Windows下将QT打包为可执行文件（exe）的完整流程，包含第三方库。

打包我的 Qt/C++ 视觉应用：从依赖部署到单文件 EXE 的踩坑之旅

一、前言

最近完成了一个基于 Qt/C++ 的桌面视觉应用项目（ proj_ai_vision_app ）。这个项目功能还挺复杂，不仅用了 Qt 做界面，还集成了 OpenCV 进行图像处理，并且通过我们自研的 AIEngine 库（它内部又调用了 MNN 推理框架）来实现 AI 功能。开发完成后，接下来的重要一步就是将它打包，以便在没有安装开发环境的其他 Windows 电脑上也能顺利运行。

通常，Windows 应用打包有两种主流方式：

1. **部署依赖**：创建一个包含 EXE 文件和所有必需的 DLL、资源文件的文件夹。用户直接运行这个文件夹里的 EXE。
2. **单文件打包**：将 EXE 和所有依赖项“塞”进一个单独的可执行文件中。用户只需双击这一个文件即可运行。

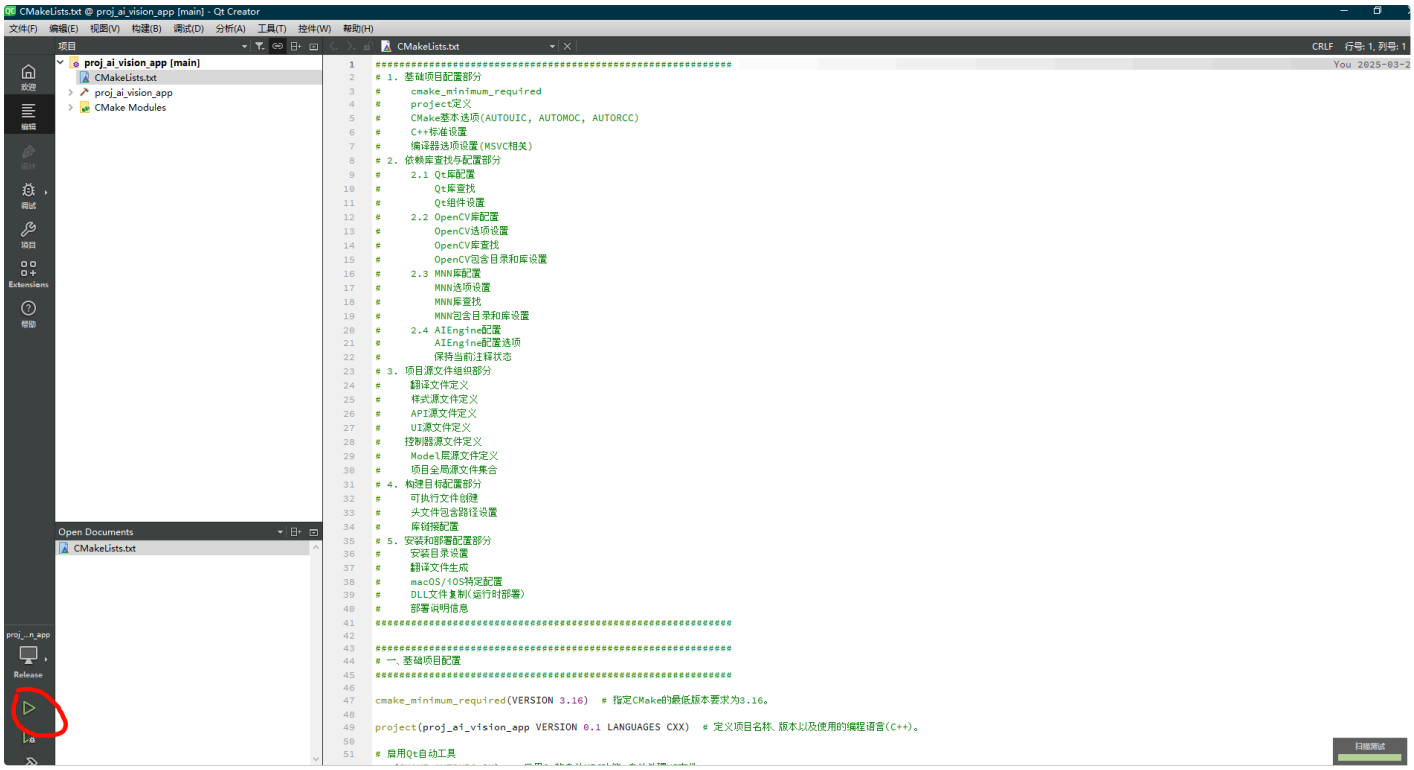
这篇文章就记录一下我为 proj_ai_vision_app 这个项目打包的过程，特别是后面尝试创建单个可执行文件时遇到的一些坑以及最终的解决方案。

二、打包 EXE 文件流程（带依赖文件夹）

这是最基础也是最标准的打包方式，主要依赖 Qt 官方提供的 windeployqt 工具。

1. 构建 Release 版本

首先，在 Qt Creator 中，确保你的项目配置是 "Release" 模式。Release 构建会进行代码优化，生成的可执行文件运行效率更高，体积也可能更小。点击构建按钮，生成 proj_ai_vision_app.exe 文件。



2. 创建部署目录

新建一个干净的文件夹，专门用来存放打包后的所有文件。我这里创建了一个名为 exe 的目录（ D:\DeskTop\QT_Program\exe\ ）。然后，将 Qt Creator 构建生成的 proj_ai_vision_app.exe 文件复制到这个 exe 目录中。

3. 使用 windeployqt 收集 Qt 依赖

windeployqt.exe 是 Qt SDK 自带的一个命令行工具，它的作用是分析你的 EXE 文件，然后自动将运行该 EXE 所需的 Qt 库 DLL、必要的插件（如平台插件 qwindows.dll 、图片格式插件 qjpeg.dll 等）、翻译文件（ .qm ）等复制到 EXE 所在的目录。

搜索windows下的qt终端，记得打开的是带有你编译器环境的终端（我这里使用的是MVSC，所以打开的也是带有MVSC环境的QT终端），切换到你的部署目录（ exe 目录），

然后执行：

```
# 确保 windeployqt 在你的系统 PATH 中，或者使用 Qt 安装目录下的完整路径
```

```
windeployqt .\proj_ai_vision_app.exe
```

执行后，你会看到类似这样的输出，提示它添加了哪些模块和插件：

```
D:\DeskTop\QT_Program\exe\proj_ai_vision_app.exe 64 bit, release executable
Adding in plugin type platforms for module: Qt6Gui
... （省略其他插件和依赖）
Updating Qt6Core.dll.
Updating Qt6Gui.dll.
Updating Qt6Widgets.dll.
...
Creating directory D:/DeskTop/QT_Program/exe/platforms.
Updating qwindows.dll.
...
Creating D:\DeskTop\QT_Program\exe\translations...
Creating qt_zh_CN.qm...
...
```

现在查看 exe 目录，你会发现多了很多 Qt6*.dll 文件，以及 platforms ， imageformats ， styles ， translations 等子文件夹，里面也包含了相应的 .dll 或 .qm 文件。

4. 手动添加非 Qt 依赖

windeployqt 只负责 Qt 自身的依赖。我的项目还依赖了 OpenCV、MNN 和我自己编译的 AIEngine。这些库的 DLL 文件需要我们手动复制到 exe 目录中。

根据我的项目配置（ CMakeLists.txt ），这些 DLL 位于：

- **AIEngine.dll**: lib/AIEngine/proj_ai_engine/Arch/x86/Windows/build/bin/Release/AIEngine.dll
- **MNN.dll**: lib/MNN/bin/MNN.dll
- **OpenCV DLLs**: lib/opencv/x64/vc17/bin/ 目录下的所有 opencv_*.dll 文件。

将这些 DLL 文件全部复制到 exe 目录下，与 proj_ai_vision_app.exe 和 Qt6*.dll 放在一起。

至此，这个 exe 文件夹就包含了运行 proj_ai_vision_app 所需的所有基本文件。理论上，将这个文件夹拷贝到其他 Windows 电脑上（前提是目标电脑安装了对应版本的 Microsoft Visual C++ Redistributable，因为 AIEngine 是用 MSVC 编译的），应该就能运行了。

三、创建单个可执行文件（使用 Enigma Virtual Box）

虽然带依赖文件夹的方式很标准，但有时分发一个单独的 EXE 文件会更方便。我选择了 **Enigma Virtual Box** 这款免费工具来尝试实现这个目标。它的原理是将所有依赖文件虚拟化地打包进主 EXE 中。

1. 下载与安装

前往 Enigma Virtual Box 官网 下载并安装。

2. Enigma Virtual Box 基本配置

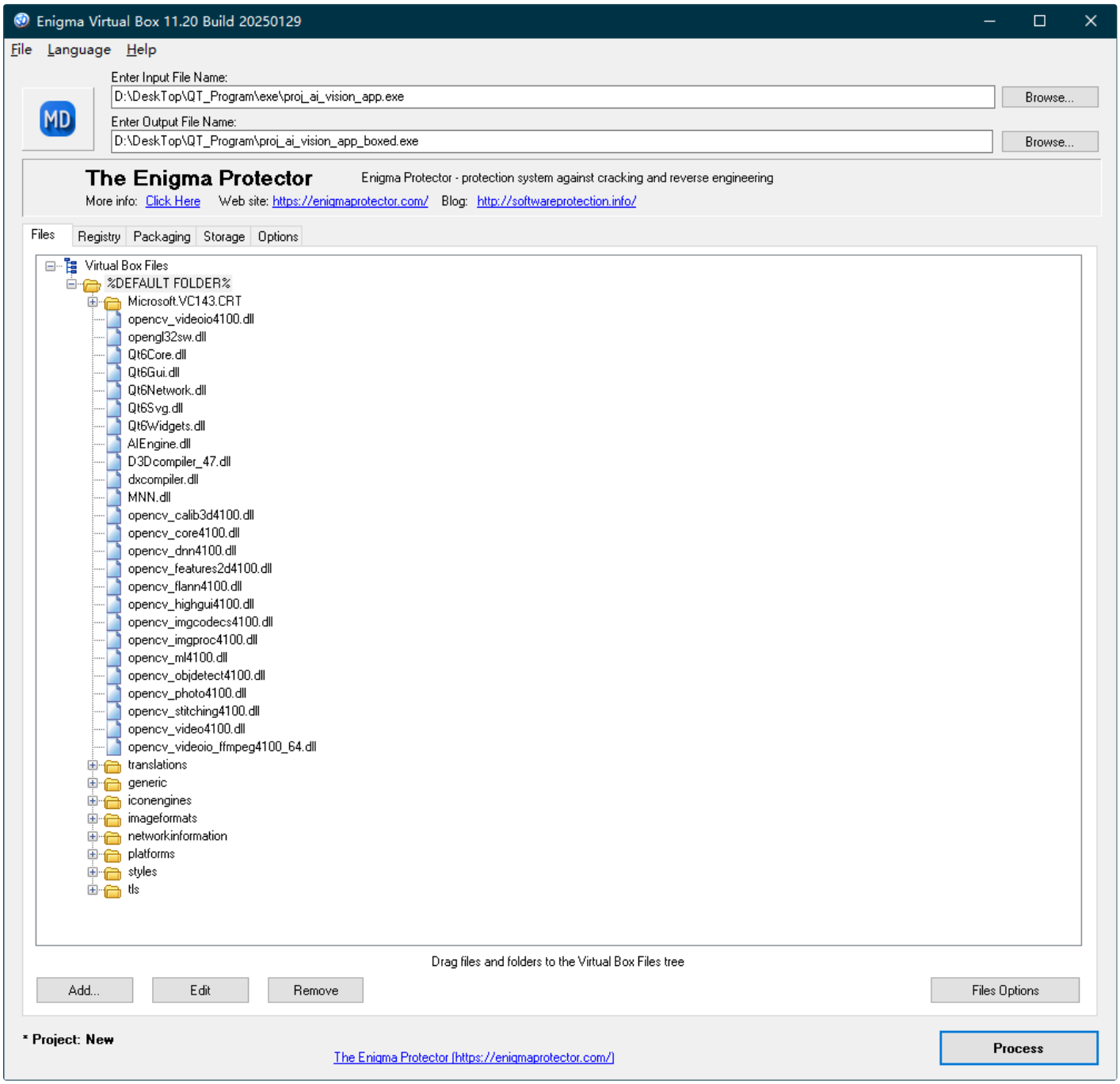
- 打开 Enigma Virtual Box。
- **Enter Input File Name**: 选择我们之前准备好的、位于 **部署目录**（ D:\DeskTop\QT_Program\exe\ ）中的主程序 proj_ai_vision_app.exe 。
- **Enter Output File Name**: 指定打包后生成的单文件名，例如 D:\DeskTop\QT_Program\exe\proj_ai_vision_app_boxed.exe 。

3. 添加虚拟文件（关键步骤与踩坑记录）

这是最关键也是我踩坑最多的地方。目标是把所有依赖项都“告诉”Enigma Virtual Box，让它打包进去。

- **错误尝试 1：直接添加整个部署目录（ exe ）作为子文件夹**
  我最初的想法是，既然所有东西都在 exe 目录里，那我就把这个 exe 文件夹整个添加到 Enigma 的虚拟文件系统 %DEFAULT FOLDER% 下。
  结果： 运行打包后的程序，直接报错 “Cannot load library AIEngine.dll”。
  原因： Enigma 创建的虚拟文件系统 %DEFAULT FOLDER% 对于运行在其中的 proj_ai_vision_app.exe 来说，就是它认为的“当前目录”。Windows 加载器默认只在 EXE 所在的当前目录查找直接依赖的 DLL。我把 DLL 都放到了虚拟的 exe 子文件夹里，程序自然在它认为的当前目录（虚拟根目录）找不到它们。
- **错误尝试 2：添加了错误的源文件（Qt 构建目录）**
  我还尝试过把 Qt Creator 的构建目录（ Desktop_Qt_6_7_3_MSVC2022_64bit-Release ）添加进去。
  结果： 同样失败。
  原因： 构建目录里包含的是编译中间产物，而不是最终部署需要的文件，结构完全不对。
- **关键依赖：VC++ 运行时库**
  反复失败后，我们排查到 AIEngine.dll 是我自己用 Visual Studio 2022 (MSVC) 编译的，它运行时依赖特定版本的 **Microsoft Visual C++ Redistributable** 运行时库。 windeployqt 不会处理这个依赖。这些运行时库（主要是 vcruntime140.dll ， msvcp140.dll ， vcruntime140_1.dll 等）也必须被打包进去！
  查找路径： 这些 DLL 可以在 VS 安装目录下的 VC\Redist\MSVC\<version_number>\x64\Microsoft.VC143.CRT 找到。
  操作： 将这个 Microsoft.VC143.CRT 文件夹下的**所有 DLL 文件**复制出来。
- **正确的做法：重建部署目录结构于虚拟根目录**
  最终的解决方案是，模拟程序运行时所需的真实文件结构，将其“平铺”到 Enigma Virtual Box 的虚拟根目录（ %DEFAULT FOLDER% ）下：
  1. **移除**之前错误添加的 exe 文件夹或构建目录。
  2. **添加 DLL 文件**：将物理部署目录（ D:\DeskTop\QT_Program\exe ）下的**所有 .dll 文件**（包括 AIEngine.dll ， MNN.dll ，所有 opencv_*.dll ，所有 Qt6*.dll ， opengl32sw.dll 等）直接添加到 %DEFAULT FOLDER% 下。

3. **添加 VC++ 运行时 DLL**：将从 VS Redist 目录找到的 `Microsoft.VC143.CRT` 文件夹下的所有 `.dll` 文件也添加到 `%DEFAULT FOLDER%` 下。
4. **添加 Qt 插件文件夹**：将物理部署目录（`D:\DeskTop\QT_Program\exe`）下的 `platforms`，`imageformats`，`styles`，`translations` 等**子文件夹**（使用 "Add Folder Recursively" 或类似选项）添加到 `%DEFAULT FOLDER%` 下，保持它们的子文件夹结构。



最终，`%DEFAULT FOLDER%` 下应该直接包含所有必需的 DLL，以及顶级的 Qt 插件文件夹和 VC++ 运行时 DLLs。

4. 打包与测试

确认 Enigma Virtual Box 的文件列表结构正确后，点击界面右下角的 "Process" 按钮。等待打包完成。然后，将生成的 `proj_ai_vision_app_boxed.exe` 文件复制到一台**没有**安装 Qt、VS 开发环境和相关库的干净 Windows 电脑上，双击运行。

经过上面正确的步骤处理后，这次终于成功运行了！

总结

创建单文件 EXE 对于分发确实更简洁，但需要对程序的运行时依赖有更清晰的认识。使用 Enigma Virtual Box 这类工具的关键在于，在它的虚拟文件系统中正确地重建程序运行时所期望的文件和目录结构。特别是对于自己编译的库，千万不要忘记它依赖的 VC++ 运行时库。同时，程序运行时需要加载的数据文件（如 AI 模型）也必须一并打包进去。这次踩坑经历虽然曲折，但也加深了对 Windows 程序部署和依赖管理的理解。