第二周 – AI 模型部署概念与容器化初探纪要

一、NGC Catalog 认知

在 NGC Catalog 网站中，包括了 Collections , Containers, Helm Charts, Models, Resources 等内容。在 Containers 目录下，包含用于 AI/ML 元宇宙和高性能计算（HPC）应用的容器，这些容器经过性能优化、测试，可随时部署在基于 GPU 的本地、云和边缘系统上。在 Models 目录下，提供了数百种针对计算机视觉、语音、推荐等领域的预训练模型。通过直接使用这些模型或将少量自定义数据用于快速构建专有模型，可更快地将人工智能推向市场。Helm Charts 可实现 Kubernetes 集群上的软件部署自动化，让用户能够专注于使用软件，而非安装软件。NGC Catalog 提供可直接在 Kubernetes 上运行的 Helm Charts，使部署强大的 NVIDIA 和第三方软件变得轻而易举。在 Resources 目录下，提供了通过 Jupyter Notebooks 提供的分步说明和脚本，适用于各种应用场景，包括机器学习、计算机视觉和对话式人工智能。这些资源能帮助我们更快速地进行分析、理解、定制、测试和构建人工智能，并充分利用最佳实践。

Containers 拥有开箱即用的优化环境，如 l4t-pytorch、l4t-tensorflow 等专为 Jetson 优化的容器。通过预装 CUDA、cuDNN、TensorRT 等 NVIDIA 加速库，避免手动配置环境带来的兼容性问题。因为支持从云端（如 DGX/A100）到边缘（Jetson）的无缝迁移，确保训练和推理环境一致。Models 可以加速 AI 应用落地，数百种涵盖各个领域的预训练模型可直接部署到 Jetson。基于 TAO Toolkit，可用少量数据微调 NGC 模型快速适配边缘场景。

二、ONNX 模型的作用和重要性

ONNX（Open Neural Network Exchange）是一种开放的模型表示标准，用于在不同深度学习框架（如 PyTorch、TensorFlow、MXNet）之间交换模型。它定义了通用的计算图格式，使模型能在不同工具链中无缝迁移。

不同框架的模型格式和 API 差异大，直接转换困难。ONNX 作为中间层：充当“通用语言”，避免复杂的直接转换。专用推理引擎（如 TensorRT）通常不支持直接加载 PyTorch/TF 模型，但支持 ONNX。比如流程为 PyTorch 训练 → 导出 ONNX → TensorRT 优化 → 部署到 NVIDIA GPU。而且，训练框架（如 PyTorch）可能不适合高吞吐、低延迟推理，ONNX 模型可被轻量化运行时加载。

图 2.1：ONNX 模型示例

Netron 是一款开源的神经网络模型可视化工具，支持 ONNX 及其他多种深度学习框架的模型格式。它通过直观的图形界面展示模型的计算图结构，帮助开发者快速理解、调试和优化模型。

三、TensorRT 核心作用

NVIDIA TensorRT 是一个 SDK，用于优化经过训练的深度学习模型以实现高性能推理。在 TensorRT 官方文档中，包含了 Getting Started, Installing TensorRT, Architecture, Inference Library, Performance, API 等目录。

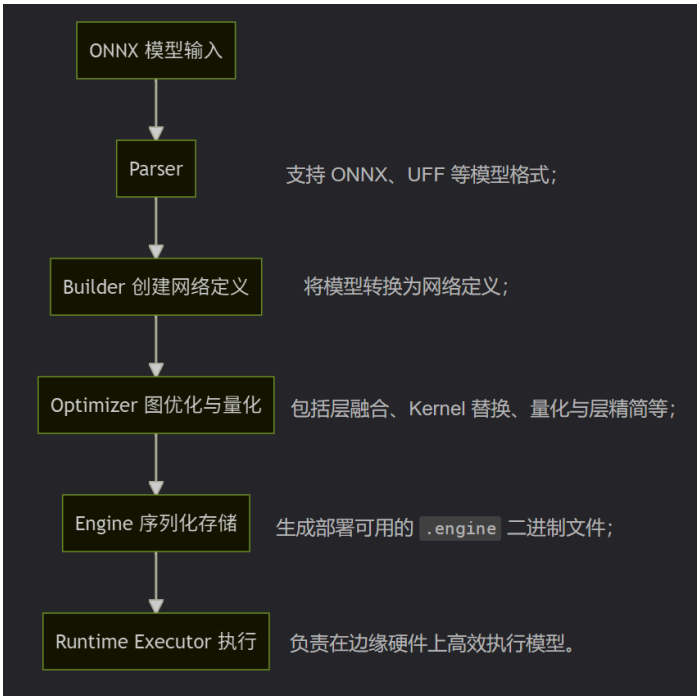

图 3.1：TensorRT 引擎结构组成

TensorRT 接收一个经过训练的网络，该网络包括网络定义和一组训练参数，并生成一个高度优化的运行时引擎，用于对该网络进行推理。TensorRT 通过 C++和 Python 提供 API，帮助通过网络定义 API 表达深度学习模型，或者通过解析器加载预定义模型，使 TensorRT 能够优化并运行这些模型在 NVIDIA GPU 上。TensorRT 进行图优化、层融合等优化操作，同时利用多样化的高度优化的内核找到该模型的最快实现方式。TensorRT 还提供了一个运行时通过它在从 Kepler 一代开始的所有 NVIDIA GPU 上执行此网络。TensorRT 还具备在 TegraTM X1 中引入的可选高速混合精度功能，并在 PascalTM、VoltaTM、TuringTM 和 NVIDIA Ampere 等 GPU 架构的支持下得到了扩展。

在边缘推理部署中，TensorRT 是提升性能的核心引擎。它通过计算图优化、低精度量化和硬件感知内核调优等手段，显著提升深度学习模型的推理效率，使其能在资源受限的边缘环境中高效运行。边缘设备通常面临算力有限、内存紧张、高实时性要求和低功耗约束等挑战。通过减少计算需求和内存带宽的使用，TensorRT 可以显著提高推理性能。它支持多种深度学习框架，如 TensorFlow、PyTorch 等，能够快速将训练好的模型部署到边缘设备上。此外，TensorRT 支持跨框架模型（如 ONNX）转换，简化了从训练到边缘部署的流程，成为边缘 AI 落地的关键工具。

四、Docker 容器化技术

Docker 镜像是一个只读模板，用于创建 Docker 容器。镜像包含了运行某个软件所需的所有文件系统、库、环境变量和配置文件。可以将其类比为传统虚拟机中的“快照”或“模板”，但镜像更加轻量级，因为它只包含必要的文件，而不是整个操作系统。

Docker 容器是镜像的运行实例。容器是轻量级的、独立的、可执行的应用包，包含了运行应用所需的所有东西：代码、运行时、系统工具、系统库和设置。容器可以看作是一个隔离的进程，运行在宿主机的操作系统之上。

表 4.1：镜像与容器的区别

维度	镜像	容器
存在形式	静态模板	动态示例
存储周期	只读分层	可写层+只读层
生命周期	长期存储	临时运行
创建方式	Docker build	Docker run/create
数量关系	1:N	1:1
磁盘占用	较大（包含完整环境）	较小（增量存储）
修改影响	创建新镜像	实时生效

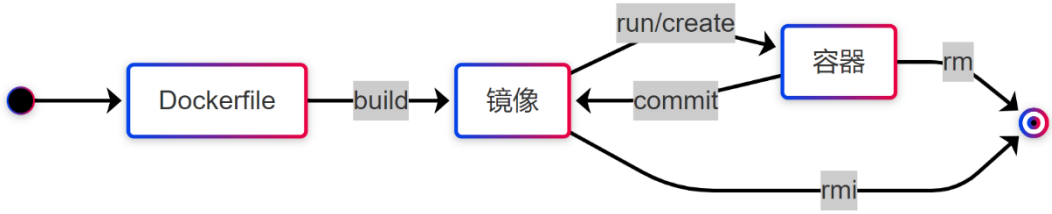

图 4.1：镜像与容器的关系

Docker 技术在软件开发和部署中的核心优势在于通过容器化实现环境一致性和高效资源利用。它将应用及其依赖打包成轻量级、可移植的镜像，确保开发、测试和生产环境完全一致，彻底解决“在我机器上能跑”的难题。容器秒级启动的特性结合 Kubernetes 等编排工具，能够快速扩展实例以应对流量高峰，同时避免传统虚拟机的资源开销。此外，Docker 镜像的分层结构和版本控制简化 CI/CD 流程，支持无缝回滚，而跨平台能力和多云兼容性则赋予部署极大的灵活性。对于微服务架构，Docker 允许每个服务独立容器化，便于拆分和管理。配合丰富的生态工具，开发者能快速搭建复杂环境并实现高效监控，显著提升开发效率和系统可靠性。

五、安装 Docker Desktop for Windows

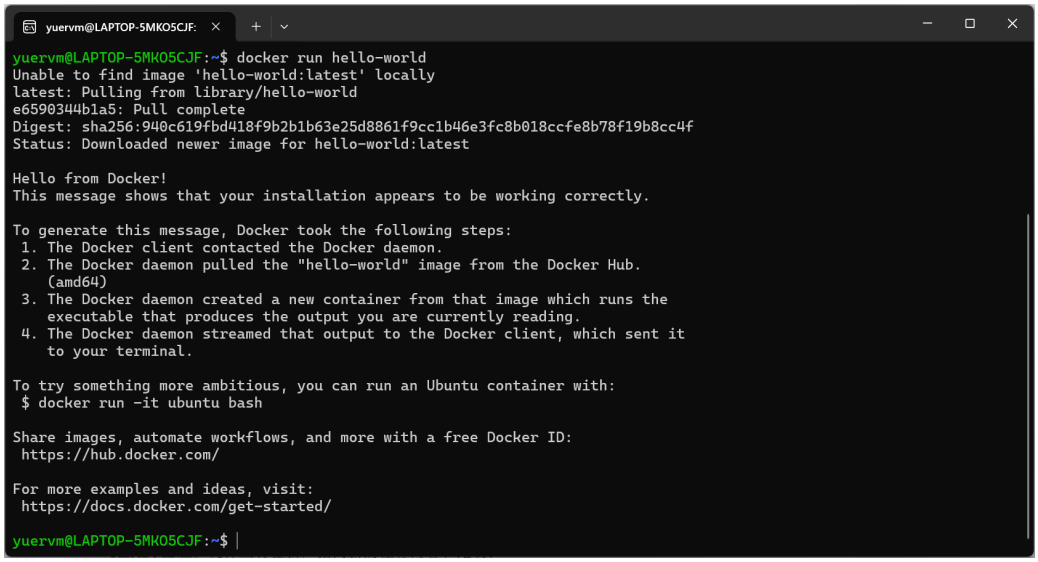

图 5.1：运行 docker run hello-world 命令

在 Docker Engine 中配置镜像站：

```
{
    "registry-mirrors": [
        "https://docker.m.daocloud.io",
        "https://docker.1panel.live",
        "https://hub.rat.dev"
    ]
}
```

图 5.2：查看本地镜像和运行中的容器

表 5.1：Docker 常用命令

操作	命令
拉取镜像	Docker pull image_name
列出镜像	Docker images
列出正在运行的容器	Docker ps
运行容器	Docker run container_name
打包容器	Docker commit container_name new_image_name
启动容器	Docker start container_name
停止容器	Docker stop container_name
推送镜像	Docker login
删除容器	Docker rm container_name
删除镜像	Docker rm image_name

常用 OPTION：

- -d：后台运行容器，例如 docker run -d ubuntu。
- -it：以交互式终端运行容器，例如 docker exec -it container_name bash。
- -t：为镜像指定标签，例如 docker build -t my-image .。

Dockerfile 是一个用来构建镜像的文本文件，包含了构建 Docker 镜像的所有指令。通过定义一系列命令和参数，Dockerfile 指导 Docker 构建一个自定义的镜像。

表 5.2：Dockerfile 基础指令

指令	作用	示例
FROM	指定基础镜像	FROM ubuntu:20.04
LABEL	添加元数据	LABEL maintainer=”your@email.com”
WORKDIR	设置工作目录	WORKDIR /app

表 5.3：环境配置指令

指令	作用	示例
ENV	设置环境变量	ENV NODE_ENV-=production
ARG	定义构建时的变量	ARG VERSION=1.0
COPY	复制文件/目录到镜像中	COPY ./src /app/src
ADD	类似 COPY，支持自动解压和远程 URL	ADD https://example.com/file

表 5.4：运行命令指令

指令	作用	示例
RUN	在构建时执行命令	RUN apt-get update
CMD	指定容器启动时默认执行的命令	CMD [“python”, “app.py”]

表 5.5：网络与存储命令

指令	作用	示例
EXPOSE	声明容器运行时监听的端口	EXPOSE 80
VOLUME	定义数据卷挂载点	VOLUME /data