实习总结与未来规划

第一周我完成了 NVIDIA JetPack SDK 核心组件概念学习和 PC 端开发环境准备。其中，我首次接触了 Linux 系统并学习了 Linux 常用命令。由于 Linux 虚拟机没有物理显卡无法使用 CUDA，我开始了解在无物理 GPU 环境下使用 CUDA 的替代方案：WSL。当开发机为 Windows 系统且配备 NVIDIA 显卡时，可通过 WSL 直接调用物理 GPU 运行 CUDA。但是我在 WSL 中安装了 CUDA 之后无法找到任务要求的 deviceQuery 和 bandwidthTest 示例，我便手动从 Github 中下载了这两个文件所在的 cuda-samples 文件夹。随后，我又遇到了无法执行 make 命令的情况，我找到了在此目录下的 Makefile 文件进行修正，最终成功地运行了这两个示例。

第二周我初步认识了 NVIDIA NGC Catalog 和 TensorRT 的核心概念。在 Docker 的学习过程中，在 WSL 中安装 Docker Engine for Linux 后尝试拉取 hello-world 镜像时遇到了网络问题。这是因为在 Windows 11 家庭版环境下使用 WSL2 运行 Ubuntu 20.04 时，发现 /etc/resolv.conf 中出现了异常的 nameserver 配置：10.255.255.254，而非预期的局域网 DNS 地址：172.x.x.x。这种 DNS 配置异常会导致 WSL 实例无法正常进行域名解析，进而影响网络连接功能。我移除了错误的 DNS 配置文件，将 DNS 设置为 114 DNS 和 阿里 DNS，从而解决了这个问题。在配置国内镜像源时，我起初尝试了最常用的阿里云镜像，使用 ping 命令检查连接良好，但在拉取镜像时却始终无法成功，这说明在 HTTPS 访问环节出了问题。经过多方搜索，我采用了 DaoCloud 和 1Panel 镜像源，成功拉取了 hello-world 镜像，并练习了常用的 Docker 命令。

第三周我进行了 AI 应用的容器化构建与本地推理实践，基于 YOLOv4.onnx 模型实现了物体检测功能。我首先在 Python 脚本中完成了图像输入的预处理，对输入图像进行保持长宽比的缩放（调整为 416×416），通过计算最小缩放比例确保图像不变形，随后将缩放后的图像置于灰色背景中央完成填充，最后进行像素值归一化（0-255→0-1）并添加批处理维度，形成符合 YOLOv4 模型要求的 (1,416,416,3) 张量格式。后处理阶段包含三个核心环节：

1）边界框解码环节通过 sigmoid 和指数运算还原预测框的精确坐标，结合锚点框和特征图步长将输出映射到原图尺度；

2）检测结果精修阶段通过逆向坐标变换将检测框映射回原始图像空间，同时进行边界裁剪和双重阈值过滤（几何尺寸+置信度）；

3）非极大值抑制（NMS）环节按类别剔除冗余检测框，提供标准 NMS 和 Soft-NMS 两种策略，最终输出格式为 [xmin, ymin, xmax, ymax, confidence, class_id] 的标准检测结果。整个流程在保持算法精度的同时实现了从模型原始输出到实用检测结果的完整转换。

最后在部署阶段，针对 Docker 容器环境的特点，我采用了 OpenCV 的无头版本替代标准版，避免了容器中缺少 GUI 系统导致的 cv2.imshow() 报错问题。

第四周我对 NVIDIA TAO Toolkit 和 NVIDIA DeepStream SDK 进行了概念学习。我在配置 NVIDIA Container Toolkit 时，遇到 curl 下载 gpg 密钥时出现“gpg: no valid OpenPGP data found”错误。怀疑是受到了网络防火墙的影响，尝试关闭 Ubuntu 默认防火墙（UFW）却依然无法连接。通过系统化排查验证基础网络连通性：ping 和 DNS 连接正常，但无论是 curl 还是 wget 命令都无法与 nvidia.github.io 建立 SSL 连接，并出现 "Connection reset by peer" 或 "Unable to establish SSL connection" 错误。这表明系统与 NVIDIA 的 GitHub Pages

服务器之间的 HTTPS 连接被阻断了。随后，我又尝试了更新 curl 和 OpenSSL 库，遇到了“does not have a Release file”的问题。经过查看，发现在 nvidia.github.io 默认的 Ubuntu 目录下只有 list 文件且内容为空，却在 amd64 目录下找到了 .deb 包，于是我修改了下载路径并进行下载，安装依赖项之后成功安装了 NVIDIA Container Toolkit。随后，由于连接问题无法使用官方命令下载，我从 GitHub 中手动下载了 tao_tutorials 包，最终完成了 TAO Launcher 的安装。

通过这四周的实践，我对嵌入式 AI 开发建立了初步的认知。最初接触 JetPack SDK 时，我还在适应 Linux 环境和 GPU 加速的基本概念，经历了从虚拟机局限到 WSL2 实战的突破，通过手动修复 CUDA 样本的 Makefile 获得了首次成功。第二周在探索 NGC 和 Docker 时，网络配置问题让我深入理解了 WSL2 的网络栈特性，通过 DNS 优化和镜像源调配掌握了容器化开发的基础。第三周的 YOLOv4 实战让我认识到嵌入式 AI 开发与传统嵌入式的本质差异，比如传统嵌入式 GPIO 输出非 0 即 1， 而 AI 开发中需处理 [x, y, w, h, conf, cls] 的浮点预测结果，引入 NMS 后处理。第四周在突破 NVIDIA 工具链安装障碍的过程中，我形成了系统化的网络问题诊断能力，从 SSL 连接到仓库结构的分析，最终通过手动部署完成 TAO Toolkit 的安装。这些经历让我意识到，嵌入式 AI 开发不再是简单的硬件编程，而是需要融合 Linux 系统管理、容器化部署、模型优化和异构计算的复合能力，这种技术栈的广度和深度，正是其区别于传统嵌入式开发的核心价值所在。

这四周的实践还是浅尝辄止，我当前对嵌入式 AI 开发的认知仍停留在工具链搭建和基础模型部署的层面，缺乏端到端的项目实战经验。接下来我可以考虑尝试做一些轻量级嵌入式 AI 仿真项目，完整覆盖我学习过的 TAO Toolkit 优化模型 → 模型剪枝/量化 → ONNX 转换 → TensorRT 引擎 → DeepStream 全流程。善用 Kaggle 等机器学习平台，在没有本地硬件的情况下进行 AI 开发。