南京航空航天大学
高性能计算 AI 深度学习平台
使用手册

2024 年 05 月

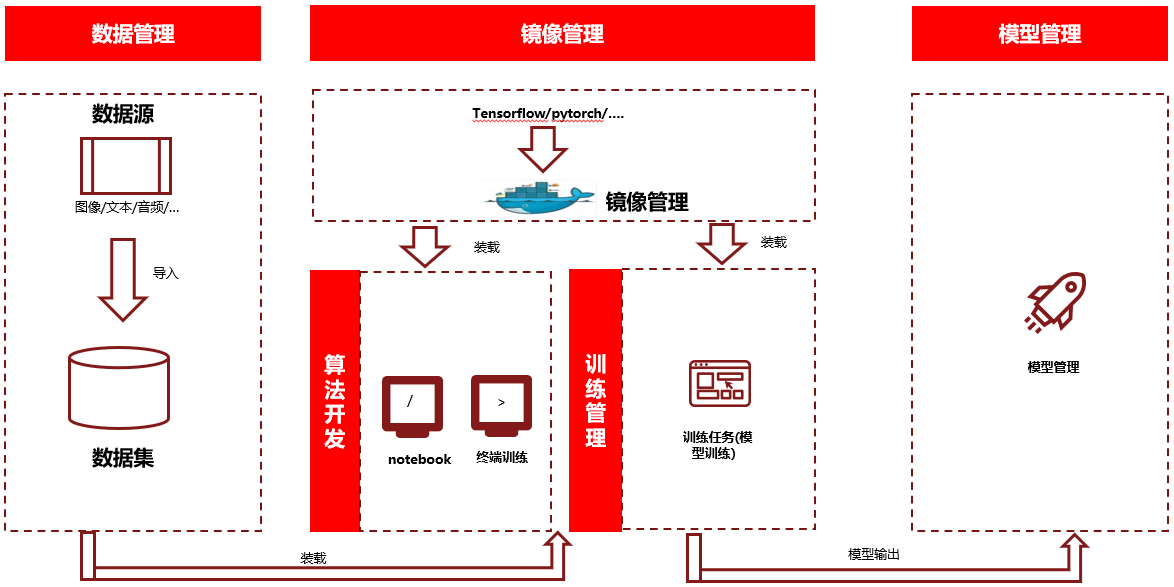
关键名词解释

数据集	数据集，又称为资料集、数据集合或资料集合。数据集是指包含标注的数据集合，可用于图像分类、目标检测、目标跟踪、自然语言处理等特定任务
镜像	将机器学习依赖的环境等借助于容器化(docker)技术进行编译打包成可供深度学习平台进行调度的文件
算法	算法是完成特定任务的步骤的描述,在计算机中表现为指令的有限序列。一般来说，机器学习算法可以分为监督学习、无监督学习、半监督学习、强化学习以及推荐这几大类
模型	模型本质上是一个函数,用以实现从一个样本到样本的标记值的映射
notebook	Jupyter Notebook 是一个能运行 Python 等代码的 Web 应用程序，它是目前进行在线机器学习实践的主流工具
训练任务	算法开发调试完毕后，通过在平台中申请资源，挂载数据集、算法，设置机器学习的其他相关参数，进行机器学习训练，整体训练过程由平台进行托管。
终端训练	通过平台申请计算资源后，启动独立的机器学习环境。让用户使用本地 IDE 工具（如 vscode、python charm 等）通过 SSH 的方式远程连接到启动后的环境进行机器学习实践
Dockerfile	DockerFile 是自定义 docker 镜像的一套规则，包含一系列的指令集合，用于打包 docker 镜像

目录

一、功能介绍

1.1、整体功能和流程说明



平台整体功能分为数据、算法、训练、模型四个部分

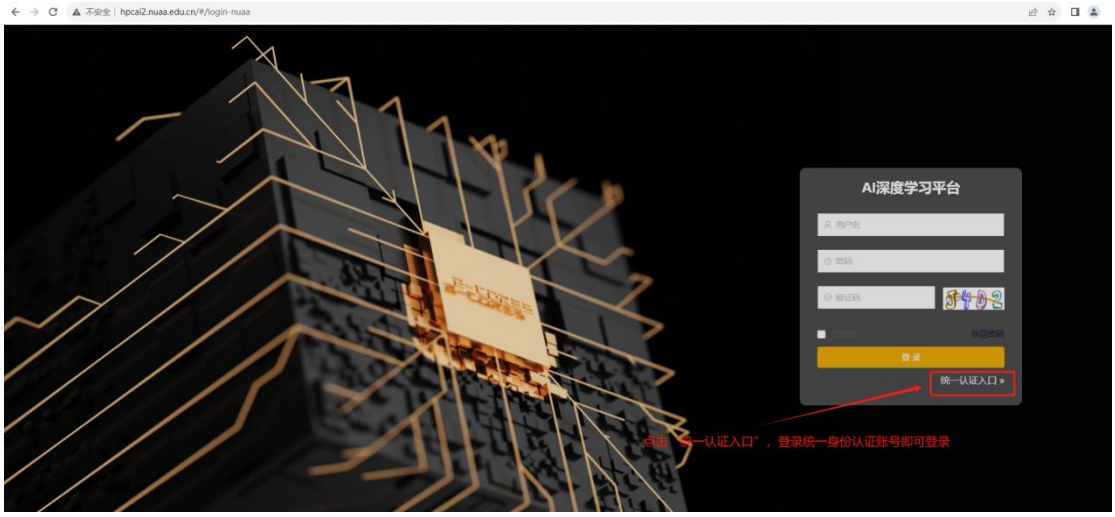
- **数据**：依据机器学习的不同场景，将标注后的图像、文本、音频等不同分类的数据集导入到平台进行统一管理，作为算法开发、训练管理的所需挂载数据集的输入。机器学习依赖的环境等借助于容器化(docker)技术进行编译打包后，通过平台进行上传、统一管理。主要功能包括数据集、镜像导入等
- **算法**：算法开发平台提供 notebook、远程训练两种方式。notebook 功能主要包括任务的查询、停止、算法保存等，提供了 jupyter、vscode、rstudio 开发工具；远程训练功能主要包括终端训练任务的创建、保存与停止、删除以及远程资源使用记录等

- **训练：**算法开发调试完毕后，通过在平台中申请资源，挂载数据集、算法，设置机器学习的其他相关参数，进行模型训练。主要功能包括训练任务的创建、修改、查询、日志查看、实时监控、模型保存等
- **模型：**管理训练过的模型，功能主要包括查询等

1.2、功能模块说明

1.2.1、平台登录

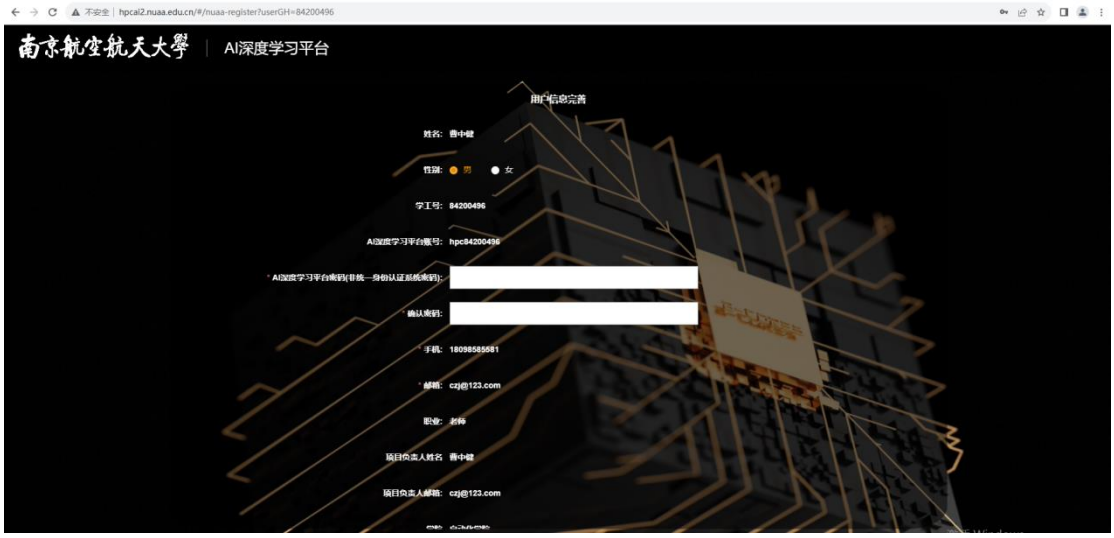
平台登录地址：http://hpcai2.nuaa.edu.cn
打开浏览器，在地址栏中输入平台登录访问地址，进入系统登录界面入。下图所示：

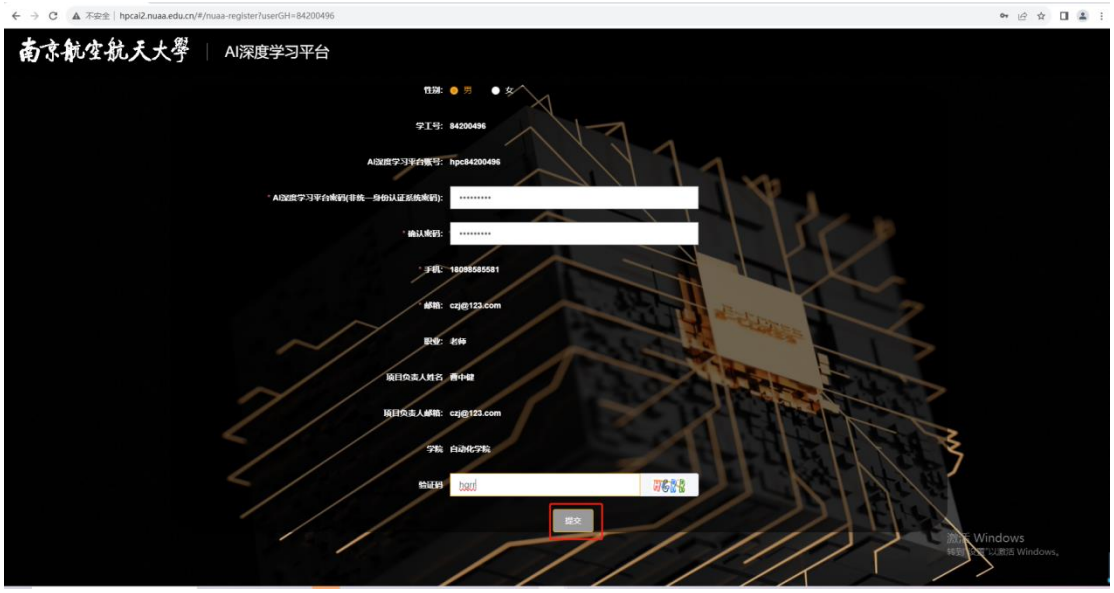


点击统一认证入口进入统一身份认证页面，如下所示：

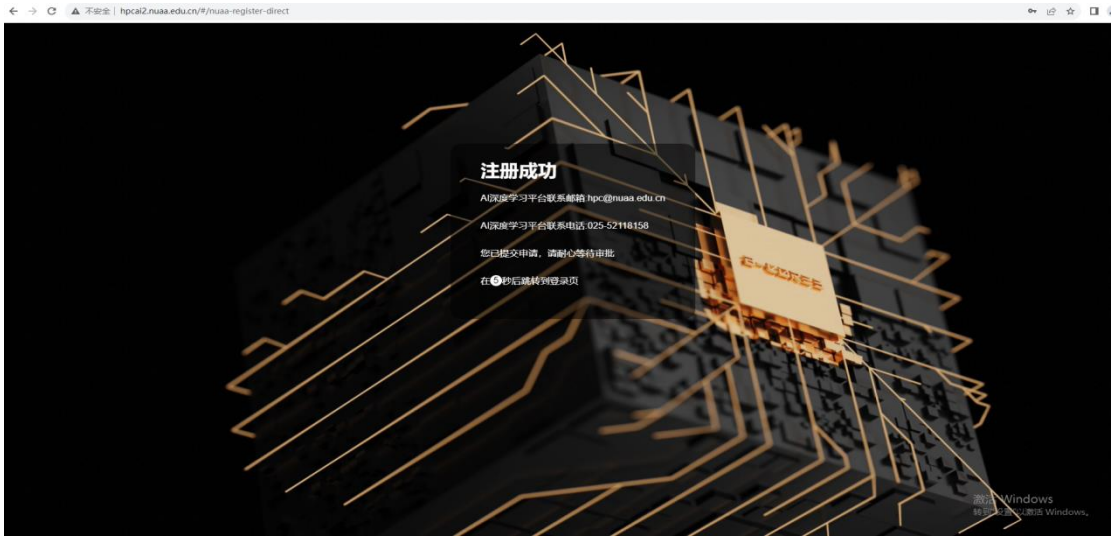
输入账号密码后点击登录，若该账号未在 ai 深度学习平台注册使用且未在学校申请使用平台流程，如下所示：

输入账号密码后点击登录，若该账号在学校申请使用平台流程，如下所示：

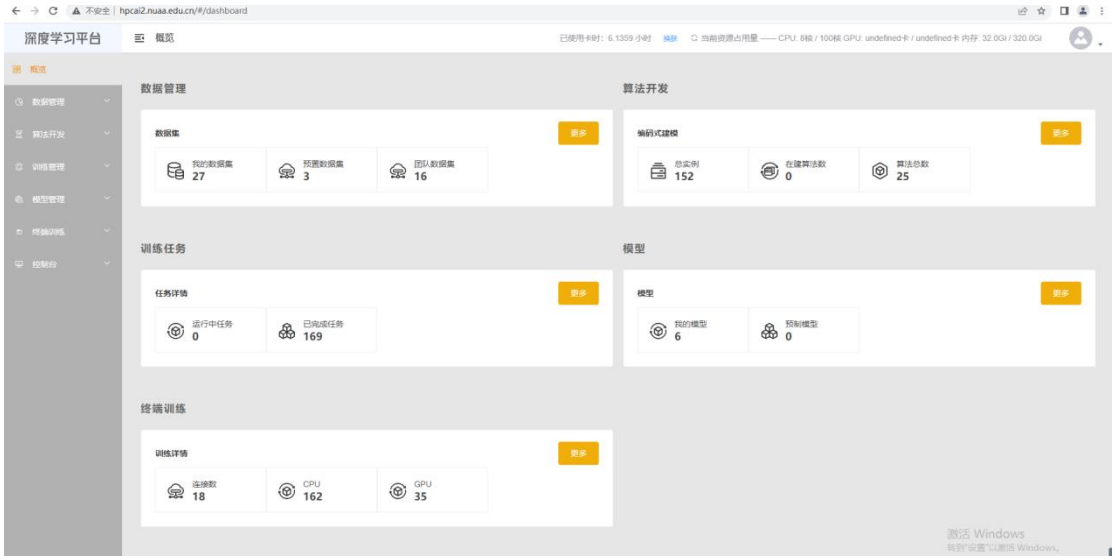




用户补全信息，选择性别、输入密码、确认密码、验证码后点击提交，等待管理员审核，如下所示：



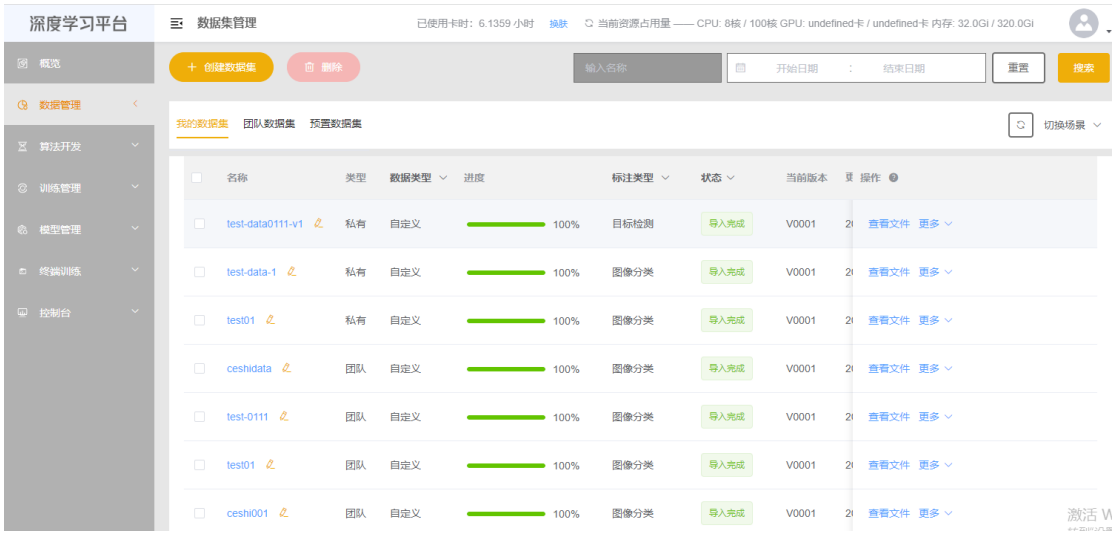
输入账号密码后点击登录成功之后跳转进入平台首页，如下所示：



1.2.2、数据

1.2.2.1、数据集管理

点击菜单“数据管理->数据集管理”，打开数据集列表页面



数据集列表

数据集包括：我的数据集、团队数据集、预置数据集

- 我的数据集：当前登录用户自行上传的数据集
- 团队数据集：当前用户归属的组成员共享的数据集
- 预置数据集：由平台管理员上传的对所有平台所有使用者可以共享使用的数据集

在当前数据集列表页面，可以进行功能操作的说明

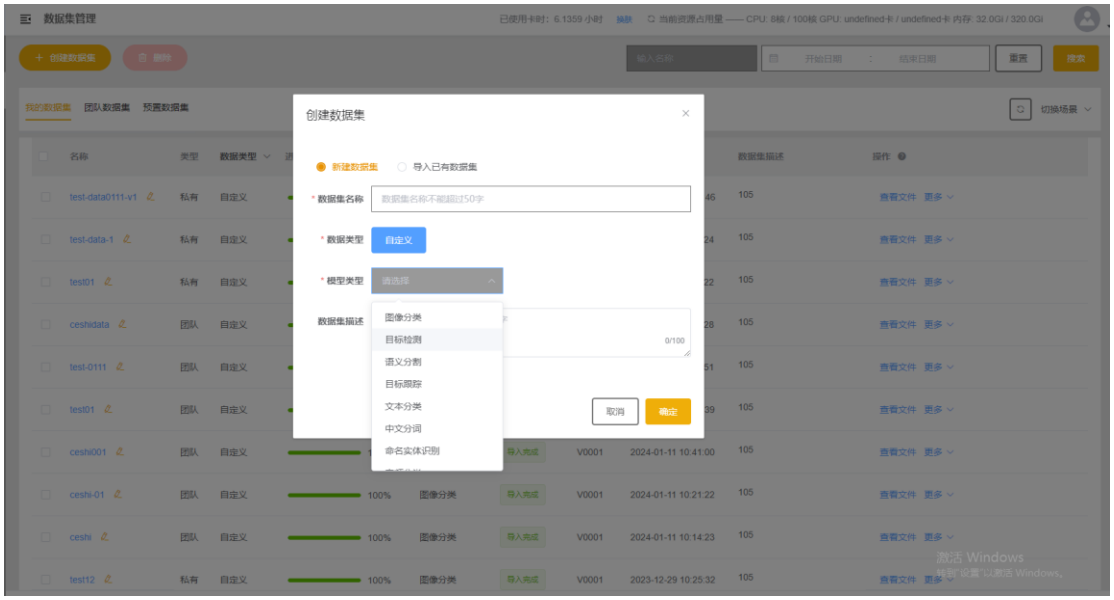
操作按钮或链接	功能说明
创建数据集	新建一个数据集的功能入口
删除	选择一个或者多个数据集，点击该按钮进行删除
查看文件	点击后页面跳转查看该数据集下的文件目录或者文件
历史版本	在平台中对数据集的信息修改、新增、删除文件均会生成一个数据集的新版本
置顶	将常用的数据集排序到最前面
修改	调整数据集的名称和描述
共享	一个组内的成员的自己的数据集实现组内共享，让其他组成员也能使用

1.2.2.1.1、创建数据集

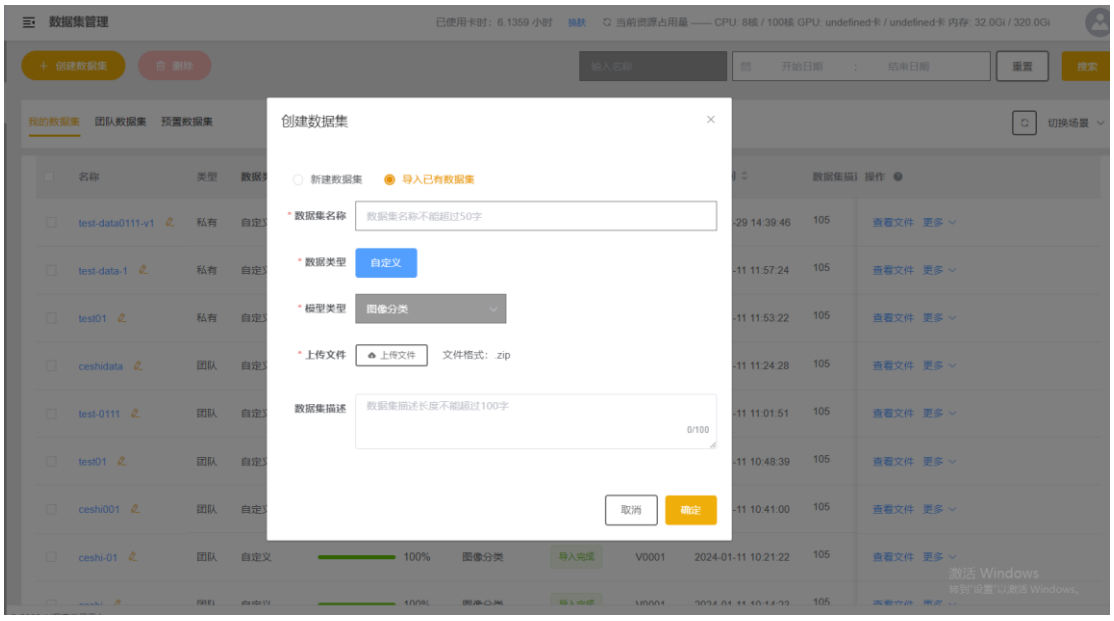
点击“创建数据集”按钮弹出数据集的信息增加框。数据集的创建分为两种方式：普通模式和导入模式。数据集的模型分类：包括图像分类、目标检测、语义分割、中文分词等

普通模式：定义数据集名称、数据类型、模型类型等基本信息，数据集包含的文件依次逐个上传

导入模式：标注好的数据集，按照目录结构进行压缩打包，通过导入压缩包的方式导入到平台，平台会自动进行解压处理，维护导入的数据集



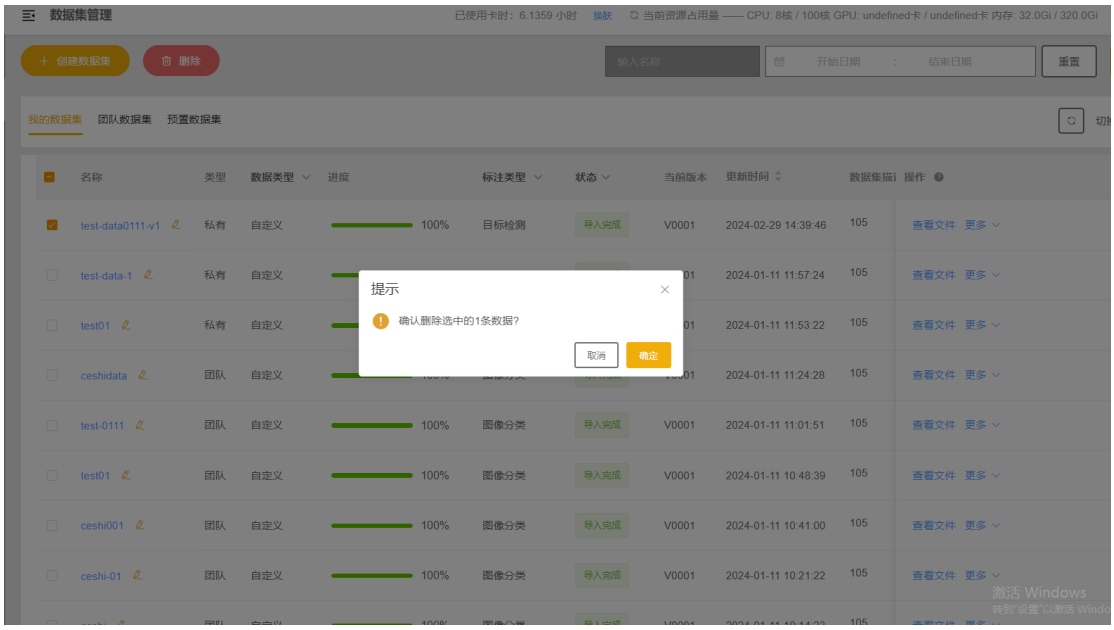
创建数据集—普通模式



创建数据集—导入模式

1.2.2.1.2、删除数据集

选择一个或者多个数据集，点击“删除”按钮，弹出删除确认框，确认后，完成数据集的删除

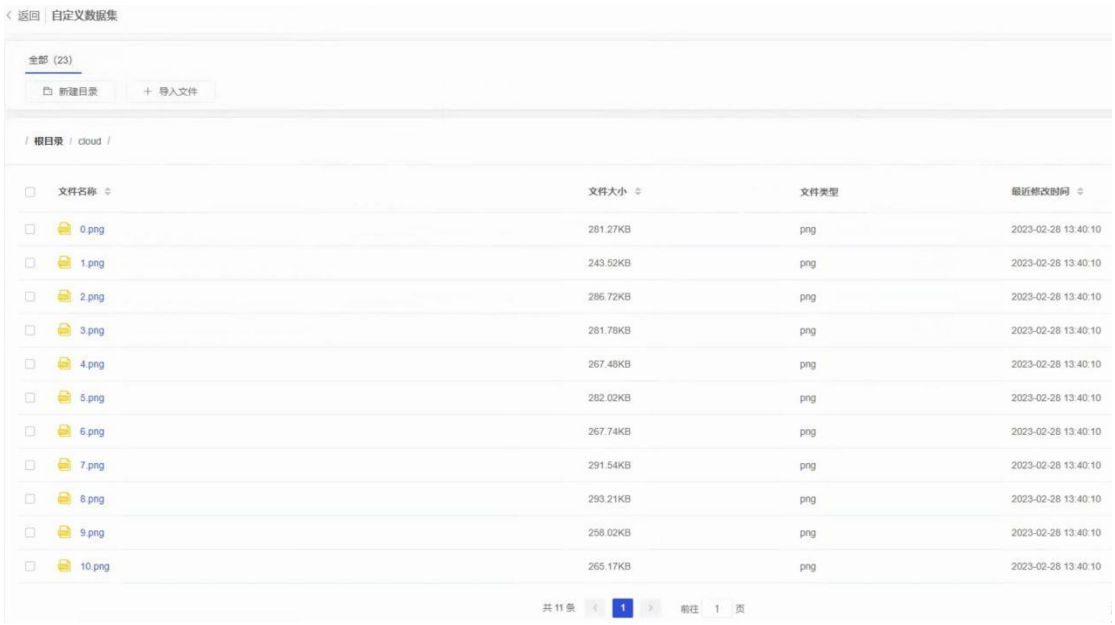


删除数据集

1.2.2.1.3、查看文件

点击”查看文件“查看该数据集下的所有标注的数据集文件

查看文件——数据集目录

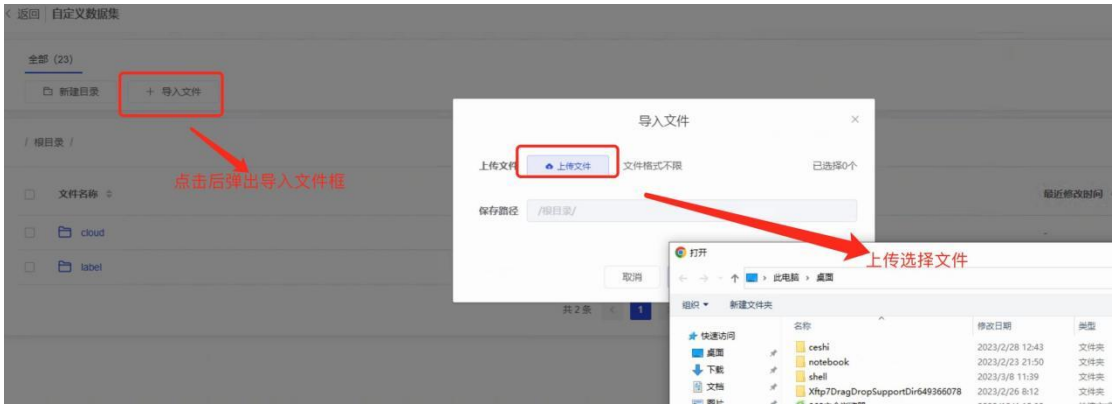


查看文件——数据集文件列表

在数据集的列表页面上，操作包括新建目录、导入文件

新建目录：在当前数据集目录下新建子文件夹

导入文件：在当前目录下可以通过导入的方式上传一个或者多个数据集文件



数据集管理——导入文件

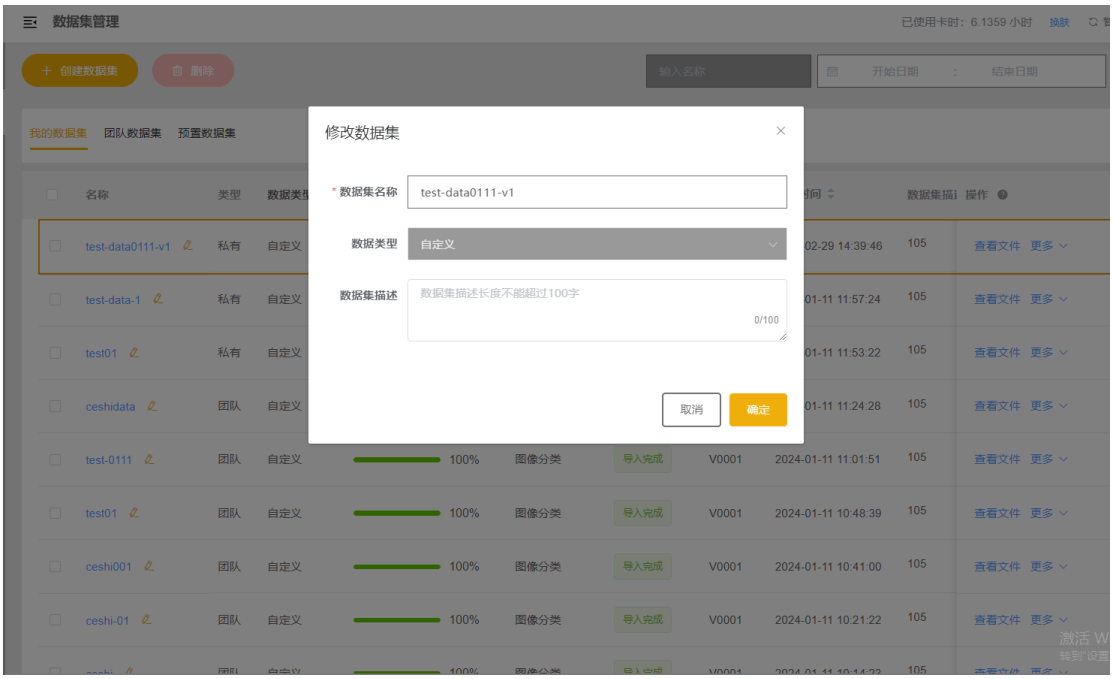
1.2.2.1.4、历史版本

点击列表中的“历史版本”进入到数据集版本管理列表页面。在列表中可以查看该数据集对应的以往的所有版本；

数据集版本管理

1.2.2.1.5、修改数据集

在数据集列表页面，点击“修改”弹出数据集的基本信息编辑框，调整数据集名称和描述，点击“确定”完成数据集信息的修改



数据集修改

1.2.2.1.6、置顶数据集

在数据集列表页面，点击“置顶”完成该数据集的顶部排序展示功能

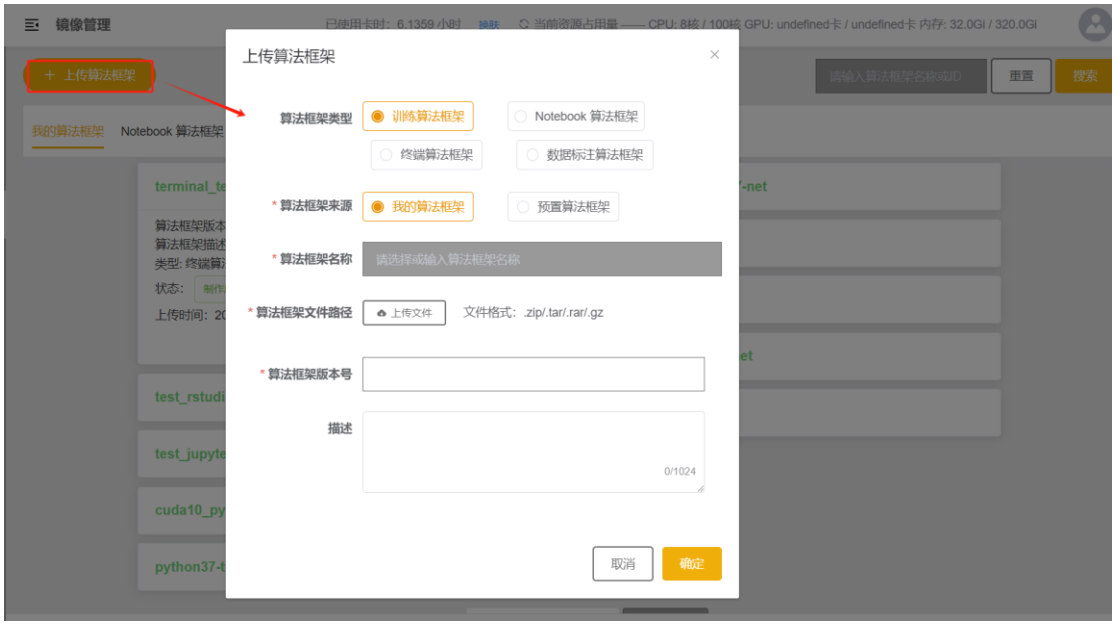
1.2.2.1.7、共享数据集

在数据集列表页面，点击“共享”确认后，实现该数据集在当前用户所在的组内成员间共享使用

1.2.2.2、镜像管理

1.2.2.2.1、上传算法框架

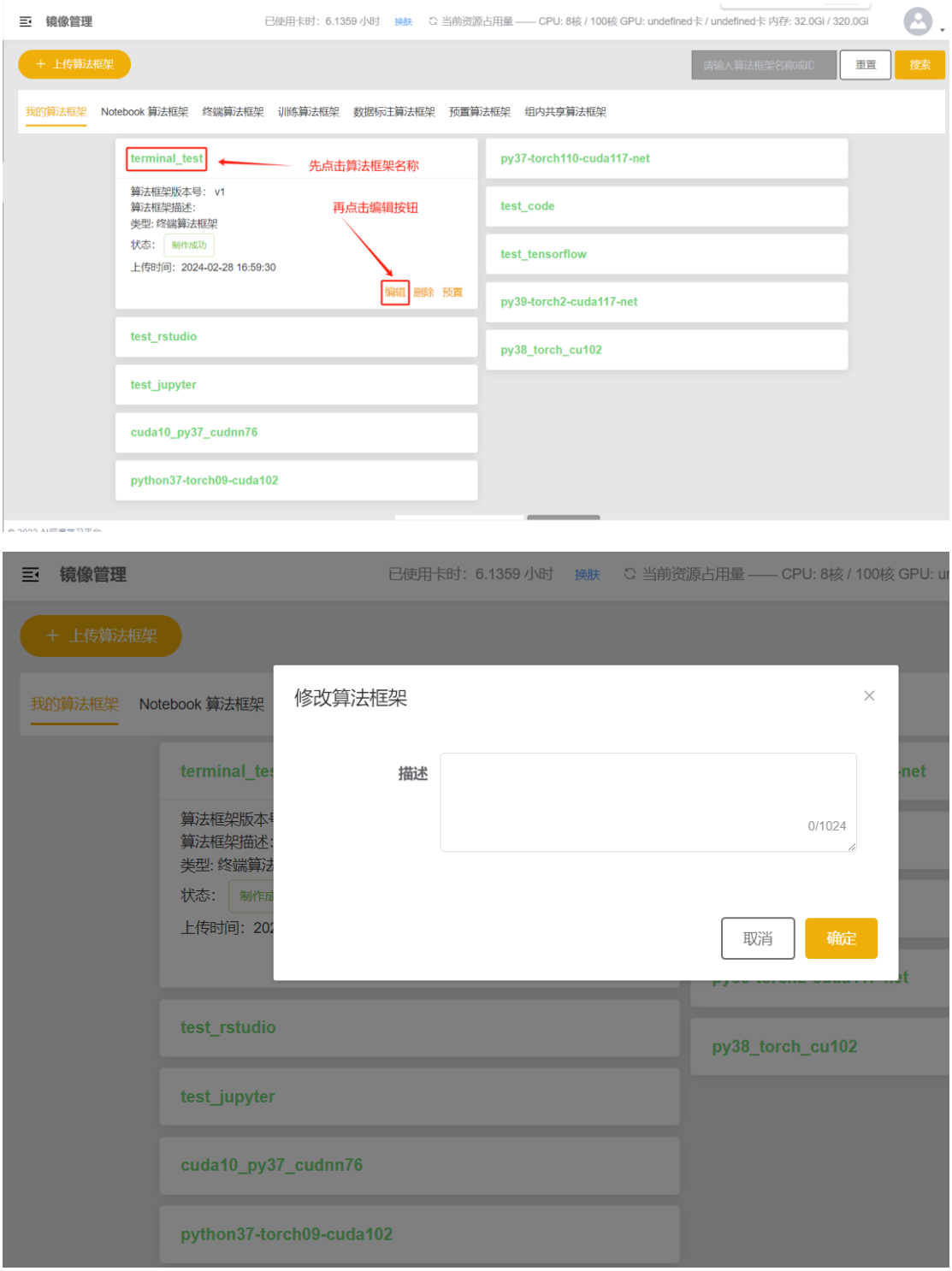
点击菜单“训练管理->镜像管理”，进入算法框架列表页面。
在算法框架列表页面，点击“上传算法框架”按钮，弹出上传算法框架的信息编辑框，输入算法框架的基本信息及选择已经打包后的算法框架文件，点击“确定”完成用户个人算法框架的上传



镜像管理——上传算法框架

1.2.2.2.2、编辑算法框架

在算法框架列表页面，先点击算法框架名称展示详情，然后点击“编辑”链接，弹出修改算法框架描述框，输入内容，点击“确定”即完成算法框架描述的修改



镜像管理——编辑算法框架

1.2.2.2.3、删除算法框架

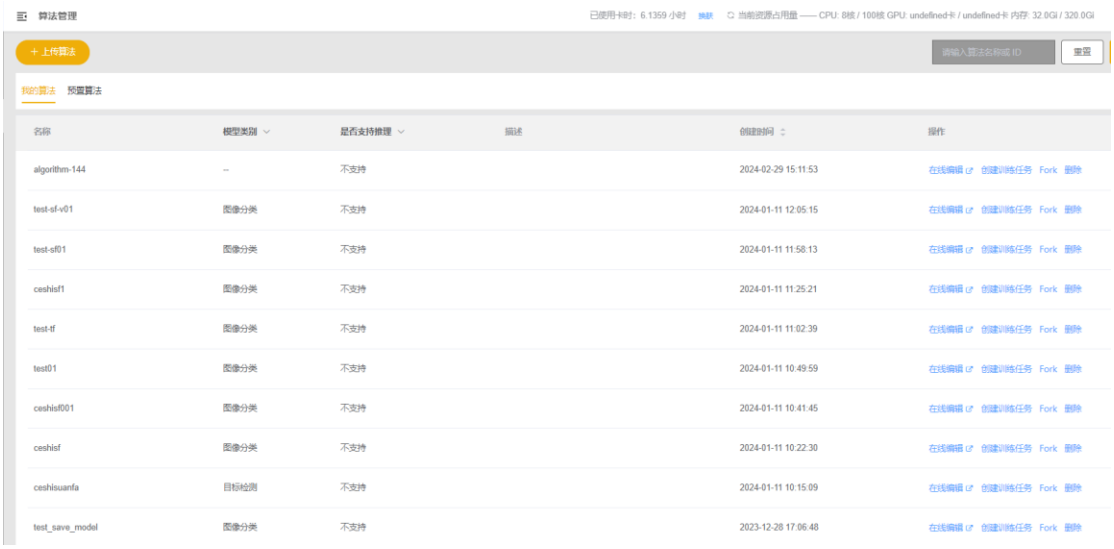
在镜像列表页面，先点击算法框架名称展示详情，然后“删除”链接，弹出删除算法框架确认框，点击“确定”即完成算法框架的删除

镜像管理——删除算法框架

1.2.3、算法

1.2.3.1、算法管理

用户点击左侧菜单“算法开发”-> “算法管理”，进入算法列表页。
用户可以查看我的算法、预置算法列表
我的算法:平台用户自己上传的算法
预置算法：由平台管理员上传的算法，对于平台的所有用户均可使用



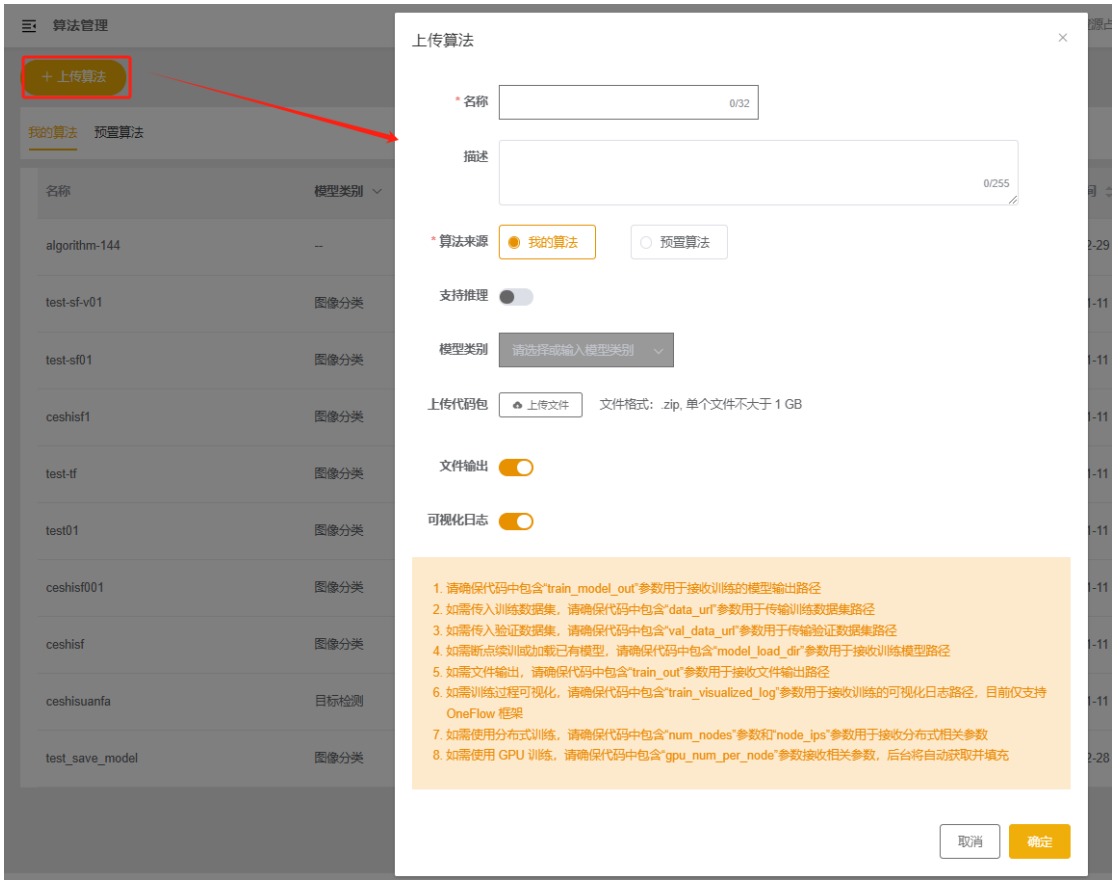
算法列表

当前算法列表页操作的按钮或者链接功能说明

操作按钮或链接	功能说明
上传算法	新建一个算法的功能入口
在线编辑	启动在线算法开发的 NOTEBOOK 的入口，提供用户在线编辑算法
创建训练任务	快速启动训练管理中训练任务的快捷入口
fork	复制拷贝算法
删除	对算法进行删除

1.2.3.1.1、上传算法

用户点击左侧菜单“算法开发”-> “算法管理”，进入算法列表页。
点击“上传算法”按钮，输入名称（不超过 32 字符）、描述（不超过 255 字符）、算法来源、模型类别。上传已打包的代码包（文件格式.zip,切单个文件不大于 1GB）
点击“确认按钮”，完成上传算法。

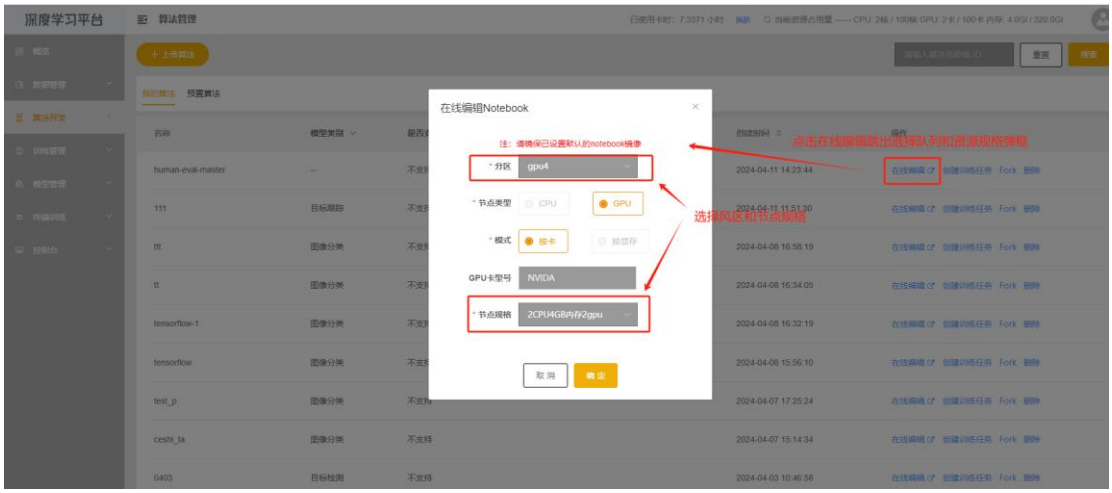


上传算法

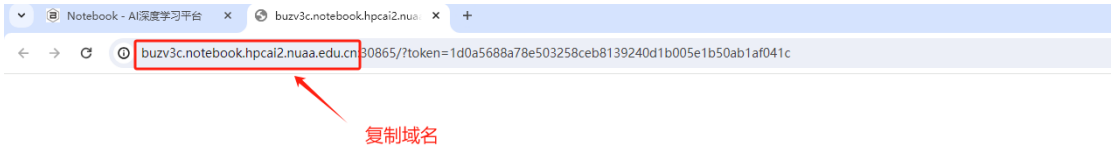
1.2.3.1.2、在线编辑（仅校内网络下使用）

说明：出于网络安全考虑，该功能仅校内网络下使用，校外网络或 VPN 均无法使用。

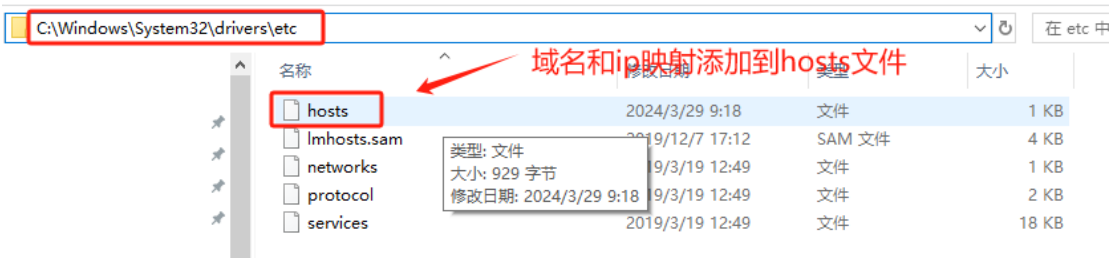
用户点击左侧菜单“算法”-> “算法管理”，进入算法列表页。
选择所需要编辑的算法，点击“在线编辑”按钮，选择队列和资源规格后点击“确认”启动 notebook 环境，浏览器会跳出新窗口，复制导航栏上的域名添加 ip（172.18.101.69）映射到本地 hosts 文件，保存后刷新浏览器即可进入在线编辑页面。
用户可以对已有文件进行打开，然后进行编辑。



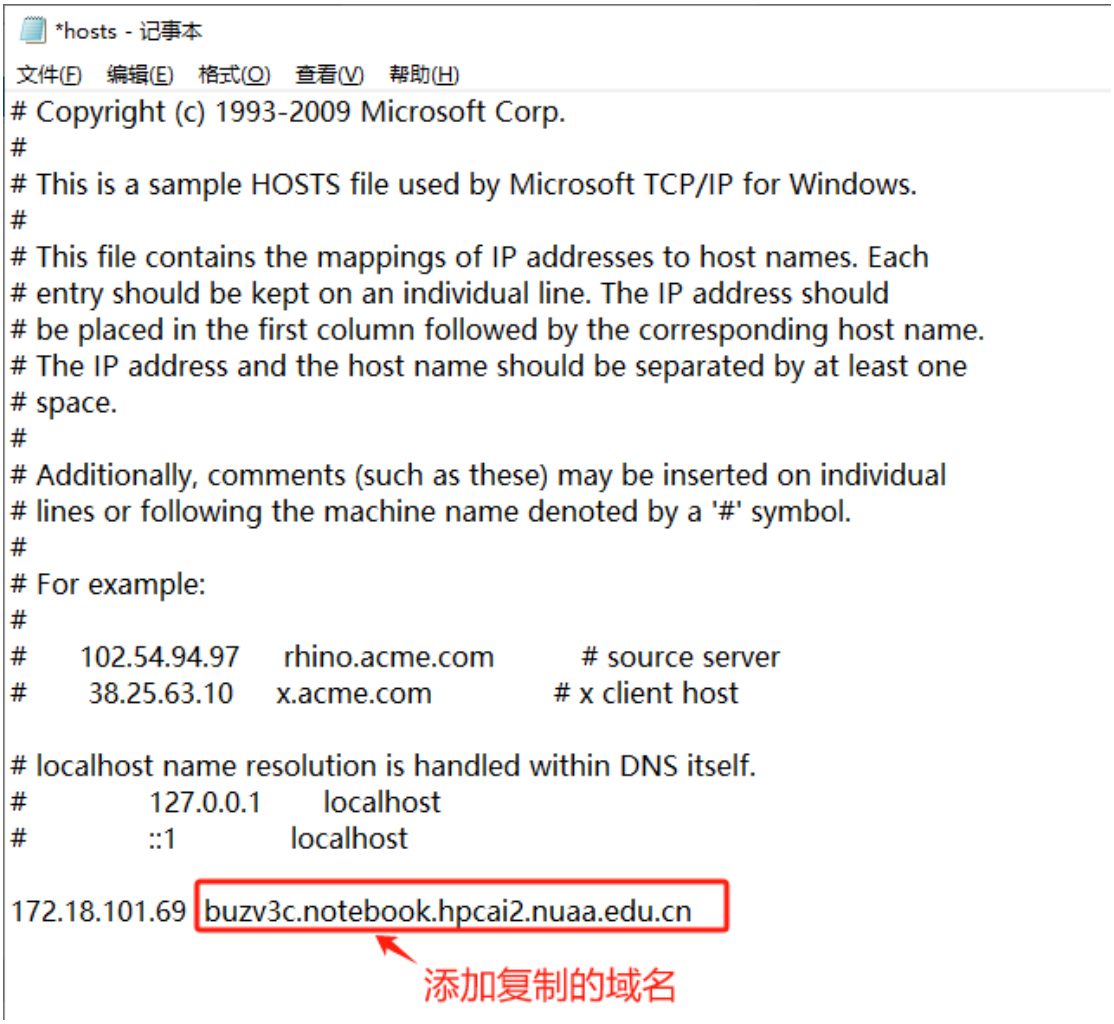
点击在线编辑选择分区和资源规格



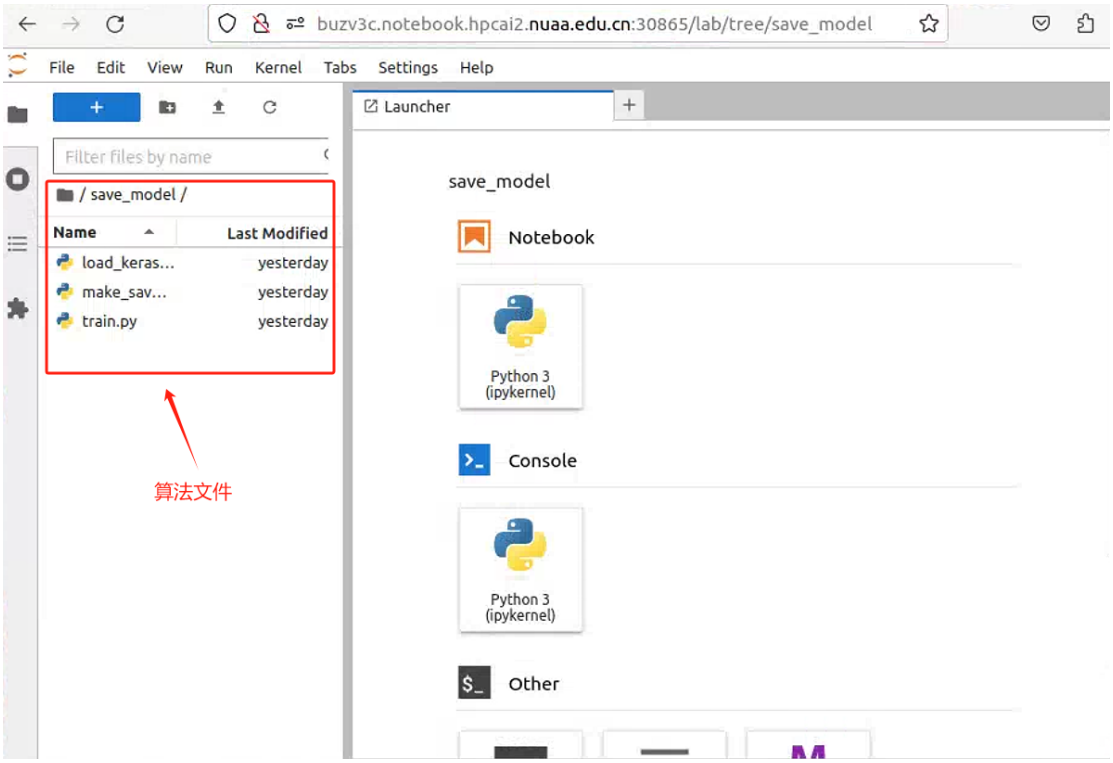
复制域名



hosts 文件添加域名 ip



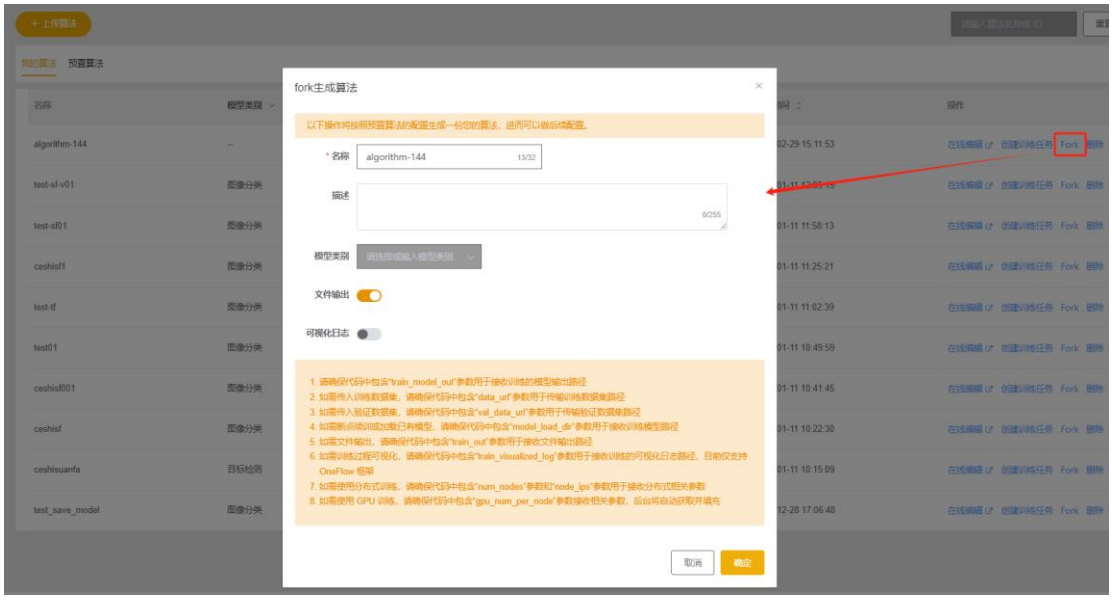
hosts 文件添加域名 ip



保存域名刷新浏览进入在线线编辑页面

1.2.3.1.3、fork 算法

用户点击左侧菜单“算法开发”-> “算法管理”，进入算法列表页。
选择所需要的算法，点击“Fork”，对已有算法进行副本拷贝。



Fork 算法

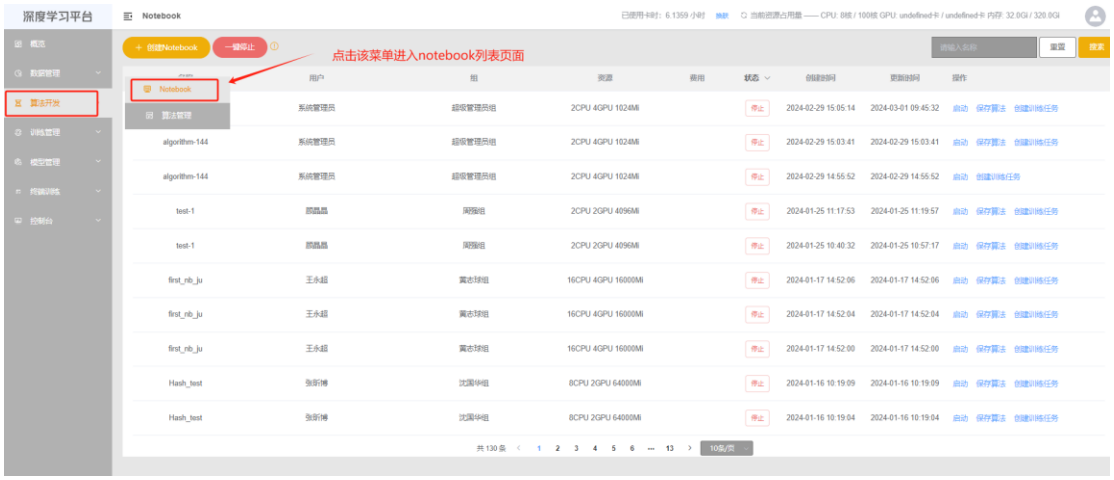
1.2.3.1.4、创建训练任务

用户点击左侧菜单“算法开发”-> “算法管理”，进入算法列表页。
选择所需要的算法，点击“创建训练任务”，进入创建训练任务界面。

1.2.3.2、 Notebook（仅校内网络下使用）

说明：出于网络安全考虑，该功能仅校内网络下使用，校外网络或 VPN 均无法使用。

NoteBook 是一个能运行 Python 等代码的 Web 应用程序，提供在线的算法开发。
用户点击左侧菜单“算法开发”-> “NoteBook”，进入 notebook 列表

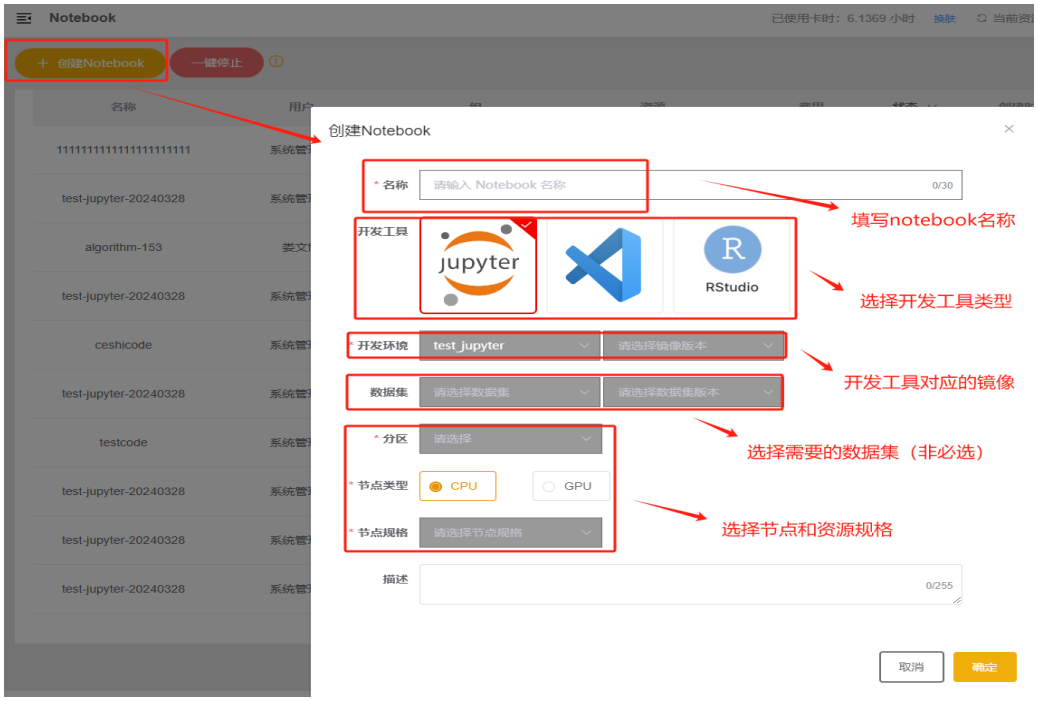


当前 notebook 列表页操作的按钮或者链接功能说明

操作按钮或链接	功能说明
创建 notebook	启动一个新的 notebook 任务的功能入口
一键停止	停止所有正在运行的在线算法开发的 NOTEBOOK 的任务
打开	快速启动 notebook 的快捷入口
停止	停止正在运行的在线算法开发的 NOTEBOOK 的任务
保存算法	将 notebook 中编辑的算法进行保存到算法库中
创建训练任务	快速启动训练管理中训练任务的快捷入口

1.2.3.2.1、创建 NoteBook

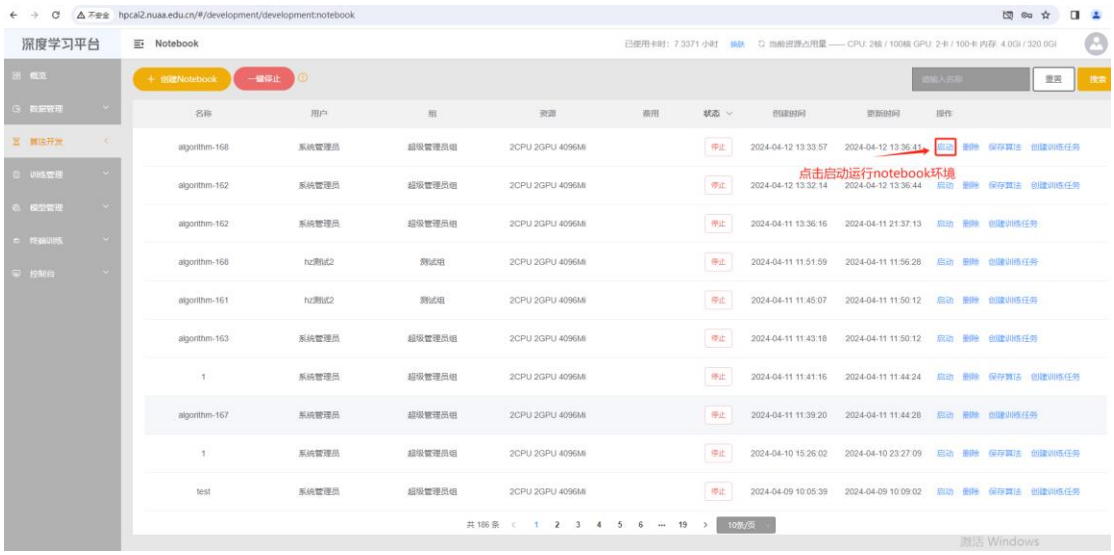
用户点击左侧菜单“算法”-> “NoteBook”，进入 notebook 列表。
点击“创建 Notebook”按钮，进入创建 Notebook 页面。
输入 notebook 名称，选择开发环境对应的镜像名称以及版本， 选择数据集，资源类型、以及节点规格。
点击“确定”按钮，创建 NoteBook 环境(创建后会自动运行)



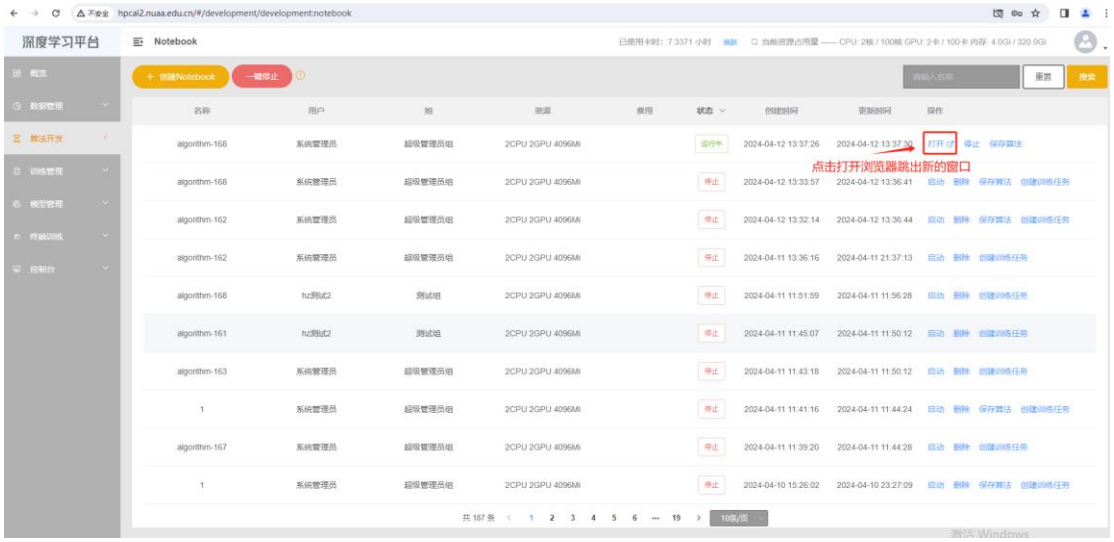
创建 notebook 开发环境

1.2.3.2.2、在线算法开发

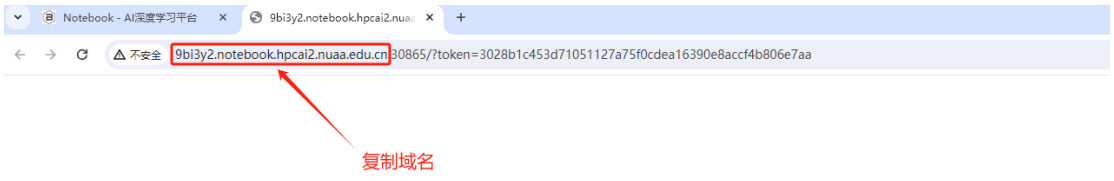
点击“启动”按钮，当 notebook 状态变成“运行中”时点击“打开”按钮，浏览器会新开一个窗口，复制浏览器导航栏的域名，添加 ip(172.18.101.69)映射添加到本地的 hosts 文件里面保存后刷新浏览器即可进入算法编辑页面，其中所选择的数据集将会 mount 到开发环境的/dataset/目录。在算法编辑页面中，用户可以使用 NoteBook、Console、terminal 环境开发，可以使用 text file 编辑脚本文件。算法编辑页面脚本会自动保存。



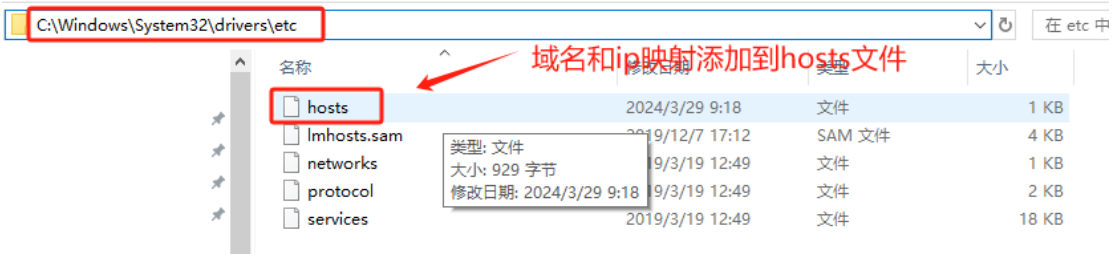
点击启动 **notebook**



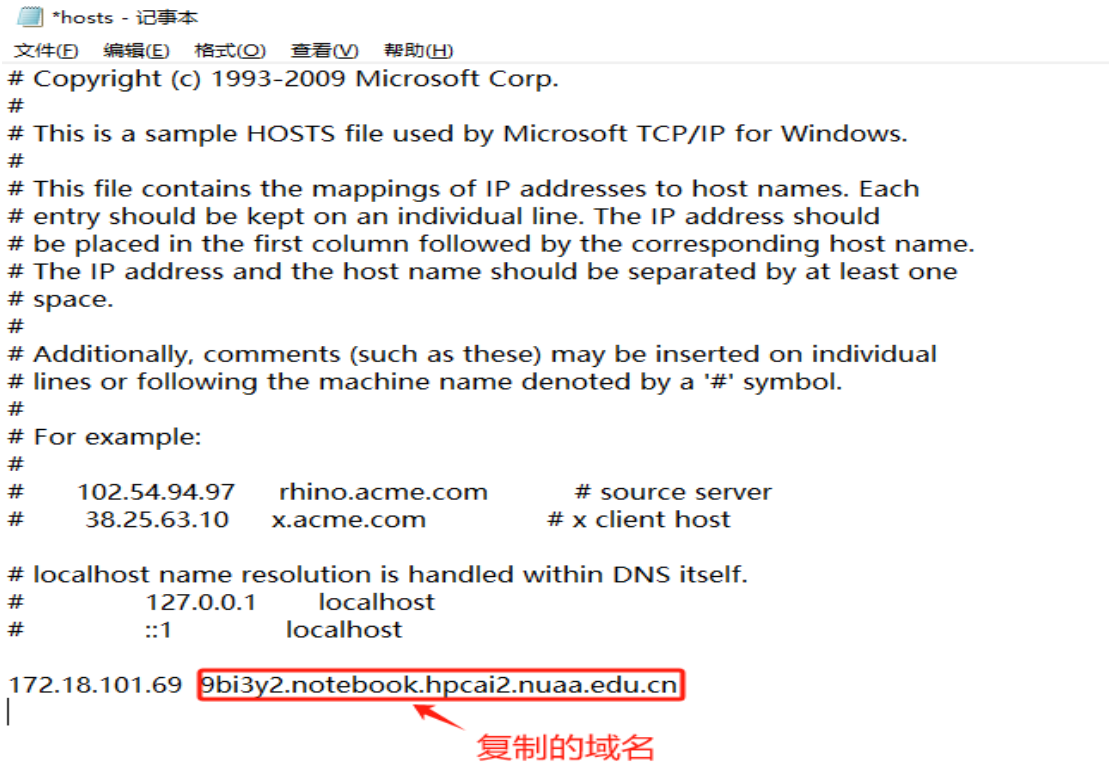
点击打开跳出新的窗口



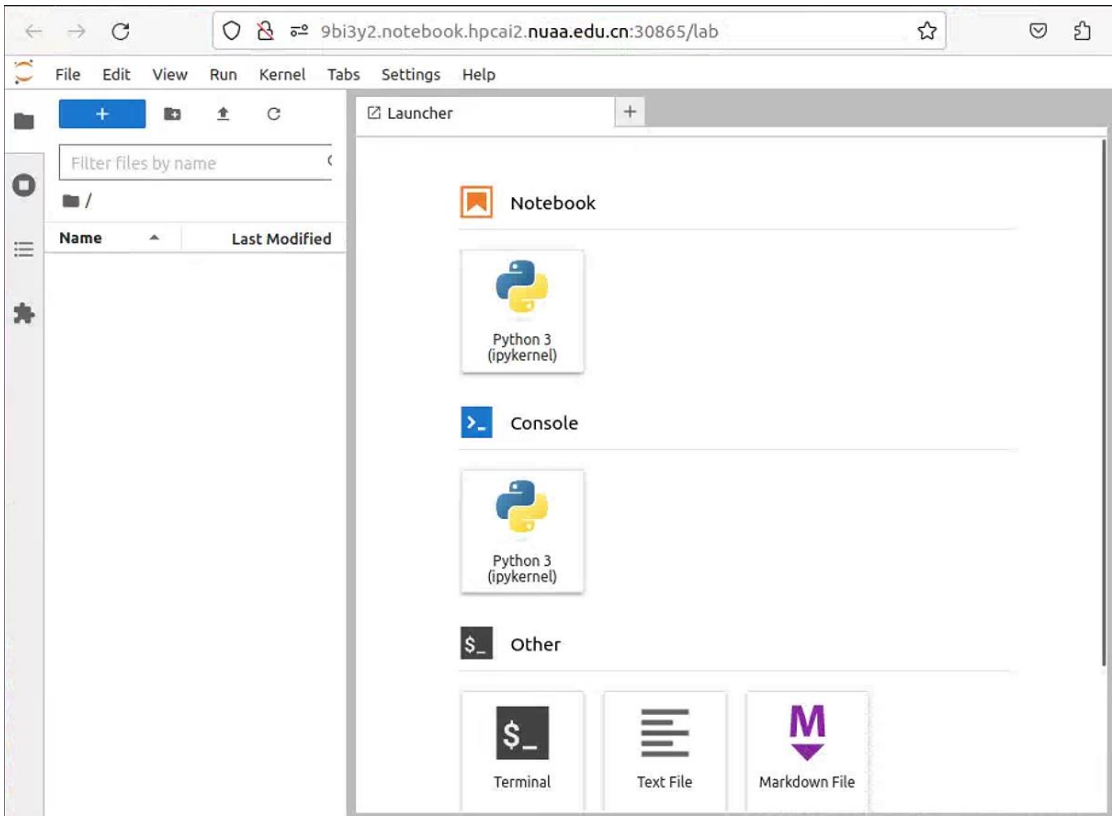
启动 notebook 之后浏览器会跳出新窗口



域名 ip 映射添加的 hosts 文件目录



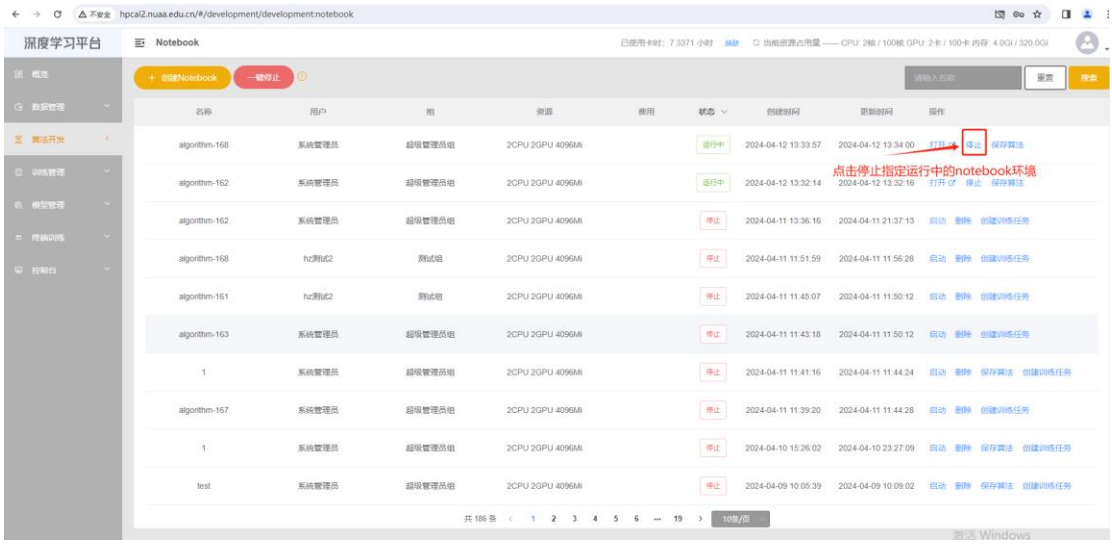
hosts 文件添加域名 ip



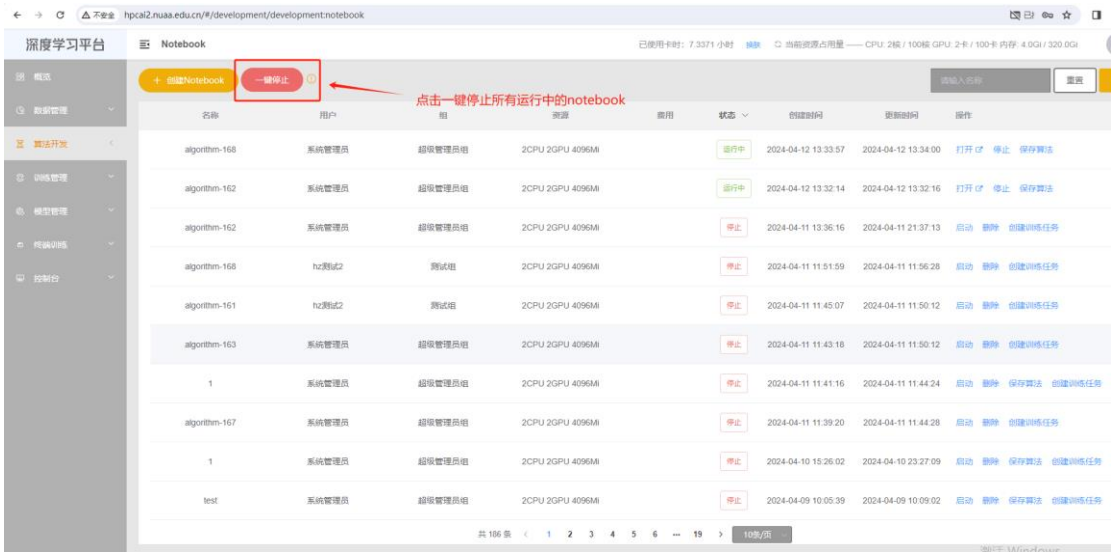
刷新浏览器进入在线算法开发环境

1.2.3.2.3、停止与一键停止

用户点击左侧菜单“算法”-> “NoteBook”，进入 notebook 列表。
选择所需要的正在运行中的 notebook 任务、点击对应的“停止”按钮，控制 NoteBook 的停止使用。
用户点击 notebook 列表页上的“一键停止”，控制用户下所有的 notebook 的停止使用



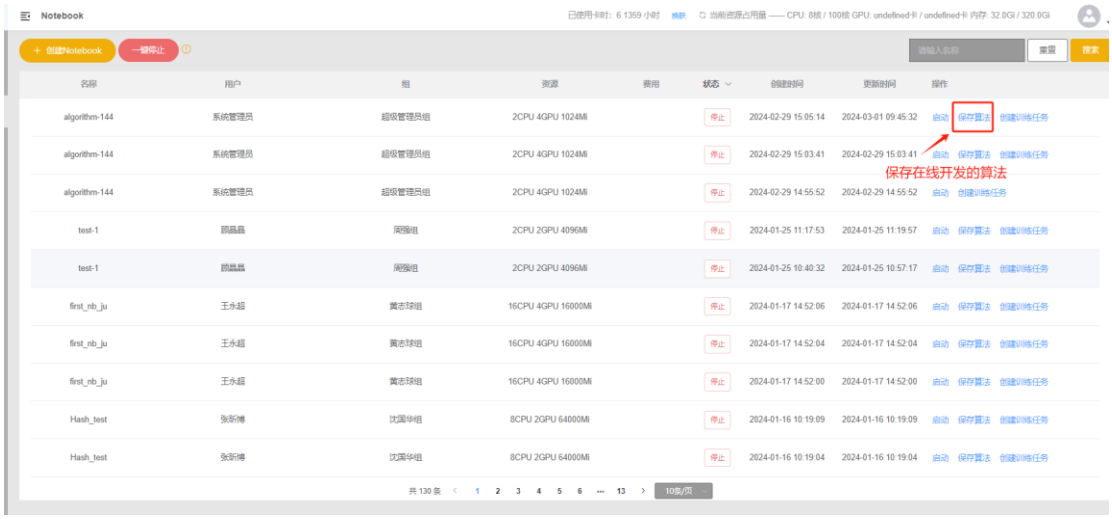
停止指定的 notebook



一键停止 notebook

1.2.3.2.4、保存算法

用户点击左侧菜单“算法”-> “NoteBook”，进入 notebook 列表。
选择所需要的 notebook，点击对应的“启动/停止”按钮，控制 NoteBook 的启动和停止使用。
通过“保存算法”按钮，将编辑好的算法,保存至算法管理



保存算法

1.2.3.2.5、创建训练任务

用户点击左侧菜单“算法”-> “算法管理”，进入算法列表页。
选择所需要的算法，点击“创建训练任务”，进入创建训练任务界面。

1.2.3.3、终端训练（仅校内网络下使用）

说明：出于网络安全考虑，该功能仅校内网络下使用，校外网络或 VPN 均无法使用。

终端训练指通过平台申请计算资源后，启动独立的机器学习环境。让用户使用本地 IDE 工具（如 vscode、python charm 等）通过 SSH 的方式远程连接到启动后的环境进行算法开发

用户点击左侧菜单“终端训练”-> “远程连接”，进入终端任务列表页。

终端训练列表

当前终端训练列表页操作的按钮或者链接功能说明

操作按钮或链接	功能说明
新建连接	启动一个新的终端训练任务的功能入口
保存与停止	直接保存与停止终端连接的任务，释放申请的资源
删除	直接删除终端连接的任务，释放申请的资源
启动	针对保存已终止的终端训练，重新启动

1.2.3.3.1、新建连接

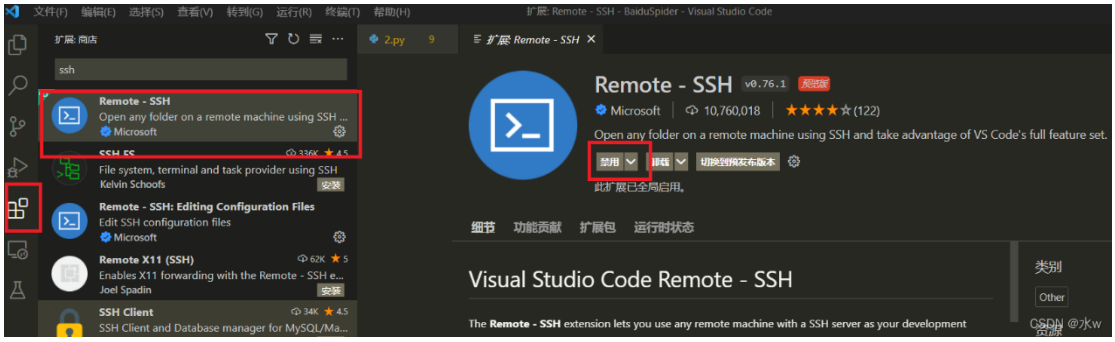
用户点击左侧菜单“终端训练”-> “远程连接”，进入任务列表页。

点击“新建连接”按钮，填写需要的资源（CPU、内存、GPU、磁盘）、数据集、远程连接的镜像、 节点数
点击确定，确认创建远程连接

1.2.3.3.2、远程连接

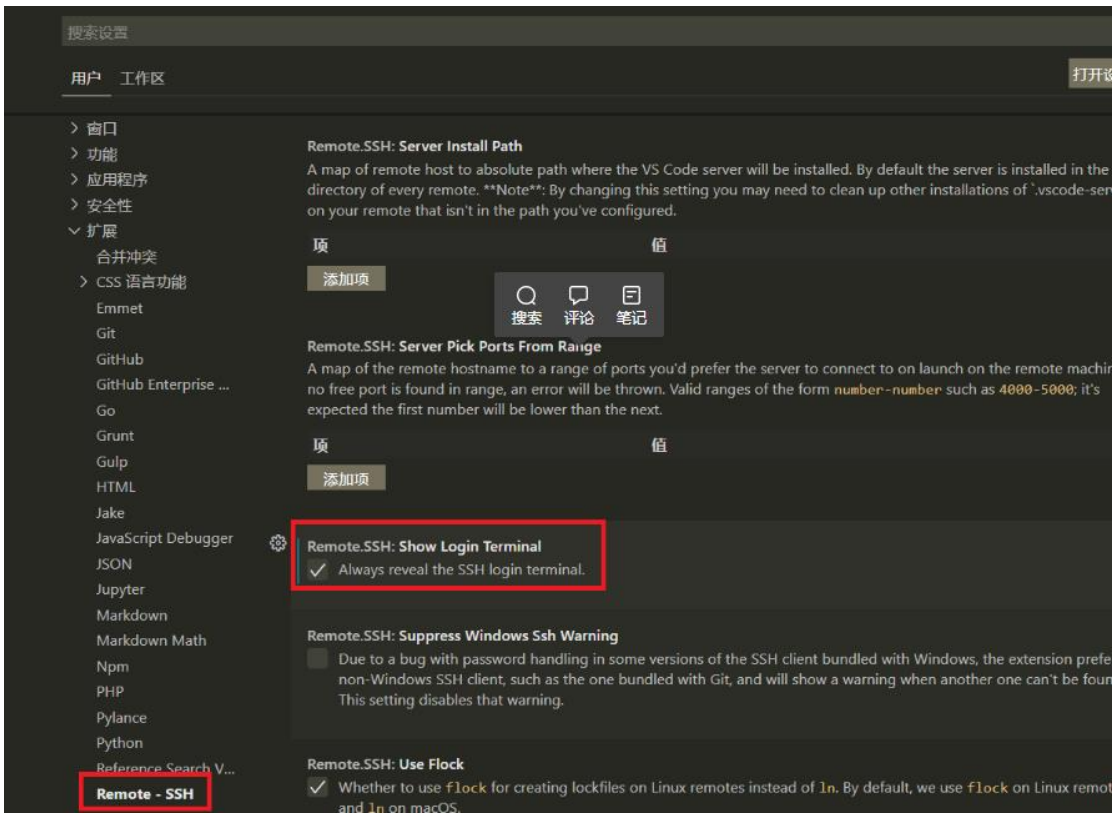
待服务启动后，通过 ssh 命令或者本地 IDE 工具、远程登录节点，进行训练操作。

1. VSCODE 远程连接示例
- 安装 SSH



- 点击右下角的“设置”按钮

- 根据截图的提示，找到“Show Login Terminal”并勾选该选项

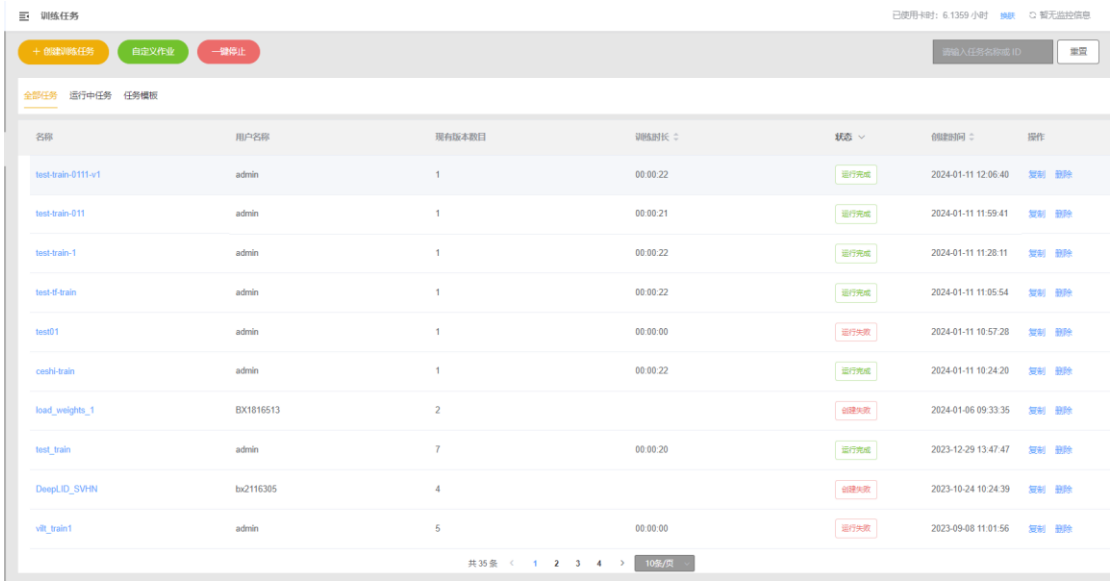


- 点击右侧工具栏中的“远程”可以查看远程连接，点击“SSH TARGETS”下面显示的服务器旁边的按钮进行连接操作，选择服务的平台，选择以后才会出现输入密码的步骤，弹出输入密码的框。输入密码之后，就会成功连接远程服务器

1.2.4、训练

1.2.4.1、训练任务(模型训练)

用户点击左侧菜单“训练管理”-> “训练任务”，进入任务列表页。分为全部任务、运行中任务、任务模板。



训练任务列表

当前训练列表页操作的按钮或者链接功能说明

操作按钮或链接	功能说明
创建训练任务	启动一个新的模型训练任务的功能入口
一键停止	批量停止该用户下所有的训练任务

1.2.4.2、创建训练任务

用户点击左侧菜单“训练管理”-> “训练任务”，进入任务列表页。
点击“创建训练任务”，进入训练任务创建页。

➢ 配置训练参数：
由用户填写任务名称、描述、创建方式、选用的算法、训练所需的镜像及版本、选择训练需要数据集、脚本启动命令。

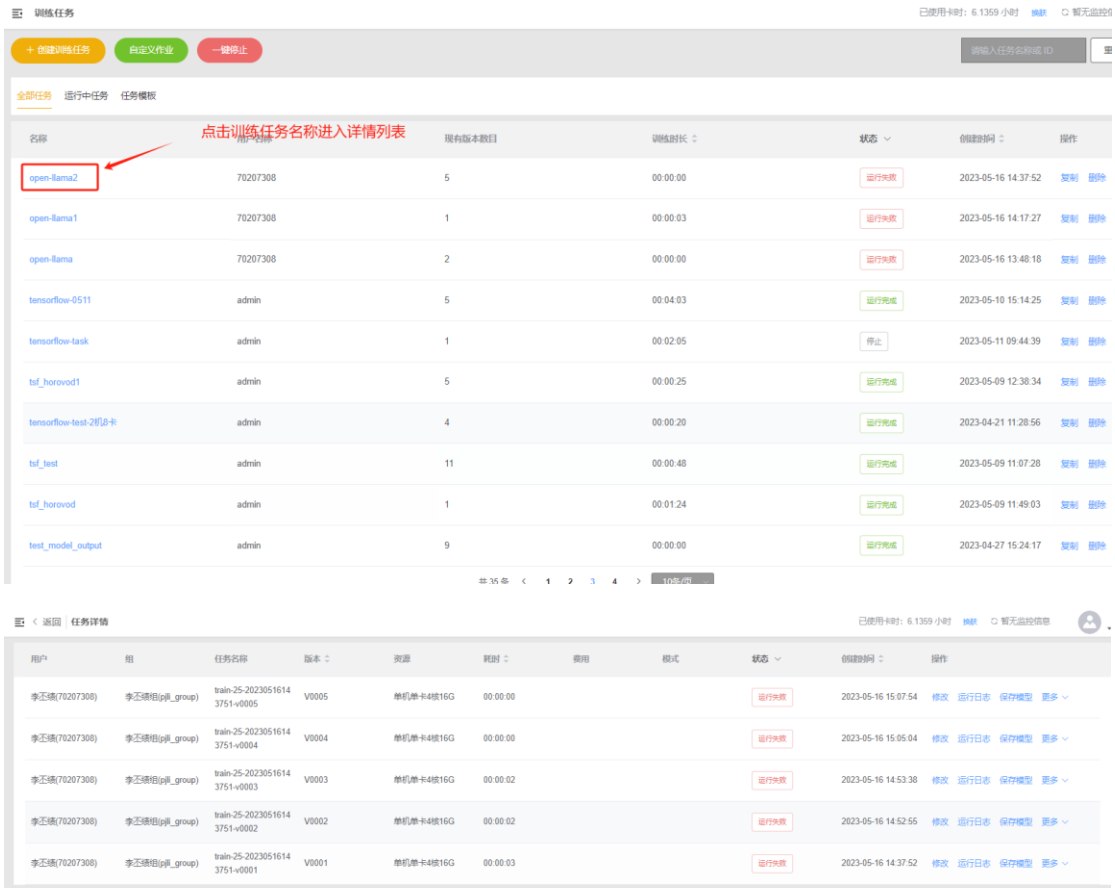
➢ 配置资源规格：
用户可以根据需要选择执行训练所需要的，节点数，节点类型（CPU/GPU），相应的节点规格，任务执行的优先级，

➢ 配置执行时间
用户可以配置延迟启动、训练时长上限

所有的训练任务设定的参数后，点击“开始训练”，生成一个新的训练任务。

1.2.4.3、查看训练任务

用户点击左侧菜单“训练管理”-> “训练任务”，进入任务列表页。
点击任务名称，进入任务版本列表。

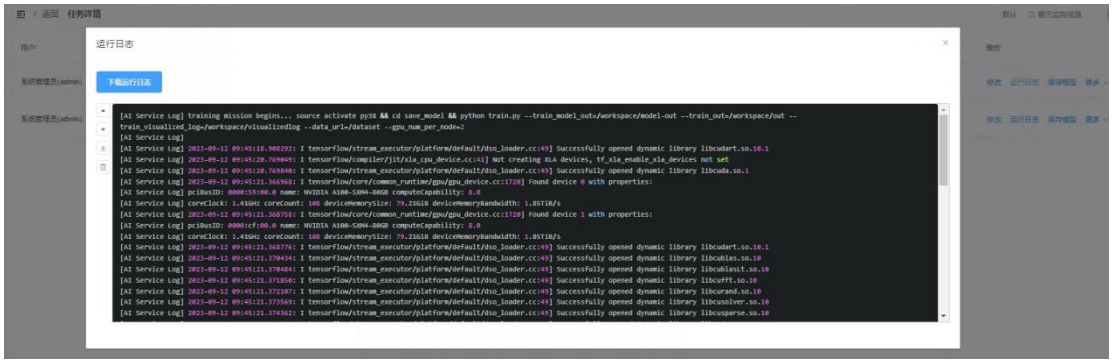


当前训练列表页操作的按钮或者链接功能说明

操作按钮或链接	功能说明
修改	启动一个新的终端训练任务的功能入口
运行日志	查看作业运行的日志信息
保存模型	运行完成的任务，对于输出的结果进行模型保存
停止	对正在运行中的任务进行停止
保存任务模板	当前训练任务的参数设置成一个模板，后续快速使用，提高创建作业的效率

1.2.4.4、运行日志

用户点击左侧菜单“训练管理”-> “训练任务”，进入任务列表页。
点击任务名称，进入任务版本列表。
点击“运行日志”查看实时监控训练任务执行情况
点击“下载运行日志”，可以下载日志到本地。



运行日志

1.2.4.5、修改训练任务

针对状态不是运行中的任务，可以点击“修改“弹出训练任务修改框，调整相应参数，点击”开始训练“，提交一个新任务

修改训练任务

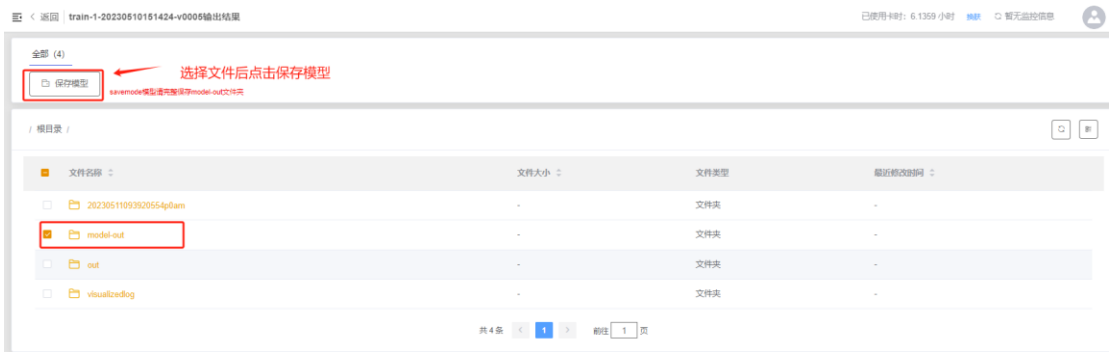
1.2.4.6、训练任务详情

点击单条任务，出现任务详情明细框，详细展示任务的名称、启动命令、申请的资源等

任务详情

1.2.4.7、保存模型

针对“运行完成“状态下的任务，可以点击”保存模型“将训练后的模型结果保存，在 model-out 目录选择要保存的模型文件。



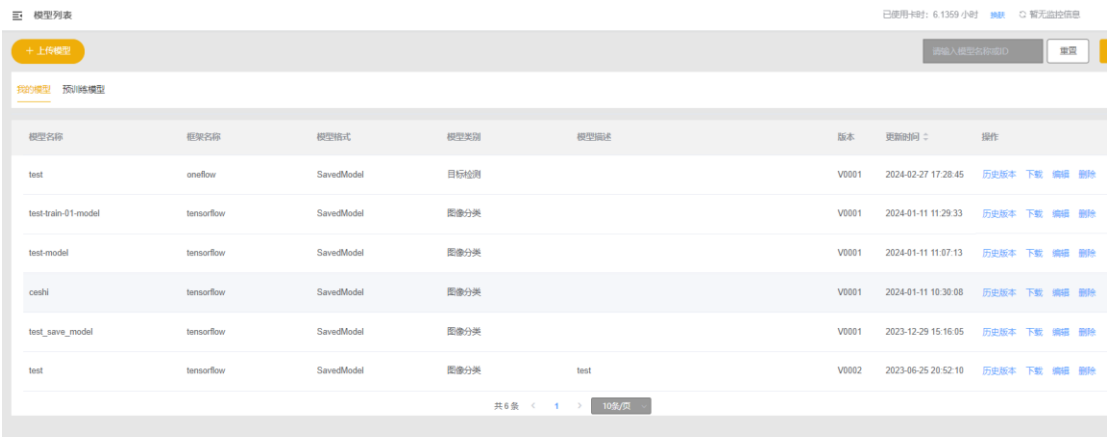
保存模型

1.2.5、模型

1.2.5.1、模型列表

用户点击左侧菜单“模型管理”-> “模型列表”，查看模型列表。分为我的模型、预训练模型。

我的模型:平台用户自己保存或者上传的模型
预训练模型：由平台管理员上传的模型，对于平台的所有用户均可使用



我的模型

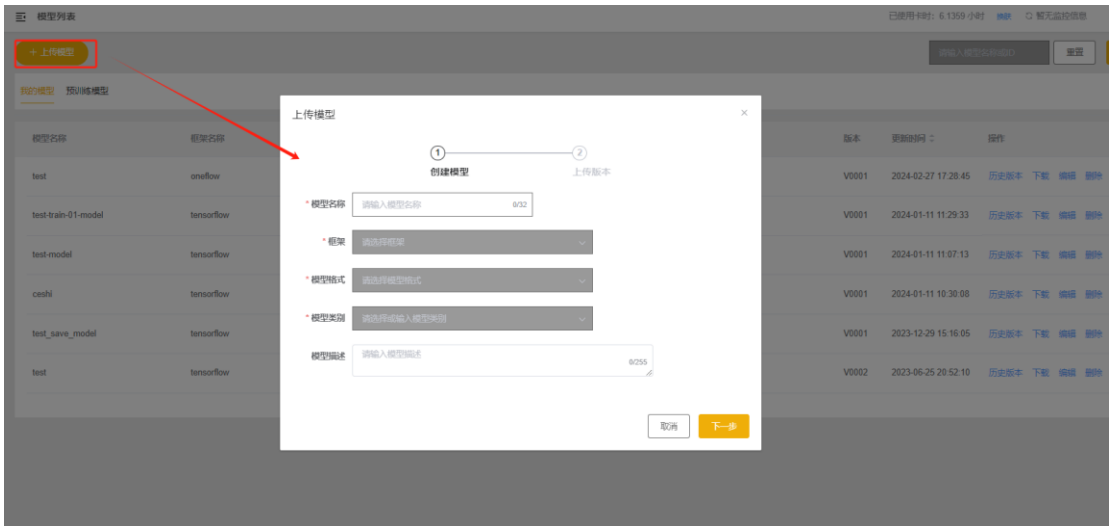
预训练模型

当前模型列表页操作的按钮或者链接功能说明

操作按钮或链接	功能说明
上传模型	上传新模型的功能入口
历史版本	查看模型的历史版本列表信息
下载	下载模型数据，以压缩包的方式下载
编辑	编辑模型的基本信息保存
删除	删除模型

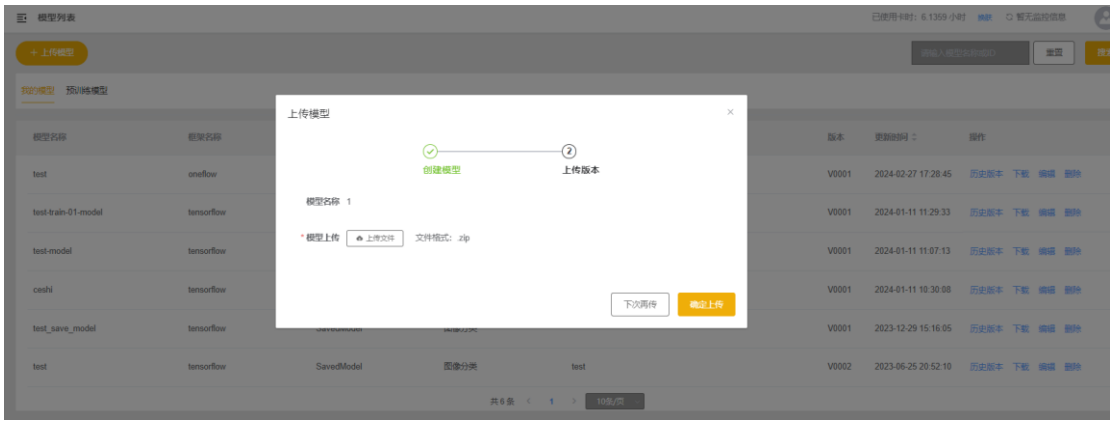
1.2.5.2、上传模型

用户点击左侧菜单“模型管理”-> “模型列表”，进入任务列表页。
点击“上传模型”，弹出模型创建编辑框。



上传模型

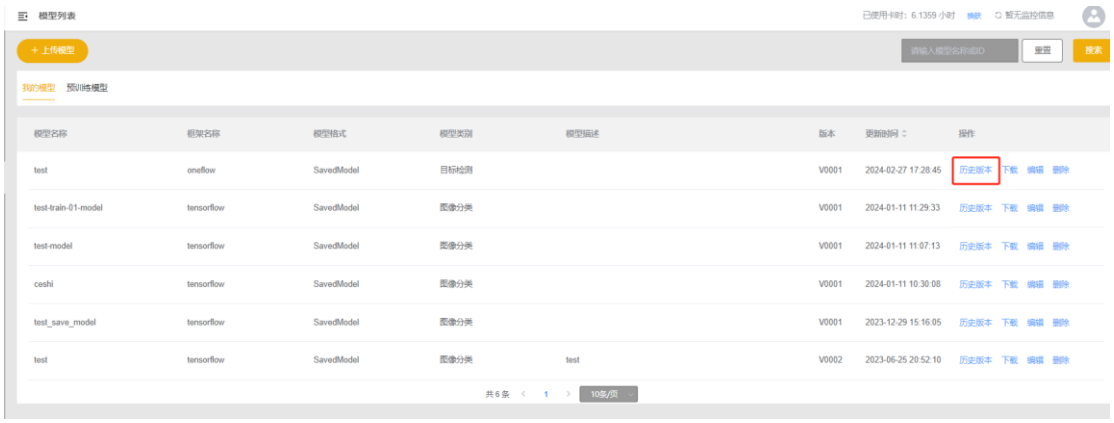
点击“下一步“，到模型上传的页面，选择需要上传的模型文件，点击”确定上传“完成模型上传



模型上传

1.2.5.3、历史版本

用户点击左侧菜单“模型管理”-> “模型列表”，进入任务列表页，点击某条模型的“历史版本“连接，进去模型历史版本页面，展示该模型对应的历史版本。

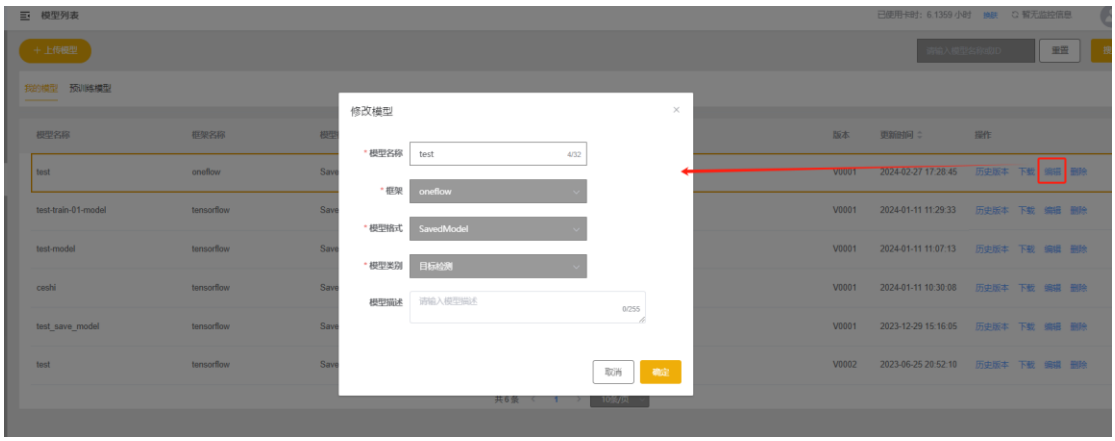


历史版本

点击“上传模型版本“展示的功能同”上传模型“功能

1.2.5.4、编辑模型

用户点击左侧菜单“模型管理”-> “模型列表”，进入任务列表页，点击某条模型的“编辑“连接，弹开模型信息编辑框，点击”确定“后保存模型信息。

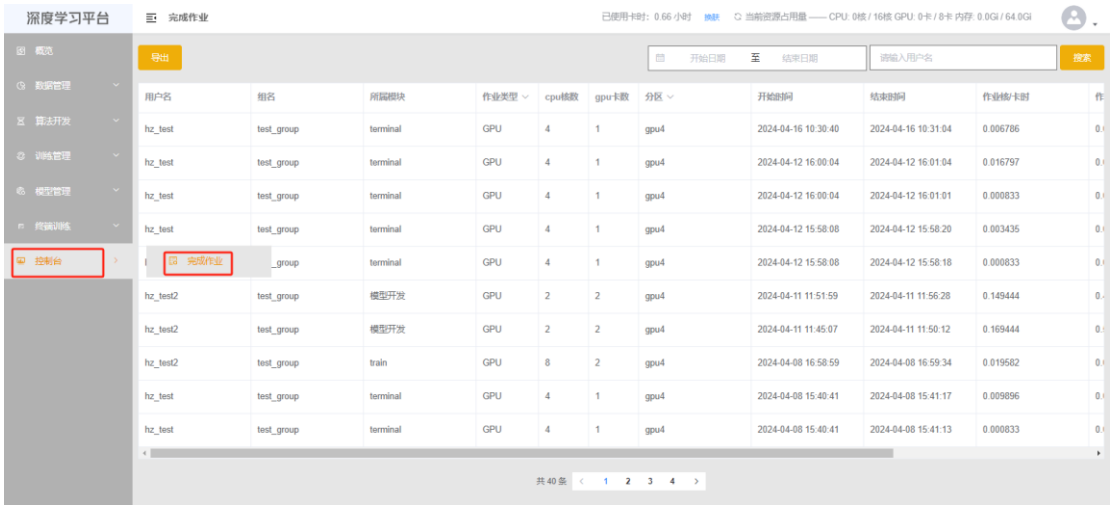


编辑模型

1.2.6、控制台

1.2.6.1、完成作业

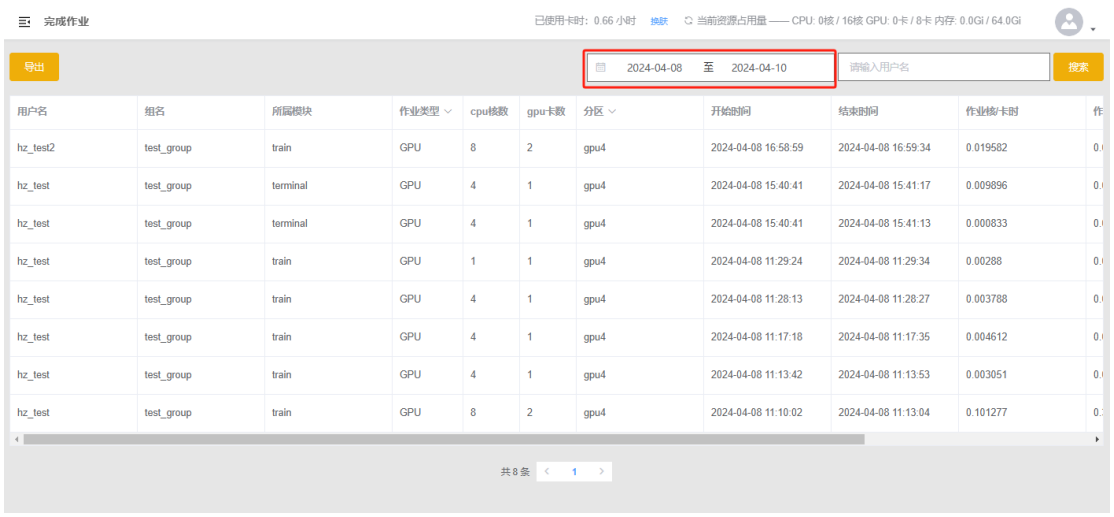
点击左侧菜单“控制台”-> “完成作业”，进入作业详情列表页。
用户可以查看作业运行详细信息、导出作业详情。



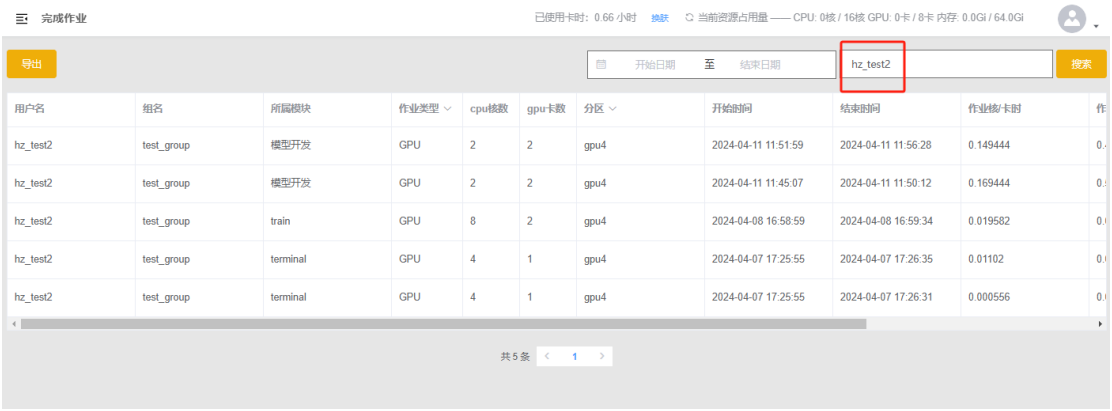
完成作业——作业列表

1.2.6.1.1、查询作业

用户点击左侧菜单“控制台”-> “完成作业”，进入作业详情列表页。
输入开始日期和结束日期或输入用户名后点击“搜索”按钮可查询相关的作业详细信息。



完成作业——按时间范围搜索

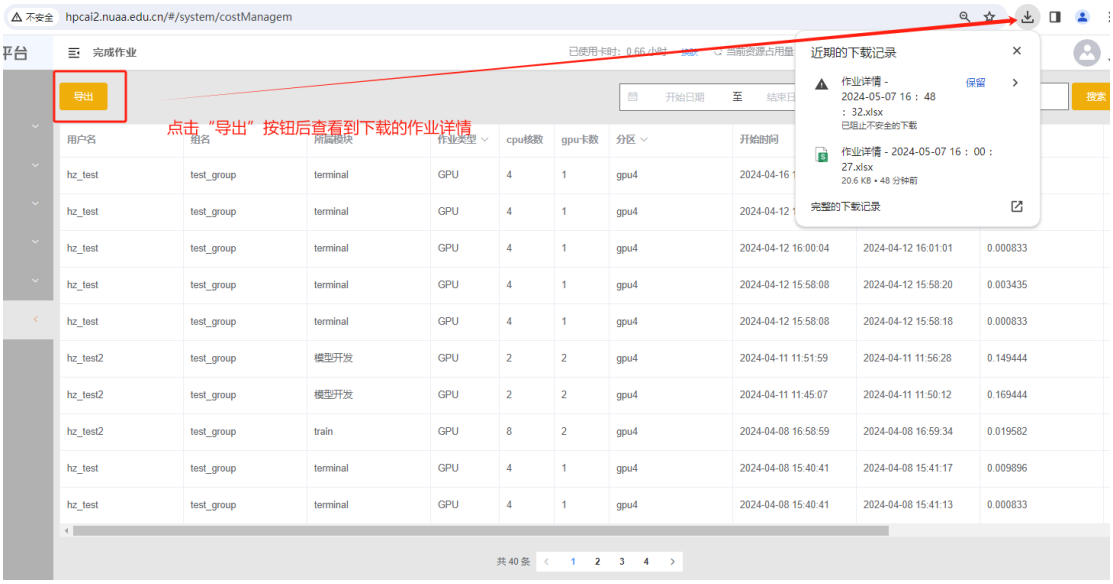


完成作业——按用户名搜索

完成作业——按时间范围和用户名搜索

1.2.6.1.2、导出作业

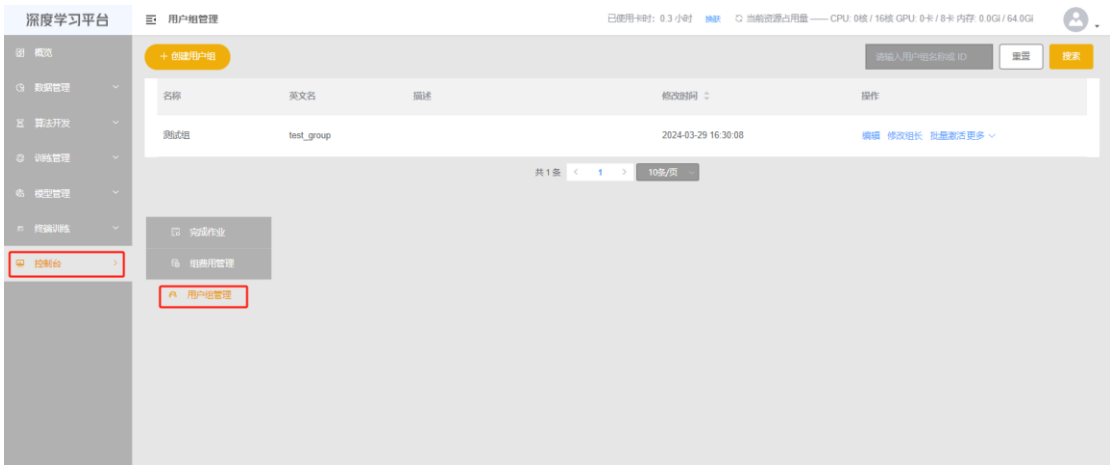
用户点击左侧菜单“控制台”-> “完成作业”，进入作业详情列表页。
点击左上角的“导出按钮”导出完成作业信息的 excel 文件。



完成作业——导出作业详情

1.2.6.2、用户组管理（仅教师用户可见）

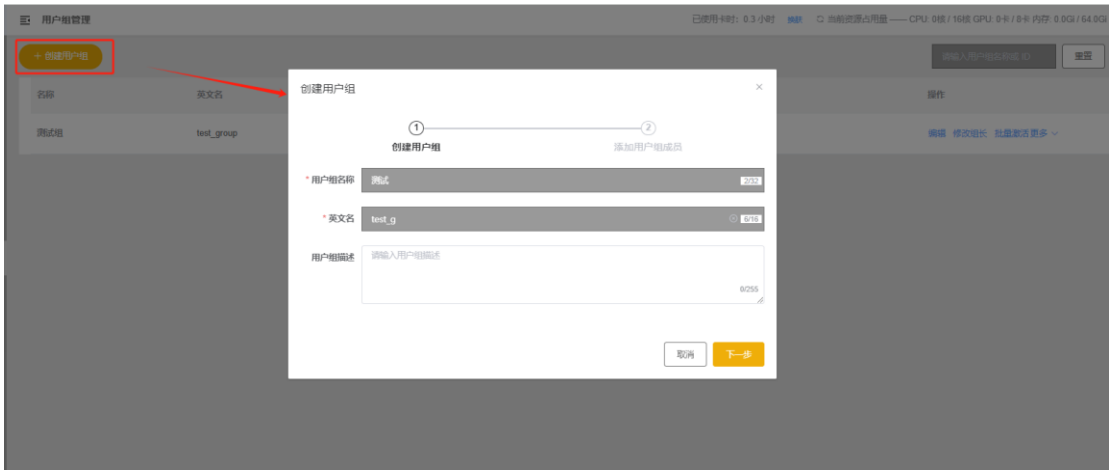
用户打开左侧菜单“控制台”-> “用户组管理”。可以对用户组进行创建、编辑用户组信息、修改组长等操作。



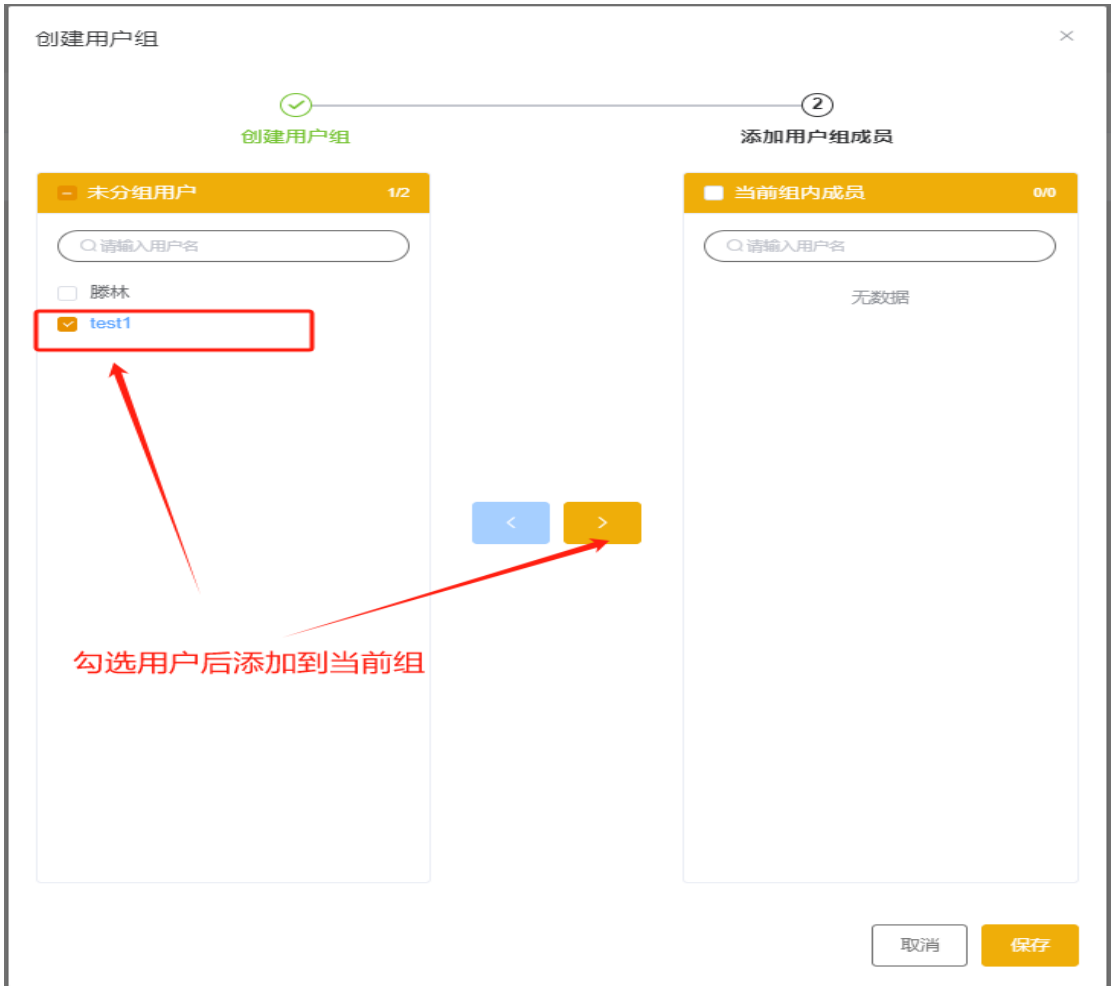
用户组列表

1.2.6.2.1、创建用户组

点击“创建用户组“按钮，弹出创建用户组信息框，维护用户组名称、英文名、描述等信息点击“下一步”按钮后选取用户添加到当前组，然后点击“保存”按钮即可。



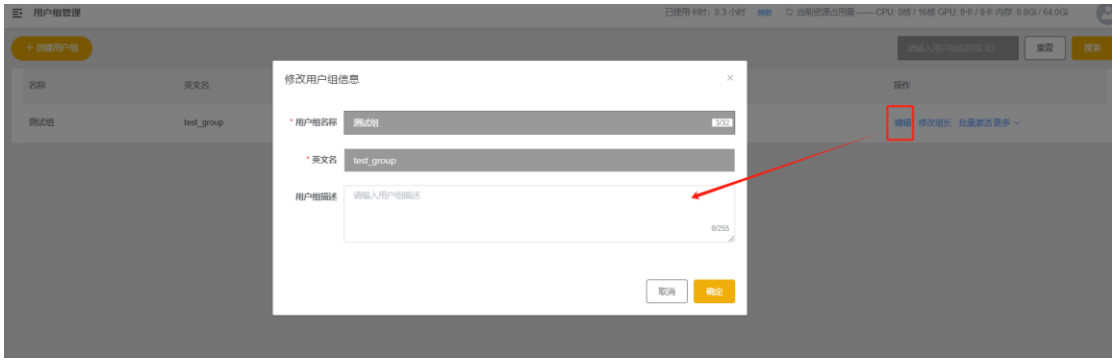
创建用户组



添加用户到当期组

1.2.6.2.2、修改用户组

在用户组列表中，选择一条用户组信息，点击“编辑”弹出用户组信息编辑框，调整用户组信息，点击“确定“完成用户组信息的编辑。



修改用户组

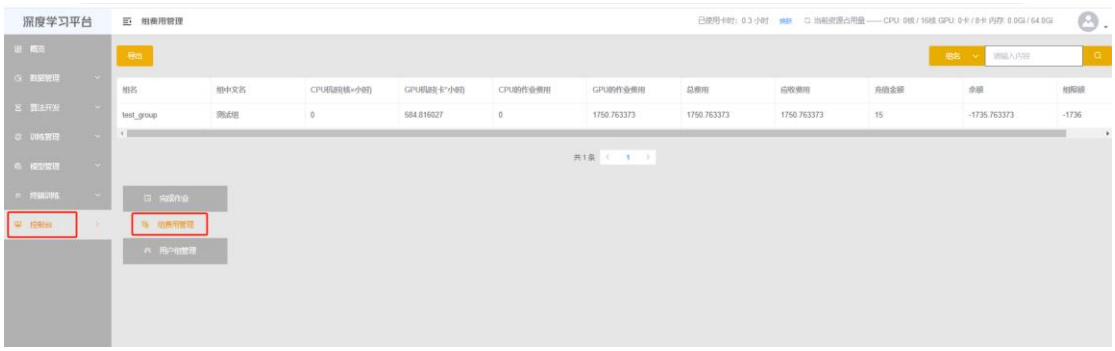
1.2.6.2.3、修改组长

在用户组列表中，选择一条用户组信息，点击“修改组长”弹出选取组长编辑框，选中组长，点击“确定“完成组长的修改。

修改组长

1.2.6.3、组费用管理（仅教师用户可见）

用户打开左侧菜单“控制台”-> “组费用管理”。可以查看每个组费用详情以及导出组费用详情。



组费用列表

1.2.6.3.1、搜索组费用详情

在组费用管理列表页面，右上角输入用户组名点击“搜索”按钮查询改组费用详情。

搜索组费用

1.2.6.3.2、导出组费用详情

在组费用管理列表页面，左上角点击“导出”按钮即可。

导出组费用

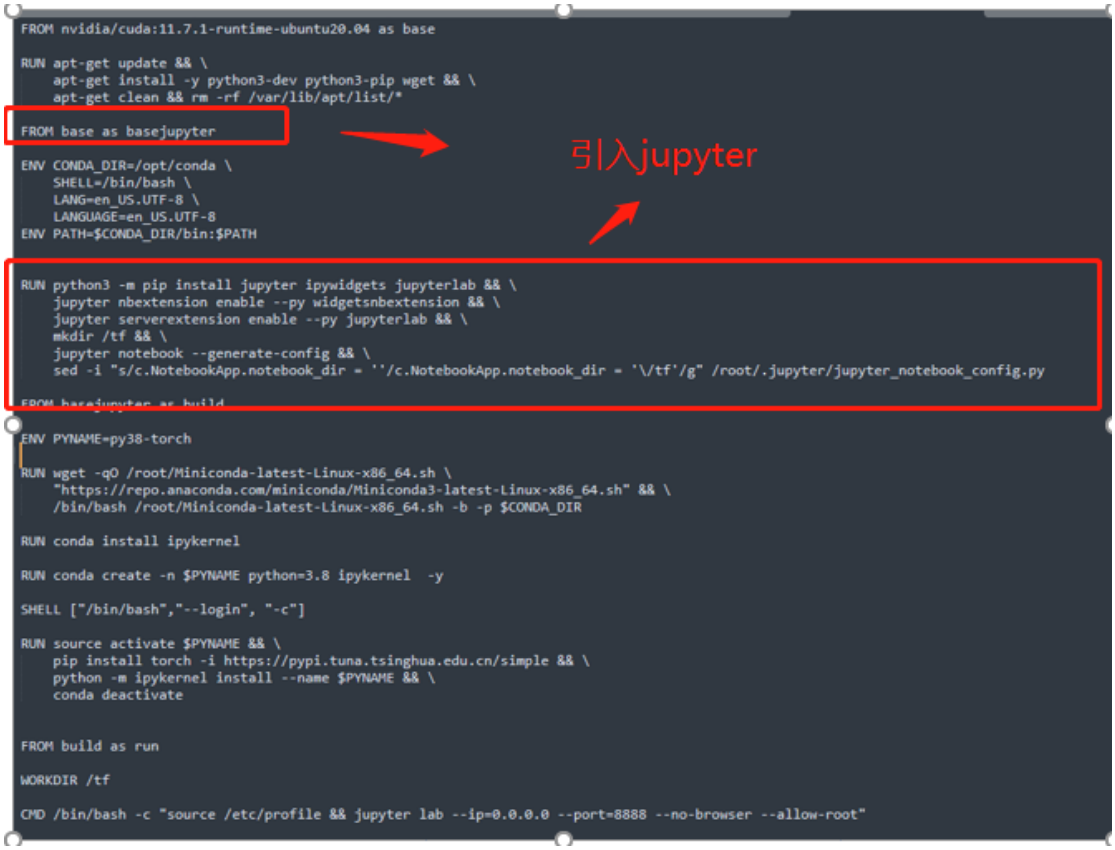
二、附录

2.1、镜像打包示例

镜像分为：notebook 镜像、训练镜像、终端镜像
notebook 镜像：主要预置了 jupyter 的安装，能够让用户可以通过浏览器进行在线算法开发
终端镜像：主要预置了 sshd 服务的支持，能够让用户通过 ssh 的方式远程登录到容器服务，进行算法开发
训练镜像：在训练管理中的进行模型训练使用的镜像，需要执行 pretreatment 脚本安装 jq, dnsutils 等系统工具

2.1.1、Notebook 镜像打包

- **编写 Dockerfile**，参考下图示例




```
FROM nvidia/cuda:11.7.1-runtime-ubuntu20.04 as base

RUN apt-get update && \
    apt-get install -y python3-dev python3-pip wget && \
    apt-get clean && rm -rf /var/lib/apt/list/*

FROM base as basejupyter

ENV CONDA_DIR=/opt/conda \
    SHELL=/bin/bash \
    LANG=en_US.UTF-8 \
    LANGUAGE=en_US.UTF-8
ENV PATH=$CONDA_DIR/bin:$PATH

RUN python3 -m pip install jupyter ipywidgets jupyterlab && \
    jupyter nbextension enable --py widgetsnbextension && \
    jupyter serverextension enable --py jupyterlab && \
    mkdir /tf && \
    jupyter notebook --generate-config && \
    sed -i "s/c.NotebookApp.notebook_dir = ''/c.NotebookApp.notebook_dir = '\/tf'/g" /root/.jupyter/jupyter_notebook_config.py

FROM basejupyter as build

ENV PYNAME=py38-torch

RUN wget -qO /root/Miniconda-latest-Linux-x86_64.sh \
    "https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh" && \
    /bin/bash /root/Miniconda-latest-Linux-x86_64.sh -b -p $CONDA_DIR

RUN conda install ipykernel

RUN conda create -n $PYNAME python=3.8 ipykernel  -y

SHELL ["/bin/bash","--login", "-c"]

RUN source activate $PYNAME && \
    pip install torch -i https://pypi.tuna.tsinghua.edu.cn/simple && \
    python -m ipykernel install --name $PYNAME && \
    conda deactivate


FROM build as run

WORKDIR /tf

CMD /bin/bash -c "source /etc/profile && jupyter lab --ip=0.0.0.0 --port=8888 --no-browser --allow-root"
```

- **针对 NoteBook 镜像必须引入 jupyter。**

安装 Jupyter 服务，以及将环境设置为容器默认启动命令如下:

```
# 安装和配置 Jupyter
RUN python3 -m pip install --upgrade --force pip && \
    python3 -m pip install  jupyter ipywidgets jupyterlab && \
    jupyter nbextension enable --py widgetsnbextension && \
    jupyter serverextension enable --py jupyterlab && \
    mkdir /tf && \
    jupyter notebook --generate-config && \
    sed -i "s/c.NotebookApp.notebook_dir = ''/c.NotebookApp.notebook_dir = '\/tf'/g" /root/.jupyter/jupyter_notebook_config.py

# 设置启动命令
WORKDIR /tf
CMD /bin/bash -c "source /etc/profile && jupyter lab --ip=0.0.0.0 --port=8888 --no-browser --allow-root"
```

- **调整容器内网络**

添加网络代理，命令如下：

```
RUN echo "export http_proxy=http://10.12.2.32:8989" >> /etc/profile
RUN echo "export https_proxy=http://10.12.2.32:8989" >> /etc/profile
```

- **调整容器内的 pip/apt 源**

指定内部 apt/pip 代理源,命令如下：

```
COPY pip.conf /root/.config/pip/pip.conf
```

COPY 10proxy /etc/apt/apt.conf.d/10proxy

- **Build 镜像**

在 dockerfile 文件所在目录，执行以下命令

docker build -t 镜像名称 .

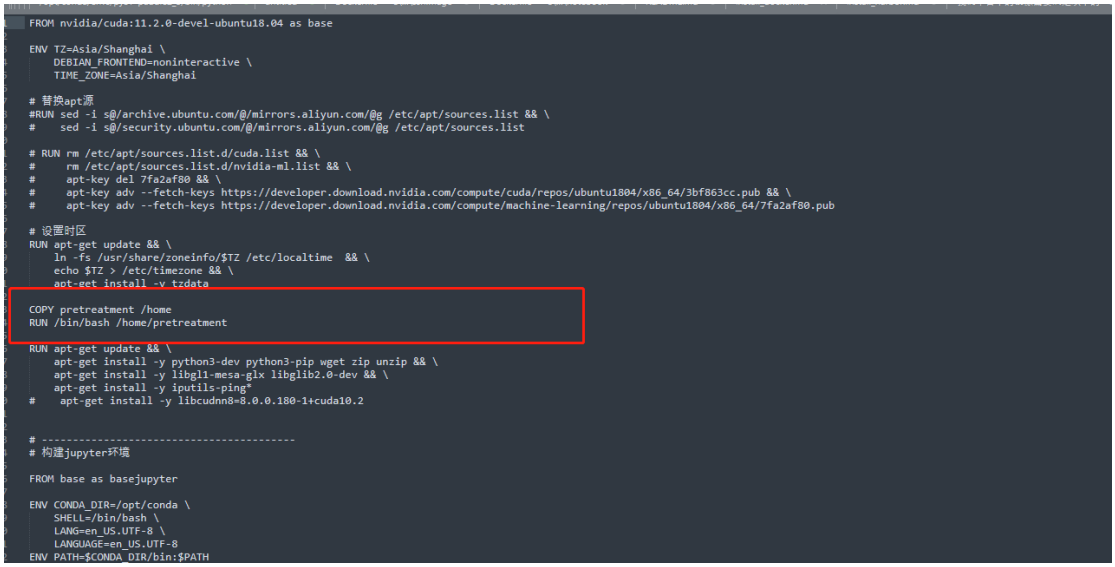
- **导出镜像为压缩包**

将镜像包导出为压缩包，提供在平台中进行上传

docker save -o xxxx.tar 镜像名称

2.2.2、训练镜像打包

- **编写 Dockerfile**



```
FROM nvidia/cuda:11.2.0-devel-ubuntu18.04 as base

ENV TZ=Asia/Shanghai \
    DEBIAN_FRONTEND=noninteractive \
    TIME_ZONE=Asia/Shanghai

# 替换apt源
#RUN sed -i s@/archive.ubuntu.com/@/mirrors.aliyun.com/@g /etc/apt/sources.list && \
#    sed -i s@/security.ubuntu.com/@/mirrors.aliyun.com/@g /etc/apt/sources.list

# RUN rm /etc/apt/sources.list.d/cuda.list && \
#     rm /etc/apt/sources.list.d/nvidia-ml.list && \
#     apt-key del 7fa2af80 && \
#     apt-key adv --fetch-keys https://developer.download.nvidia.com/compute/cuda/repos/ubuntu1804/x86_64/3bf863cc.pub && \
#     apt-key adv --fetch-keys https://developer.download.nvidia.com/compute/machine-learning/repos/ubuntu1804/x86_64/7fa2af80.pub

# 设置时区
RUN apt-get update && \
    ln -fs /usr/share/zoneinfo/$TZ /etc/localtime  && \
    echo $TZ > /etc/timezone && \
    apt-get install -y tzdata

COPY pretreatment /home
RUN /bin/bash /home/pretreatment

RUN apt-get update && \
    apt-get install -y python3-dev python3-pip wget zip unzip && \
    apt-get install -y libgl1-mesa-glx libglib2.0-dev && \
    apt-get install -y iputils-ping*
#    apt-get install -y libcudnn8=8.0.0.180-1+cuda10.2


# ----------------------------------------
# 构建jupyter环境

FROM base as basejupyter

ENV CONDA_DIR=/opt/conda \
    SHELL=/bin/bash \
    LANG=en_US.UTF-8 \
    LANGUAGE=en_US.UTF-8
ENV PATH=$CONDA_DIR/bin:$PATH
```

图示中针对训练镜像，

- **需要执行 pretreatment 脚本安装 jq, dnsutils 等系统工具**

将 pretreatment 放于 Dockerfile 同级目录，并在 Dockerfile 中添加以下命令

COPY pretreatment /home

RUN /bin/bash /home/pretreatment

- **调整容器内网络**

添加网络代理，命令如下：

RUN echo "export http_proxy=http://10.12.2.32:8989" >> /etc/profile

RUN echo "export https_proxy=http://10.12.2.32:8989" >> /etc/profile

- **调整容器内的 pip/apt 源**

指定内部 apt/pip 代理源，命令如下：

COPY pip.conf /root/.config/pip/pip.conf

COPY 10proxy /etc/apt/apt.conf.d/10proxy

- **Build 镜像**

在 dockerfile 文件所在目录，执行以下命令

docker build -t 镜像名称 .

- **导出镜像为压缩包**

将镜像包导出为压缩包，提供在平台中进行上传
docker save -o xxxx.tar 镜像名称

2.2.3、终端镜像打包

- 编写 Dockerfile

```
FROM nvidia/cuda:11.2.0-devel-ubuntu18.04 as base

ENV TZ=Asia/Shanghai \
    DEBIAN_FRONTEND=noninteractive \
    TIME_ZONE=Asia/Shanghai

# 替换apt源
#RUN sed -i s@/archive.ubuntu.com/@/mirrors.aliyun.com/@g /etc/apt/sources.list && \
#    sed -i s@/security.ubuntu.com/@/mirrors.aliyun.com/@g /etc/apt/sources.list

#RUN rm /etc/apt/sources.list.d/cuda.list && \
#    rm /etc/apt/sources.list.d/nvidia-ml.list && \
#    apt-key del 7fa2af80 && \
#    apt-key adv --fetch-keys https://developer.download.nvidia.com/compute/cuda/repos/ubuntu1804/x86_64/3bf863cc.pub && \
#    apt-key adv --fetch-keys https://developer.download.nvidia.com/compute/machine-learning/repos/ubuntu1804/x86_64/7fa2af80.pub

# 设置时区
RUN apt-get update && \
    ln -fs /usr/share/zoneinfo/$TZ /etc/localtime  && \
    echo $TZ > /etc/timezone && \
    apt-get install -y tzdata


RUN apt-get update && \
    apt-get install -y python3-dev python3-pip wget zip unzip openssh-server && \
    apt-get install -y libgl1-mesa-glx libglib2.0-dev && \
    apt-get install -y iputils-ping*
#    apt-get install -y libcudnn8=8.0.0.180-1+cuda10.2

#RUN apt-get install -y libnccl2=2.7.3-1+cuda10.2 libnccl-dev=2.7.3-1+cuda10.2 --allow-change-held-packages && \
#    apt-get clean

# ----------------------------------------
# 构建jupyter环境

FROM base as basejupyter
```

```
    filterpy


# ------------------------------------------------
# 启动ssh

FROM build as run

RUN echo "http_proxy=http://192.168.71.1:8989" >> /etc/profile
COPY pip.conf /root/.config/pip/pip.conf
COPY 10proxy /etc/apt/apt.conf.d/10proxy

WORKDIR /tf

RUN service ssh start

RUN echo 'root:1qaz@WSX' | chpasswd

RUN sed -ri 's/^#PermitRootLogin\s+.*/PermitRootLogin yes/' /etc/ssh/sshd_config
RUN sed -ri 's/UsePAM yes/#UsePAM yes/g' /etc/ssh/sshd_config

RUN useradd -rm -d /home/ubuntu -s /bin/bash -g root -G sudo -u 1000 test
RUN usermod -aG sudo test
RUN echo 'test:test' | chpasswd


EXPOSE 22
CMD ["/usr/sbin/sshd","-D"]
```

- 预装 sshd 服务

安装和启动 ssh 命令如下

```
RUN apt-get install -y openssh-server
EXPOSE 22
CMD ["/usr/sbin/sshd","-D"]
```

- 调整容器内网络

添加网络代理，命令如下：

```
RUN echo "export http_proxy=http://10.12.2.32:8989" >> /etc/profile
RUN echo "export https_proxy=http://10.12.2.32:8989" >> /etc/profile
```

- **调整容器内的 pip/apt 源**

指定内部 apt/pip 代理源,命令如下：

```
COPY pip.conf /root/.config/pip/pip.conf
COPY 10proxy /etc/apt/apt.conf.d/10proxy
```

- **Build 镜像**

在 dockerfile 文件所在目录，执行以下命令

```
docker build -t 镜像名称 .
```

- **导出镜像为压缩包**

将镜像包导出为压缩包，提供在平台中进行上传

```
docker save -o xxxx.tar 镜像名称
```

2.2、场景使用示例

2.2.1、使用 Paddle 进行 COCO 训练

- **准备工作**

- 训练集
- Notebook 镜像
- 训练镜像

- **下载数据集**

依据开源地址 https://github.com/PaddlePaddle/PaddleDetection 数据集的下载说明，通过代码自动化下载 COCO 数据集

```
# 参考执行代码自动化下载 COCO 数据集
python dataset/coco/download_coco.py
```

- **上传数据集**

将下载下来的数据集通过深度学习平台的数据集管理功能进行上传。

查询数据集:

新建数据集:

填写数据机名称、选择数据类型以及模型类型创建新的数据集

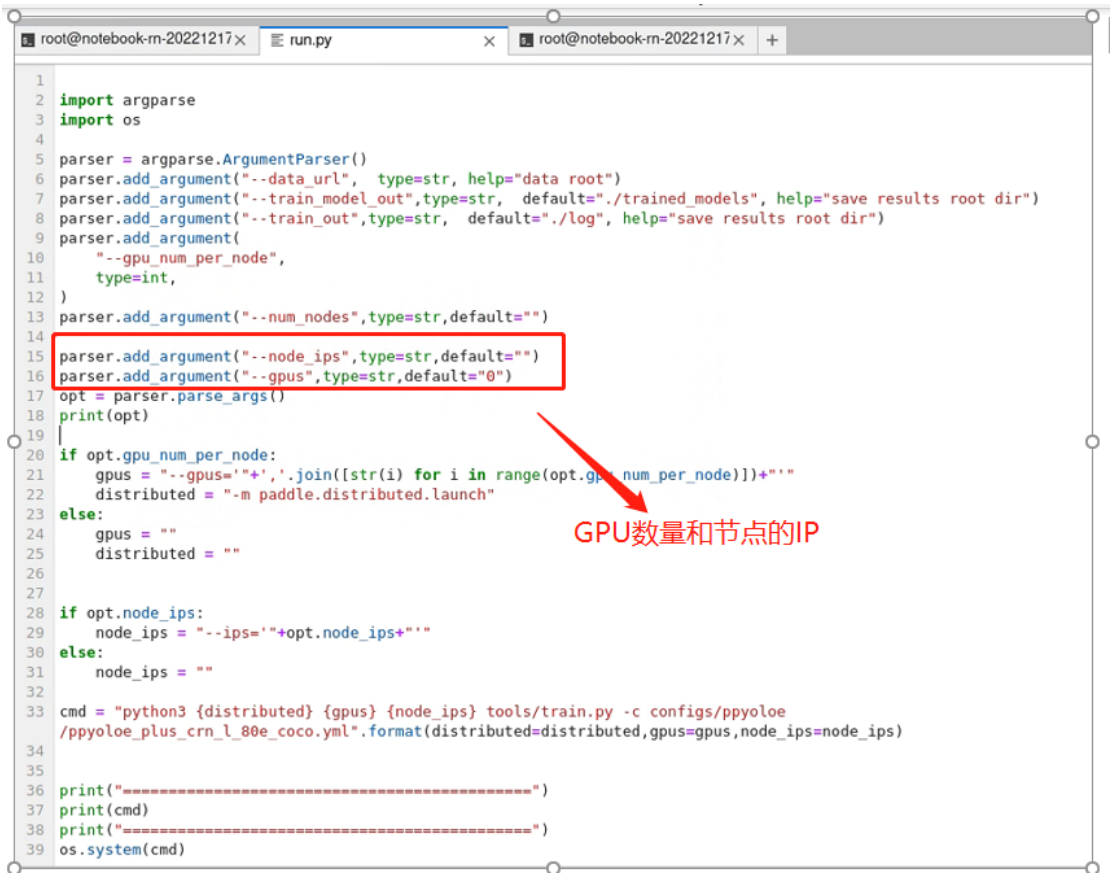
- **算法编写**

➢ 从 https://github.com/PaddlePaddle/PaddleDetection 获取 Paddle 模型算法

➢ 选择 NoteBook 镜像 jupyter-paddle. 创建 notebook 环境

➢ 上传模型算法到 Notebook 环境

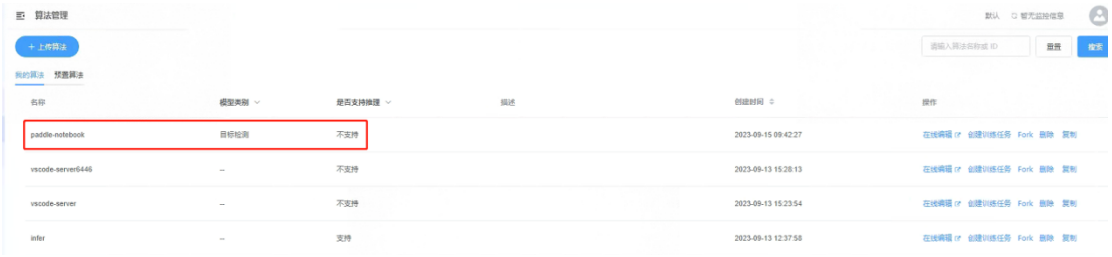
➢ 修改其中的启动代码 run.py 文件,用于接收深度学习平台传递的 GPU 数量和节点 IP 参数




```
import argparse
import os

parser = argparse.ArgumentParser()
parser.add_argument("--data_url",  type=str, help="data root")
parser.add_argument("--train_model_out",type=str,  default="./trained_models", help="save results root dir")
parser.add_argument("--train_out",type=str,  default="./log", help="save results root dir")
parser.add_argument(
    "--gpu_num_per_node",
    type=int,
)
parser.add_argument("--num_nodes",type=str,default="")

parser.add_argument("--node_ips",type=str,default="")
parser.add_argument("--gpus",type=str,default="0")
opt = parser.parse_args()
print(opt)

if opt.gpu_num_per_node:
    gpus = "--gpus='"+','.join([str(i) for i in range(opt.gpu_num_per_node)])+"'"
    distributed = "-m paddle.distributed.launch"
else:
    gpus = ""
    distributed = ""


if opt.node_ips:
    node_ips = "--ips='"+opt.node_ips+"'"
else:
    node_ips = ""

cmd = "python3 {distributed} {gpus} {node_ips} tools/train.py -c configs/ppyoloe
/ppyoloe_plus_crn_l_80e_coco.yml".format(distributed=distributed,gpus=gpus,node_ips=node_ips)


print("==========================================")
print(cmd)
print("==========================================")
os.system(cmd)
```

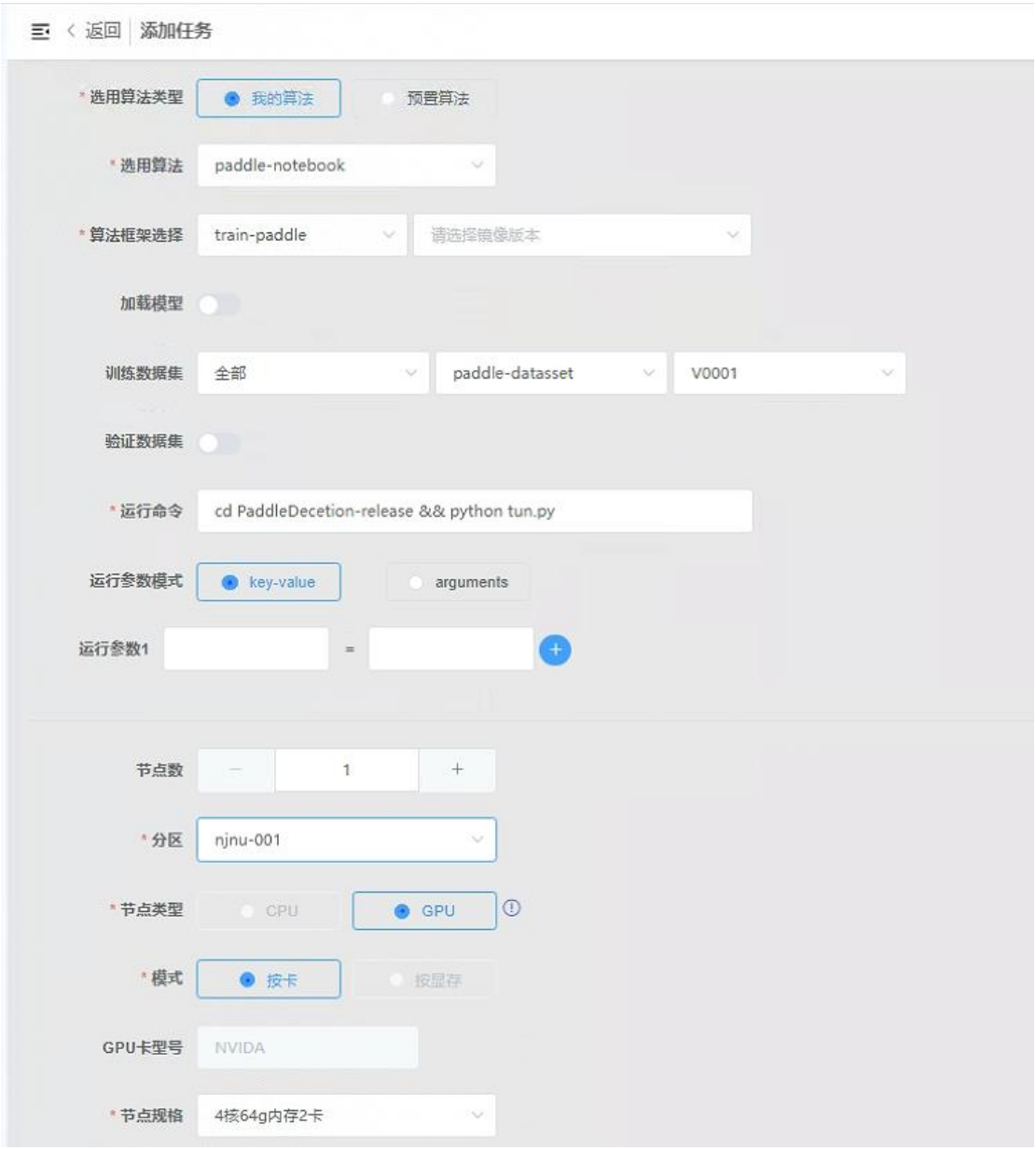
- **保存算法**

修改完毕后，点击保存即可完成新算法的保存



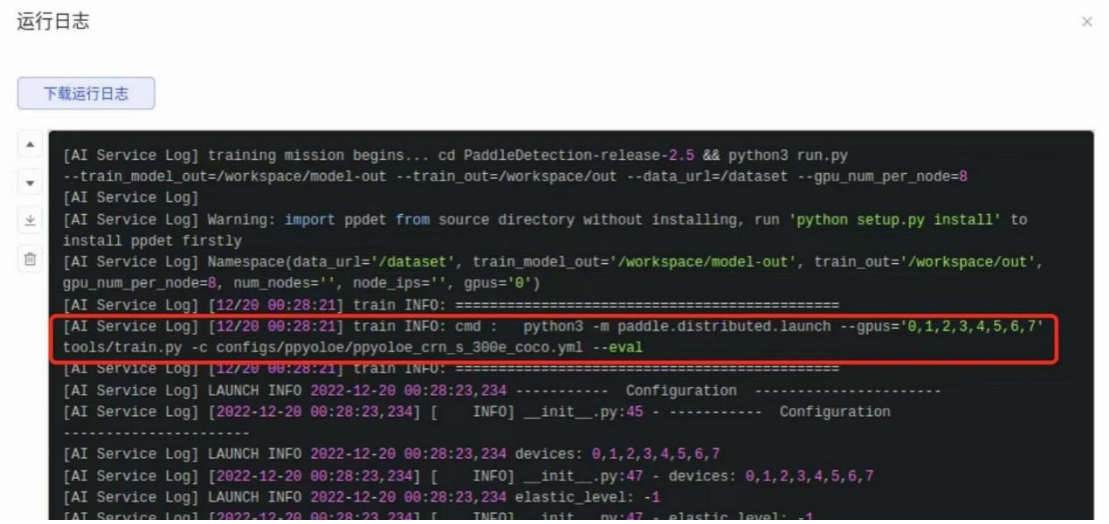
- 9.2.4 提交任务

选择算法 Paddle_NodeBook 以及训练集 paddle-dataset，选择节点数目 1 以及对应的资源规格，点击“开始训练”完成任务的提交

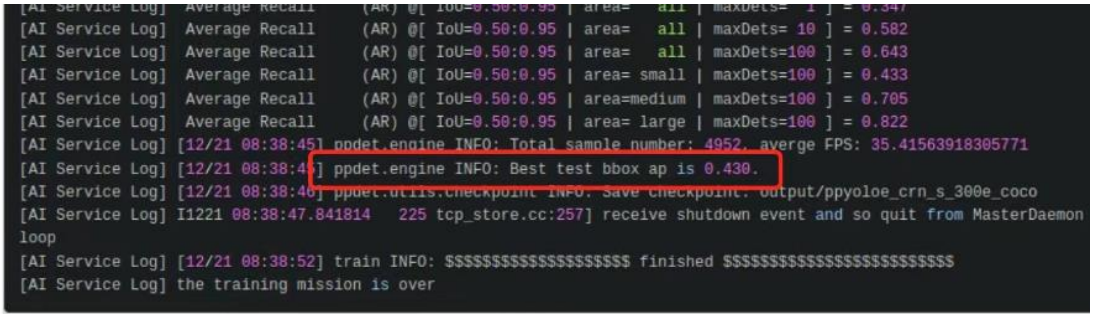


- **查询作业的实施运行日志**

点击作业名称，弹出作业详情，点击“运行日志”，查看作业的实时运行日志内容



- **结果查看**
  结果精度： 43%



2.2.2. Gdal 环境训练

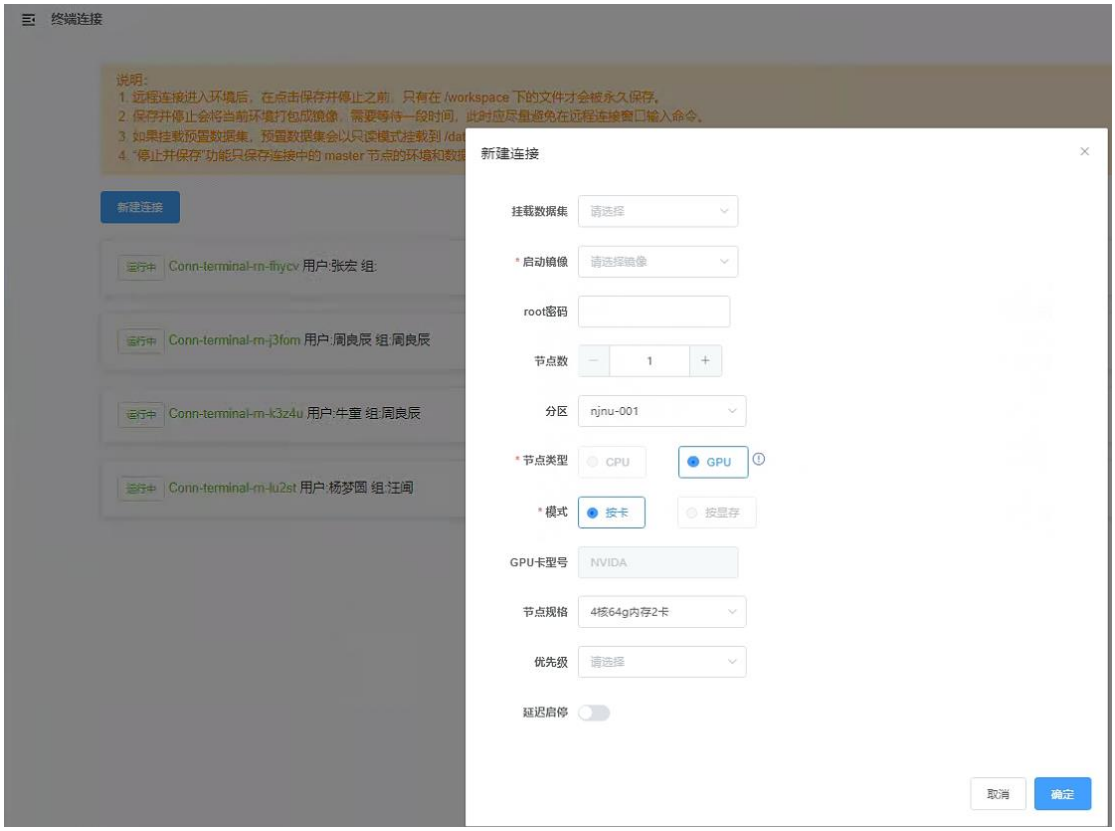
2.2.2.1、终端训练

- **准备工作**

打包终端镜像上传到平台

- **新建远程连接**

用户点击左侧菜单“终端训练”-> “远程连接”，进入任务列表页。
点击“新建连接”按钮，填写需要的资源（CPU、内存、GPU、磁盘）、数据集、远程连接的镜像、 节点数， 选择 Gdal 的平台镜像
点击确定，确认创建远程连接



10.1.2 使用 ssh 进入环境进行相关程序开发

```
exit
root@mgt:/data/Workspace/ImageScript/C_gbol/train# ssh -p 31078 terminal.gpu-cluster.njnu.local
root@terminal.gpu-cluster.njnu.local's password:
Last login: Thu Mar  2 23:16:37 2023 from 10.244.0.0
[root@terminal-rn-ruab70-tigkkarx-d64784c98-fj27t ~]# nvcc --version
nvcc: NVIDIA (R) Cuda compiler driver
Copyright (c) 2005-2022 NVIDIA Corporation
Built on Wed_Jun__8_16:49:14_PDT_2022
Cuda compilation tools, release 11.7, V11.7.99
Build cuda_11.7.r11.7/compiler.31442593_0
[root@terminal-rn-ruab70-tigkkarx-d64784c98-fj27t ~]#
```

2.2.2.2、训练任务训练

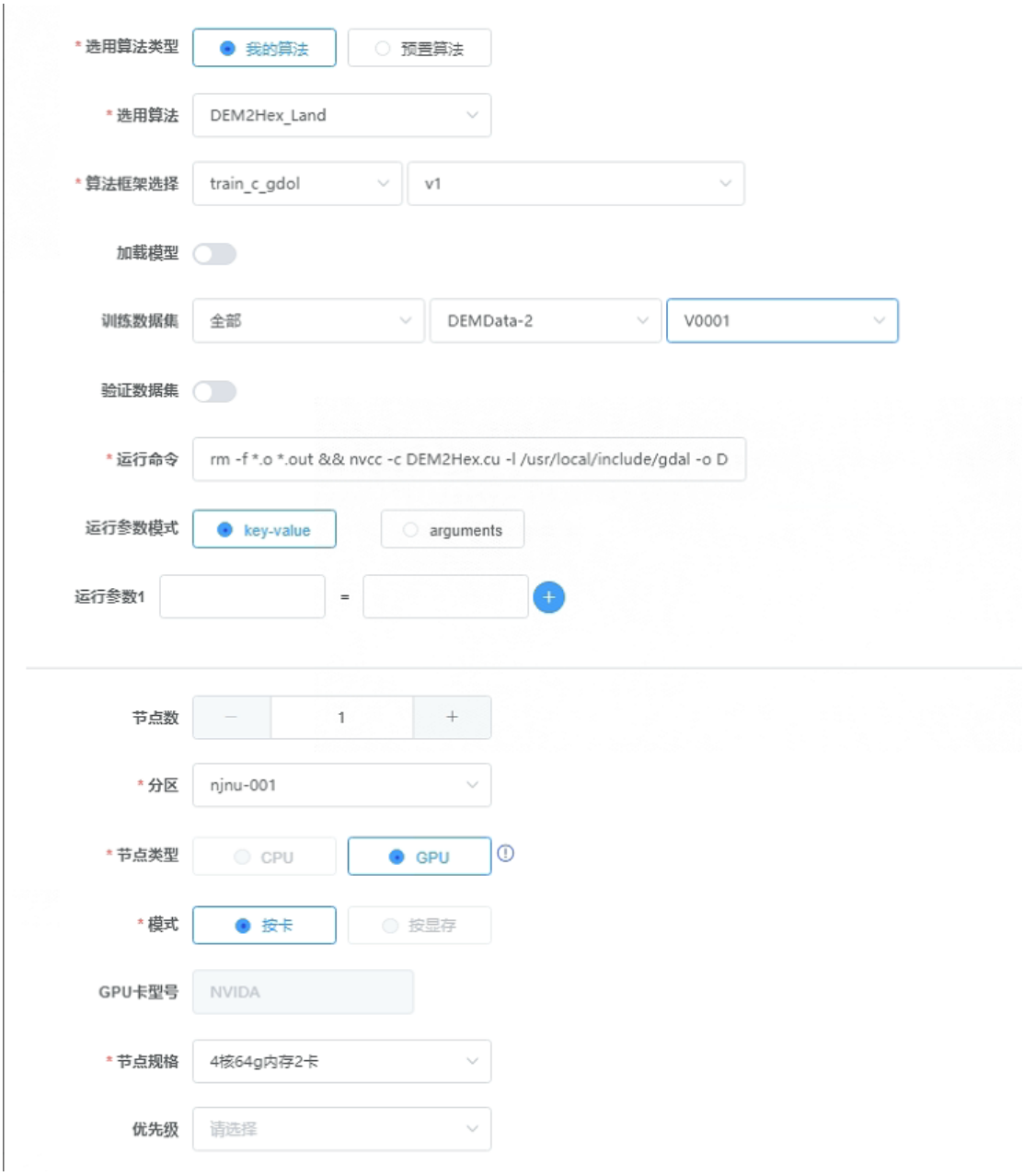
- **准备工作**

打包训练镜像并上传到平台

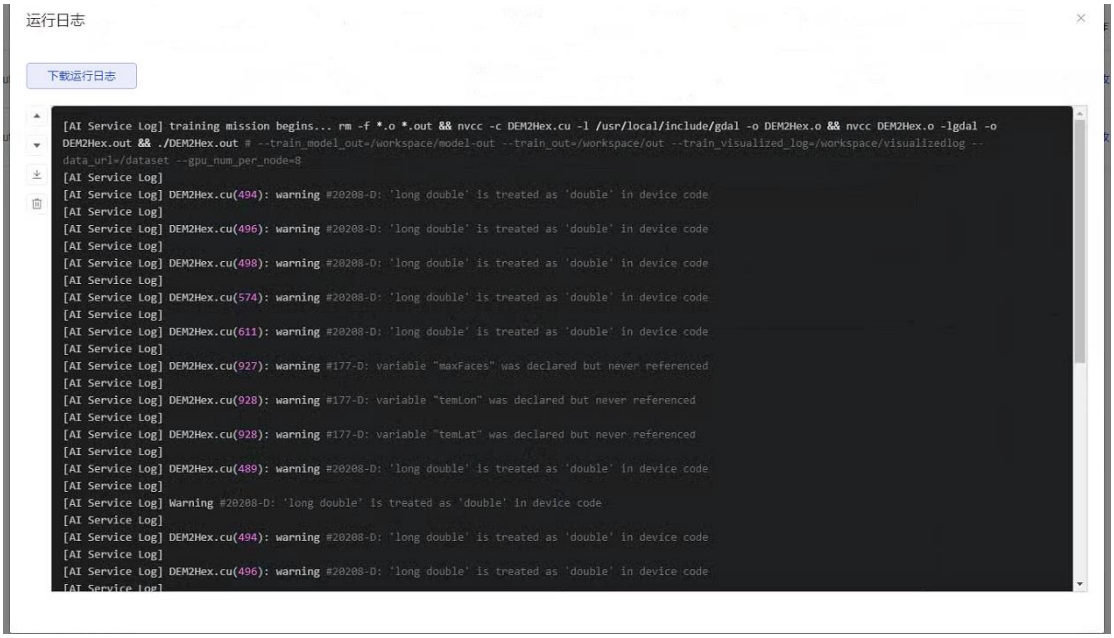
- **上传算法代码**

打开“算法管理”，点击“上传算法”按钮，上传算法

- **创建训练任务**


* 选用算法类型
我的算法
预置算法
* 选用算法
DEM2Hex_Land
* 算法框架选择
train_c_gdol
v1
加载模型
训练数据集
全部
DEMData-2
V0001
验证数据集
* 运行命令
rm -f *.o *.out && nvcc -c DEM2Hex.cu -I /usr/local/include/gdal -o D
运行参数模式
key-value
arguments
运行参数1
=
节点数
1
* 分区
njnu-001
* 节点类型
CPU
GPU
* 模式
按卡
按显存
GPU卡型号
NVIDA
* 节点规格
4核64g内存2卡
优先级
请选择

- 查看训练日志


运行日志
下载运行日志
[AI Service Log] training mission begins... rm -f *.o *.out && nvcc -c DEM2Hex.cu -l /usr/local/include/gdal -o DEM2Hex.o && nvcc DEM2Hex.o -lgdal -o DEM2Hex.out && ./DEM2Hex.out # --train_model_out=/workspace/model-out --train_out=/workspace/out --train_visualized_log=/workspace/visualizedlog --data_url=/dataset --gpu_num_per_node=8
[AI Service Log]
[AI Service Log] DEM2Hex.cu(494): warning #20208-D: 'long double' is treated as 'double' in device code
[AI Service Log]
[AI Service Log] DEM2Hex.cu(496): warning #20208-D: 'long double' is treated as 'double' in device code
[AI Service Log]
[AI Service Log] DEM2Hex.cu(498): warning #20208-D: 'long double' is treated as 'double' in device code
[AI Service Log]
[AI Service Log] DEM2Hex.cu(574): warning #20208-D: 'long double' is treated as 'double' in device code
[AI Service Log]
[AI Service Log] DEM2Hex.cu(611): warning #20208-D: 'long double' is treated as 'double' in device code
[AI Service Log]
[AI Service Log] DEM2Hex.cu(927): warning #177-D: variable "maxFaces" was declared but never referenced
[AI Service Log]
[AI Service Log] DEM2Hex.cu(928): warning #177-D: variable "temLon" was declared but never referenced
[AI Service Log]
[AI Service Log] DEM2Hex.cu(928): warning #177-D: variable "temLat" was declared but never referenced
[AI Service Log]
[AI Service Log] DEM2Hex.cu(489): warning #20208-D: 'long double' is treated as 'double' in device code
[AI Service Log]
[AI Service Log] Warning #20208-D: 'long double' is treated as 'double' in device code
[AI Service Log]
[AI Service Log] DEM2Hex.cu(494): warning #20208-D: 'long double' is treated as 'double' in device code
[AI Service Log]
[AI Service Log] DEM2Hex.cu(496): warning #20208-D: 'long double' is treated as 'double' in device code
[AI Service Log]