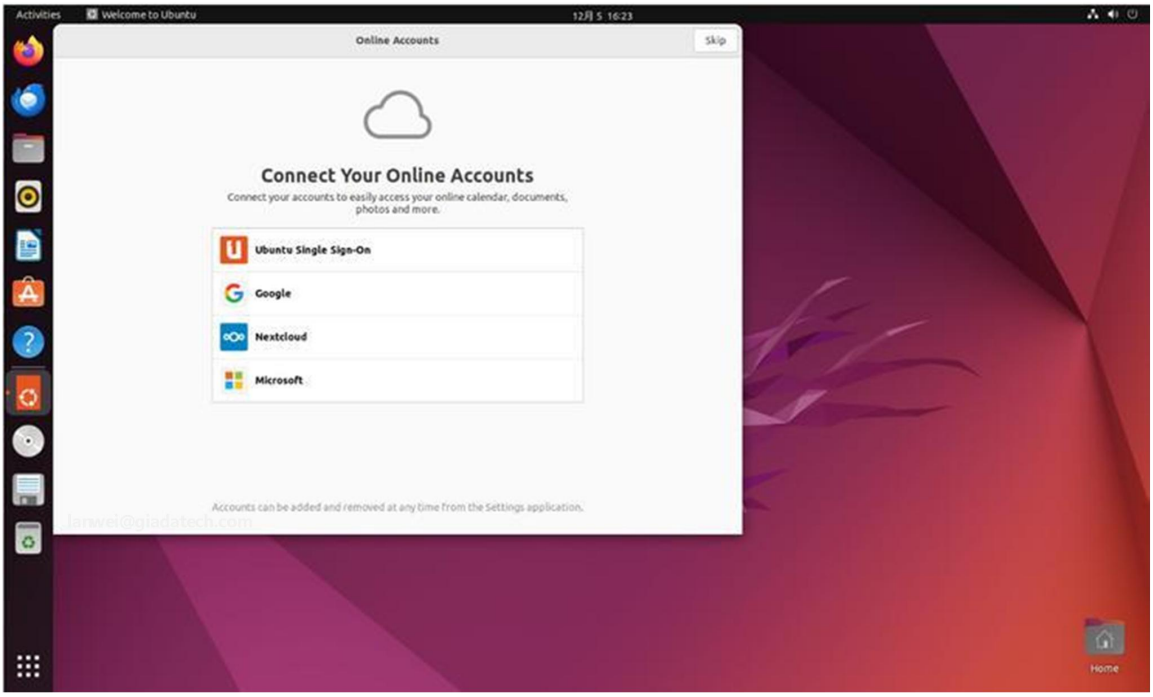


④ 设备接入键盘、鼠标和网络，U盘插入设备，开机时，按F12选择U盘进行系统安装

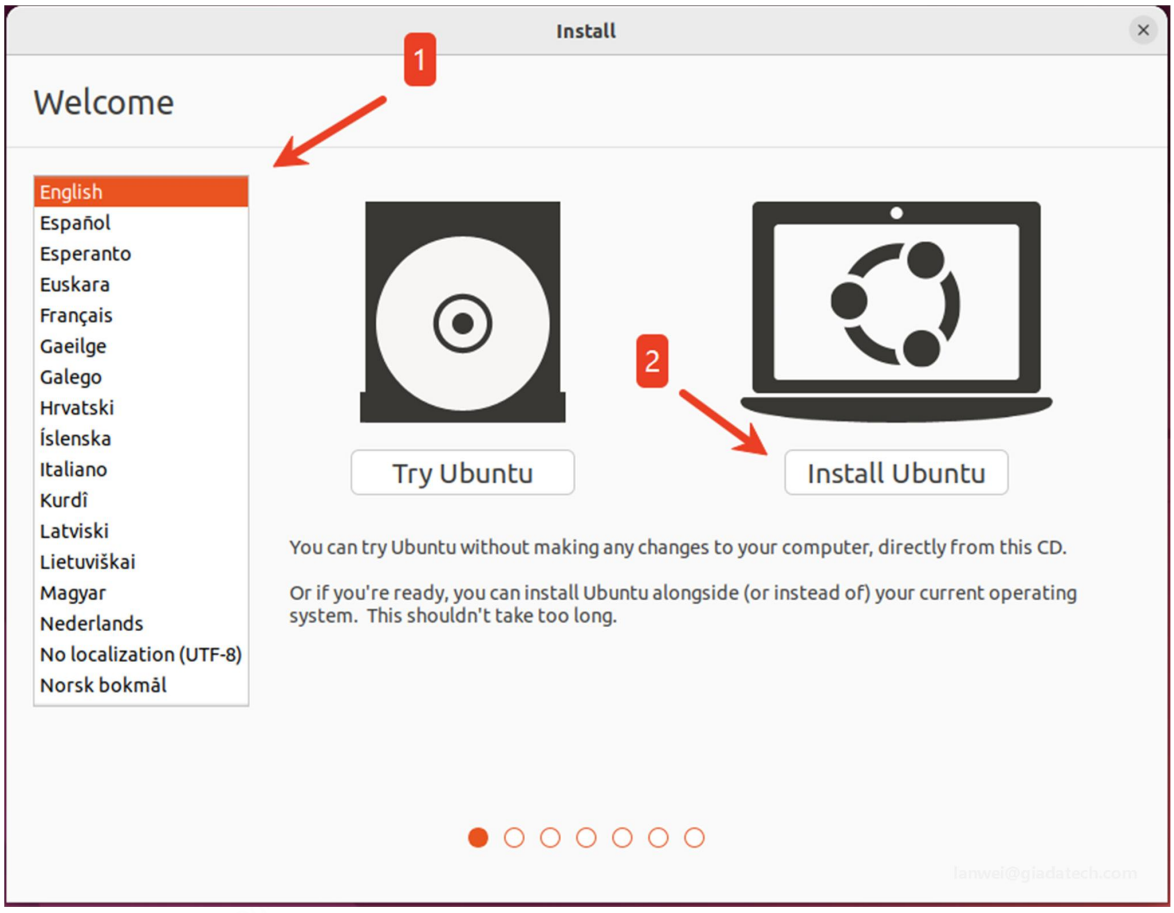


⑤ 进入到 Ubuntu Desktop OS 安装介面

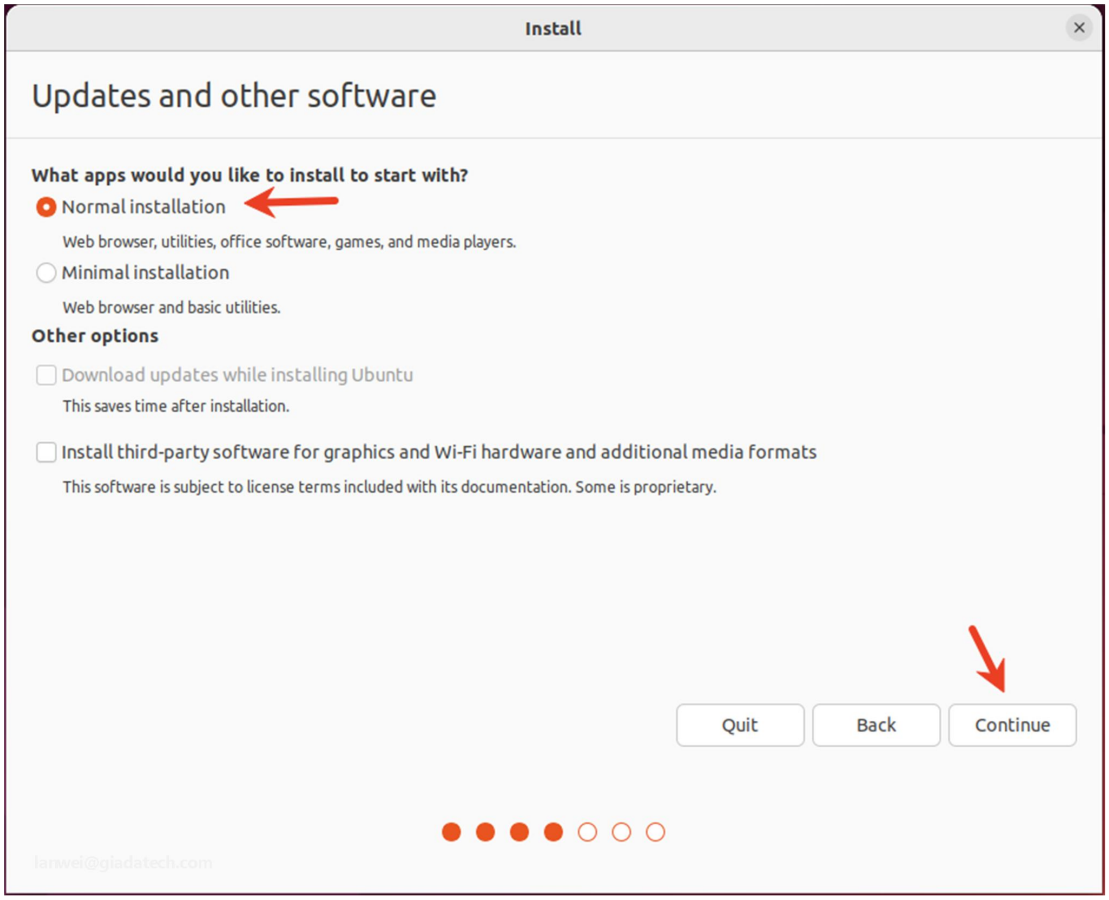




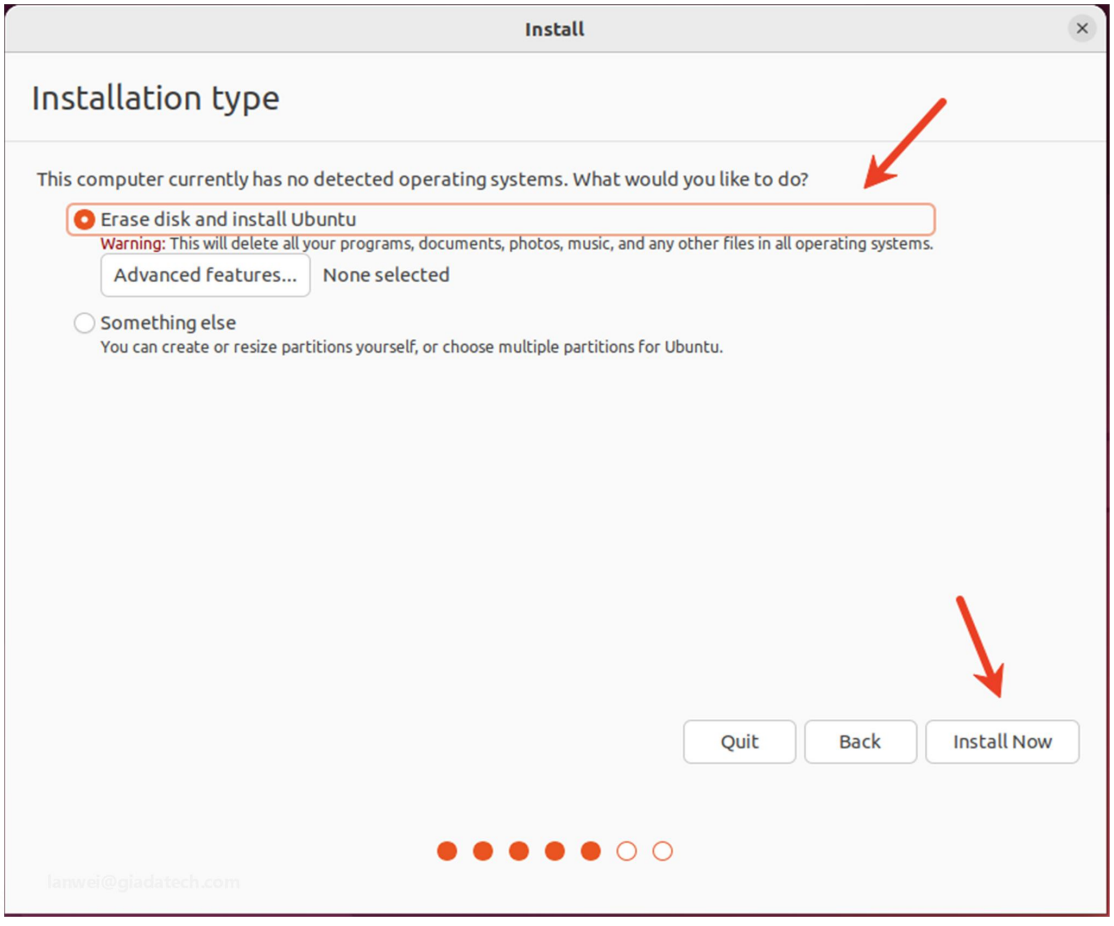
⑥ 选择系统语言，安装Ubuntu，键盘布局默认



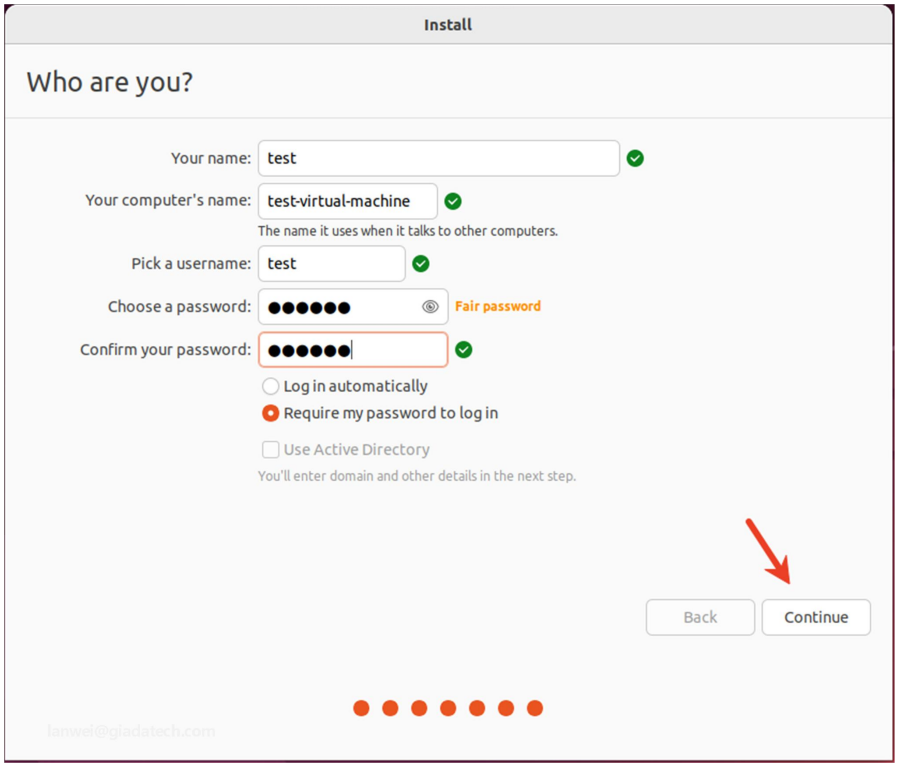
选择正常安装，点击继续



选择清除整个磁盘并安装Ubuntu，点击现在安装



选择地区继续，输入用户信息，开始安装系统



2.3 安装必要套件

包括安装与更新套件等前置准备，开启终端机(terminal) 进行以下操作

2.3.1 x86_64安装方式

更新软件源索引

```
sudo apt update -y
```

升级已安装软件（可选，推荐执行）

```
sudo apt upgrade -y
```

安装核心编译工具、交叉编译器、依赖库

```
sudo apt install -y \
    gcc-12 g++-12 \ # 原生 x86_64 GCC 12 编译器
    gcc-aarch64-linux-gnu g++-aarch64-linux-gnu \ # 交叉编译器
    git meson ninja-build pkg-config build-essential \ # 构建工具链
    libncurses5-dev libncursesw5-dev libtinfo-dev \ # 终端界面库依赖
    python3-dev python3-pip \ # Python 开发环境
    python3-pyqt5 python3-pyqt5.qtquick python3-pyqt5.qtsvg \# Qt5 GUI 界面库

sudo ln -s /usr/bin/python3 /usr/bin/python
```

2.3.2 ARM64 安装方式

更新软件源索引

```
sudo apt update -y
```

升级已安装软件（可选，推荐执行）

```
sudo apt upgrade -y
```

安装核心编译工具、交叉编译器、依赖库

```
sudo apt install -y \
    gcc-12 g++-12 \ # 原生 x86_64 GCC 12 编译器
    gcc-aarch64-linux-gnu g++-aarch64-linux-gnu \ # ARM64 交叉编译器
    git meson ninja-build pkg-config build-essential \ # 构建工具链
    libncurses5-dev libncursesw5-dev libtinfo-dev \ # 终端界面库依赖
    python3-dev python3-pip \ # Python 开发环境
    python3-pyqt5 python3-pyqt5.qtquick python3-pyqt5.qtsvg \# Qt5 GUI 界面库

sudo apt install librga-dev librga2

sudo ln -s /usr/bin/python3 /usr/bin/python
```

2.4 验证 linux-headers

ARM64 需要确认是否安装 linux-headers 环境，x86_64跳过

① 创建文件夹

```
sudo mkdir -p /home/tmp_verify
```

② 复制全部执行（内容复制到文本编辑中按如下排版调整后，拷贝设备上执行）

```
sudo tee /home/tmp_verify/check_headers.c <<-'EOF'
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/init.h>

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Test");
MODULE_DESCRIPTION("Kernel headers verification module");

static int __init check_headers_init(void)
{
    printk(KERN_INFO "check_headers: Kernel headers are working correctly!\n");
    return 0;
}

static void __exit check_headers_exit(void)
{
    printk(KERN_INFO "check_headers: Module unloaded successfully.\n");
}

module_init(check_headers_init);
module_exit(check_headers_exit);
EOF
```

③ 复制全部执行（内容复制到文本编辑中按如下排版调整后，拷贝设备上执行）

```
sudo tee /home/tmp_verify/Makefile <<-'EOF'
obj-m += check_headers.o

all:
	make -C /lib/modules/$(shell uname -r)/build M=$(CURDIR) modules

clean:
	make -C /lib/modules/$(shell uname -r)/build M=$(CURDIR) clean
EOF
```

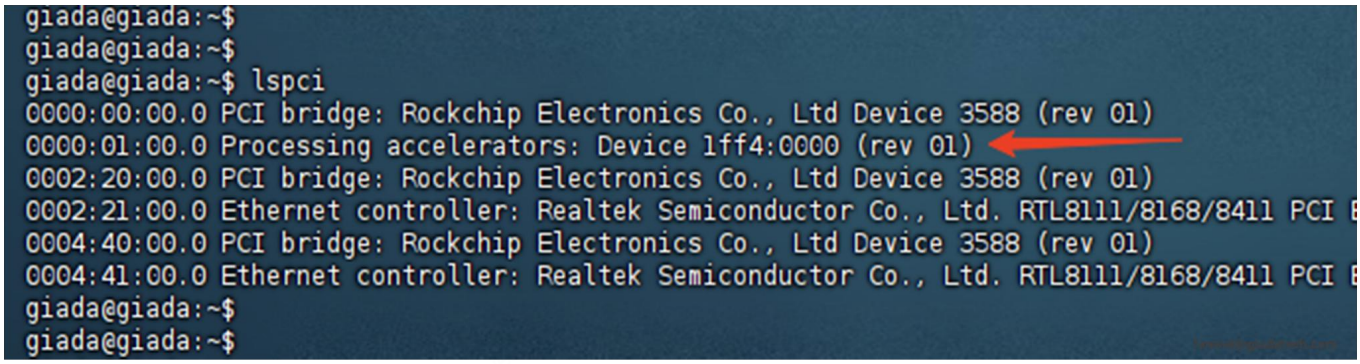
④ 编译并验证

```
cd /home/tmp_verify

sudo sed -i 's/^[[:space:]]*make/\tmake/' Makefile

sudo make

# 如文件存在，者代表 linux-headers 环境正常
ls /home/tmp_verify/check_headers.ko
```

如以上验证失败，需要检查 linux-headers 是否正确安装。未安装或环境缺失，请联系系统厂商提供完整的 linux-headers 文件安装

2.5 检查设备

```
lspci


# 输出包含 1ff4 或 DEEPX，代表读取 LM2-100 设备成功
```



该命令未输出指定内容，说明 PCIe 链路未正常建立，请检查物理连接和系统 BIOS 设置

2.6 准备部署包

①使用本地部署包

```
A. 拷贝部署包到 home 目录下

B. 解压部署包 (*替换为实际版本号)
cd /home
sudo tar -xvzf dx-all-suite_v*.tar.gz

C. 目录所属用户修改为当前用户
sudo chown -R ${USER}:${USER} /home/dx-all-suite
```

② 在线下载部署包

```
A. 进入目录
cd /home

B. 拉取代码 (如网络导致下载失败, 请稍后重试, 或网络查找 GitHub 加速服务下载)
sudo git clone --recurse-submodules https://github.com/DEEPX-AI/dx-all-suite.git

C. 目录所属用户修改为当前用户
sudo chown -R ${USER}:${USER} /home/dx-all-suite

D. 进入部署包
cd /home/dx-all-suite

E. 切换到指定版本
git checkout v2.4.1
# 查看版本 (输出对应版本则成功)
git describe --tags

F. 初始化并更新所有子模块
git submodule update --init --recursive
# 查看子模块状态 (前缀不含 - 或 + 符合, 者代表同步成功)
git submodule status
```

以上部署方式二选一

2.7 安装并验证

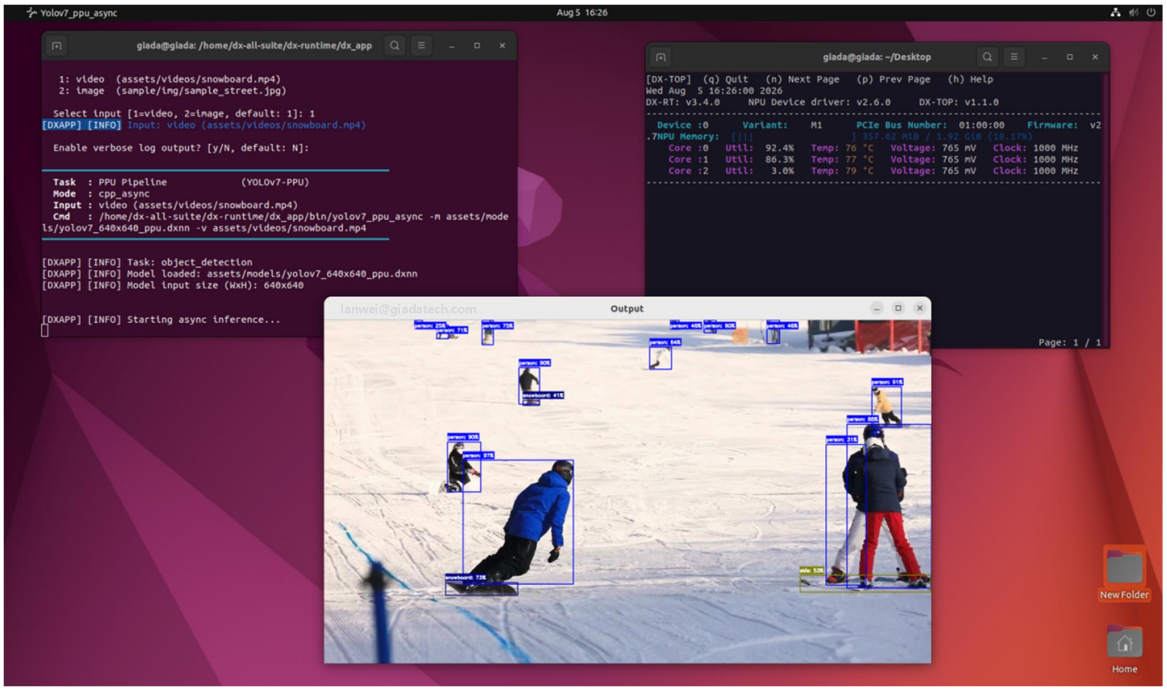
重要：剩余可用内存不足8G，需要创建交换空间(Swap)或增加内存后再安装（临时编译使用）

```
A. 进入到项目目录
cd /home/dx-all-suite

B. 一键安装
./dx-runtime/install.sh --all

# 如一键安装报错，可按如下单独安装每个组件（一键安装报错，才执行这一步）
./dx-runtime/uninstall.sh
./dx-runtime/install.sh --target=dx_rt_npu_linux_driver
./dx-runtime/install.sh --target=dx_rt
./dx-runtime/install.sh --target=dx_fw
./dx-runtime/install.sh --target=dx_app
./dx-runtime/install.sh --target=dx_stream

C. 安装完成后，请关机断开电源再上电开机

D. 验证
dxrt-cli -s

【正常输出示例】
DXRT v3.3.2
=======================================================
 * Device 0: M1, Accelerator type
---------------------   Version   ---------------------
 * RT Driver version   : v2.4.1
 * PCIe Driver version : v2.2.0
-------------------------------------------------------
... (Continued)
```

2.8 运行Demo示例(dx_app)

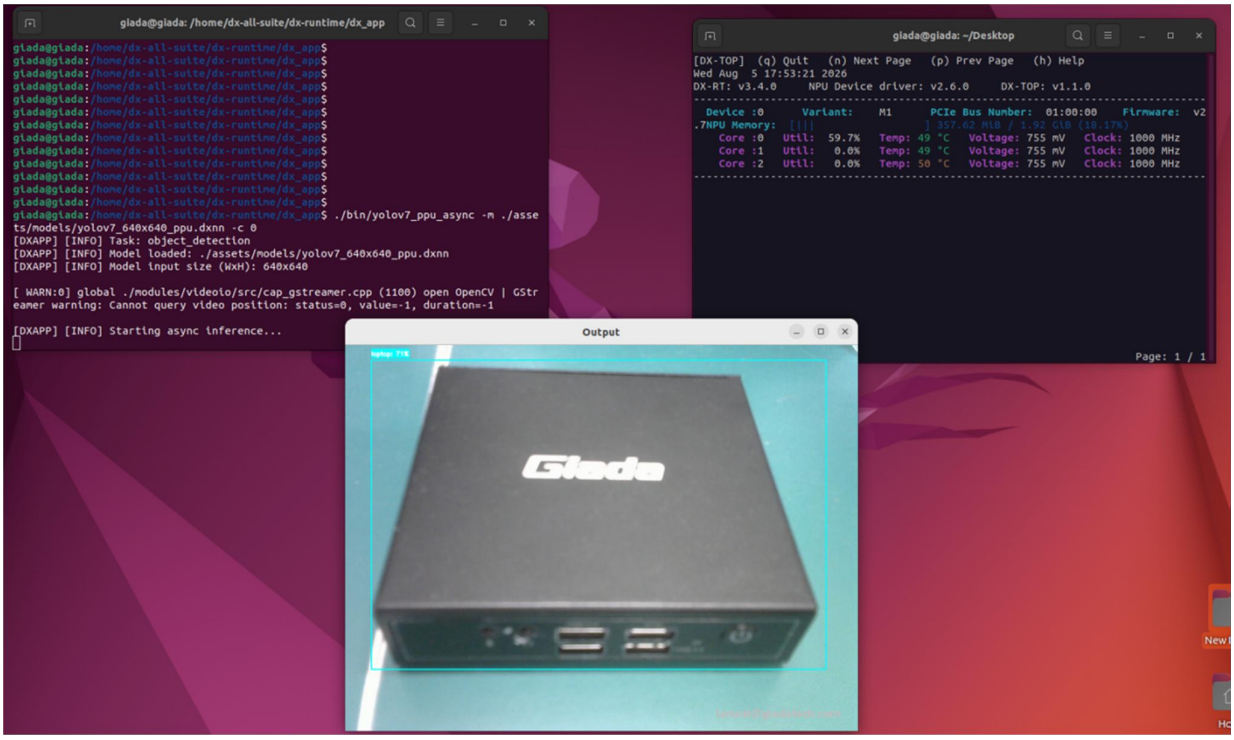
①运行Demo

```
# 进入目录
cd /home/dx-all-suite/dx-runtime/dx_app

# 运行本地识别（首次必须运行，需要下载模型和视频）
./run_demo.sh
```



② 摄像头识别

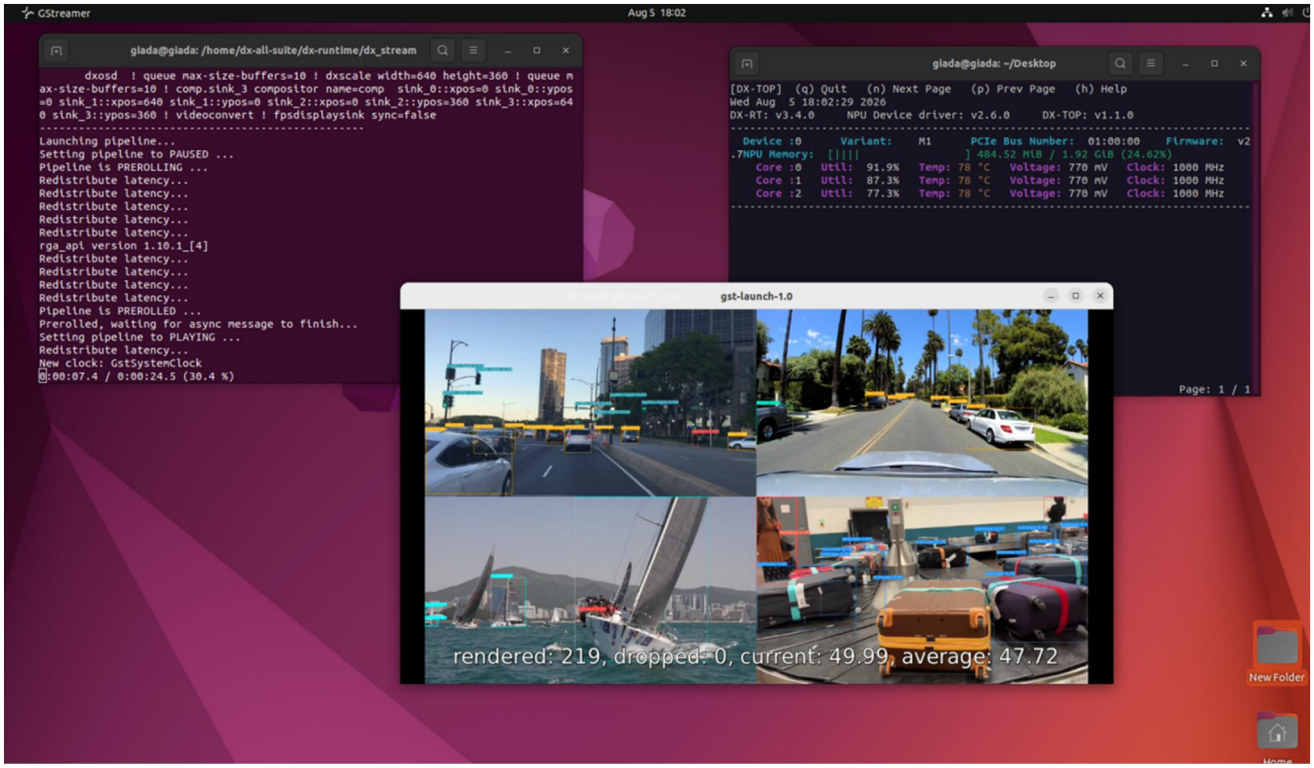
```
# 进入目录
cd /home/dx-all-suite/dx-runtime/dx_app

#  USB摄像头识别
./bin/yolov7_ppu_async -m ./assets/models/yolov7_640x640_ppu.dxnn -c 0

# 网络摄像头识别
./bin/yolov7_ppu_async -m ./assets/models/yolov7_640x640_ppu.dxnn \
-r
"rtsp://admin:123456@192.168.13.233:554/cam/realmonitor?channel=1&sub
type=0"

>以大华为例： rtsp://账号:密码@IP:554/cam/realmonitor?channel=1&subtype=0
```



2.9 运行Demo示例(dx_stream)

① 运行Demo

```
# 进入目录

cd /home/dx-all-suite/dx-runtime/dx_stream

# 运行本地识别（首次运行，需要下载模型和视频）

./run_demo.sh
```



3

第三章 详细部署参考

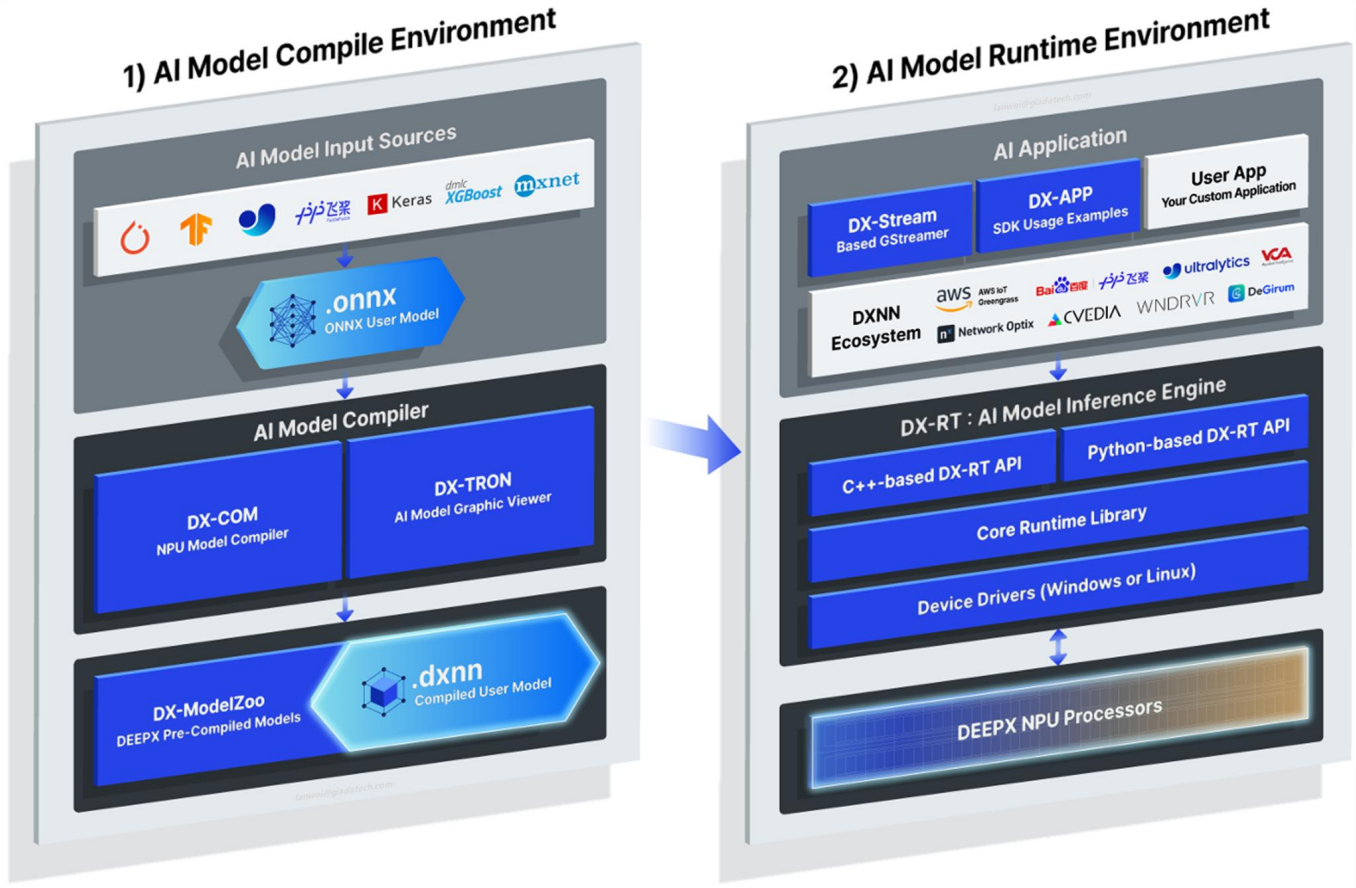
3.1 引言

为更好地支持用户使用本产品，提供更多的方法与案例，本章基于DEEPX的AI算力芯片本身的各类产品（不局限于LM2-100本身），包括基于DEEPX算力芯片的模组、板卡、开发板、AI边缘整机等，提供围绕DX-AS（DEEPX官方套件）的相关AI推理部署方法及应用案例参考

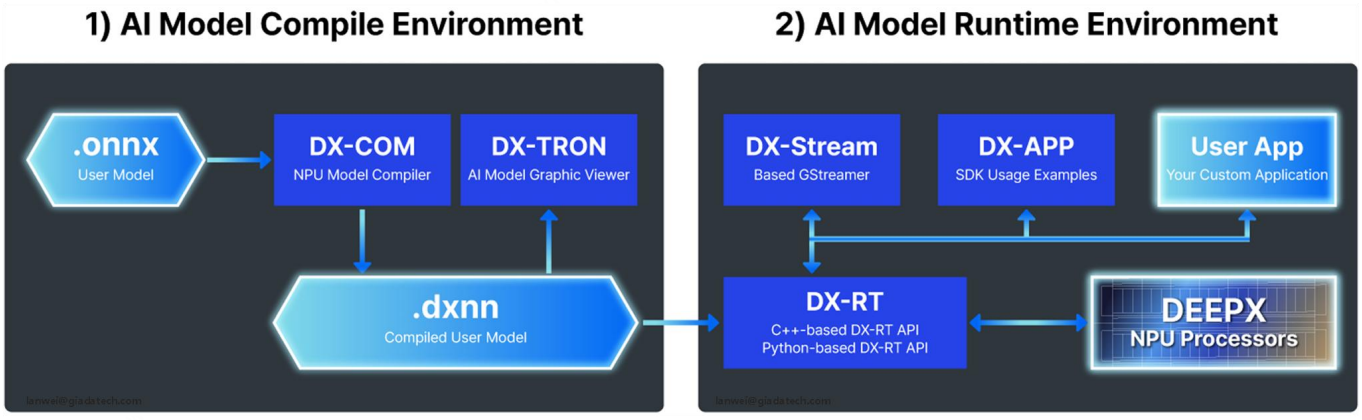
3.2 DX-AS简介

3.2.1 概述

DX-AS（DEEPX全套件）是一套集成式框架与工具环境，可基于DEEPX硬件加速卡完成AI模型编译与推理部署。



用户可单独安装各工具组件，但DX-AS会统一对齐所有组件版本，保障全链路最优兼容性。



3.2.1 核心组件介绍

DX-COM（NPU编译器）

DEEPX NPU编译器（DX-COM）读取ONNX模型与配置文件，生成NPU专用指令集；输出文件graph.dxnn包含完整指令集与模型权重，是NPU硬件执行的核心文件。

DX-RT（推理运行时）

- DX-RT是适配DEEPX硬件的AI推理SDK，可加载DEEPX官方模型库预编译模型，或DX-COM编译生成的DXNN模型。
- DX-RT提供C/C++原生API，支持开发者自研应用；
- DX-RT配套Python封装接口，可通过Python脚本快速开发业务程序；
- DX-RT内置命令行工具（CLI）：查看模型信息、虚拟文件基准测试、实时监控NPU硬件状态；
- DX-RT通过PCIe总线调用DX NPU驱动，向NPU传输输入数据并读取推理结果。

DX-ModelZoo（官方模型库）

- DEEPX开放模型库帮助开发者零门槛使用NPU加速算力，内置大量经过硬件适配验证的神经网络模型。
- 所有上架模型均配套：预训练ONNX文件、配置JSON、预编译DXNN二进制文件。
- 开发者也可自行将ONNX模型编译为DXNN格式，快速搭建NPU加速应用。
- 配套完整性能基准测试工具，可对比INT8量化NPU推理与GPU/CPU FP32高精度推理的精度、速度差异。

DX-APP（应用示例工程）

- DX-RT API的官方示例工程，帮助开发者快速搭建运行环境、直观演示推理效果。
- DX-APP可快速实现目标检测、人脸检测、图像分类等视觉任务的NPU加速推理演示，开发者可直接参考
- DX-APP源码进行自有业务开发。

DX-Stream（流媒体推理插件）

- 基于GStreamer开发的自定义插件，简化视觉AI流媒体应用开发。
- DX-Stream以模块化流水线架构实现NPU加速推理，支持自定义预处理、推理、后处理单元，适配各类视频AI业务场景。

3.2.2 DX-AS（DEEPX全套件）环境与安装指南

① AI模型编译环境（主机编译平台）

- 用途：安装在用于模型转换的主机上，负责将ONNX模型编译为DEEPX私有DXNN格式。
- 核心组件：DX-COM，将ONNX模型转换为高度优化、可直接在NPU运行的二进制文件。
- 兼容性：操作系统：仅支持Debian系Linux；硬件架构：仅x86_64
- 便捷安装：一键自动化脚本完成全套部署，安装后所有编译组件可直接使用。编译工具安装指引可查看对应链接。

② AI模型运行环境（部署目标平台）

- 用途：安装在搭载DEEPX M1 M.2加速模块的设备上，用于执行.dxnn推理模型。
- 核心组件：DX-RT、固件、NPU驱动（硬件控制底层基础软件）；DX-APP（C++/Python示例工程，快速启动项目）；DX-Stream（GStreamer多媒体流水线集成插件）。
- 兼容性：操作系统：Debian系Linux；硬件架构：x86_64、arm64双架构支持
- 便捷安装：自动化脚本一键部署，安装完成后需重启一次设备完成驱动初始化。运行时环境安装指引可查看对应链接。

3.3 DEEPX智能体驱动开发—dx-agent-dev（测试版）

测试版功能说明：智能体驱动开发功能仍在持续迭代，功能定义、任务调度逻辑会随版本更新调整

3.3.1 简介

DEEPX智能体驱动开发工具dx-agent-dev（测试版），支持使用自然语言搭建NPU应用：只需用通俗语言描述应用功能或模型任务，AI代码智能体（Claude Code、Cursor、GitHub Copilot、OpenCode、Codex）会基于DEEPX知识库完成全流程开发：需求梳理 → 方案规划 → 测试驱动开发 → 结果校验，覆盖从ONNX/.pt模型编译到DX-M1 NPU端侧部署全链路。

该工具专为Ultralytics模型生态适配DEEPX NPU打造，文档内所有落地案例均通过该功能自动生成，配套原始提示词、实测性能数据、完整构建日志。

支持开发流程

- 独立推理程序：基于IFactory、同步执行器SyncRunner、异步执行器AsyncRunner开发
- GStreamer视频流水线：调用DEEPX自研13个自定义插件，覆盖6大功能分类
- 跨项目工程：同时调用dx_app、dx_stream、dx-runtime多模块
- 模型编译：通过dx-compiler内DX-COM完成ONNX转DXNN格式

3.3.2 落地案例

以下所有案例均仅通过一段自然语言提示词，全自动生成完整DEEPX NPU应用；每个案例仓库内均存放原始提示词、实测指标、一键复现脚本、完整构建会话日志。

① NPU体感交互小游戏

仅需20分钟、极低开发成本，通过自然语言生成完整NPU体感小游戏，带街机风格可视化界面。

案例名称	功能说明	开发耗时	智能体交互轮次	输出Token总量	预估开发成本
深蹲计数小游戏	通过膝盖、髋关节关键点角度统计深蹲次数，带计分、动作提示可视化界面	约12分钟	132轮	约109K	7.3美元
拉伸指导小游戏	3套标准拉伸动作，搭配动画虚拟教练演示标准姿态	约15分钟	130轮	约142K	8.1美元

2.2.2 Ultralytics YOLO生态适配

一行命令将Ultralytics YOLO模型迁移至DEEPX NPU，也可针对行业场景重新训练；支持四维度对比评估（原始模型/微调模型 × GPU FP32/NPU INT8），INT8量化推理精度与FP32基本持平，行业定制模型在NPU上推理速度更快。

案例名称	功能说明	开发耗时	智能体交互轮次	输出Token总量	预估开发成本
Ultralytics YOLO转DEEPX模型导出	单条yolo导出命令，将.pt权重转为可部署NPU专用.dxnn模型，自动执行推理校验	约12分钟	108轮	约84K	2.4美元
野生动物监测模型	基于yolo26n训练非洲野生动物识别（水牛/大象/犀牛/斑马），用于野外保护摄像头，四维度精度速度评估	约7分钟	78轮	约85K	3.2美元
工地劳保安全检测	yolo26n微调工地安全帽、反光背心识别，用于工地安防摄像头，四维度评估	约17分钟	102轮	约93K	4.0美元
脑部肿瘤CT/MRI筛查	yolo26n医疗影像微调，用于边缘医疗设备病灶检测，四维度评估	约9分钟	91轮	约103K	3.7美元
药片计数质检模型	yolo26n医药包装药片识别，用于产线计数工位，四维度评估	约8分钟	118轮	约103K	5.1美元

2.2.3 PaddlePaddle生态适配

PP-OCRv5百度文字识别模型部署至DEEPX NPU，单段提示词实现视频/摄像头实时文字识别，完整流程：文字检测 → 方向校正 → 文字识别。

案例名称	功能说明	开发耗时	智能体交互轮次	输出Token总量	预估开发成本
视频/摄像头实时OCR（PP-OCRv5）	NPU端完成整套文字识别，视频文件、实时摄像头共用一套代码，自动标注文字框与识别文本；单帧最多识别14处文本，推理速度约2.8帧/秒，单帧耗时341ms	约18分钟	175轮	约184K	12.0美元

2.2.4 RapidAI文档解析生态适配

单段自然语言生成PDF转Markdown文档解析应用，基于RapidAI RapidDoc（PP-StructureV3），完成版面分析、OCR文字识别、表格、公式解析，全部在DX-M1 NPU上加速运行。

案例名称	功能说明	开发耗时	智能体交互轮次	输出Token总量	预估开发成本
PDF转Markdown文档解析程序	支持电子版/扫描版PDF，输出结构化Markdown+JSON；9页财务报告端侧完整解析，保留21个标题、9张HTML表格；自动解析模式耗时12.6秒，纯OCR解析14.7秒	约12分钟	133轮	约148.5K	6.2美元

2.2.5 案例复现方式

```
Bash
cd dx-agent-dev-showcase/对应案例文件夹
bash setup.sh && bash run.sh
# 体感小游戏额外执行方式
./setup.sh && ./run.sh --camera 0
# 模型导出/微调案例直接执行一键脚本
```

运行前置条件：x86_64 Linux系统 + 完整DEEPX运行时dx_engine；每个案例README内附带完整前置依赖、原始提示词，同步提供韩文版README-ko.md。

3.3.3 运行逻辑说明

底层运行逻辑：调度框架而非单纯模型调用

每个自动生成的应用都会完整保存智能体构建会话日志，可直观看到整套流程核心依赖调度框架的指令、工具、校验机制，而非仅依赖基础模型：

- 指令遵循：智能体严格遵循套件硬性约束：会话起止标记[DX-AGENT-DEV: START]/[DX-AGENT-DEV: DONE]；所有生成代码隔离至独立会话目录，不会覆盖原有工程代码；不生成占位桩代码。
- 智能体与工具调用：严格按固定流程调用工具：任务路由工具 → 需求梳理智能体 → 开发方案生成 → 测试驱动开发 → 结果校验，而非仅文本描述流程。
- 完整推理链路：自动检索匹配现有参考案例、读取知识库内官方API文档、先编写校验用例（红测），逐文件生成并校验后才标记开发完成。

3.3.4 环境前置要求

需求项	详细说明
DEEPX开发环境	完整安装DX-RT SDK，执行setup_env.sh加载环境变量
AI代码智能体（任选其一）	Claude Code、VS Code GitHub Copilot、Cursor、OpenCode、Codex CLI
Python环境	Python 3.10及以上，完整安装dx-all-suite套件依赖包

3.3.5 架构总览

智能体驱动知识库分为三层独立模块，每层自带.deepx/目录存放功能套件、指令、会话记忆文件，智能体执行任务时自动读取。

3.3.5.1 dx_app：独立推理程序层

不依赖GStreamer的Python/C++推理应用，核心抽象类：

- IFactory：生成模型专属预处理/后处理流水线
- SyncRunner/AsyncRunner：同步阻塞推理、异步非阻塞推理执行器
- DxInfer：推理引擎底层封装接口

.deepx/知识库覆盖模型加载、DXNN文件寻址、批量推理、结果可视化全流程。

3.3.5.2 dx_stream：GStreamer流媒体流水线层

基于GStreamer构建实时视频分析应用，智能体内置13个DEEPX专用插件，分为6大功能分类（数据源、推理、画面叠加、编码、推流、输出），可通过单条自然语言生成多分支复杂流水线。

3.3.5.3 dx-runtime：跨模块调度中间层

统一跨项目路由与标准化校验，位于另外两层上层，自动分发任务至对应子项目构建器，统一编码规范、测试流程、模型管理规则。

3.3.5.4 dx-compiler：模型编译层

DX-COM驱动DXNN模型编译全流程，智能体完整掌握编译链路：ONNX模型合法性校验、自动生成配置json、校准数据集准备、INT8量化、PPU硬件预处理单元配置。编译前会自动发起需求确认提问：无NMS模型检测、ONNX模型简化、PPU编译配置，确保参数完全适配硬件。

3.3.6 可用智能体与功能套件

仓库各层级均配置专属智能体与功能工具，顶层dx-all-suite内置路由智能体，自动识别任务并分发至对应子模块。

3.3.6.1 分层智能体列表

层级	智能体名称	功能描述
dx-all-suite顶层	@dx-suite-builder	全局路由智能体：识别任务类型，分发至对应子模块
dx-all-suite顶层	@dx-suite-validator	全套件校验智能体：跨三层执行框架完整性检测
dx-runtime	@dx-runtime-builder	跨项目构建智能体：分发任务至dx_app或dx_stream
dx-runtime	@dx-validator	统一校验调度器，带错误反馈修复闭环
dx_app	@dx-app-builder	独立推理程序构建器，分发至细分专业工具
dx_app	@dx-python-builder	Python推理程序构建器（4种变体：同步、异步、C++后处理、异步C++后处理）

dx_app	@dx-cpp-builder	C++推理程序构建器
dx_app	@dx-model-manager	模型下载、仓库管理智能体
dx_stream	@dx-stream-builder	GStreamer流水线构建器，分发细分功能工具
dx_stream	@dx-pipeline-builder	完整流水线搭建（含6大类插件、消息队列）
dx_stream	@dx-validator	dx_stream流水线校验与反馈闭环
dx-compiler	@dx-compiler-builder	模型编译路由，分发格式转换/编译工具
dx-compiler	@dx-model-converter	PyTorch模型转ONNX工具
dx-compiler	@dx-dxnn-compiler	ONNX转DXNN编译工具（DX-COM）

3.3.6.2 OpenCode专属功能套件（斜杠指令调用）

层级	智能体名称	功能描述
dx-runtime	/dx-agent-runtime-validate	全局校验、收集错误、自动修复、二次验证
dx_app	/dx-agent-app-build-python	生成Python推理程序
dx_app	/dx-agent-app-build-cpp	生成C++推理程序
dx_app	/dx-agent-app-build-async	生成高性能异步推理程序
dx_app	/dx-agent-app-model-management	模型下载、参数配置
dx_app	/dx-agent-app-validate	推理程序完整性校验
dx_stream	/dx-agent-stream-build-pipeline	生成GStreamer视频流水线
dx_stream	/dx-agent-stream-build-mqtt-kafka	生成带MQTT/Kafka消息推送的流水线
dx_stream	/dx-agent-stream-validate	流媒体流水线校验
dx_stream	/dx-agent-stream-model-management	流媒体场景模型配置
dx-compiler	/dx-agent-compiler-convert	PyTorch转ONNX
dx-compiler	/dx-agent-compiler-compile	ONNX编译DXNN

dx-compiler	/dx-agent-compiler-validate	DXNN编译结果校验
dx-all-suite顶层	/dx-swe-brainstorm	全流程需求梳理：所有开发前统一方案设计
dx-all-suite顶层	/dx-swe-tdd	测试驱动开发：逐文件增量校验
dx-all-suite顶层	/dx-swe-verify	开发完成校验：产出完整验证结果再标记完工

使用提示：不确定任务归属哪个子模块时，直接调用顶层@dx-suite-builder，自动识别并分发任务。

3.3.7 支持的AI开发工具

智能体驱动开发兼容5款主流AI代码工具，各工具通过自身配置机制自动加载.deepx/全套知识库。

3.3.7.1 工具自动加载规则

工具	全局配置文件	分文件路由规则	智能体调用方式	功能套件调用方式
Claude Code CLI	项目根目录CLAUDE.md	内置上下文路由表	@智能体名称 自由对话	直接读取.deepx/套件文件
GitHub Copilot VS Code	.github/copilot-instructions.md	.github/instructions/匹配glob规则文件	Copilot对话内@智能体名称	套件内容内联复制至.github目录
Cursor IDE	.cursor/rules/目录下dx-*.mdc，alwaysApply自动加载	.cursor/rules/glob匹配文件	自由对话自动触发规则	同目录mdc规则内置套件逻辑
OpenCode CLI	AGENTS.md + opencode.json	无分文件路由	@智能体名称	斜杠/套件名指令调用
Codex CLI	AGENTS.md + .codex/配置	直接读取.deepx/套件SKILL.md	终端自由对话	直接读取套件文件

3.3.7.2 首次使用配置

无需额外手动配置，直接在对应工具打开项目根目录，配置文件自动加载：

```
Bash
# Claude Code
cd dx-all-suite
claude
# OpenCode
```

```
cd dx-all-suite
opencode
# Codex CLI
~/bin/codex
cd dx-all-suite
# GitHub Copilot VS Code
code dx-all-suite
# Cursor CLI
cd dx-all-suite
cursor-agent
```

3.3.7.3 项目文件加载规则

- Git子模块边界限制：Copilot、Claude Code、Codex CLI仅能读取当前git根目录文件；在dx-all-suite根目录打开时，不会自动加载dx-compiler、dx-runtime等子模块文件；OpenCode通过opencode.json显式路径引用突破该限制。
- 自动加载文件：套件全局配置文件，打开项目自动载入
- 智能体文件：需手动@提及调用，不会自动加载
- 功能套件文件：仅OpenCode支持斜杠指令直接调用，其余工具内置至全局配置

共享知识库 .deepx/ 目录说明

.deepx/是所有工具的统一标准知识库，作为唯一数据源；dx-agent-gen生成器会将该目录内容转换为各工具专属配置文件（Copilot对应.github/、Claude对应.claude/、OpenCode对应.opencode/、Cursor对应.cursor/rules/），禁止直接修改生成后的工具配置文件。

目录结构：

- agents/：顶层路由、校验智能体定义
- skills/：13套开发功能套件（通用开发流程套件+DEEPX领域专用套件）
- templates/{en,ko}/.tmpl：中英文指令模板
- templates/fragments/：多仓库复用通用片段
- memory/：跨会话持久化记忆
- knowledge/：结构化SDK参考数据
- instructions/：智能体底层执行指令
- toolsets/：工具配套参考文档

平台配置文件生成

所有工具专属配置均由.deepx/通过dx-agent-gen自动生成，禁止手动编辑生成文件：

```
Bash
# 安装生成工具包
pip install -e .deepx/tools
# 一键生成全平台配置文件
dx-agent-gen generate
# 校验生成文件与源文件无偏差
dx-agent-gen check
```

配套预提交钩子脚本，保障生成文件与源文件同步：

```
Bash
.deepx/tools/scripts/install-hooks.sh # 一次性安装钩子
```

3.3.8 按工具快速上手

3.3.8.1 在dx-all-suite顶层目录使用（自动跨子模块调度）

示例提示词：Compile yolo26n.onnx to DXNN and build a person detection Python app with it

工具	使用方式
Claude Code	打开dx-all-suite目录，直接输入提示词；CLAUDE.md自动路由至dx-compiler编译、dx_app生成应用
GitHub Copilot	Copilot对话输入@dx-suite-builder，后接提示词，自动分发任务
Cursor	打开dx-all-suite目录直接输入提示词，全局alwaysApply规则自动路由
OpenCode	打开根目录输入@dx-suite-builder + 提示词，自动跨子模块调度
Codex CLI	进入根目录直接输入提示词，AGENTS.md自动读取并跨模块执行

3.3.8.2 直接进入子模块目录使用（限定单模块任务）

各子模块专属提示词示例：

子模块	示例提示词
dx-compiler	Convert my yolo26x.pt to ONNX and compile it to DXNN for DX-M1
dx_app	Build a yolo26n person detection app using Python
dx_stream	Build a detection pipeline with RTSP camera and tracking

相关软件工具：

工具	使用方式
Claude Code	进入子模块目录直接输入提示词，CLAUDE.md自动加载模块知识库
GitHub Copilot	Copilot对话输入@dx-app-builder/@dx-stream-builder/@dx-compiler-builder，搭配任务描述
Cursor	打开子模块文件夹直接输入提示词，glob匹配规则自动加载对应套件
OpenCode	子模块目录调用对应智能体@xxx-builder，或斜杠功能套件指令
Codex CLI	子模块目录直接输入提示词，自动读取AGENTS.md，可手动读取对应套件SKILL.md

3.3.9 端到端跨项目业务流程

以下场景覆盖多子模块联动开发；单模块专属流程可查看对应子模块独立开发文档。

场景1：自定义模型转换 + SDK适配 + 结果校验

提示词示例：I have yolo26x-custom.onnx at ./models/ and my inference code at ./inference.py using onnxruntime. Convert it to DXNN and port my code to DEEPX SDK.

完整流程：

- dx-compiler：将自定义ONNX编译为DXNN，自动生成适配配置文件
- dx_app：将原有onnxruntime推理代码迁移至DEEPX SDK
- 自动校验：对比迁移后应用与原始代码推理输出，验证精度一致

场景2：模型编译 + 独立推理应用生成

提示词示例：Compile yolo26n.onnx to DXNN and generate a Python detection app that uses the compiled model

流程：dx-compiler编译模型 → dx_app生成加载DXNN的Python检测程序

场景3：模型编译 + 流媒体GStreamer流水线生成

提示词示例：Compile yolo26n.onnx to DXNN and build a detection streaming pipeline with RTSP output

流程：dx-compiler编译模型 → dx_stream生成带RTSP推流的视频检测流水线

场景4：PPU硬件预处理模型编译 + 检测应用生成

提示词示例：Compile yolo26n.onnx with PPU support and generate a detection app for the PPU model

流程：

- dx-compiler开启PPU硬件加速编译，自动识别YOLO锚框/无锚框类型配置PPU参数
- dx_app生成PPU专用检测程序，硬件完成后处理解码，简化上层代码逻辑

3.3.10 跨项目任务调度

dx-all-suite顶层总调度器自动匹配任务归属子模块，无需手动切换目录：

- dx-runtime：跨项目构建、统一校验流程
- dx_app：独立Python/C++推理应用开发
- dx_stream：GStreamer视频流水线开发
- dx-compiler：模型编译、格式转换流程

提示：无需手动切换子文件夹，顶层目录直接调用@dx-suite-builder即可自动分发至任意子模块。

3.3.11 子项目独立开发文档

每个子模块配套专属智能体开发完整文档，包含全部功能套件、插件清单、实操案例：

子项目	文档路径
dx-runtime	dx-runtime/docs/source/agent_development.md
dx_app	dx_app/docs/source/docs/13_DX-APP_Agent_Driven_Development.md
dx_stream	dx_stream/docs/source/docs/08_DX-STREAM_Agent_Driven_Development.md
dx-compiler	dx-compiler/source/docs/05_DX-COMPILER_Agent_Driven_Development.md

3.3.12 内部参考文档

（面向开发者贡献者，普通用户无需阅读）

用于详解.deepx/知识库底层、生成器流水线、调度框架底层逻辑：

- .deepx/docs/dx-agent-dev-overview.md：全仓库.deepx目录完整拆解
- .deepx/README.md：顶层知识库总索引
- .deepx/docs/skill-architecture.md：三层功能套件架构说明
- .deepx/tools/README.md：dx-agent-gen生成工具使用手册
- .deepx/tools/scripts/README.md：自动化脚本、全流程循环工具说明

3.3.13 输出文件隔离机制

3.3.13.1 默认目录

默认所有智能体生成代码统一存放至目标子项目dx-agent-dev/会话ID/目录，避免覆盖现有正式工程代码；仅用户显式指定时，才会生成至src/正式源码目录。

- 默认隔离目录：dx-agent-dev/<session_id>/（永久生效）
- 正式源码目录src/：仅用户明确要求输出生产代码时使用
- 会话ID格式：年月日-时分秒_智能体类型_模型名_任务类型，智能体类型包含claude、codex、copilot、cursor、opencode
- 会话目录内置文件：
  - README.md：会话元数据、生成文件清单、一键运行脚本

- session.json：机器可读会话配置文件

- git配置：dx_app、dx_stream内dx-agent-dev/目录已加入git忽略清单，不会提交至版本库

3.3.13.2 dx-compiler专属会话目录

编译模块会话目录额外包含：

- calibration_dataset：校准数据集软链接，指向dx_com校准数据目录
- config.json：自动生成DX-COM编译配置，校准路径使用相对路径
- compiler.log：开启日志输出时生成完整编译日志

智能体自动配置校准数据集：校验dx_com/calibration_dataset/目录、自动执行初始化脚本、生成相对路径软链接。

3.3.13.3 顶层跨项目会话统一访问

在dx-all-suite顶层执行跨模块任务（编译+部署）时，各子模块各自生成会话目录，同时在顶层dx-all-suite/dx-agent-dev/创建统一软链接，集中查看所有产出文件：

软链接命名规则：{子模块名}_{会话ID}

3.3.14 会话标记机制

智能体在每次任务首尾输出固定标记，用于自动化测试识别会话起止：

起始标记：[DX-AGENT-DEV: START]

硬性规则：必须作为智能体第一条输出内容，位于所有文本、工具调用、推理逻辑之前；自动化测试缺失该标记会直接判定任务失败，用户要求简化输出也不可省略。

完成标记：[DX-AGENT-DEV: DONE (output-dir: 相对路径)]

位于所有开发工作完成后的最后一行；无任何文件生成时，省略括号内路径参数。

重要说明：DONE标记代表完整交付全部产出：业务代码、执行脚本、配置文件、校验结果；若仅生成需求文档、方案规划，未编写可运行代码，禁止输出DONE标记。

3.4 智能体驱动开发落地案例

dx-agent-dev 完整案例

全部案例均由AI代码智能体仅通过一段自然语言提示词生成完整DEEPX NPU SDK应用；每个案例仓库保存原始提示词、实测性能指标、一键复现脚本、完整构建会话日志。

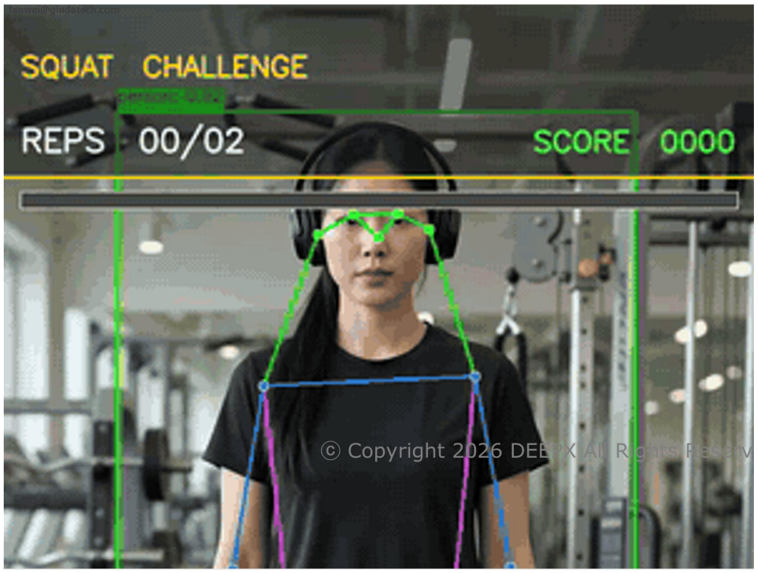
说明

文档内标注的开发耗时、Token输出量、预估成本均为对应会话日志真实统计数据，基于对应智能体官方计费标准；AI代码智能体存在非确定性，相同提示词每次执行的Token消耗、耗时会存在小幅浮动。

3.4.1 NPU体感交互小游戏

深蹲计数小游戏

通过人体关键点识别膝盖、髋关节角度统计深蹲次数，搭配街机可视化界面（计数、得分、动作提示DOWN·UP·GOOD）。



亮点：姿态识别+可视化界面 | Claude Opus 4.8 | 开发约12分钟 | 预估成本7.3美元

拉伸指导小游戏

3套标准拉伸动作，内置动画虚拟教练演示标准姿态，实时检测人体关键点判定动作标准度。

亮点：虚拟教练动画、三阶段动作引导 | Claude Opus 4.8 | 开发约15分钟 | 预估成本8.1美元

3.4.2 Ultralytics YOLO生态适配

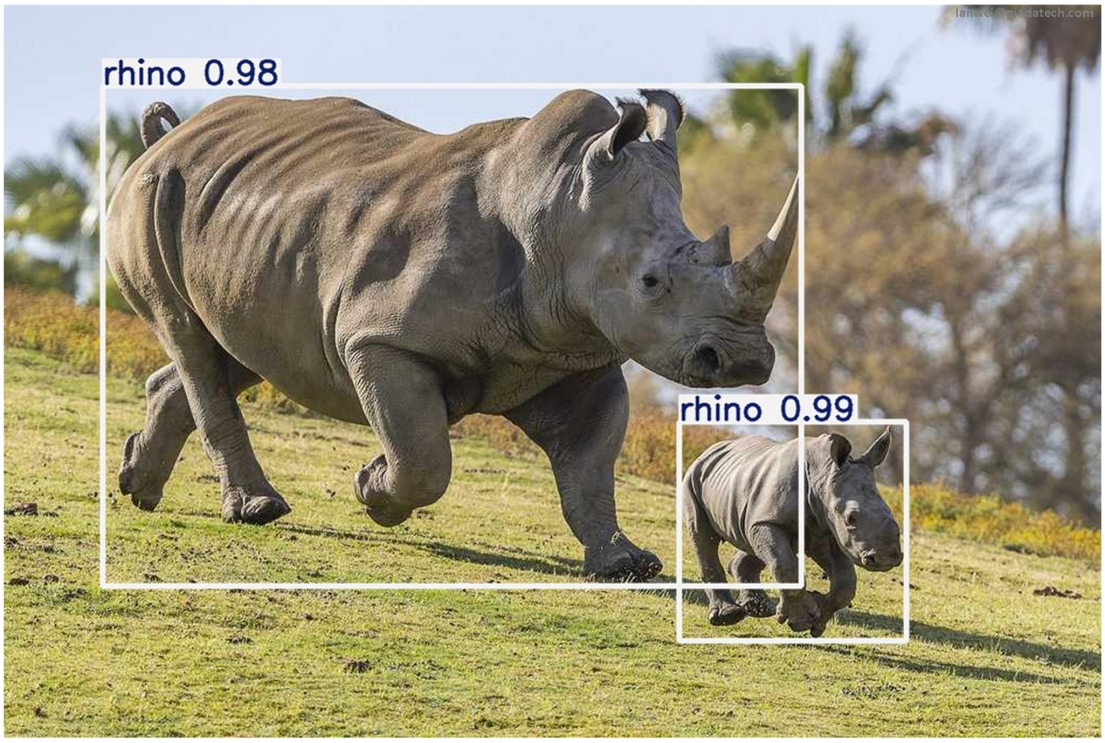
Ultralytics YOLO转DEEPX模型导出

单条yolo导出命令将.pt训练权重转为可部署NPU专用.dxnn模型，自动完成推理精度校验。

亮点：一行命令完成模型迁移 | Claude Sonnet 4.6 | 开发约12分钟 | 预估成本2.4美元

野生动物监测模型

基于yolo26n微调非洲野生动物（水牛、大象、犀牛、斑马）识别，用于野外安防摄像头；四维度评估：原始模型/微调模型 × GPU FP32/NPU INT8。



效果指标：mAP从0.0007提升至0.79，推理帧率59帧/秒提升至80帧/秒 | Claude Opus 4.8 | 开发约7分钟 | 预估成本3.2美元

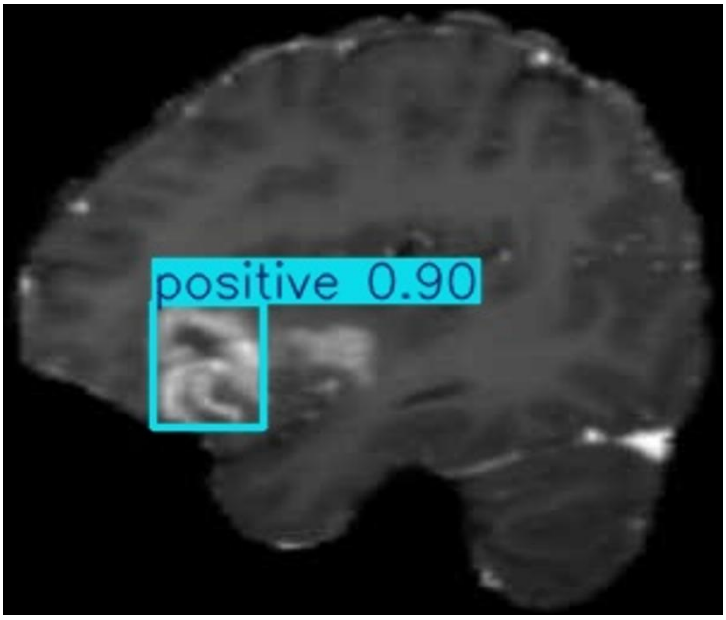
工地劳保安全检测

yolo26n微调工地安全帽、反光背心、人员识别，用于工地安全监控摄像头；四维度精度速度评估。

效果指标：mAP从0.0001提升至0.257，帧率58→80帧/秒 | Claude Opus 4.8 | 开发约17分钟 | 预估成本4.0美元

脑部肿瘤CT/MRI筛查

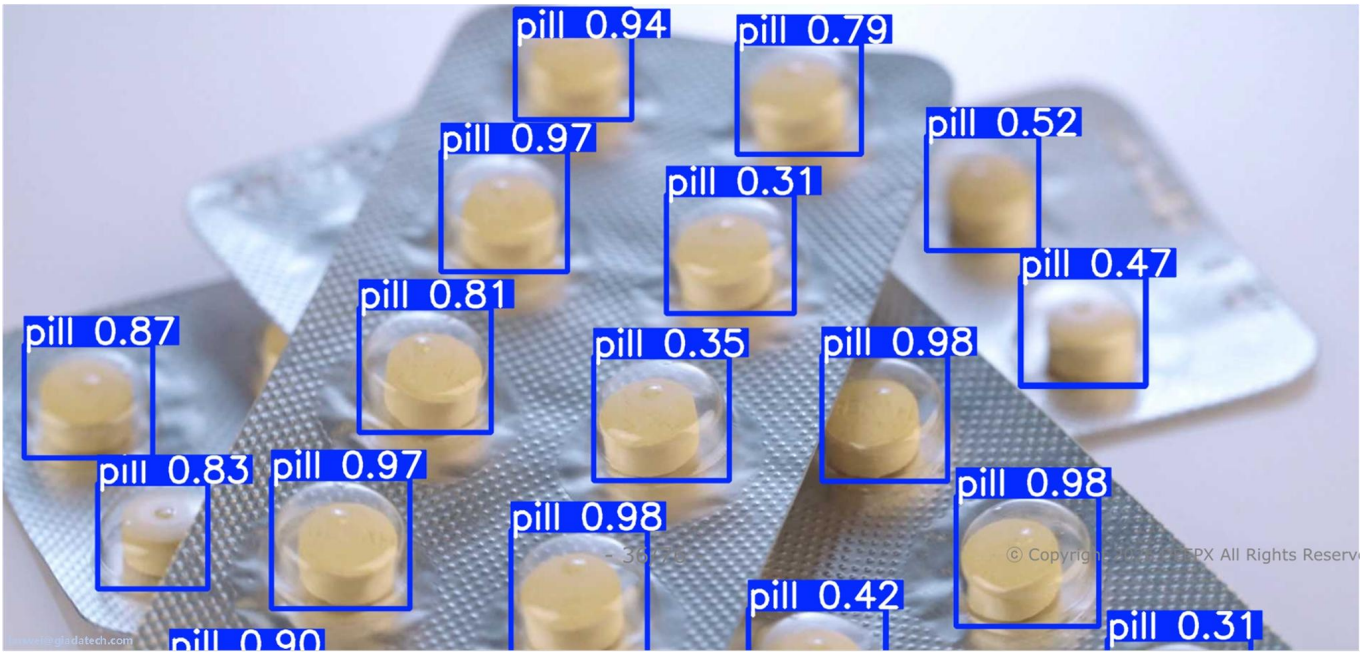
医疗影像病灶检测，边缘医疗设备部署；四维度评估精度速度。



效果指标：mAP从0.0005提升至0.40，帧率59→83帧/秒 | Claude Opus 4.8 | 开发约9分钟 | 预估成本3.7美元

药片质检计数模型

医药铝塑板药片识别，产线自动计数工位；四维度评估。

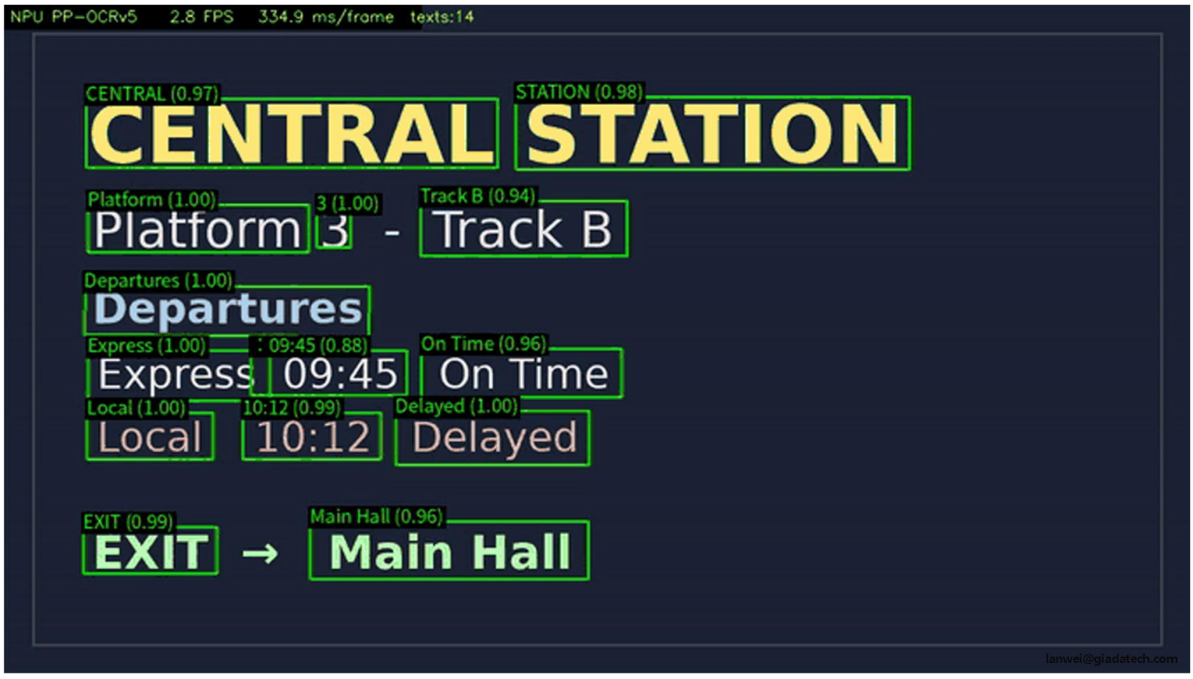


效果指标：mAP从0.001提升至0.75，mAP50达0.97，帧率55→78帧/秒 | Claude Opus 4.8 | 开发约8分钟 | 预估成本5.1美元

3.4.3 PaddlePaddle OCR生态适配

视频/摄像头实时PP-OCRv5文字识别

DEEPX DX-M1 NPU端完成整套文字识别链路：文字检测 → 文字方向校正 → 文字识别；视频文件、实时摄像头共用一套代码，自动叠加识别框与文本。

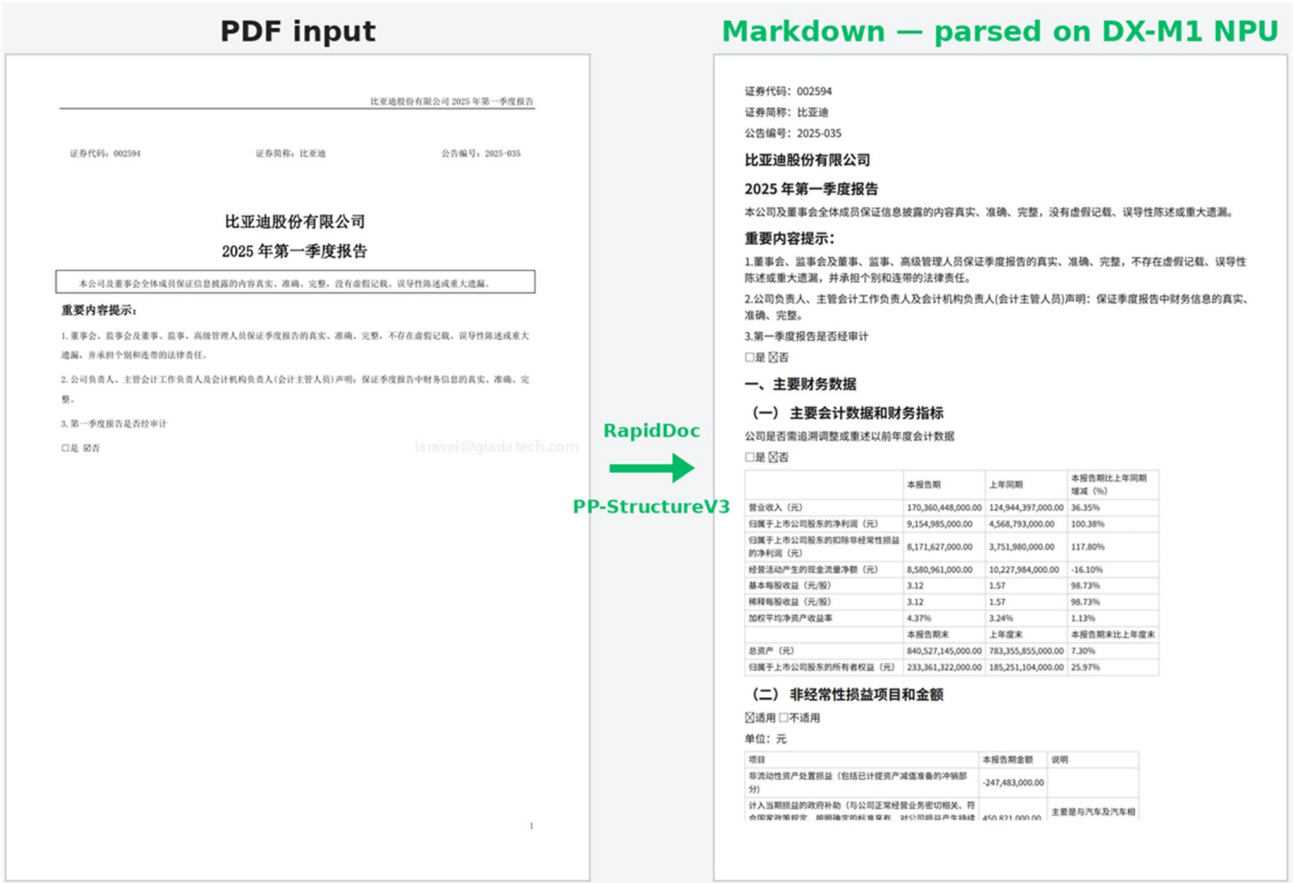


性能指标：单帧最多识别14处文本，推理速度2.8帧/秒，单帧耗时341ms | Claude Opus 4.8 | 开发约18分钟 | 预估成本12.0美元

3.4.4 RapidAI文档解析生态适配

PDF转Markdown结构化解析程序

电子版/扫描版PDF自动解析，输出Markdown+JSON结构化数据；完成版面分析、OCR文字、表格、公式提取，全部在DX-M1 NPU加速。支持解析模式：auto自动识别/txt纯文本/ocr扫描件识别。



效果指标：9页财务报告完整解析，保留21个标题、9张表格；自动模式12.6秒，纯OCR模式14.7秒 | Claude Opus 4.8 | 开发约12分钟 | 预估成本6.2美元

3.4.5 案例一键复现操作

```
Bash
cd dx-agent-dev-showcase/对应案例文件夹
bash setup.sh && bash run.sh
# 体感小游戏额外操作
./setup.sh && ./run.sh --camera 0
# 模型导出/微调案例直接执行脚本
```

前置条件：x86_64 Linux系统 + 完整DEEPX运行时dx_engine；每个案例README存放完整前置依赖、原始提示词，配套韩文README-ko.md。

3.5 DX-AllSuite整体架构总览

本章完整介绍DXNN（DEEPX神经网络）SDK整体架构、核心模块、全平台兼容适配方案。

3.5.1 为什么选择DX-AllSuite?

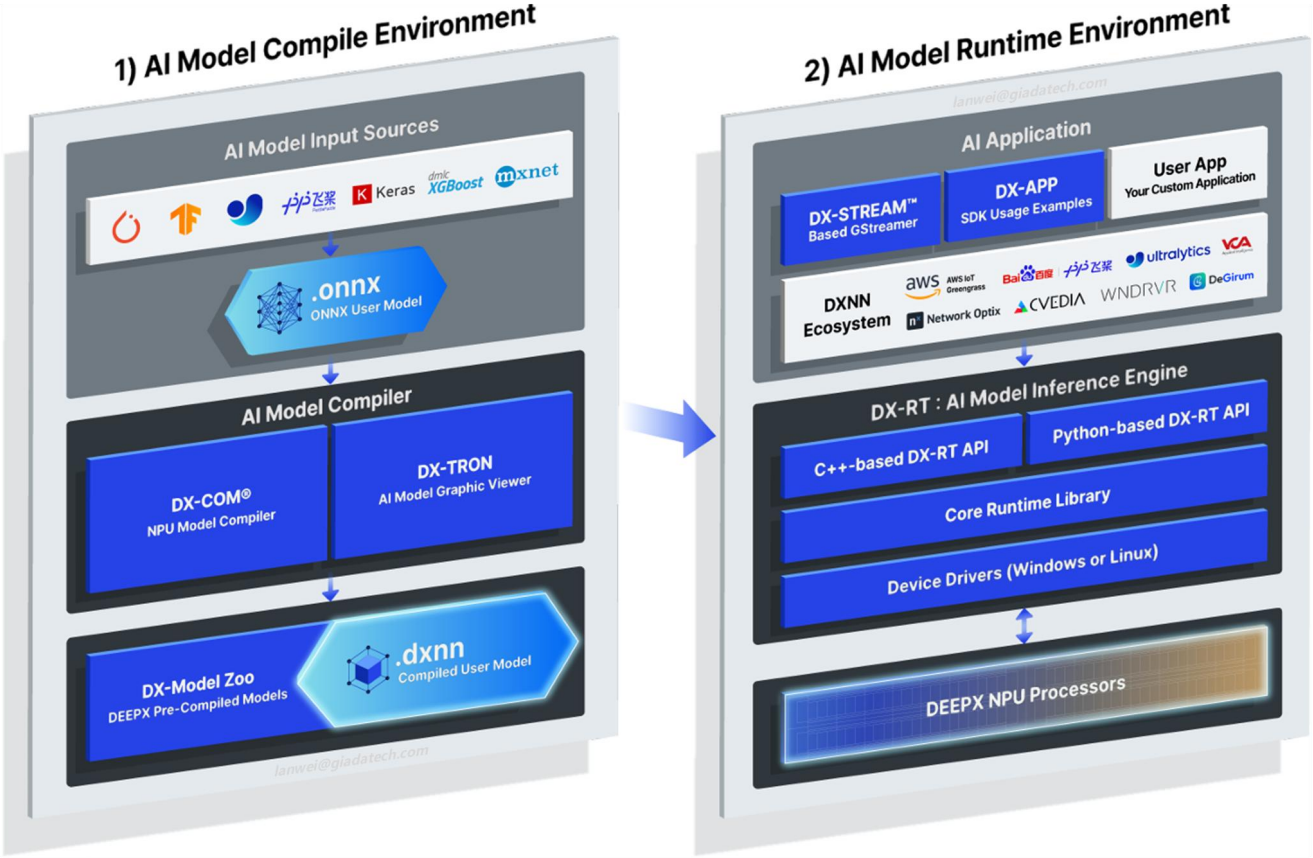
DX-AllSuite将从模型优化到硬件部署的完整链路整合为统一工作流：

① 零代码部署：在桌面主机验证完成的推理逻辑，无需修改代码直接迁移至边缘硬件设备；

② 资源优化：大幅减少环境搭建、系统集成耗时，开发者可专注AI业务逻辑开发；

③ 端到端完整方案：单套件覆盖模型编译、仿真、推理运行、硬件监控全流程。

3.5.2 系统架构与核心模块



3.5.2.1. AI模型编译环境（主机平台）

运行于x86_64 PC主机，负责将训练模型转换、优化为DEEPX NPU私有二进制文件.dxnn。

- DX-COM编译器：核心编译工具，读取ONNX模型与用户配置JSON，通过自研算法将模型转换为硬件最优指令集，量化过程最小化精度损失，实现低延迟、高算力推理；
- DX-TRON可视化工具：GUI模型可视化分析软件，可视化.dxnn模型网络结构，彩色区分NPU/CPU算力负载分布，帮助开发者优化性能；
- DX-ModelZoo官方模型库：预训练硬件适配模型仓库，配套ONNX文件、配置JSON、预编译DXNN二进制，开发者可直接测试部署，跳过编译流程。

3.5.2.2. AI模型运行环境（目标部署平台）

运行优化后的.dxnn模型，对接硬件开发上层应用；同时支持x86_64、aarch64双架构。

- DX-RT推理运行时：对接固件、驱动，在DEEPX NPU执行模型；管理模型加载、IO缓冲区、推理执行、硬件实时监控，提供灵活C/C++、Python双API；
- DX-APP示例程序：基于DX-RT开发的参考工程，覆盖目标检测、人脸识别、图像分类等视觉任务，开发者可基于模板快速自研业务；
- DX-Stream流媒体插件：GStreamer自定义插件，对接实时视频流完成NPU推理；模块化流水线包含预处理、推理、后处理单元，面向智能摄像头等高性能视觉AI场景；
- 驱动与固件：通过PCIe高速总线传输数据，管理NPU硬件调度、功耗；打通软硬件底层，最大化硬件算力、保障推理稳定。

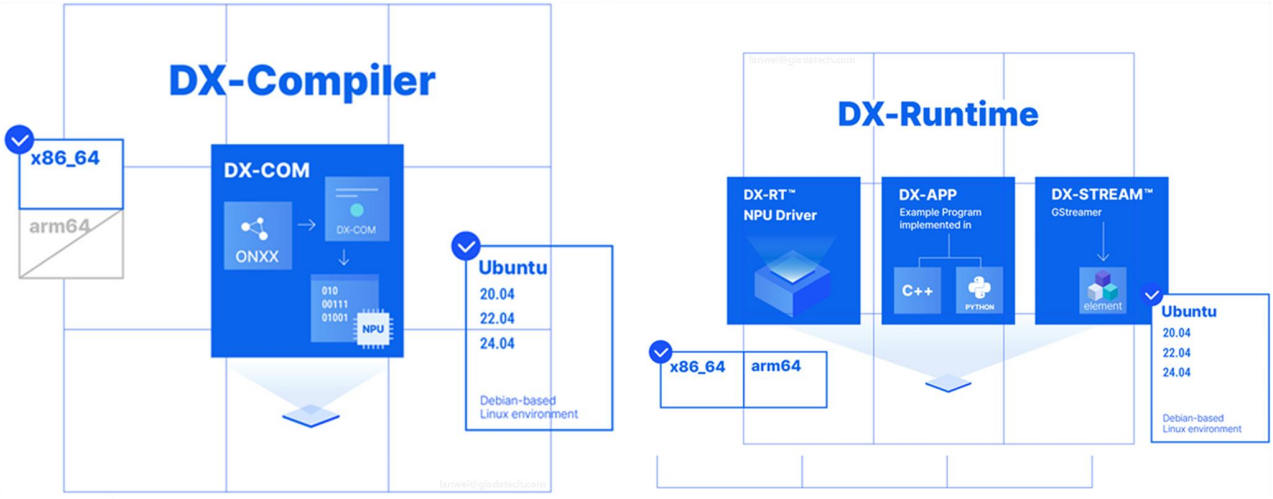
3.5.2.3. 标准开发流程

从训练模型到NPU加速推理分为4步：

① 模型预处理：将PyTorch/TensorFlow训练模型导出为ONNX格式；
② 主机端优化：DX-COM编译，量化、硬件优化生成.dxnn二进制文件；
③ 设备部署：将生成的.dxnn文件传输至目标硬件；
④ 硬件推理：通过DX-RT在DEEPX NPU执行高速推理。

3.5.3 技术兼容适配矩阵

DX-AllSuite适配主流操作系统、硬件架构，打通主机编译与边缘部署全链路，保障稳定开发环境。



3.5.3.1 硬件与操作系统适配

分类	DX-Compiler主机编译环境	DX-Runtime目标部署环境
CPU架构	x86_64	x86_64、aarch64
操作系统	Ubuntu 20.04/22.04/24.04/26.04、Fedora、Redhat、CentOS	Ubuntu 20.04/22.04/24.04/26.04、Debian12/13、Windows10/11
开发语言	Python3.8及以上	Python3.8~3.14；C++14及以上（Windows MSVC推荐C++17）

3.5.3.2 模型与AI生态适配

支持AI训练框架

Ultralytics YOLO、TensorFlow、PyTorch、Keras；统一交换格式为ONNX。

行业生态合作伙伴

- 云平台：AWS IoT Greengrass、百度、DeGirum
- 视觉算法：Ultralytics YOLO全系列、CVEDIA
- 视频管理安防：Network Optix、VCA智能视觉
- 嵌入式实时系统：Wind River VxWorks

3.5.4 DEEPX模型库与支持任务

DEEPX ModelZoo内置345个经过硬件验证的预训练模型，开发者无需手动编译，可直接测试硬件精度、速度。

支持任务与代表模型

① 图像分类：ResNet、ResNeXt、MobileNet、EfficientNet、ViT、MobileViT、VGG等
② 目标检测：YOLO全系列（v3~v11、YOLOX、YOLO26）、SSD、EfficientDet、NanoDet
③ 3D目标检测：专用3D检测模型
④ 实例分割：YOLACT、YOLOv5-Seg、YOLOv8-Seg、YOLO26-Seg
⑤ 语义分割：DeepLabV3/V3+、SegFormer、BiSeNet、UNet
⑥ 全景自动驾驶感知：全景感知专用模型
⑦ 旋转框检测OBB：YOLO26-OBB
⑧ 零样本实例分割：FastSAM
⑨ 人脸检测：RetinaFace、SCRFD、YOLO人脸系列
⑩ 人脸识别：ArcFace、MobileFaceNet
⑪ 人脸关键点：TDDFA v2轻量化版本
⑫ 人脸属性识别：FaceAttrResNetV1-18
⑬ 手部检测、手部关键点：MediaPipeHandsLite

⑭ 人体姿态识别：CenterPose、YOLO26-Pose、YOLOv8-Pose

⑮ 物体姿态估计、关键点检测：专用估计算法

⑯ 行人属性识别、行人重识别：DeepMAR、CasViT

⑰ 深度估计：FastDepth、SCDepthV3

⑱ 图像降噪、低光增强、超分辨率：DnCNN、ZeroDCE、ESPCN(x2/x3/x4)

模型库核心优势

① YAML统一编排：预处理、后处理、评估、编译参数统一写入单份YAML，高可复现、易维护；

② 统一CLI工作流：模型列表、信息查看、精度评估、编译、基准测试整合为一套命令行工具，流程自由组合；

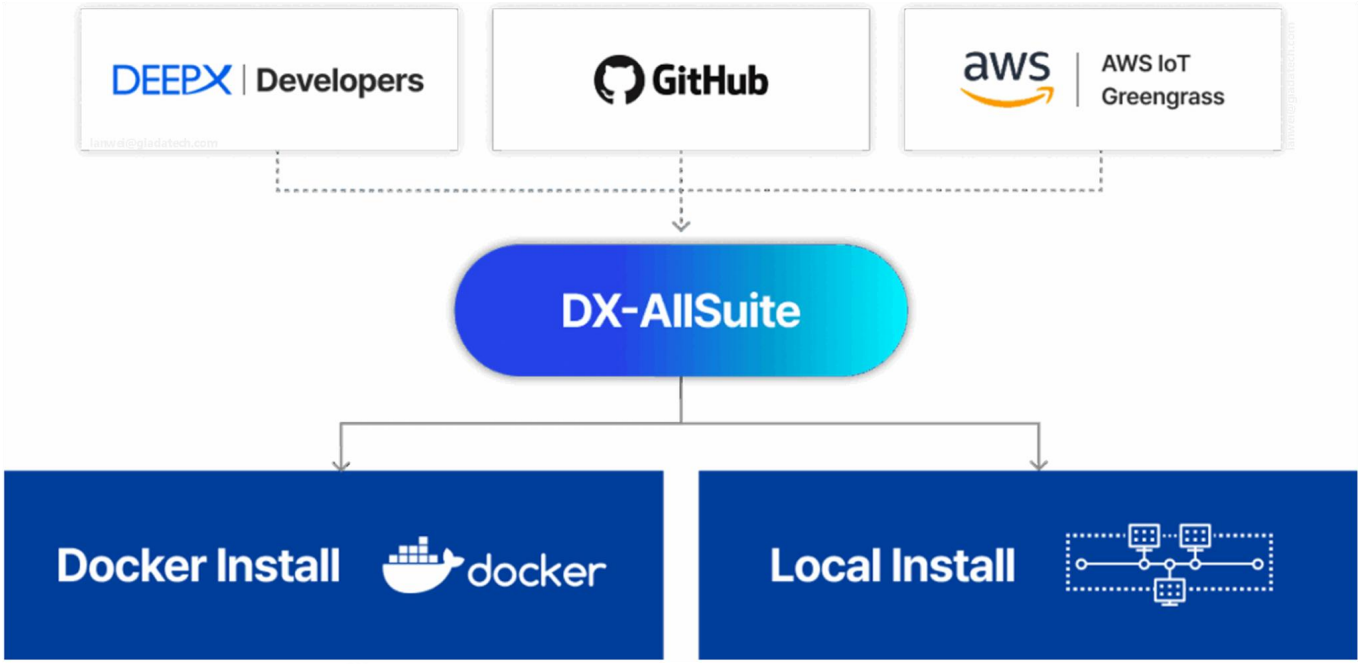
③ 多硬件配置优化：所有模型完成INT8量化、NPU架构适配，从ONNX仿真到硬件推理全程统一校验标准；

④ 可扩展插件架构：预处理、后处理、数据集、评估器支持插件拓展，快速接入自定义模型；

⑤ 全场景视觉任务覆盖：基础分类、检测之外，完整支持人脸分析、旋转框检测、图像画质增强等商用业务模型。

3.6 环境部署

DX-AllSuite是一体化环境搭建工具，统一验证、调用DEEPX硬件，解决复杂依赖冲突，主机、Docker容器环境提供一致开发体验。



安装两种方案

方案A：Docker容器安装（隔离环境）

方案B：本地系统直接安装（生产环境推荐，性能最优）

3.6.1 前置依赖

3.6.1.1 仓库克隆与子模块同步

DX-AllSuite由多个独立子模块组成，子模块缺失会导致编译、推理功能失效，严格执行以下命令：

1、完整克隆仓库（同步所有子模块）

```
Bash
# HTTPS推荐
git clone --recurse-submodules https://github.com/DEEPX-AI/dx-all-suite.git
# SSH
git clone --recurse-submodules git@github.com:DEEPX-AI/dx-all-suite.git
cd dx-all-suite
```

2、已有仓库同步缺失子模块

```
Bash
# 初始化并更新所有子模块至最新版本
git submodule update --init --recursive
# 校验子模块状态（无'-'前缀代表同步成功）
git submodule status
```

3、Docker环境一键安装脚本

未安装Docker时执行脚本自动部署Docker与Docker Compose：

```
Bash
./scripts/install_docker.sh
```

3.6.1.2 自动化虚拟环境管理

DX-AllSuite自动创建独立Python虚拟环境，避免包版本冲突，无需手动创建venv：

- 编译器虚拟环境：dx-compiler/venv-dx-compiler
- 运行时虚拟环境：dx-runtime/venv-dx-runtime

注意：单独安装dx-rt等子模块时，必须手动激活对应虚拟环境，否则安装会失败。

3.6.1.3 SDK两种开发工作流

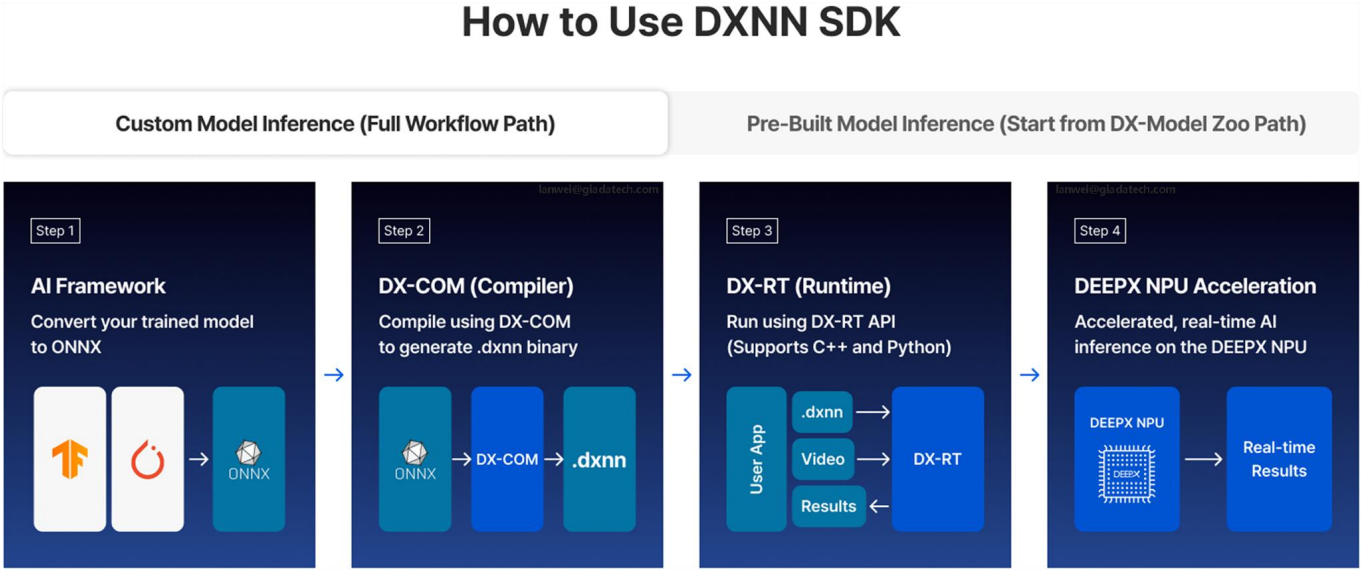
流程A：自定义模型完整开发链路（自有训练模型）

① 训练框架导出：PyTorch/TensorFlow等导出ONNX文件

② DX-COM编译：量化、硬件优化生成.dxnn二进制

③ DX-RT推理：C++/Python API加载模型，读取视频/图片输入，输出推理结果

④ DEEPX NPU硬件加速：实时高速推理输出识别结果



流程B：预编译模型快速验证链路（快速性能测试）

① 从DX-ModelZoo选择预编译.dxnn模型

② DX-RT直接加载模型，无需额外编译

③ NPU硬件加速推理，直接测试FPS、延迟等性能指标

3.6.2 Docker容器安装

Docker提供隔离开发环境，无需处理复杂依赖；关键前置：Docker容器共享主机内核，NPU驱动必须提前安装在主机，容器内无法识别硬件。

3.6.2.1 主机前置操作

1、主机安装NPU驱动

```
Bash
./dx-runtime/install.sh --target=dx_rt_npu_linux_driver
```

2、闭主机dxrtd进程（单例限制，主机/容器仅能运行一个）

```
Bash
sudo systemctl stop dxrt.service
```

3.6.2.2 构建镜像、启动容器

1、构建完整镜像（编译器+运行时+模型库，Ubuntu24.04）

```
Bash
./docker_build.sh --all --ubuntu_version=24.04
# 仅构建运行时镜像
./docker_build.sh --target=dx-runtime --ubuntu_version=24.04
```

2、启动容器

```
Bash
./docker_run.sh --all --ubuntu_version=24.04
```

GUI工具报错提示：Wayland桌面会触发X11挂载告警，参考8.2故障排查章节解决。

3.6.2.3 容器内操作指引

A. DX-Compiler编译容器

1. 进入容器终端

```
Bash
docker exec -it dx-compiler-24.04 bash
# 容器内工作目录
cd /deepx/dx-compiler/dx_com
```

路径区分：/deepx/为容器内绝对路径，主机不存在该目录，注意区分终端环境。

2. 批量编译示例模型

```
Bash
../example/3-compile_sample_models.sh
```

3. 手动单模型编译（激活虚拟环境）

```
Bash
source ../venv-dx-compiler/bin/activate
dxcom -m sample_models/onnx/YOLOV5S-1.onnx -c
sample_models/json/YOLOV5S-1.json -o output/YOLOV5S-1
```

4. 编译产物：output目录生成YOLOV5S-1.dxnn，传输至运行时容器执行推理。

B. DX-Runtime推理容器

1. 进入容器、校验硬件识别

```
Bash
docker exec -it dx-runtime-24.04 bash
# 查看NPU硬件状态
dxrt-cli -s
```

2. 运行dx_app独立推理示例

```
Bash
cd /deepx/dx-runtime/dx_app
# 一键下载模型、素材
./setup.sh
# 运行C++示例
./run_demo.sh
# 运行Python示例
./run_demo_python.sh
```

执行后输入对应序号启动不同检测任务。

3. 运行dx_stream流媒体流水线

```
Bash
cd /deepx/dx-runtime/dx_stream
./setup.sh
./run_demo.sh
```

3.6.2.4 多容器并行运行高级排障

dxrtd进程全局单例，多容器同时启动会冲突，两种解决方式：

1. 修改Dockerfile，关闭容器自动启动dxrtd

```
Dockerfile
# 注释原有自动启动
# ENTRYPOINT [ "/usr/local/bin/dxrtd" ]
# 容器常驻等待
ENTRYPOINT ["tail", "-f", "/dev/null"]
```

2. docker-compose覆盖启动命令

```
YAML
services:
  dx-runtime:
    entrypoint: ["/bin/sh", "-c"]
    command: ["sleep infinity"]
```

容器启动后手动执行进程：

```
Bash
dxrtd &
```

3.6.2.5 Docker安装校验

容器内执行硬件校验命令，满足以下三点即部署成功：

1. 识别硬件：输出Device 0: M1
2. 版本正常：驱动、固件显示有效版本号
3. 无进程冲突报错：无Other instance of dxrtd is running提示

```
Bash
dxrt-cli -s
# 完整环境校验脚本
./dx-runtime/scripts/sanity_check.sh
```

3.6.3 本地系统直接安装

推荐生产环境、性能基准测试使用，硬件兼容性、算力损耗最优。

3.6.3.1 DX-Compiler编译器本地安装（DX-COM、DX-TRON）

前置系统依赖

```
Bash
sudo apt-get update
sudo apt-get install -y --no-install-recommends libgl1-mesa-glx
libglib2.0-0 make
```

一键安装脚本

```
Bash
./dx-compiler/install.sh
```

安装校验、使用方式

```
Bash
# 激活编译器虚拟环境
source ./dx-compiler/venv-dx-compiler/bin/activate
# 查看版本
dxcom --version
python3 -c "import dx_com; print(dx_com.__version__)"
# 查看帮助文档
dxcom -h
```

示例模型素材下载脚本：

```
Bash
# 下载ONNX示例模型
./dx-compiler/example/1-download_sample_models.sh
# 下载量化校准数据集
./dx-compiler/example/2-download_sample_calibration_dataset.sh
```

DX-TRON模型可视化工具

1. 桌面本地启动

```
Bash
dxtron
# 或执行脚本
./dx-compiler/run_dxtron_appimage.sh
```

2. Web远程访问（Docker/无桌面）

```
Bash
./dx-compiler/run_dxtron_web.sh --port=8080
# 浏览器访问 http://localhost:8080
```

Windows系统可从DEEPX开发者门户下载独立安装包。

3.6.3.2 DX-Runtime运行时完整安装（驱动、固件、APP、Stream）

三种安装模式

```
Bash
# 完整安装所有组件（驱动、固件、DX-RT、dx_app、dx_stream）
./dx-runtime/install.sh --all
# 完整安装，跳过固件更新（已有最新固件时使用）
./dx-runtime/install.sh --all --exclude-fw
# 仅安装指定组件
./dx-runtime/install.sh --target=组件名
```

固件更新关键步骤

1. 更新固件

```
Bash
# 脚本更新
./dx-runtime/install.sh --target=dx_fw
# CLI手动指定固件文件更新
dxrt-cli -u ./dx-runtime/dx_fw/m1/X.X.X/mdot2/fw.bin
```

2. 硬件重启（必做）

最优方案：完全关机、断开电源断电冷却后重启；远程服务器可执行系统重启：

```
Bash
sudo reboot
```

3.6.3.3 本地安装完整性校验

```
Bash
# 查看NPU硬件、驱动、固件版本
dxrt-cli -s
# 全模块完整性批量校验
./dx-runtime/scripts/sanity_check.sh
```

校验标准：识别M1硬件、驱动/固件版本正常、无模块缺失FAIL提示。

3.7 运行首个NPU模型

完整6步标准化流程，从模型编译到硬件推理全链路自动化脚本。

3.7.1 完整流程总览

步骤	任务	配套脚本	说明
步骤0	环境校验	compiler-0、runtime-0脚本	校验SDK、NPU驱动安装完整性
步骤1	下载示例模型	compiler-1_download_onnx.sh	获取原始ONNX模型文件
步骤2	准备量化校准数据集	compiler-2_setup_calibration_dataset.sh	量化精度校准素材
步骤3	配置输出路径	compiler-3_setup_output_path.sh	定义DXNN模型存储目录
步骤4	NPU模型编译	compiler-4_model_compile.sh	生成硬件优化.dxnn文件
步骤5	NPU硬件推理运行	runtime-3_run_example_using_dxrt.sh	加载模型推理，输出性能指标

3.7.2 DX-Compiler编译脚本操作（步骤0~4）

执行顺序（getting-started目录内）

```
Bash
./getting-started/compiler-0_install_dx-compiler.sh
./getting-started/compiler-1_download_onnx.sh
./getting-started/compiler-2_setup_calibration_dataset.sh
./getting-started/compiler-3_setup_output_path.sh
./getting-started/compiler-4_model_compile.sh
```

各脚本功能说明

1. compiler-0_install_dx-compiler.sh：安装编译器环境，校验已安装则跳过，--force强制重装
2. compiler-1_download_onnx.sh：下载YOLOV5S、人脸检测、MobileNetV2示例ONNX，创建软链接统一访问
3. compiler-2_setup_calibration_dataset.sh：下载量化校准图片，保障INT8量化精度损失可控
4. compiler-3_setup_output_path.sh：统一逻辑输出路径，主机/Docker环境自动适配物理存储地址
5. compiler-4_model_compile.sh：核心编译步骤，组合ONNX模型、配置JSON、校准数据，生成.dxnn二进制文件，输出至dx-compiler/output目录，通过软链接在getting-started/dxnn直接访问

3.7.3 DX-Runtime推理脚本操作（步骤0、5）

本节重点介绍步骤0（环境设置）和步骤5（推理），在实际的DEEPX NPU硬件上部署和运行优化后的模型时，为确保稳定运行，请在目录中按顺序执行这些脚本

执行顺序

```
Bash
./getting-started/runtime-0_install_dx-runtime.sh
./getting-started/runtime-1_setup_input_path.sh
./getting-started/runtime-2_setup_assets.sh
./getting-started/runtime-3_run_example_using_dxrt.sh
```

脚本功能说明

1. runtime-0_install_dx-runtime.sh：部署运行时环境、校验驱动，--exclude-fw跳过固件更新
2. runtime-1_setup_input_path.sh：同步编译生成的.dxnn模型路径，主机/Docker自动适配
3. runtime-2_setup_assets.sh：下载推理示例图片、视频素材，初始化dx_app、dx_stream资源
4. runtime-3_run_example_using_dxrt.sh：加载模型执行推理，自动运行三类任务并输出平均FPS、单帧延迟：
   - YOLOV5S_Face人脸检测：30轮推理
   - YOLOV5S通用目标检测：300轮推理
   - MobileNetV2图像分类：300轮推理

前置要求：必须完成步骤4编译生成对应dxnn文件，否则会报文件不存在错误；建议完整编译全部三个示例模型再执行推理脚本。

【步骤 0】runtime-0_install_dx-runtime.sh（软件包安装）

搭建用于 DEEPX 神经网络处理器（NPU）控制的核心软件栈。

- 核心功能：安装运行时库，并校验驱动能否正常工作。
- 优化机制：若sanity_check.sh检测到驱动已经正确加载，就跳过重复的安装步骤。
- 省时参数用法：可附加--exclude-fw参数执行脚本，在安装软件栈时跳过固件更新；适合固件已是最新版本的场景。

【步骤 5 前置准备】runtime-1_setup_input_path.sh（模型路径同步）

统一路径映射关系，让运行时引擎能够加载编译阶段生成的.dxnn格式模型。

运行环境	物理路径（实际存储位置）	逻辑路径（程序访问入口）
Docker 容器环境	${DOCKER_VOLUME_PATH}/dxnn	getting-started/dxnn/
本地环境	${DX_AS_PATH}/workspace/dxnn	getting-started/dxnn/

【步骤 5 前置准备】runtime-2_setup_assets.sh（资源准备）

准备推理示例运行所需的依赖文件与样例数据。

- 执行动作：依次调用dx_app、dx_stream两个目录内的setup.sh脚本。
- 注意事项：该脚本仅在各自模块目录内管理资源，不会在forked_dx_app_example文件夹中重复拷贝资源。

【步骤 5】runtime-3_run_example_using_dxrt.sh（运行示例并核验结果）

该脚本将模型加载至 NPU，实测推理性能，针对三类核心任务输出可视化结果：

模型名称	输入数据	对应任务	循环推理次数
YOLOV5S_Face-1	face_sample.jpg	①人脸检测	30 次
YOLOV5S-1	1.jpg	②通用目标检测	300 次
MobileNetV2-1	1.jpeg	③图像分类	300 次

结果输出说明：脚本执行完毕后，会在终端打印平均帧率（FPS，每秒处理帧数）与平均延迟（单位：毫秒）。

3.8 版本兼容说明

DXNN SDK各组件版本强绑定，升级前必须查阅兼容对照表，DX-AllSuite统一维护经过验证的稳定版本组合。

3.8.1 DXNN SDK版本兼容对照表

文档内完整表格记录从2025.07至2026.07所有正式版本，包含DX-AllSuite总版本、DX-COM、DX-TRON、DX-FW固件、NPU驱动、DX-RT、DX-Stream、DX-APP对应配套版本。

DXNN SDK 版本兼容性矩阵如下：

Release Date	DX-AllSuite						
	DX-Compiler		DX-Runtime				
	DX-COM	DX-TRON	DX-FW	NPU Driver	DX-RT	DX-Stream	DX-APP
2026-07-28	v2.4.1						
	v2.4.1		v2.4.1				
	v2.4.0	v2.0.1	v2.7.3	v2.5.1	v3.4.1	v3.1.1	v3.2.1
2026-07-22	v2.4.0						
	v2.4.0		v2.4.0				
	v2.4.0	v2.0.1	v2.7.3	v2.5.1	v3.4.0	v3.1.0	v3.2.0
2026-05-14	v2.3.3						
	v2.3.1		v2.3.3				
	v2.3.0	v2.0.1	v2.5.6	v2.4.1	v3.3.2	v3.0.1	v3.1.1
2026-05-11	v2.3.2						
	v2.3.1		v2.3.2				
	v2.3.0	v2.0.1	v2.5.6	v2.4.1	v3.3.2	v3.0.1	v3.1.1
2026-05-06	v2.3.1						
	v2.3.1		v2.3.1				
	v2.3.0	v2.0.1	v2.5.6	v2.4.1	v3.3.1	v3.0.1	v3.1.1
2026-04-10	v2.3.0						
	v2.3.0		v2.3.0				
	v2.3.0	v2.0.1	v2.5.6	v2.4.1	v3.3.0	v3.0.0	v3.1.0
2026-02-26	v2.2.2						
	v2.2.1		v2.2.2				
	v2.2.1	v2.0.1	v2.5.0	v2.1.0	v3.2.0	V2.2.1	v3.0.2
2026-02-09	v2.2.1						
	v2.2.0		v2.2.1				
	v2.2.0	v2.0.1	v2.5.0	v2.1.0	v3.2.0	V2.2.0	v3.0.1
2026-01-16	v2.2.0						
	v2.2.0		v2.2.0				
	v2.2.0	v2.0.1	v2.5.0	v2.1.0	v3.2.0	V2.2.0	v3.0.0

2025-11-28	v2.1.0						
	v2.1.0		v2.1.0				
	v2.1.0	v2.0.0	v2.4.0	V1.8.0	v3.1.0	V2.1.0	V2.1.0
2025-09-08	v2.0.0						
	v2.0.0		v2.0.0				
	v2.0.0	v2.0.0	v2.1.4	V1.7.1	v3.0.0	V2.0.0	v2.0.0
2025-07-23	V1.0.0						
	V1.0.0		V1.0.0				
	V1.60.1	V0.0.8	v2.1.0	v1.5.0	V2.9.5	V1.7.0	V1.11.0

最新稳定版2026-07-28 v2.4.1全套组件均为v2.4.1配套版本。

3.8.2 查看当前系统各组件版本

为保障系统稳定运行，请先确认当前已安装各组件的版本，再将版本信息和前文给出的版本兼容矩阵进行比对。

步骤 1：硬件与驱动（设备端）

可通过 NPU 状态工具快速查询固件（FW）与NPU 驱动的版本信息：

```
Bash
# 查看NPU运行状态与驱动相关信息
dxrt-cli -s
```

步骤 2：编译器（主机端）

核查模型转换工具dxcom的版本；注意：必须先激活虚拟环境，系统才能识别这条命令。

```
Bash
# 请在已激活的虚拟环境中执行该指令
dxcom -v
```

备选方案（Python 方式查询）

若你以 Python 库的形式调用编译器，也可直接通过 Python 代码查看版本：

```
Bash
python3 -c "import dx_com; print(dx_com.__version__)"
```

术语补充说明

- Target Side（设备端）：搭载 NPU、实际运行推理业务的嵌入式设备
- Host Side（主机端）：用来训练、转换模型的开发电脑

- Virtual Environment（虚拟环境）：Python 隔离运行环境，避免不同工具包版本冲突