

# GPU 云服务器

## AI 优化

## 产品文档



腾讯云

**【版权声明】**

©2013-2024 腾讯云版权所有

本文档著作权归腾讯云单独所有，未经腾讯云事先书面许可，任何主体不得以任何形式复制、修改、抄袭、传播全部或部分本文档内容。

**【商标声明】**

及其它腾讯云服务相关的商标均为腾讯云计算（北京）有限责任公司及其关联公司所有。本文档涉及的第三方主体的商标，依法由权利人所有。

**【服务声明】**

本文档意在向客户介绍腾讯云全部或部分产品、服务的当时的整体概况，部分产品、服务的内容可能有所调整。您所购买的腾讯云产品、服务的种类、服务标准等应由您与腾讯云之间的商业合同约定，除非双方另有约定，否则，腾讯云对本文档内容不做任何明示或模式的承诺或保证。

---

## 文档目录

### AI 优化

- 训练加速引擎 TACO Train  
部署及实践

# AI 优化

## 训练加速引擎 TACO Train

### 部署及实践

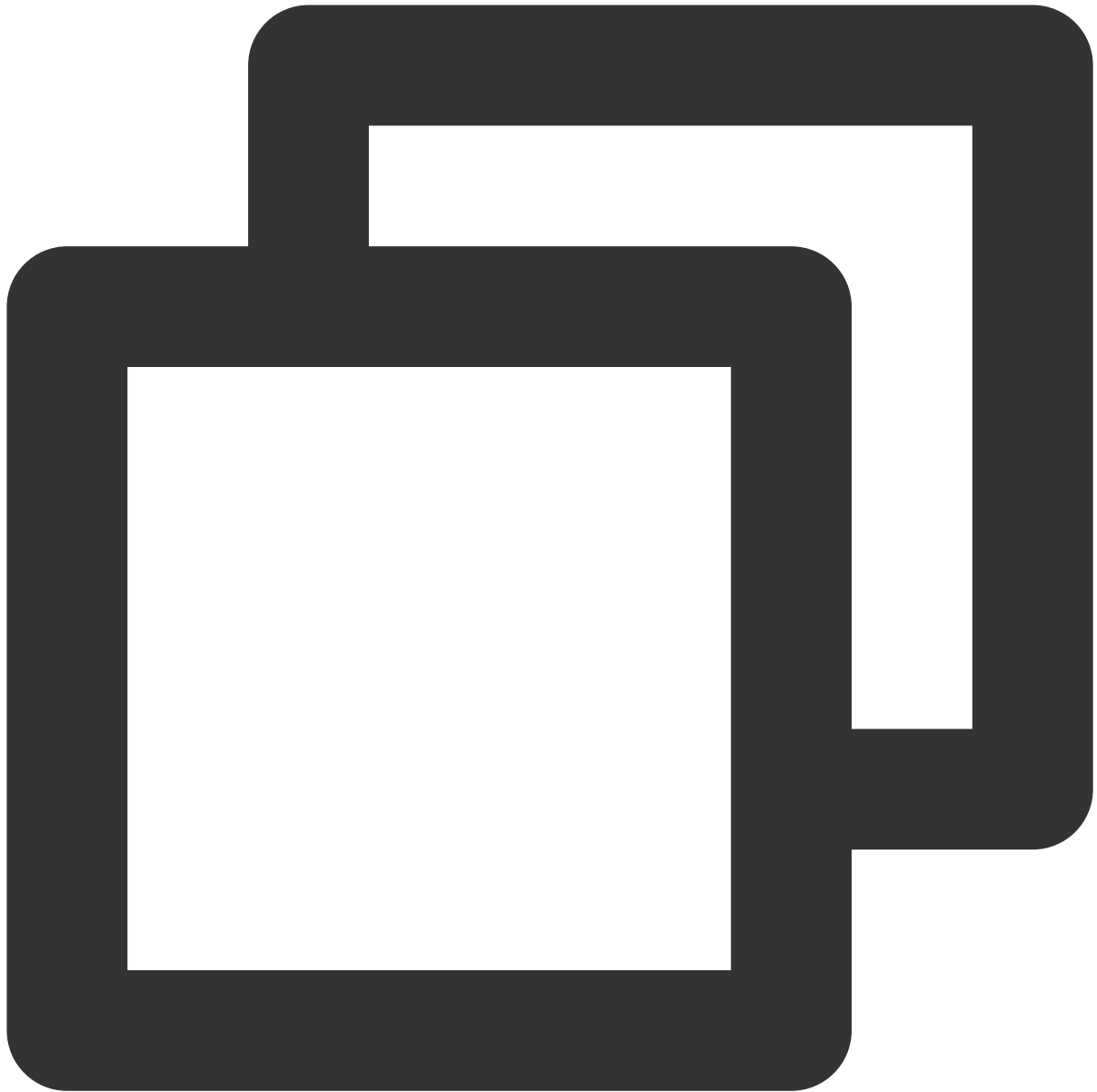
最近更新时间：2024-03-27 14:22:10

本文介绍如何在 GPU 云服务器实例上部署及使用 TACO-Training。

## 说明事项

目前 TACO-Training 仅支持 GPU 云服务器。

目前 TACO-Training 的三个加速组件已集成至同一个 Docker 镜像中，拉取镜像地址如下：



```
ccr.ccs.tencentyun.com/qcloud/taco-training:cu112-cudnn81-py3-0.3.2
```

## 操作步骤

### 实例环境准备

1. 参考购买 NVIDIA GPU 实例创建至少两台实例。其中：  
**实例**：推荐选择 [计算型 GT4](#) 实例 GT4.41XLARGE948 八卡机型。

**镜像：**请选择 CentOS 7.8或 Ubuntu 18.04及以上版本。并勾选“后台自动安装GPU驱动”，使用自动安装功能安装 GPU 驱动。

CUDA 及 cuDNN 的自动安装非本次部署的必选项，您可根据实际情况选择。如下图所示：



**系统盘：**考虑到 Docker 镜像的大小以及训练中间状态文件的存储，推荐配置100G以上的系统盘。

2. 请对应实例的操作系统类型，参考以下文档安装 Docker。

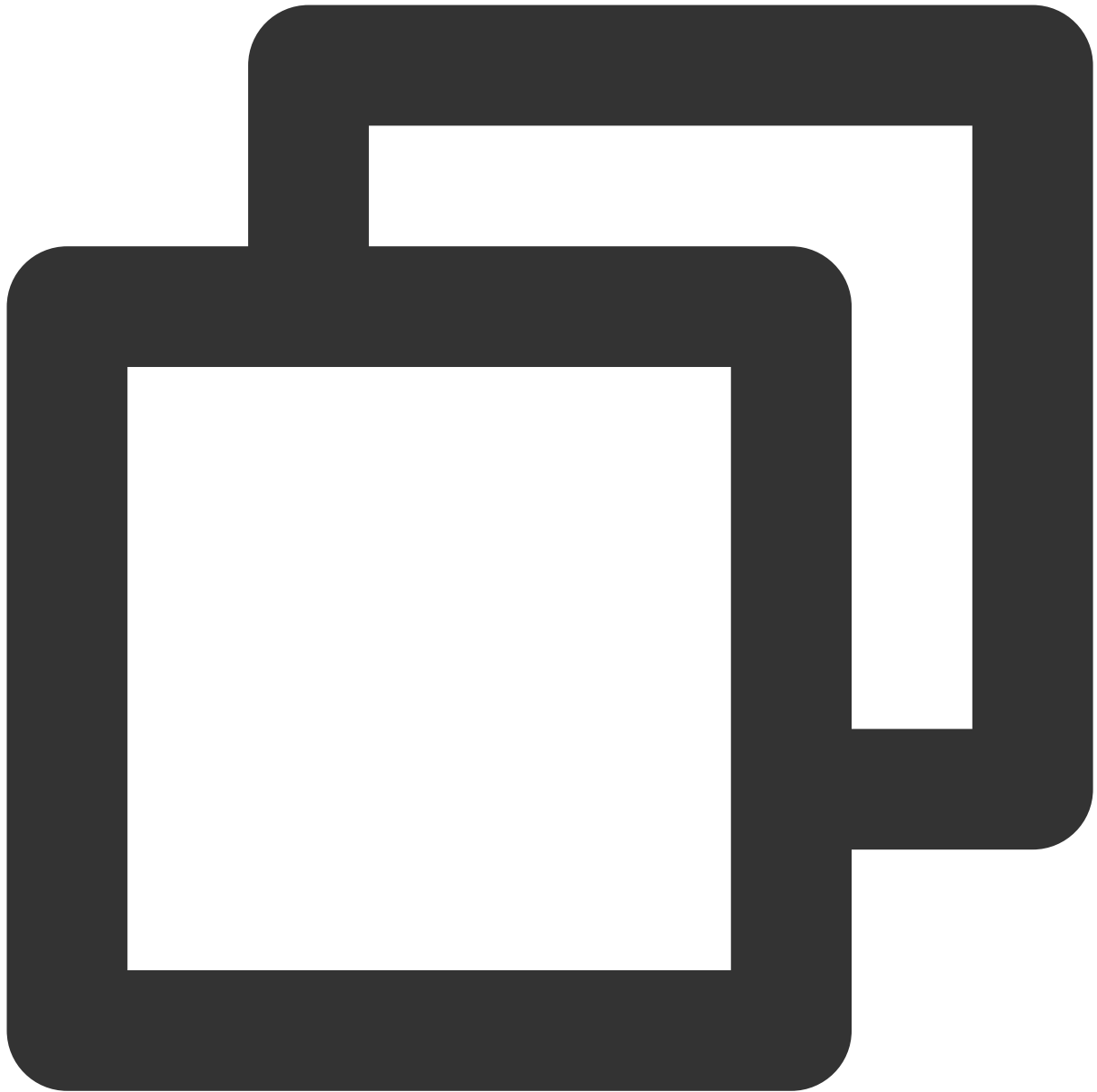
操作系统	说明
CentOS	参考 <a href="#">Docker 官方文档 - 在 CentOS 中安装 Docker</a> 进行安装。
Ubuntu	参考 <a href="#">Docker 官方文档 - 在 Ubuntu 中安装 Docker</a> 进行安装。

3. 安装 nvidia-docker，详情请参见 [NVIDIA 官方文档 - 安装nvidia-docker](#)。

## 使用 TTF

### 安装

1. 以 root 用户身份执行以下命令，通过 Docker 镜像的方式安装 TTF。



```
docker pull ccr.ccs.tencentyun.com/qcloud/taco-training:cu112-cudnn81-py3-0.3.2
```

2. 执行以下命令，启动 Docker。



```
docker run -it --rm --gpus all --shm-size=32g --ulimit memlock=-1 --ulimit stack=67
```

3. 执行以下命令，查看 TTF 版本。





```
pip show ttensorflow
```

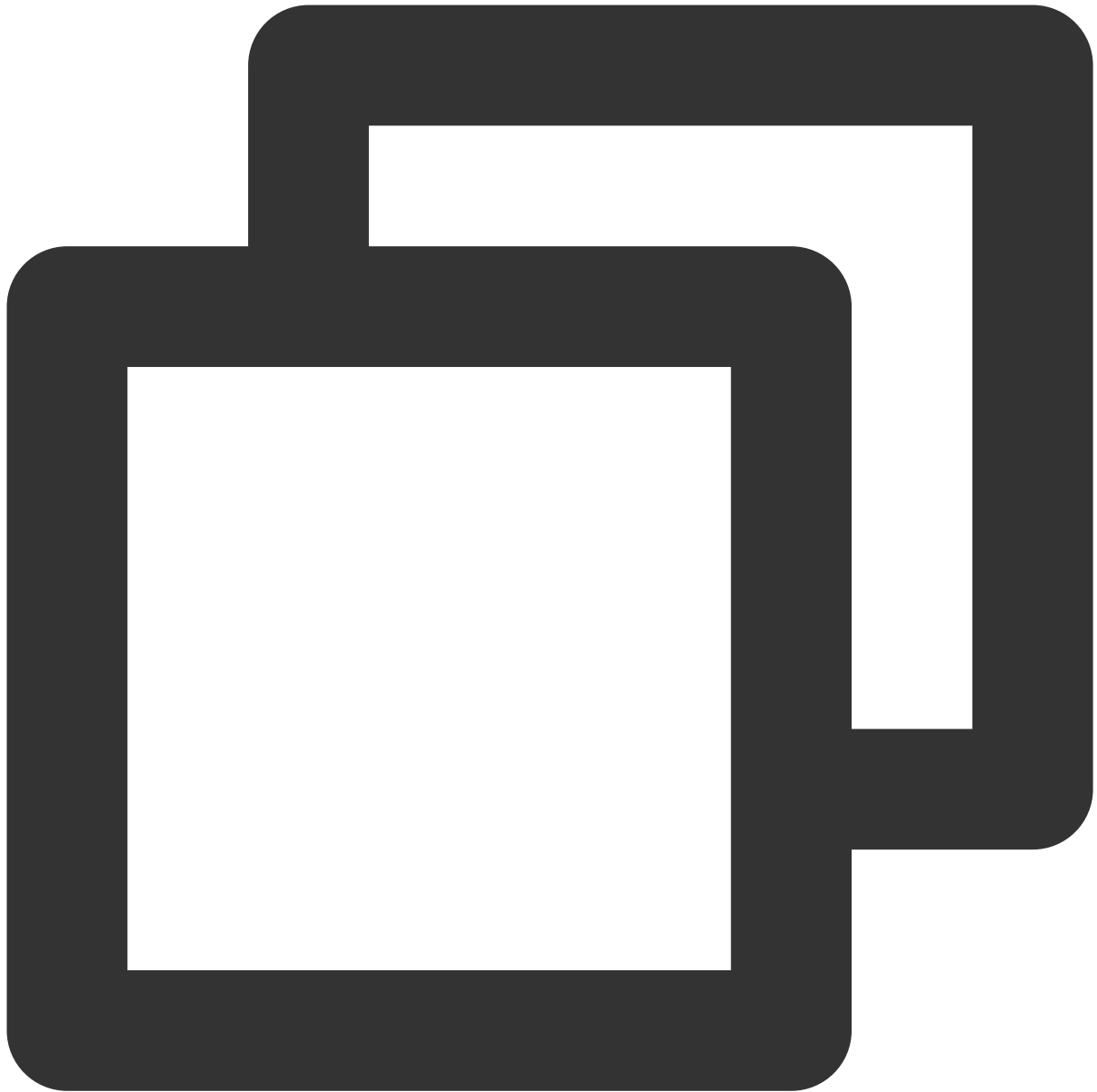
## 模型适配

### 动态 embedding

TF 原生的静态 embedding 及 TTF 提供的动态 embedding 代码如下：

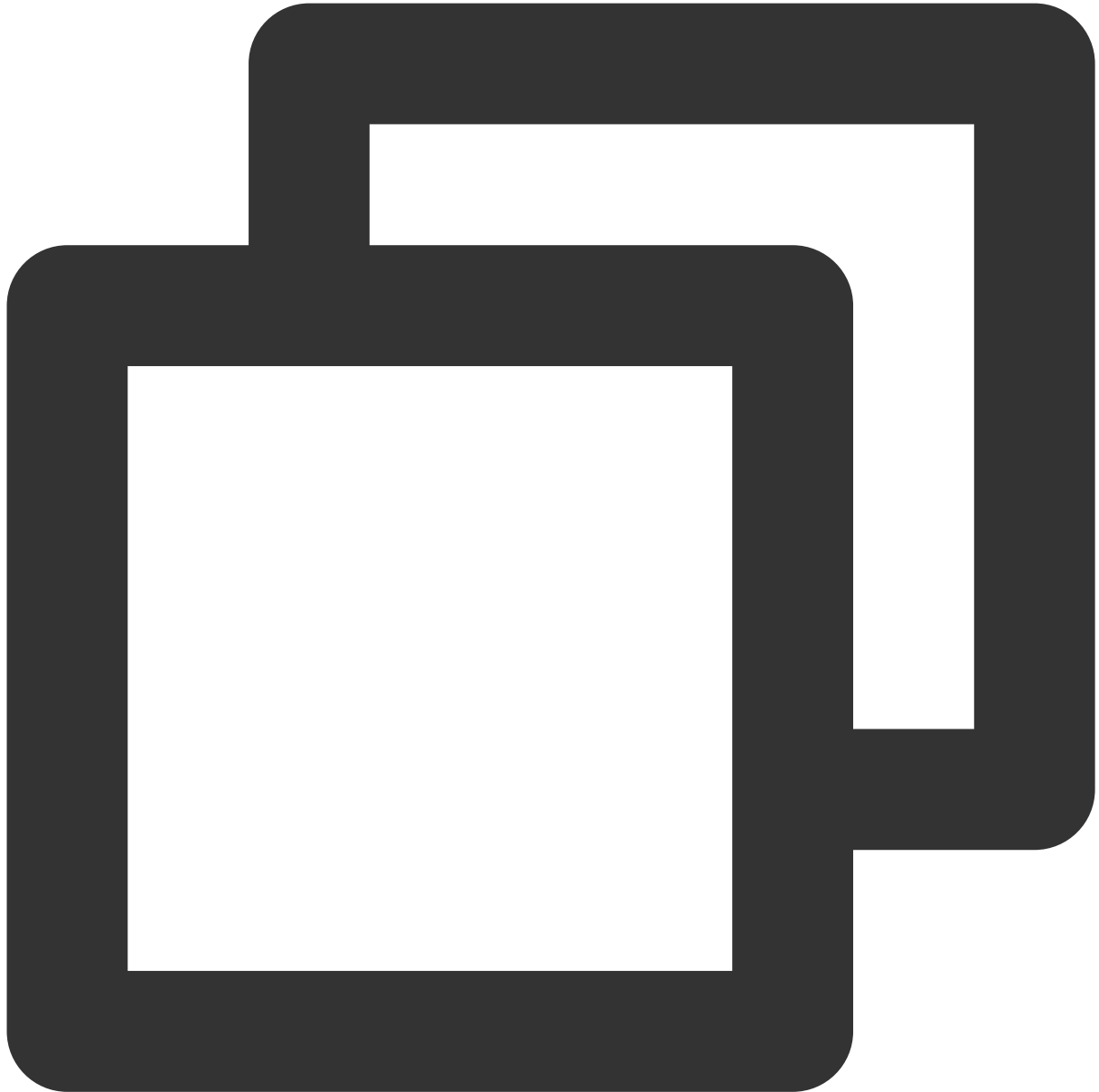
TF 原生的静态 embedding

TTF 提供的动态 embedding



```
deep_dynamic_variables = tf.get_variable(  
    name="deep_dynamic_embeddings",  
    initializer=tf.compat.v1.random_normal_initializer(0, 0.005),  
    shape=[100000000, self.embedding_size])  
deep_sparse_weights = tf.nn.embedding_lookup(  
    params=deep_dynamic_variables,  
    ids=ft_sparse_val,  
    name="deep_sparse_weights")  
deep_embedding = tf.gather(deep_sparse_weights, ft_sparse_idx)  
deep_embedding = tf.reshape(  
    deep_embedding,
```

```
shape=[self.batch_size, self.feature_num * self.embedding_size])
```



```
deep_dynamic_variables = tf.dynamic_embedding.get_variable(  
    name="deep_dynamic_embeddings",  
    initializer=tf.compat.v1.random_normal_initializer(0, 0.005),  
    dim=self.embedding_size,  
    devices=["/{}:0".format(FLAGS.device)],  
    init_size=100000000)  
deep_sparse_weights = tf.dynamic_embedding.embedding_lookup(  
    params=deep_dynamic_variables,  
    ids=ft_sparse_val,
```

```
name="deep_sparse_weights")
deep_embedding = tf.gather(deep_sparse_weights, ft_sparse_idx)
deep_embedding = tf.reshape(
    deep_embedding,
    shape=[self.batch_size, self.feature_num * self.embedding_size])
```

对比后可查看 TTF 主要对以下两部分内容进行了替换，使用非常简单：

embedding 使用 `tf.dynamic_embedding.get_variable`，详情请参见 [tensorflow 动态 embedding 说明文档](#)。

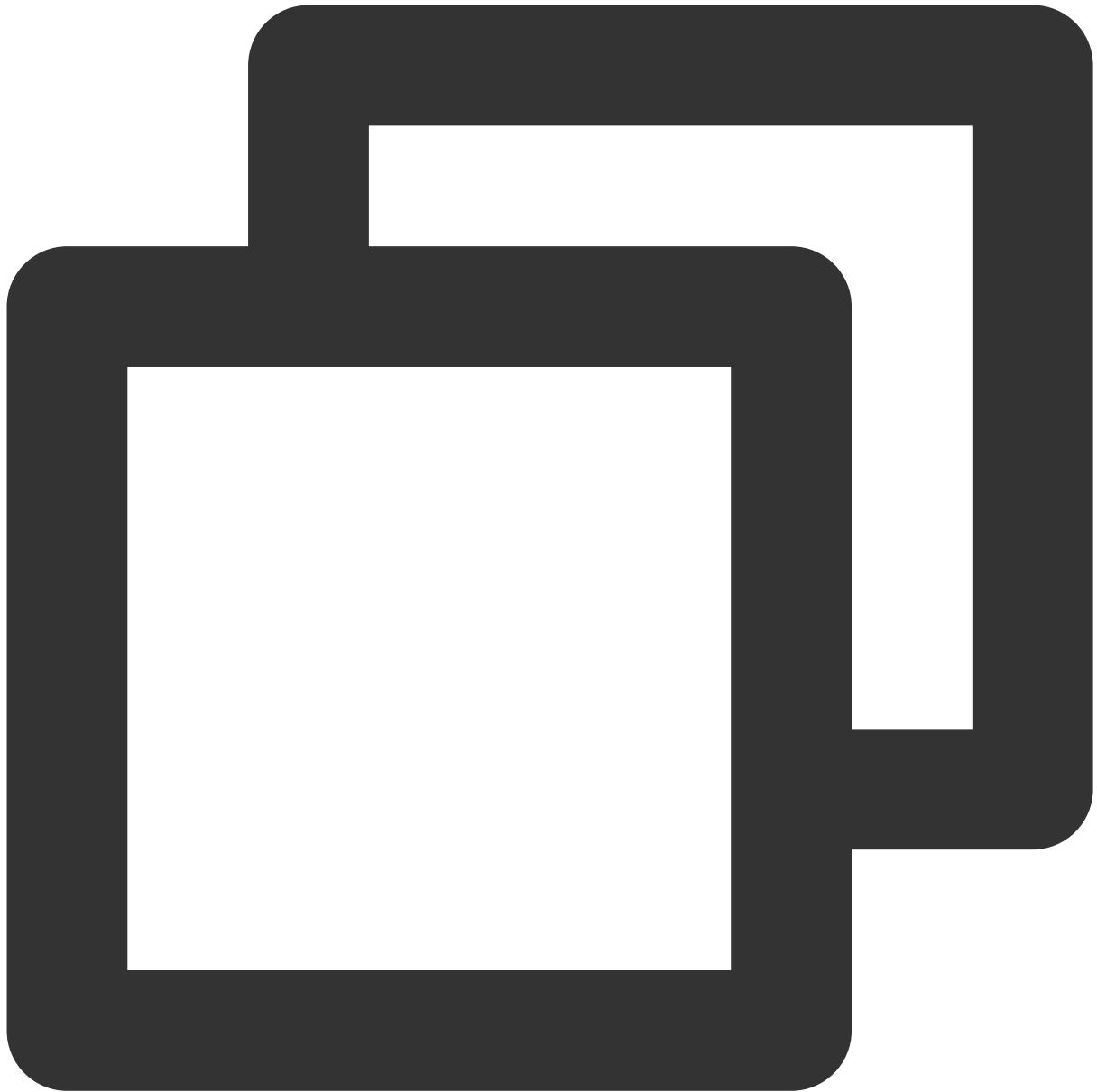
lookup 使用 `tf.dynamic_embedding.embedding_lookup`，详情请参见 [tensorflow 动态 embedding 说明文档](#)。

详细的 API 使用说明文档请参见 [tensorflow 动态 embedding](#)。

### 混合精度

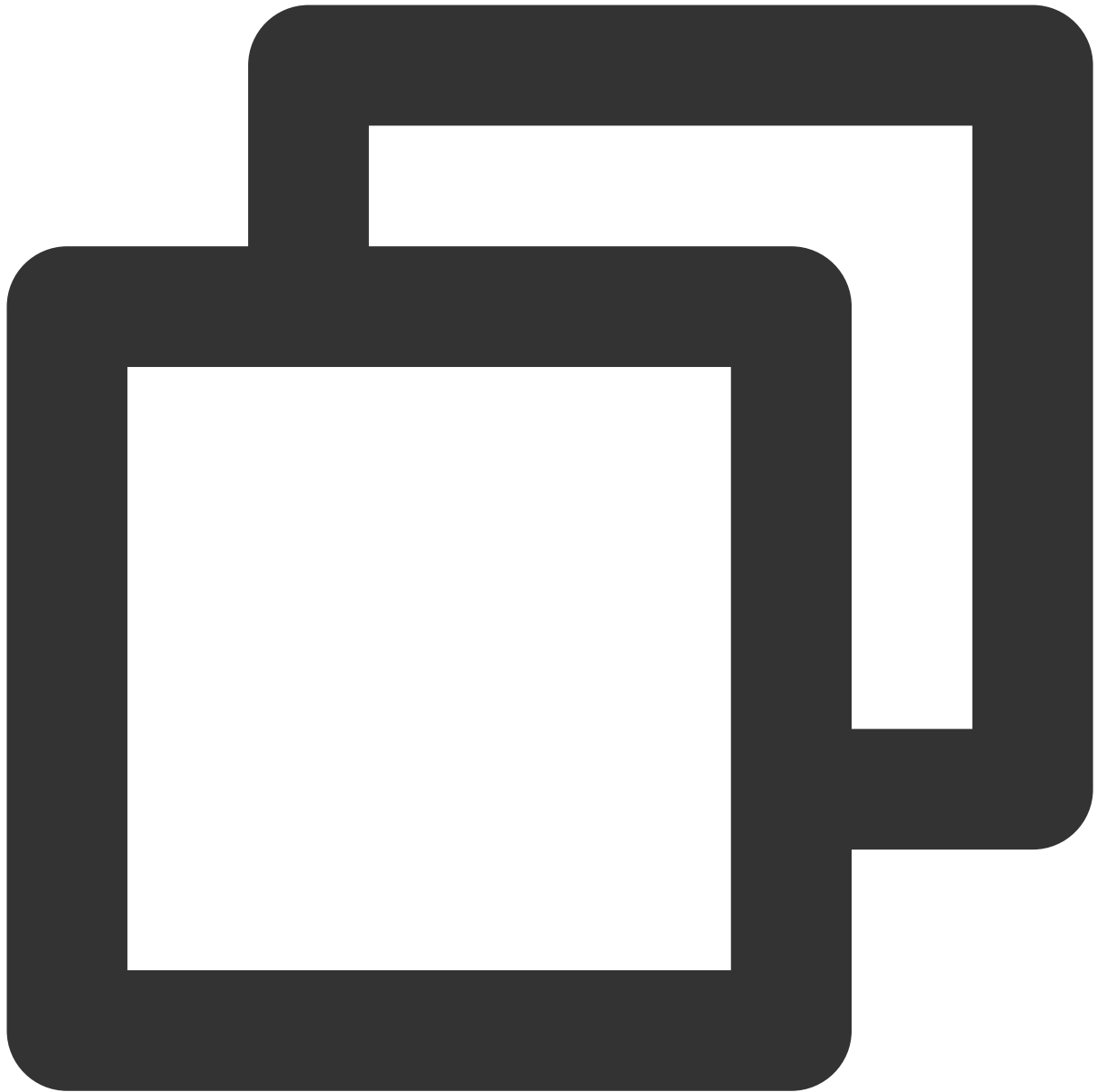
混合精度既可以通过代码对优化器进行重写，也可通过修改环境变量实现。如下所示：

代码修改的方式



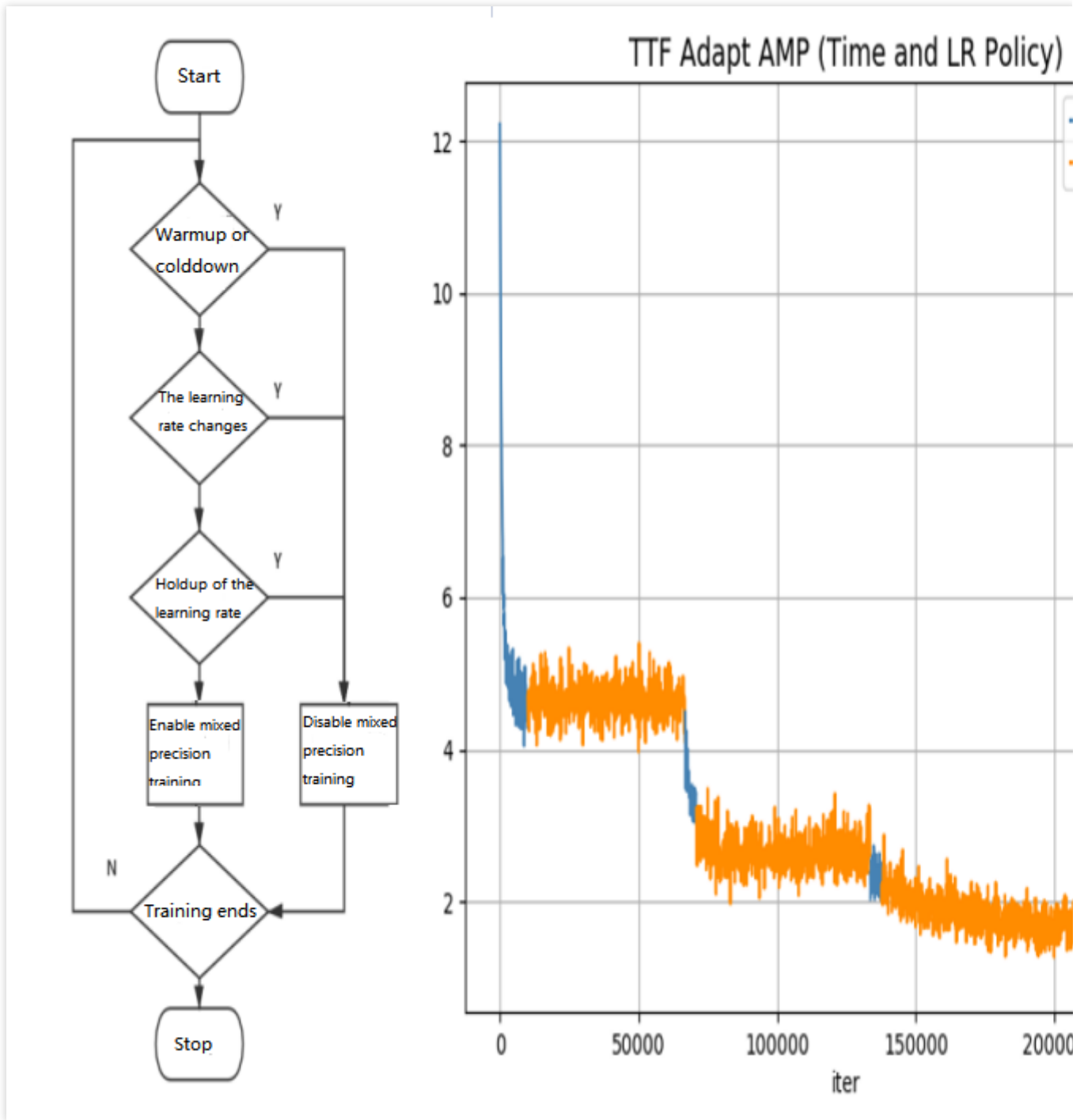
```
opt = tf.train.experimental.enable_mixed_precision_graph_rewrite(opt)
```

环境变量的方式



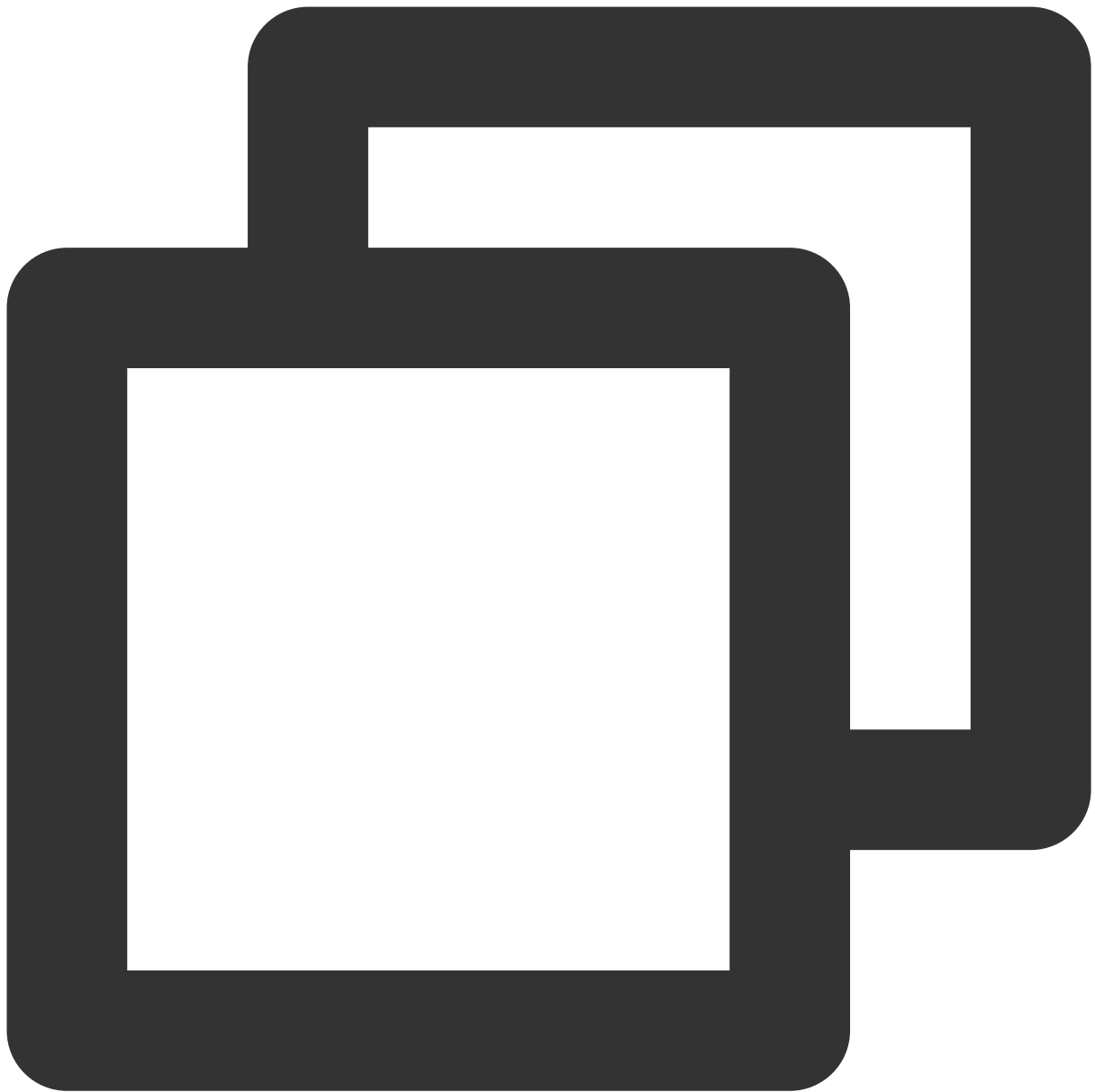
```
export TF_ENABLE_AUTO_MIXED_PRECISION=1
```

TTF 混合精度在 imagenet 数据集上训练 resnet50 的 loss 变化曲线示例。如下图所示：



**XLA**

XLA 既可以通过代码进行配置，也可以通过环境变量设置。如下所示：  
 代码修改的方式



```
config = tf.ConfigProto()  
config.graph_options.optimizer_options.global_jit_level = tf.OptimizerOptions.ON_1  
sess = tf.Session(config=config)
```

环境变量的方式



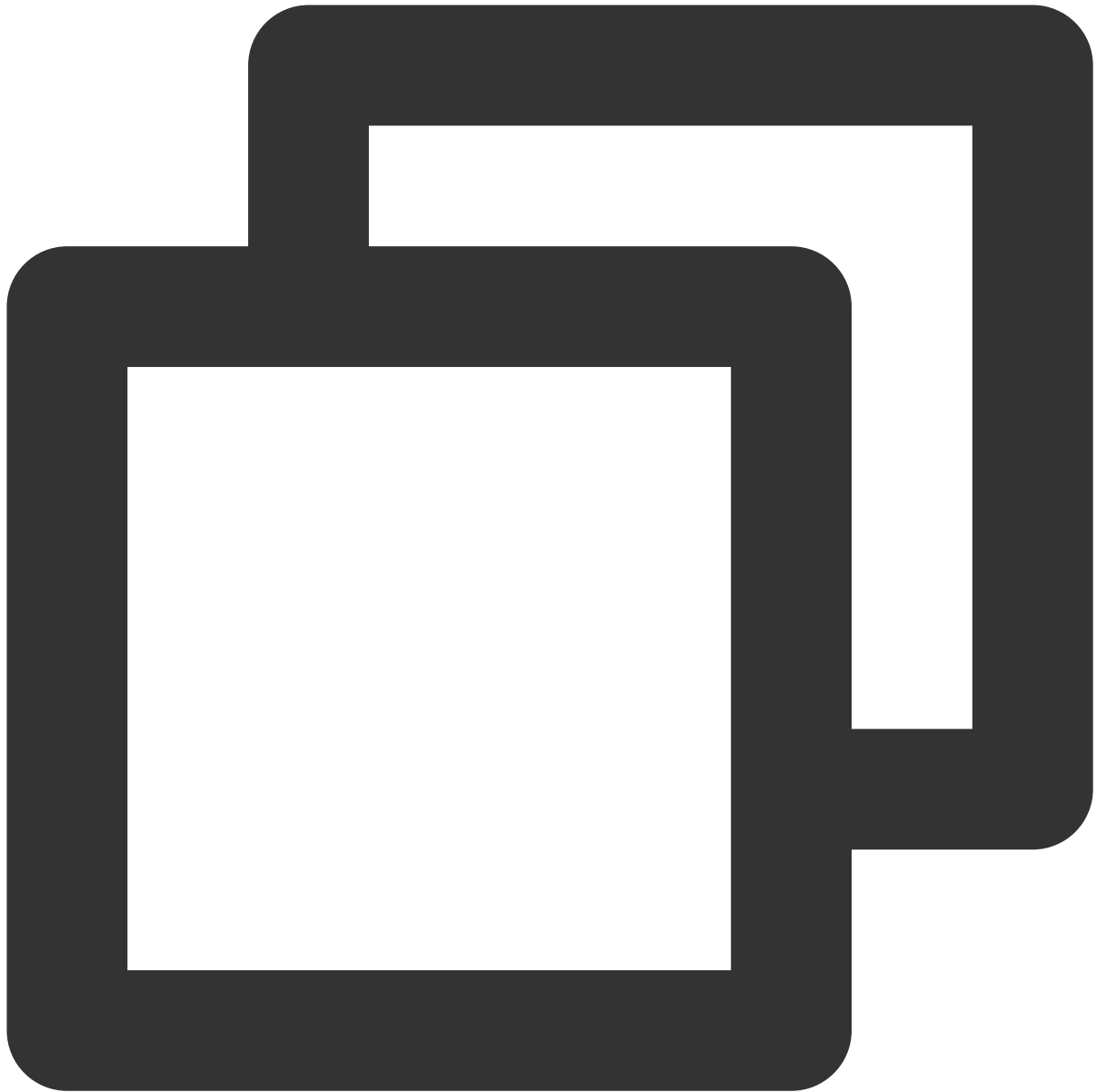


```
TF_XLA_FLAGS=--tf_xla_auto_jit=1
```

## Demo

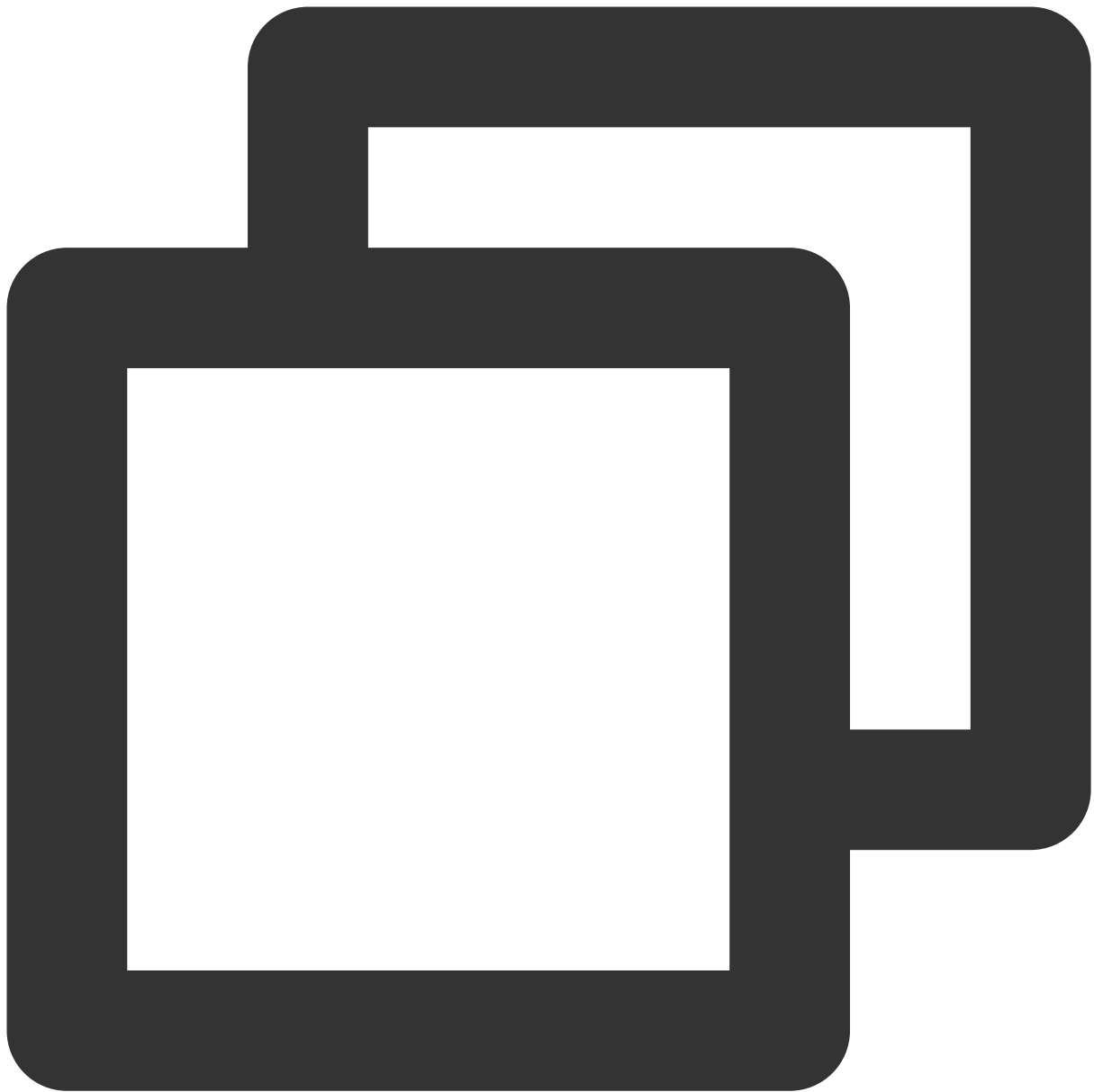
在运行 Demo 前：

1. 执行以下命令，进入 demo 目录。



```
cd /opt/dynamic-embedding-demo
```

2. 执行以下命令，下载数据集。



```
bash download_dataset.sh
```

您可根据以下 Demo，快速了解并使用 TTF。

### **benchmark**

此 Demo 用来对比测试动态 embedding 和原生静态 embedding 的性能：



```
#进入benchmark目录
cd benchmark
#按照默认配置运行
python train.py

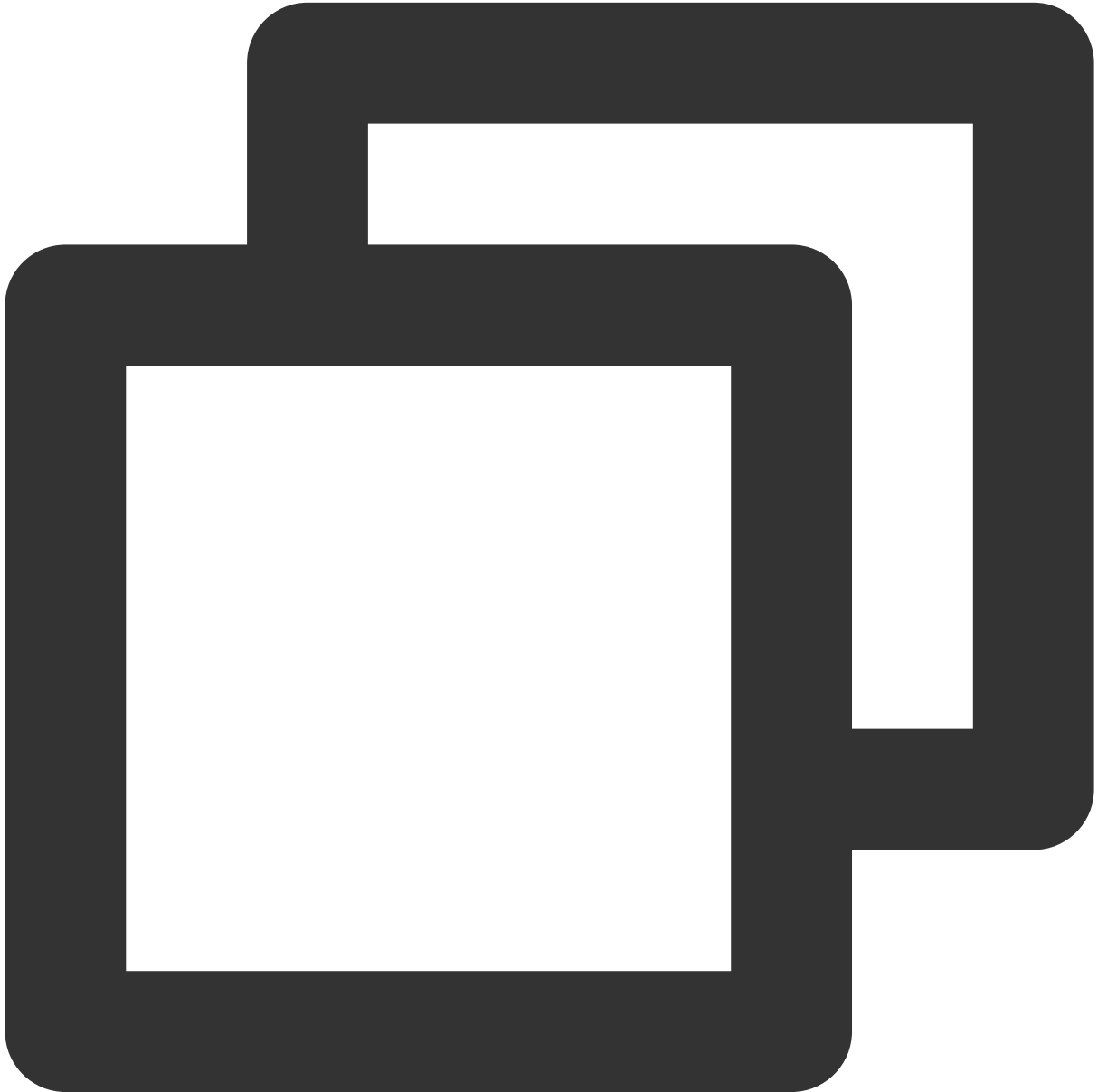
#每次修改batch size, 需要将本地数据集缓存文件删掉
rm -f .index .data-00000-of-00001
python train.py --batch_size=16384

#分别使用静态embedding和动态embedding进行DeepFM模型训练
python train.py --batch_size=16384 --is_dynamic=False
```

```
python train.py --batch_size=16384 --is_dynamic=True  
  
#调整Deep部分的fc层数  
python train.py --batch_size=16384 --dnn_layer_num=12
```

## ps

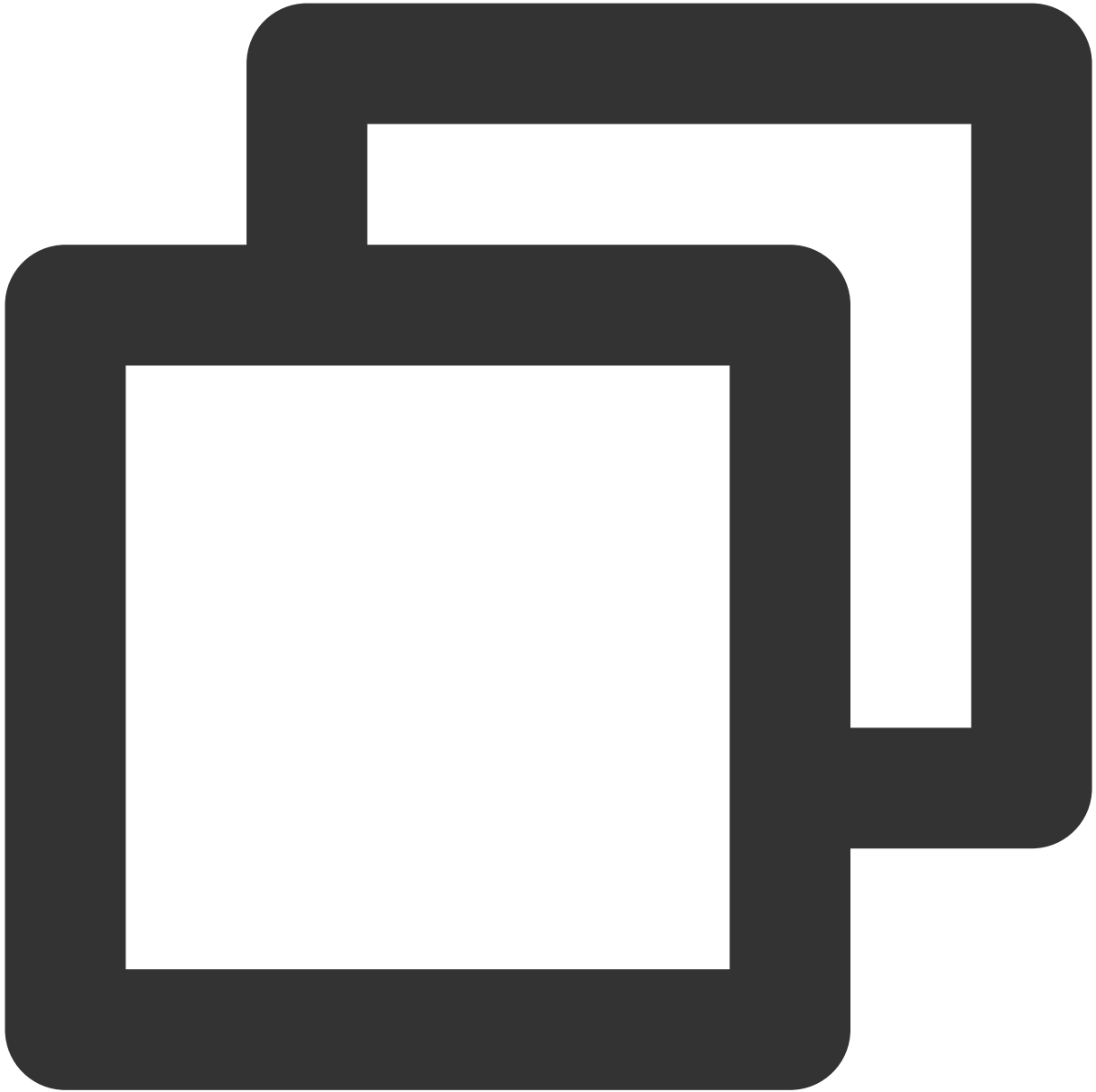
此 Demo 用来展示如何在 ps 模式下使用动态 embedding :



```
cd ps && bash start.sh
```

## Estimator

此 Demo 用来展示如何在 Estimator 模式下使用动态 embedding：

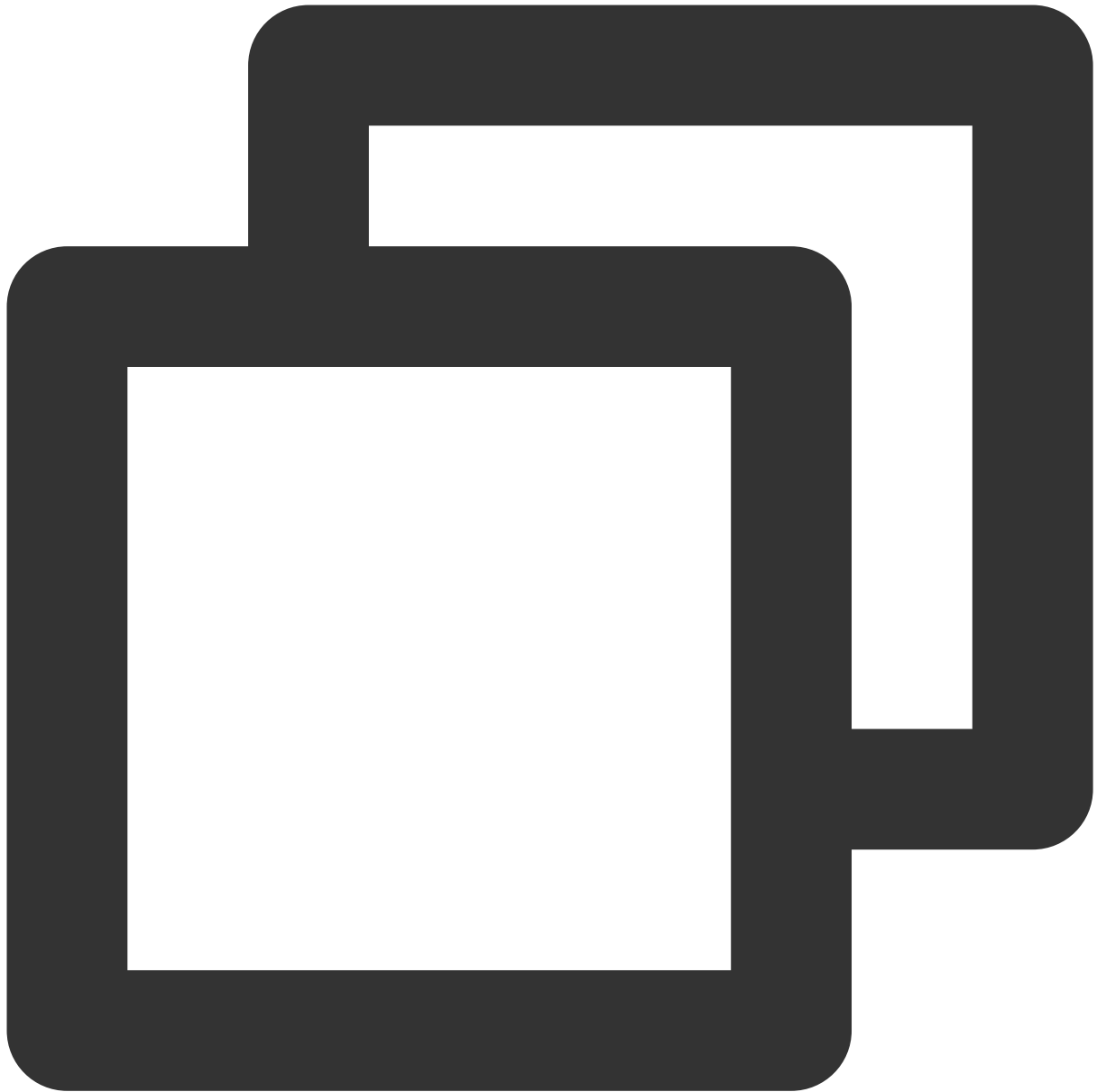


```
cd estimator && bash start.sh
```

## 使用 LightCC

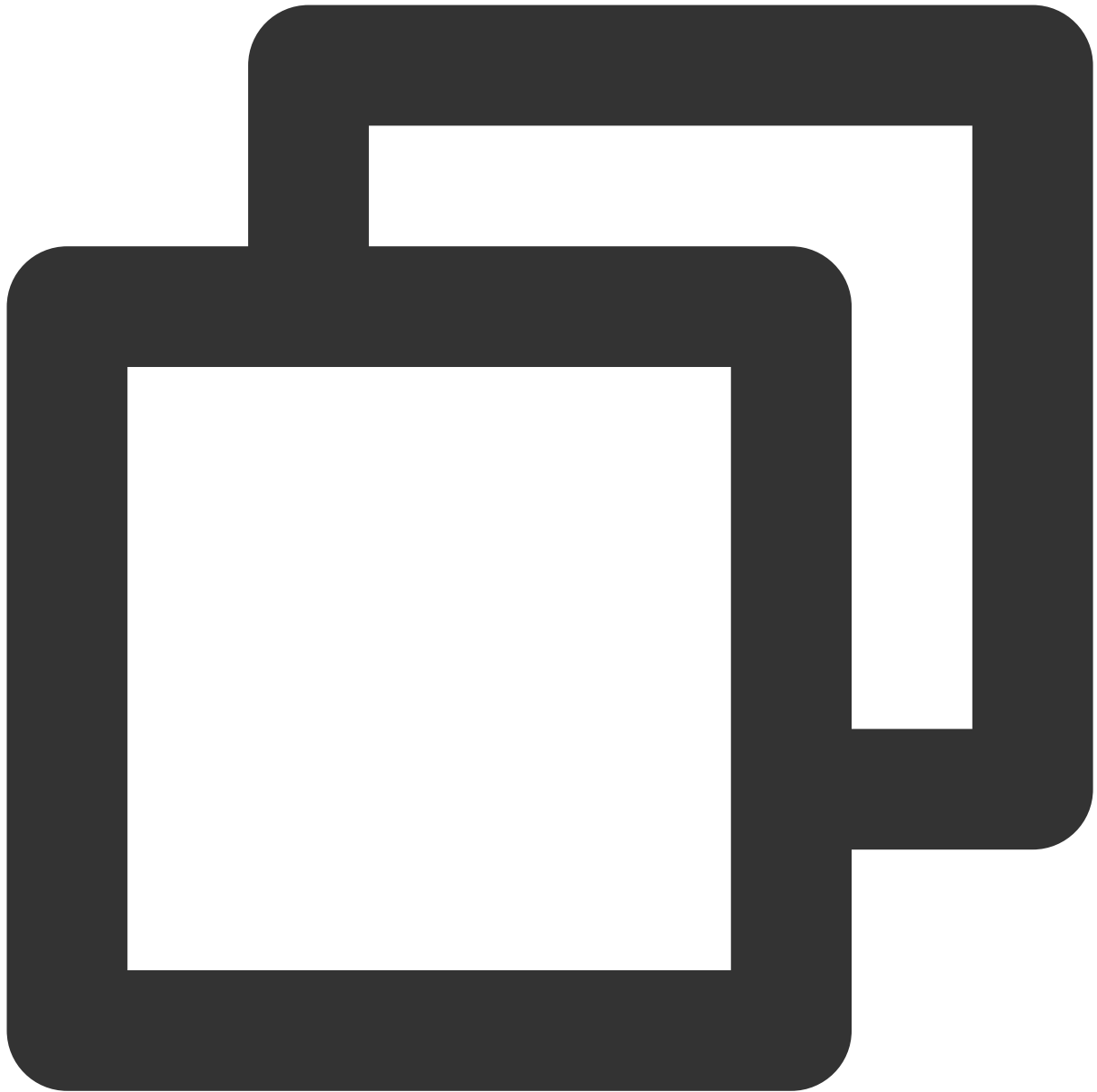
### 安装

1. 以 root 用户身份执行以下命令，通过 Docker 镜像的方式安装 LightCC。



```
docker pull ccr.ccs.tencentyun.com/qcloud/taco-training:cu112-cudnn81-py3-0.3.2
```

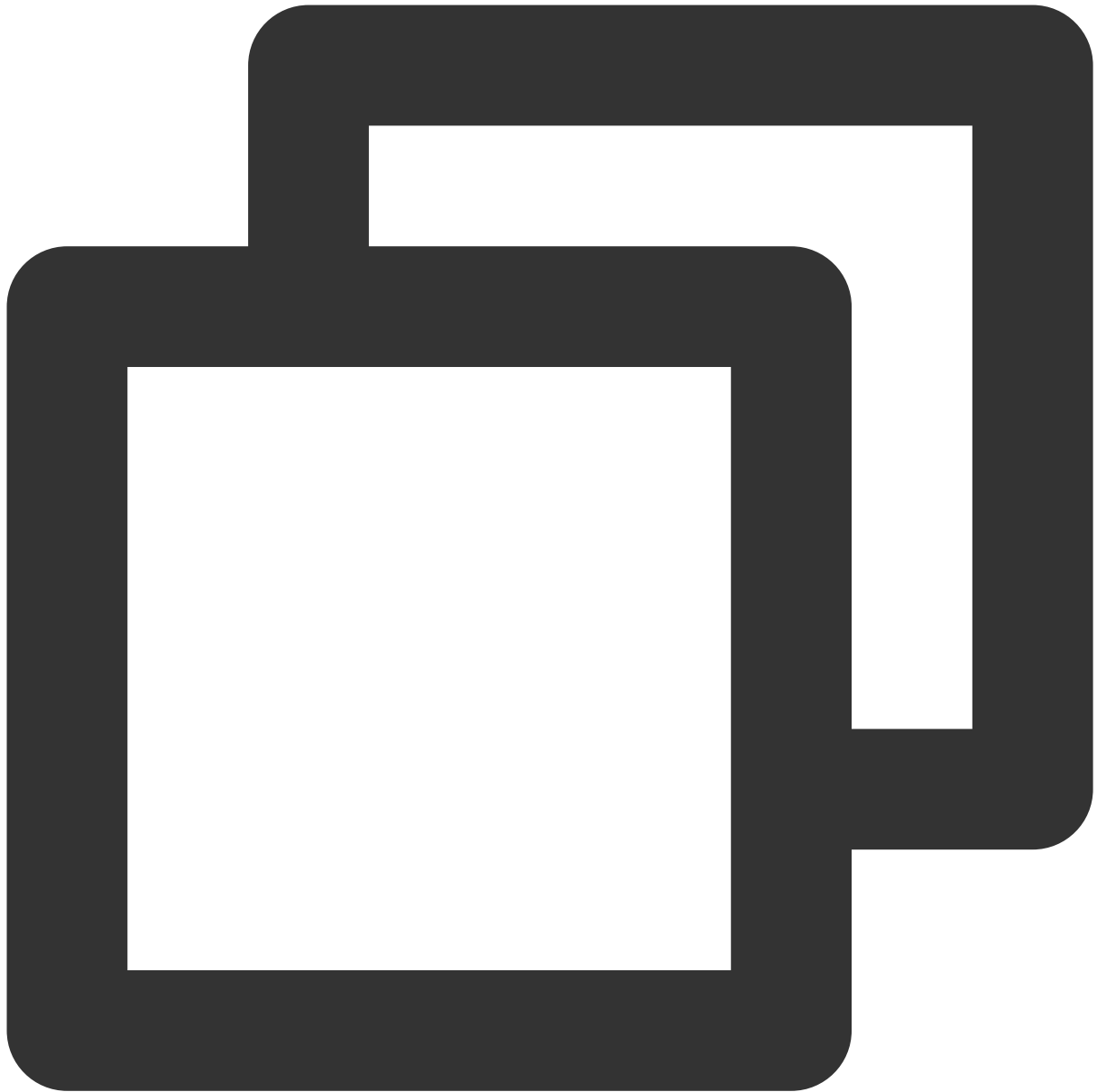
2. 执行以下命令，启动 Docker。



```
docker run --network host -it --rm --gpus all --privileged --shm-size=32g --ulimit
```

3. 执行以下命令，查看 LightCC 版本。





```
pip show light-horovod
```

#### 注意：

当使用内核协议栈进行 NCCL 网络通信，或未配置好 HARP 协议栈的运行环境时，需要将镜像内部的 `/usr/lib/x86_64-linux-gnu/libnccl-net.so` 文件移动到系统 `lib` 库目录之外（例如 `/root` 目录下）。该 `lib` 库中在 `init` 时会检查特定目录下的 HARP 配置文件是否存在，如果不存在程序会报错。

#### 环境变量配置

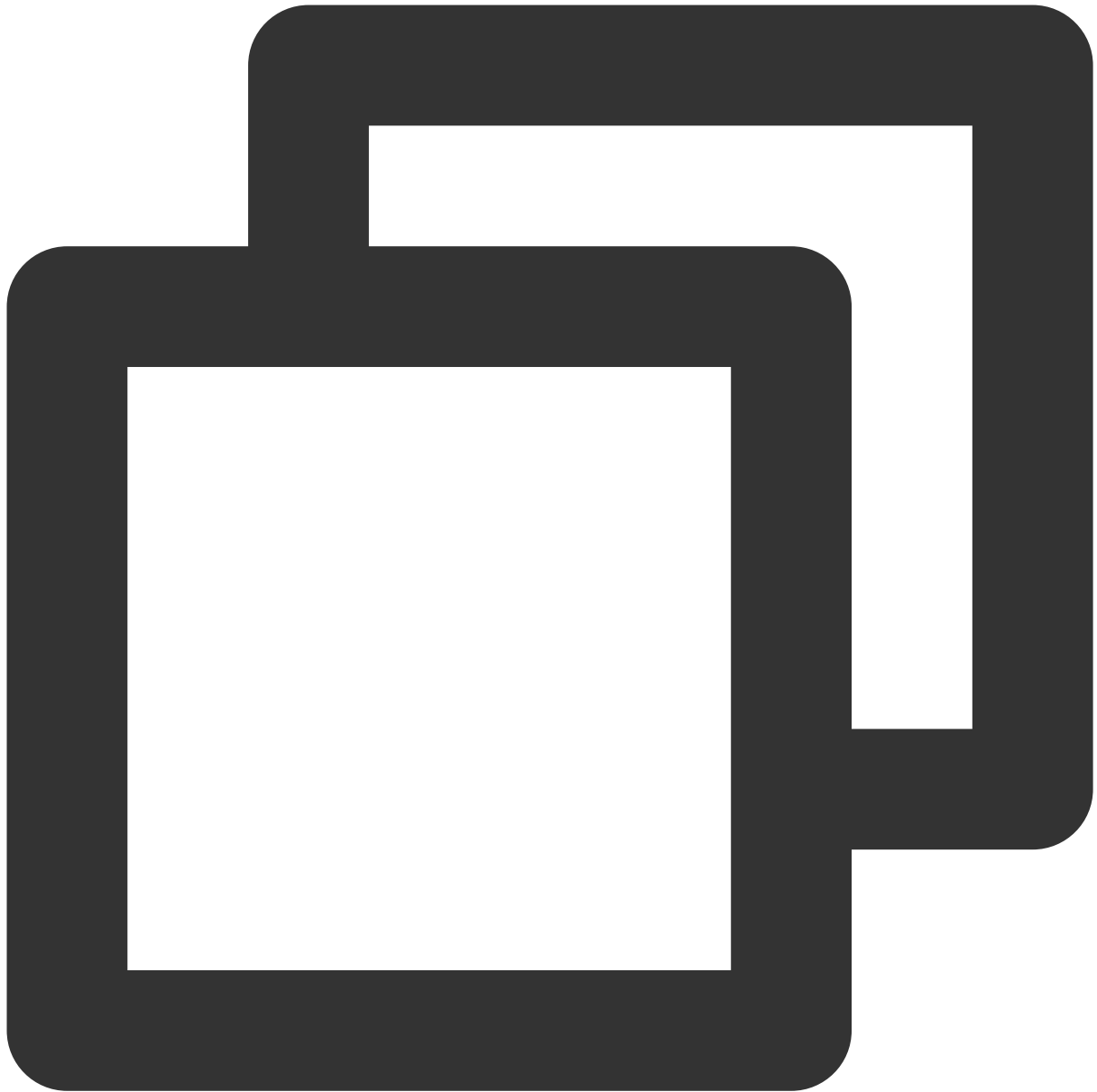
下表为 LightCC 相关环境变量说明，请按需进行配置。

--	--	--

环境变量	默认值	说明
LIGHT_2D_ALLREDUCE	0	是否使用 2D-Allreduce 算法
LIGHT_INTRA_SIZE	8	2D-Allreduce 组内 GPU 数
LIGHT_HIERARCHICAL_THRESHOLD	1073741824	2D-Allreduce 的阈值，单位是字节，小于等于该阈值的数据才使用 2D-Allreduce
LIGHT_TOPK_ALLREDUCE	0	是否使用 TOPK 压缩通信
LIGHT_TOPK_RATIO	0.01	使用 TOPK 压缩的比例
LIGHT_TOPK_THRESHOLD	1048576	TOPK 压缩的阈值，单位是字节，大于等于该阈值的数据才使用topk压缩通信
LIGHT_TOPK_FP16	0	压缩通信的 value 是否转为 FP16

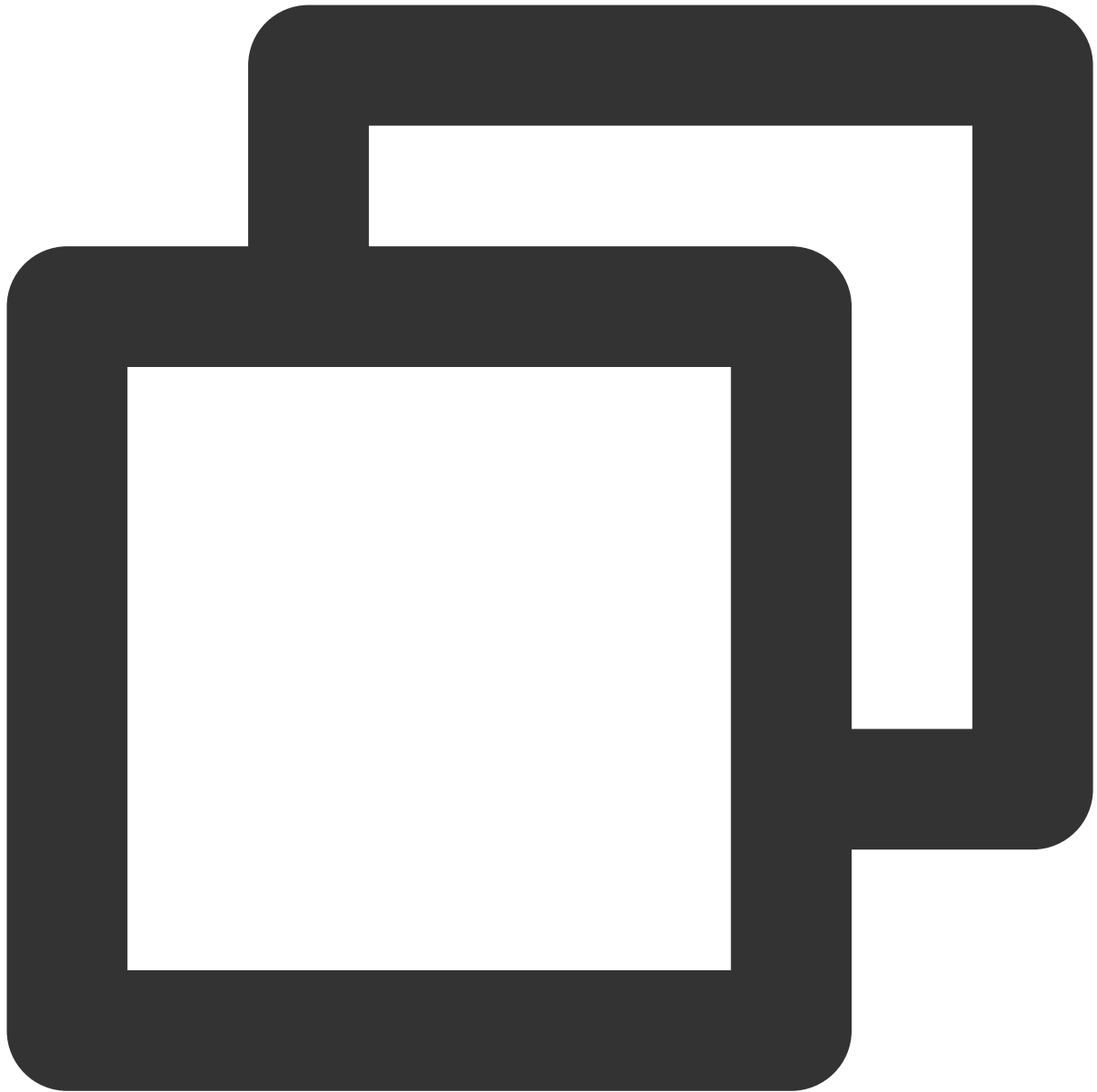
## Demo

1. 创建两台 GPU 实例后，依次按照上述步骤安装 LightCC 并配置环境变量。
2. 在容器中依次执行以下命令，对两台实例进行 SSH 免密处理。

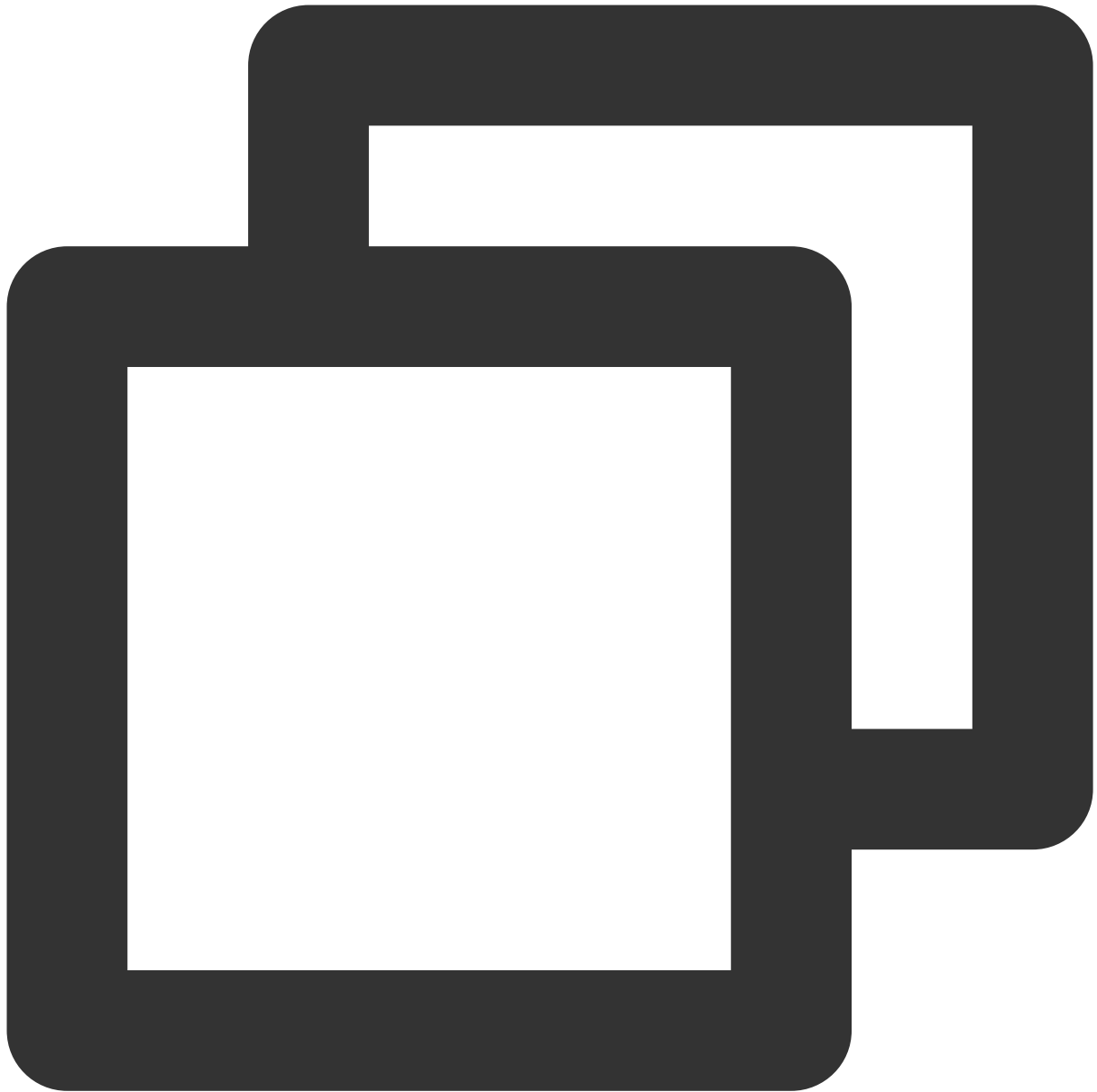


#允许root使用ssh服务, 并启动服务 (默认端口: 22)

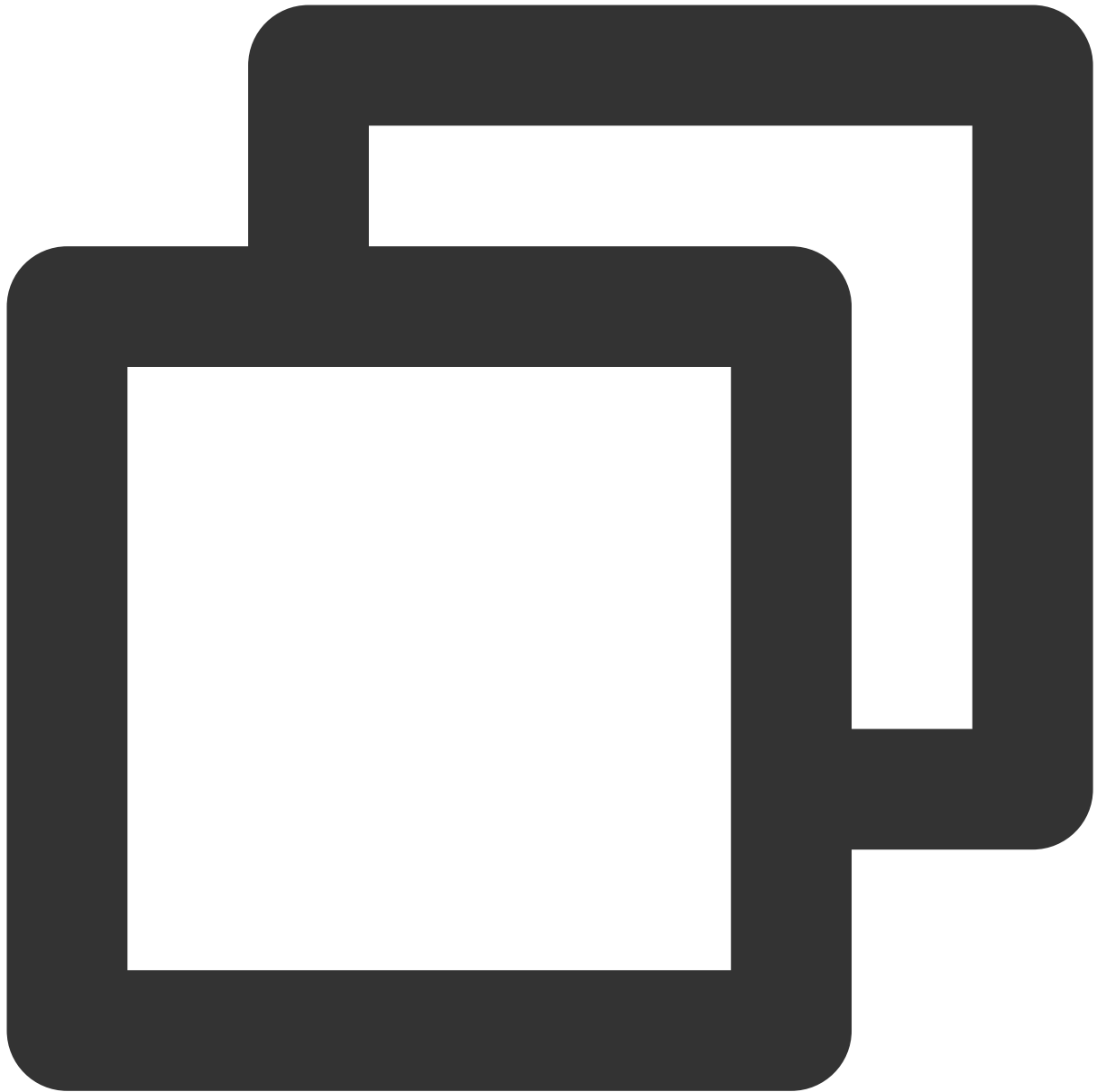
```
sed -i 's/#PermitRootLogin prohibit-password/PermitRootLogin yes/' /etc/ssh/sshd_config  
service ssh start && netstat -tulpn
```



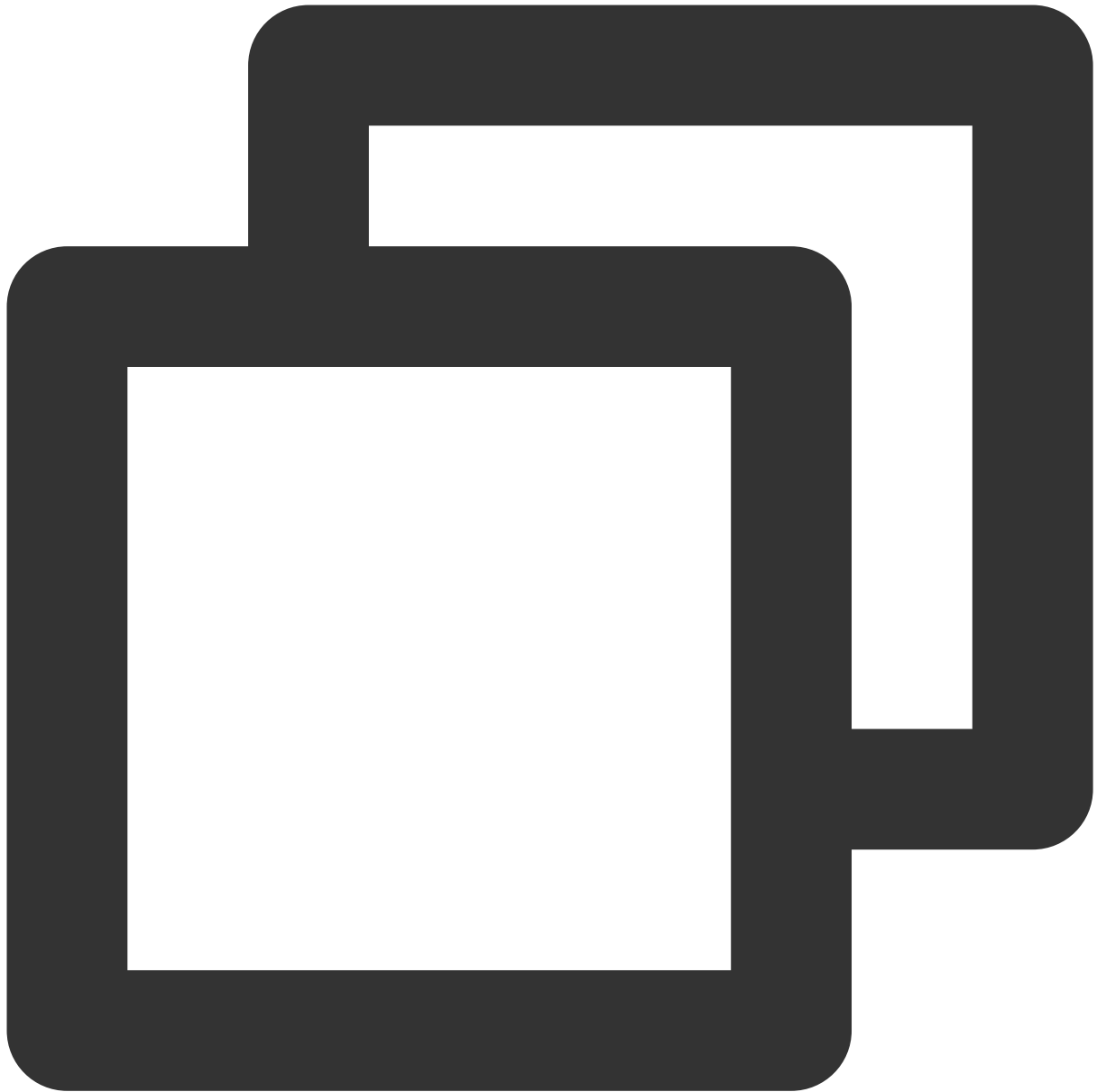
```
#修改容器内ssh默认端口为2222，防止与host冲突
sed -i 's/#Port 22/Port 2222/' /etc/ssh/sshd_config
service ssh restart && netstat -tulpn
```



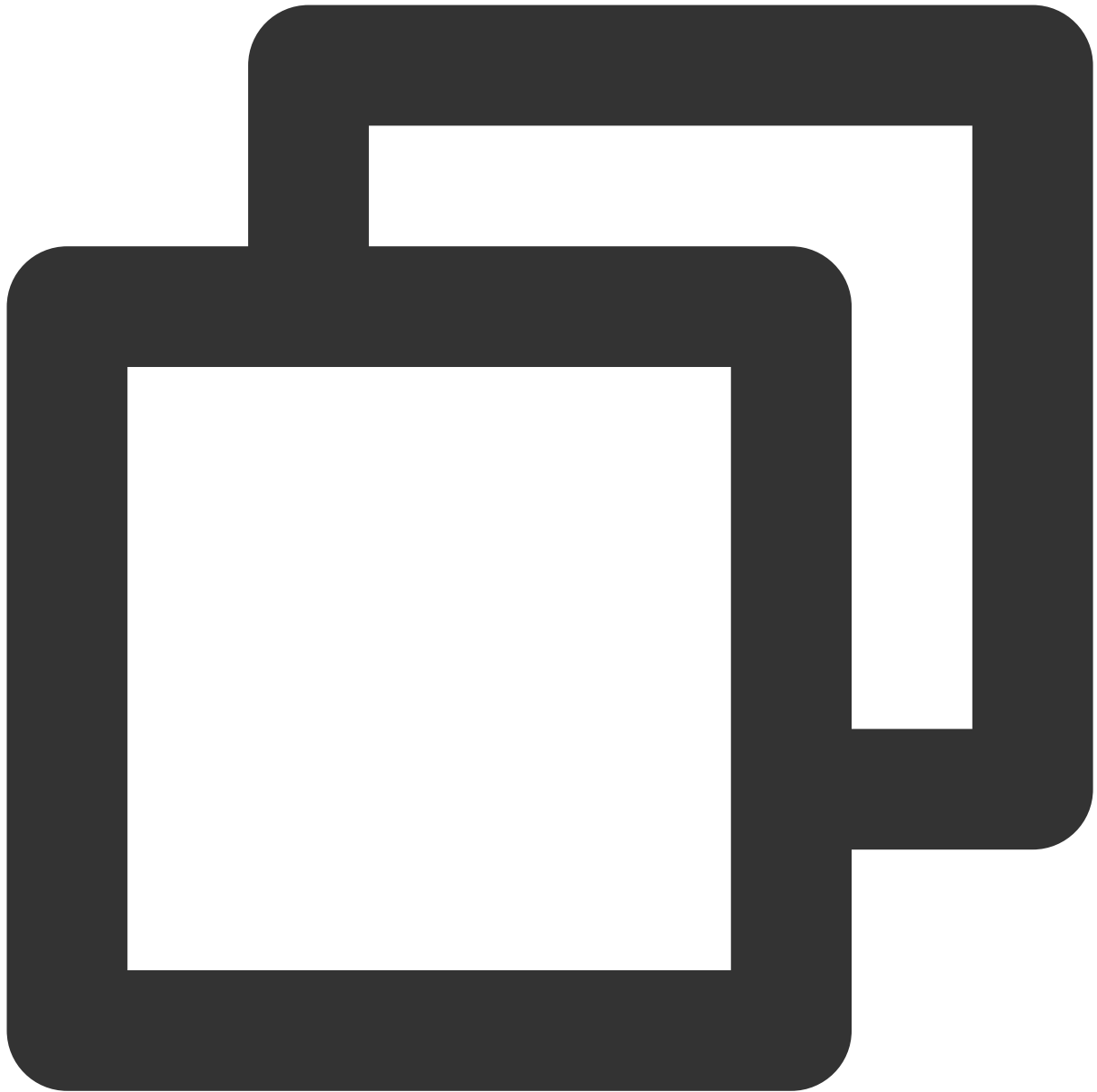
```
#设置root passwd  
passwd root
```



```
#产生ssh key  
ssh-keygen
```



```
#配置ssh使其默认使用2222端口
#创建 ~/.ssh/config, 并添加以下内容, 保存退出
#注意: 这里使用的ip是两台机器`ifconfig eth0`显示的ip
Host gpu1
  hostname 10.0.2.8
  port 2222
Host gpu2
  hostname 10.0.2.9
  port 2222
```



```
#两台机器互相免密，同时本机对自己做免密  
ssh-copy-id gpu1  
ssh-copy-id gpu2
```





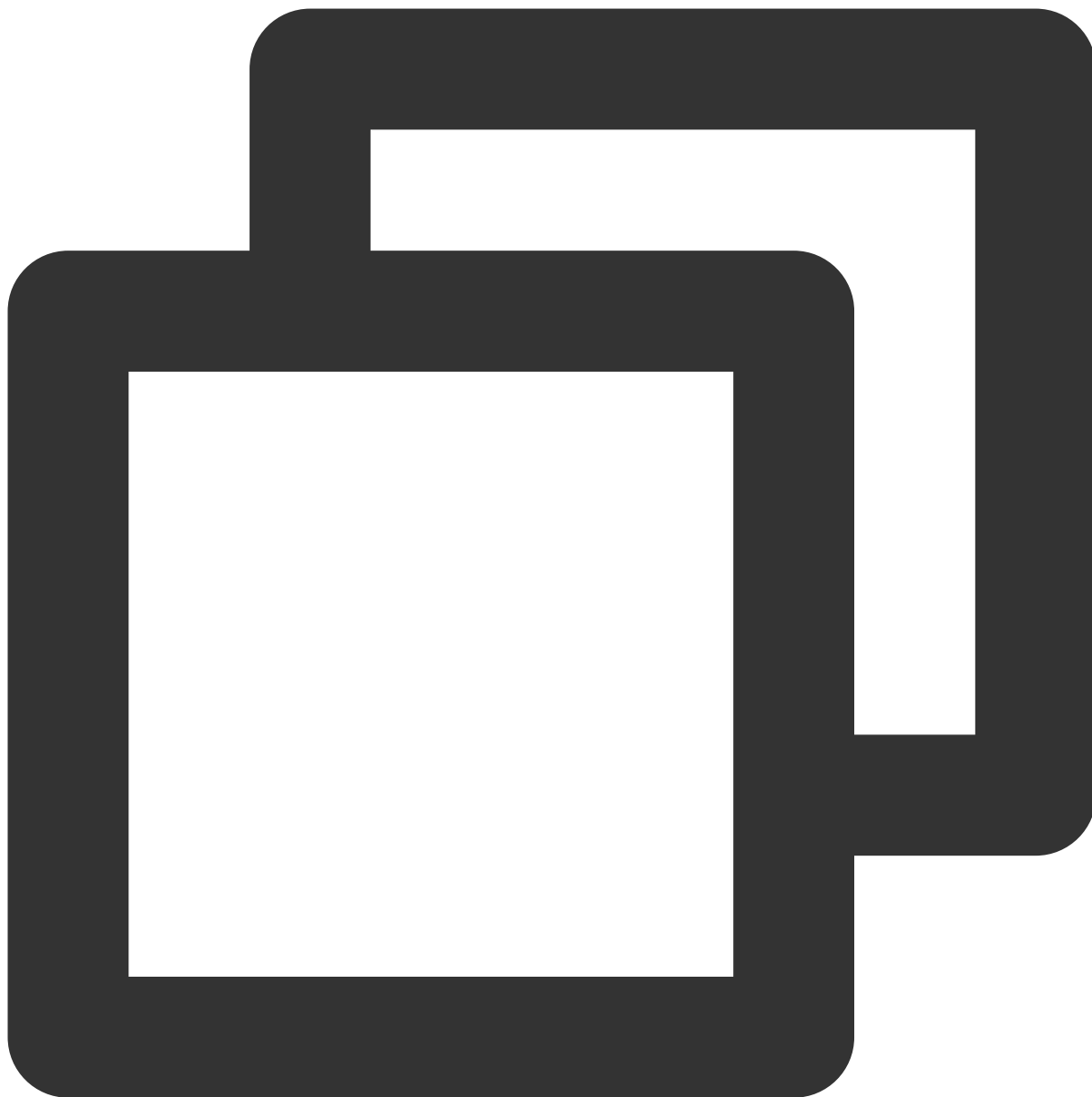
```
#测试是否建立成功免密  
ssh gpu1  
ssh gpu2
```

3. 执行以下命令，下载 Horovod 的 benchmark 测试脚本。



```
wget https://raw.githubusercontent.com/horovod/horovod/master/examples/tensorflow/t
```

4. 执行以下命令，开始 ResNet50的多机训练 benchmark。



```
/usr/local/openmpi/bin/mpirun -np 16 -H gpu1:8,gpu2:8 --allow-run-as-root -bind-to
```

此处命令参数针对八卡机型，如果有其他配置则需修改 `-np` 和 `-H` 参数。其他主要参数说明如下：

`NCCL_ALGO=RING`：选择 NCCL 中的通信算法为 ring 算法。

`NCCL_DEBUG=INFO`：开启 NCCL 中的 debug 输出。

`-mca btl_tcp_if_include eth0`：选择 eth0 设备作为 MPI 多机通信的网络设备。在有多个网卡时需要特殊指定，有些网卡无法通信，不指定 MPI 选到无法通信的网卡会产生报错。

两台 GT4.41XLARGE948 实例下 LightCC 多机训练 benchmark 的吞吐率参考数据，如下表所示：

机型：CVM GT4.41XLARGE948 (A100*8 + 50G VPC)
---

GPU 驱动：460.27.04

容器：ccr.ccs.tencentyun.com/qcloud/taco-training:cu112-cudnn81-py3-0.3.2

network	ResNet50 Throughput(images/sec)	
	horovod 0.21.3	lightcc 3.0.0
NCCL + HRAP	8970.7	10229.9
NCCL + kernel socket	5421.9	7183.5

## 使用 HARP

### 实例环境准备

1. 以 root 用户身份执行以下命令，修改内核的 `cmdline`，配置50G大页内存。



```
sed -i '/GRUB_CMDLINE_LINUX/ s/"$/ default_hugepagesz=1GB hugepagesz=1GB hugepages=
```

**注意：**

针对八卡机器可配置 `hugepages=50`，其他机型建议按照 `hugepages=（卡数 × 5+10）` 进行配置。

2. 结合实际使用的操作系统版本，执行以下命令，使配置生效并重启实例。

Ubuntu

CentOS 或 TencentOS



```
sudo update-grub2 && sudo reboot
```



```
sudo grub2-mkconfig -o /boot/grub2/grub.cfg && sudo reboot
```

3. 绑定弹性网卡，弹性网卡数量为 GPU 卡数量。具体步骤如下：

3.1 登录 [云服务器控制台](#)，选择实例所在行右侧的**更多 > IP/网卡 > 绑定弹性网卡**。

3.2 在弹出的“绑定弹性网卡”窗口中，按需选择绑定已创建的网卡，或新建弹性网卡并绑定。

3.3 单击**确定**即可完成绑定。

4. 执行以下命令，初始化配置信息。



```
curl -s -L http://mirrors.tencent.com/install/GPU/taco/harp_setup.sh | bash
```

若输入结果包含 "Set up HARP successfully", 并且 `/usr/local/tfabric/tools/config` 目录下已生成配置文件 `ztcp*.conf`, 则表示配置成功。

## 安装

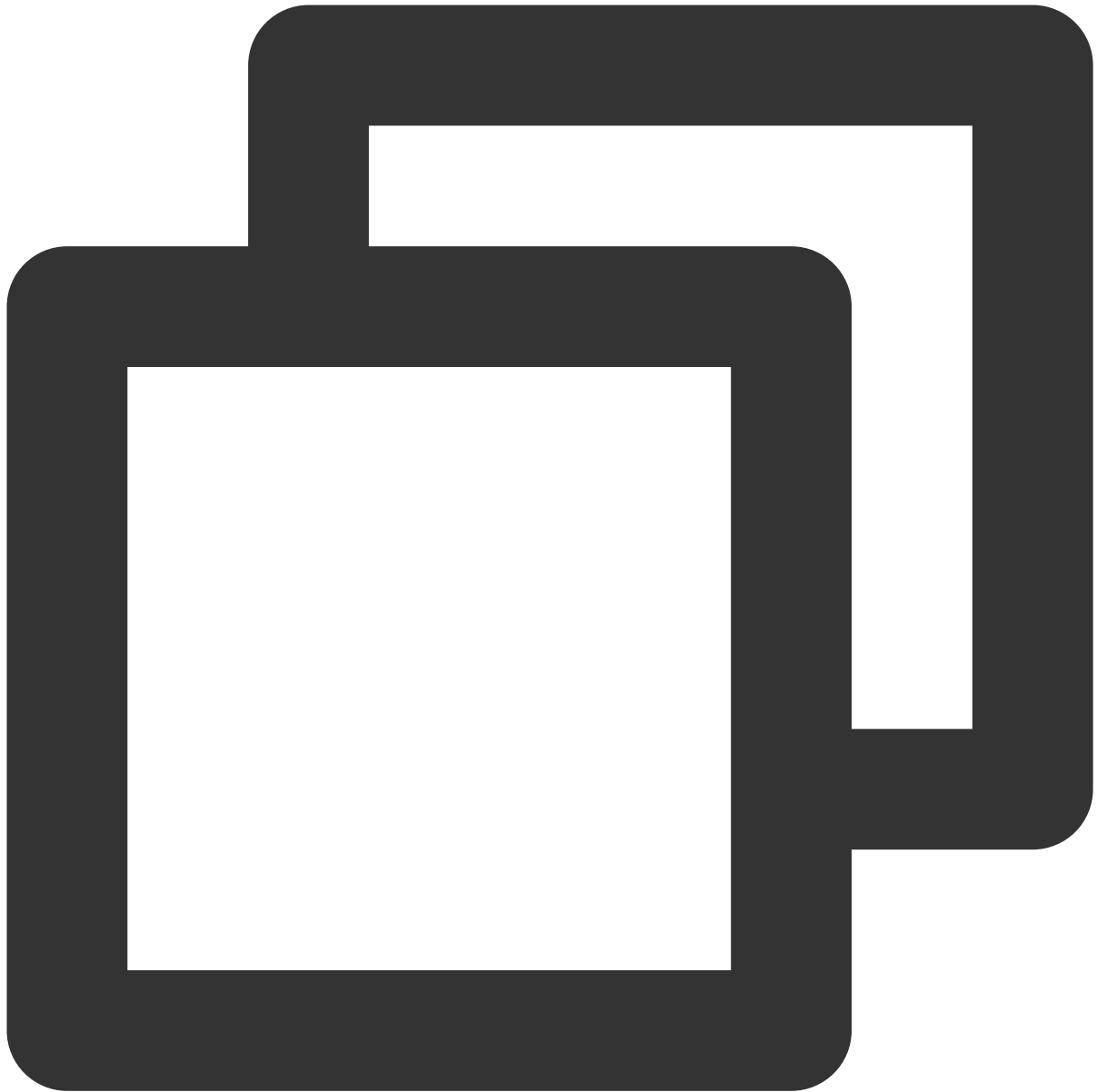
1. 以 root 用户身份执行以下命令, 通过 Docker 镜像的方式安装 HARP。





```
docker pull ccr.ccs.tencentyun.com/qcloud/taco-training:cu112-cudnn81-py3-0.3.2
```

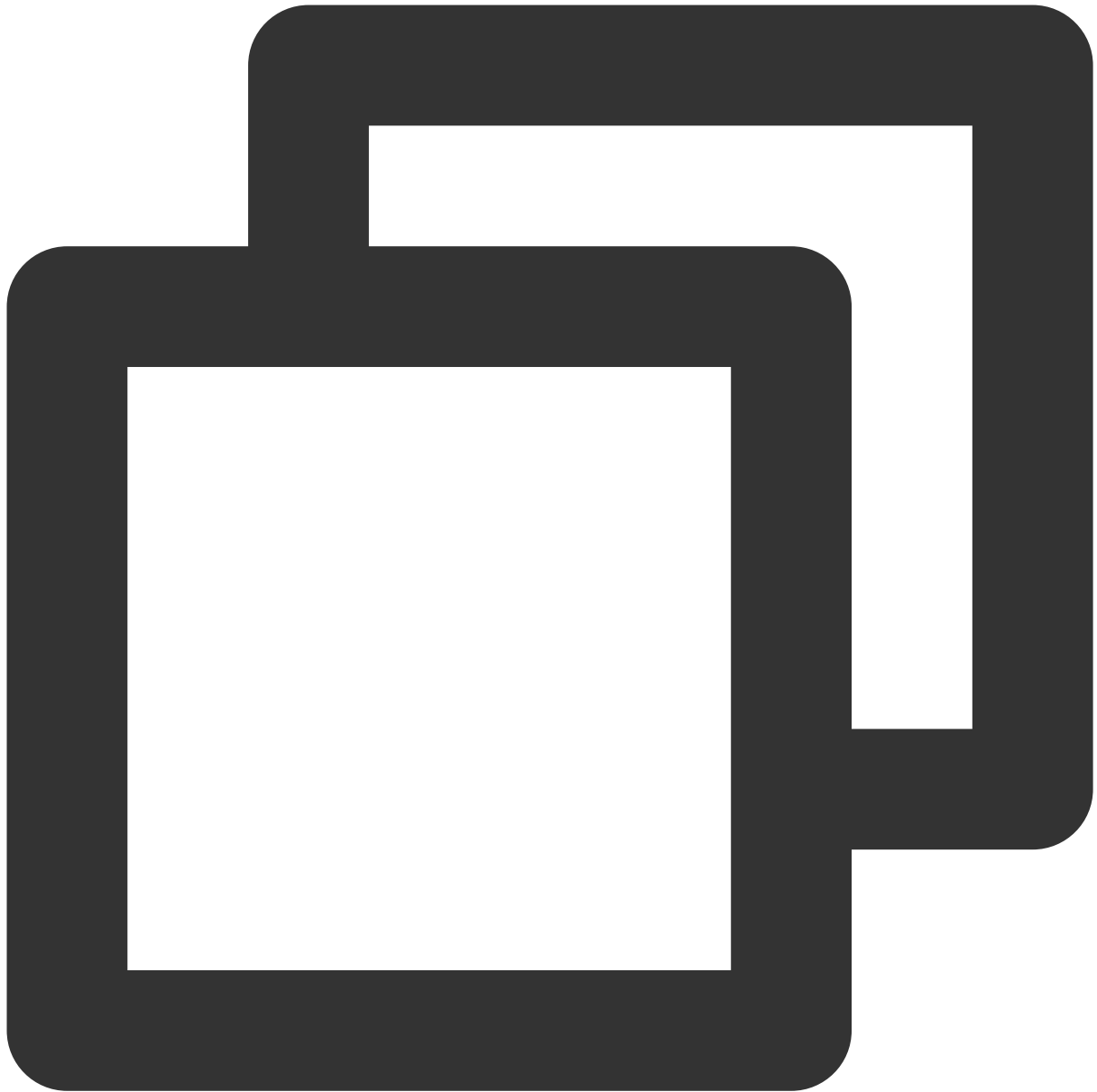
2. 执行以下命令，启动 Docker。



```
docker run -it --rm --gpus all --privileged --net=host -v /sys:/sys -v /dev/hugepag
```

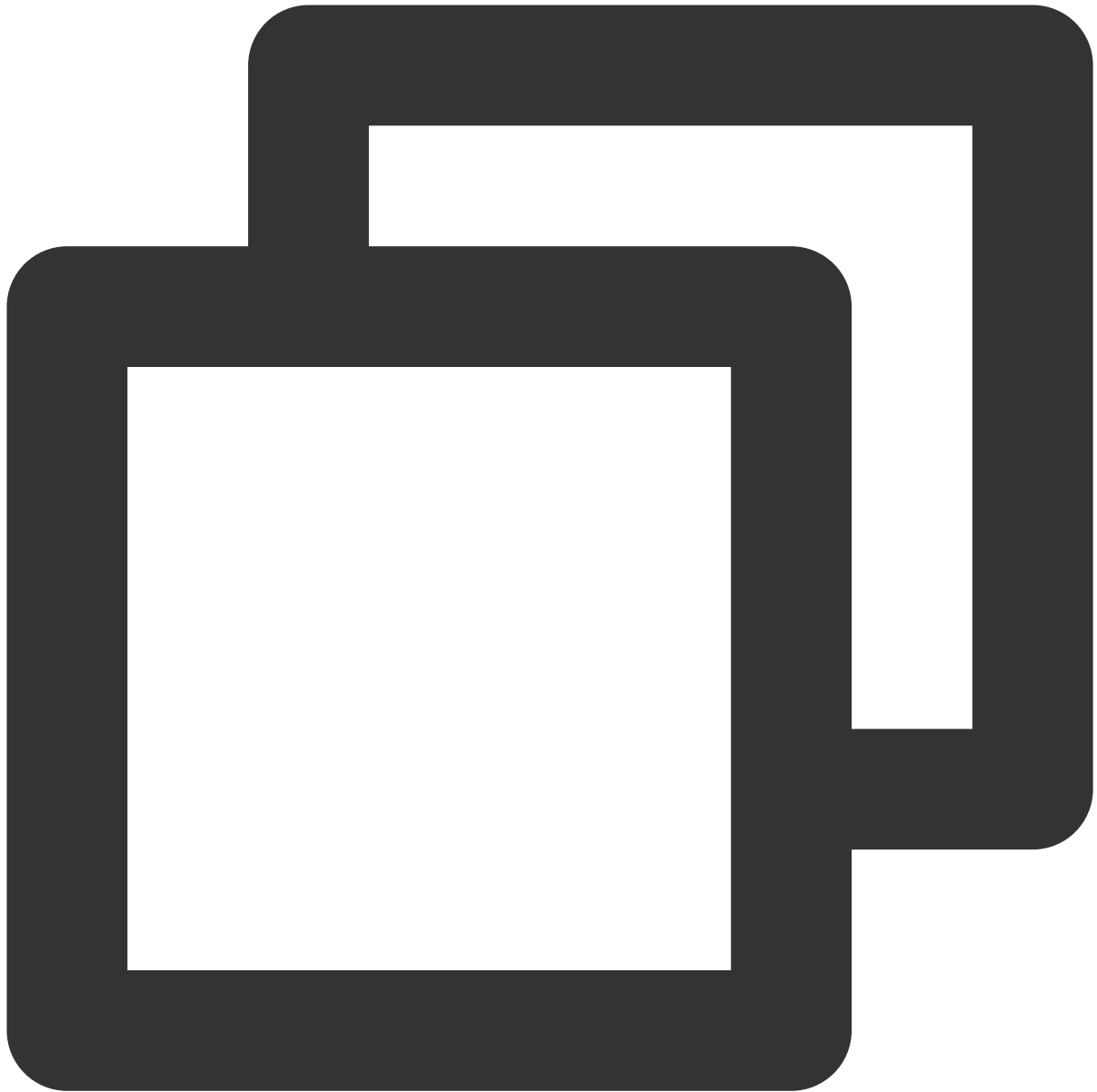
### Demo

1. 创建两台 GPU 实例后，依次按照上述步骤配置实例环境并安装 HARP。
2. 在 Docker 中依次执行以下命令，对两台实例进行 SSH 免密处理。

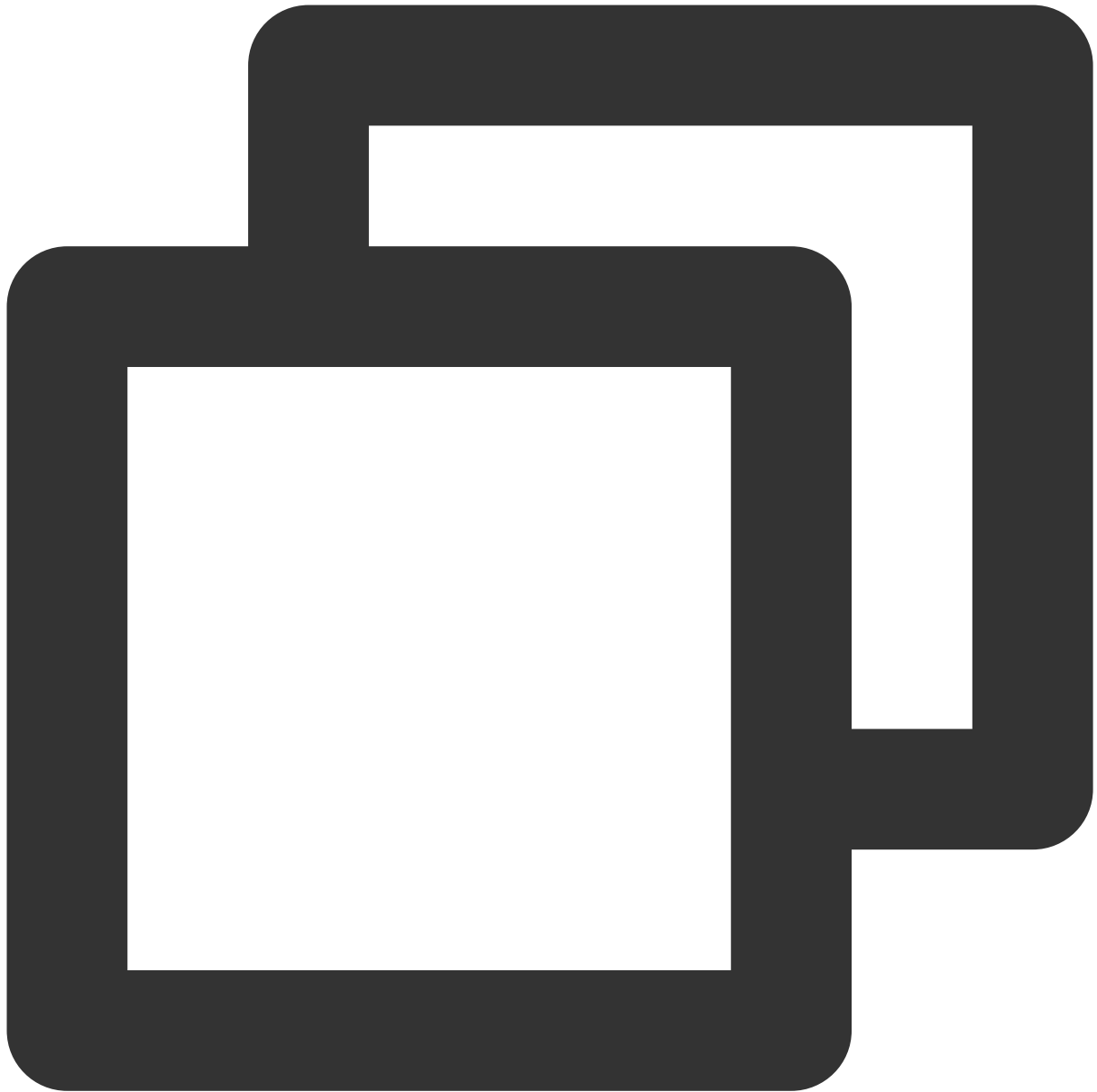


#允许root使用ssh服务，并启动服务（默认端口：22）

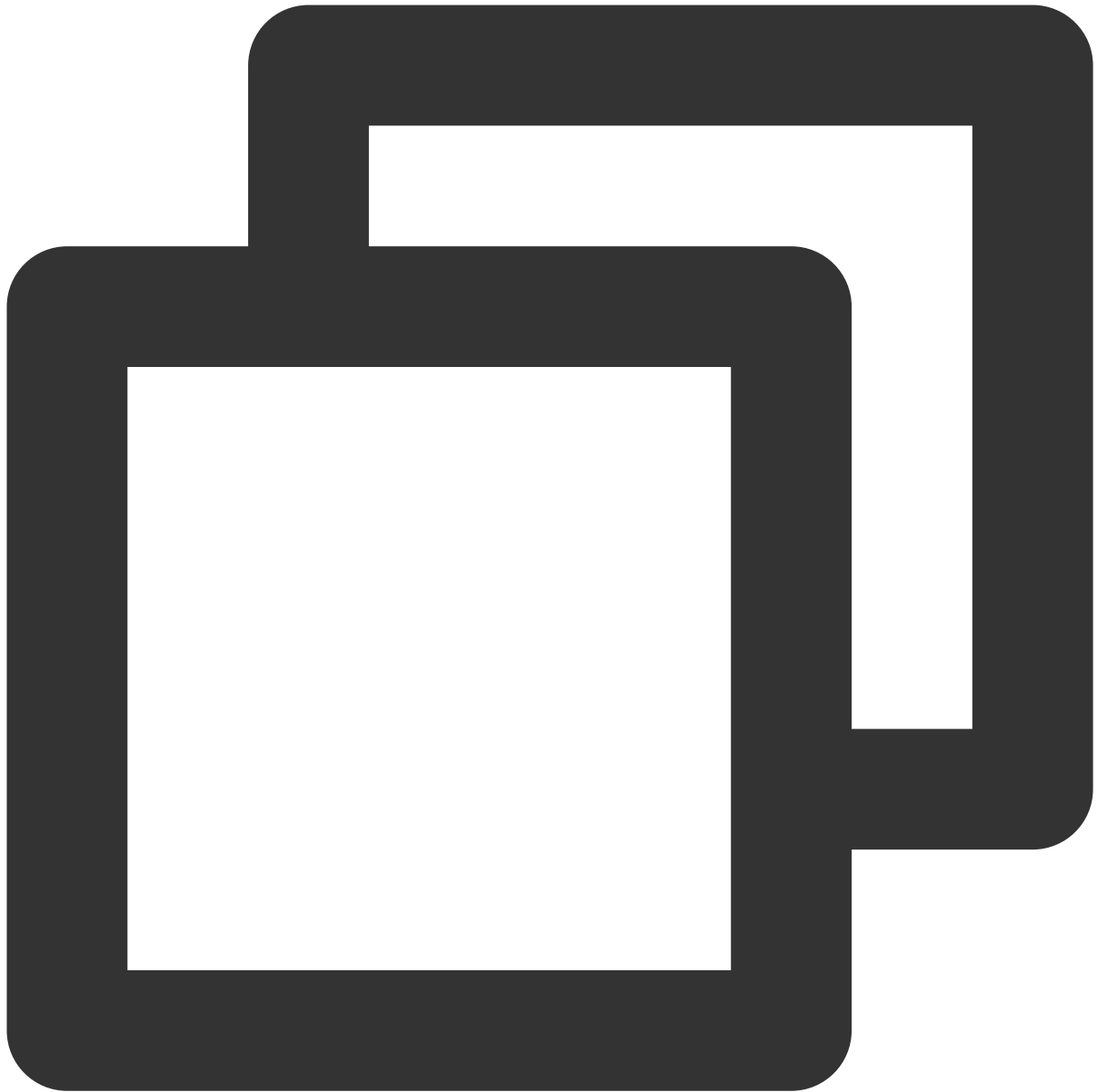
```
sed -i 's/#PermitRootLogin prohibit-password/PermitRootLogin yes/' /etc/ssh/sshd_config  
service ssh start && netstat -tulpn
```



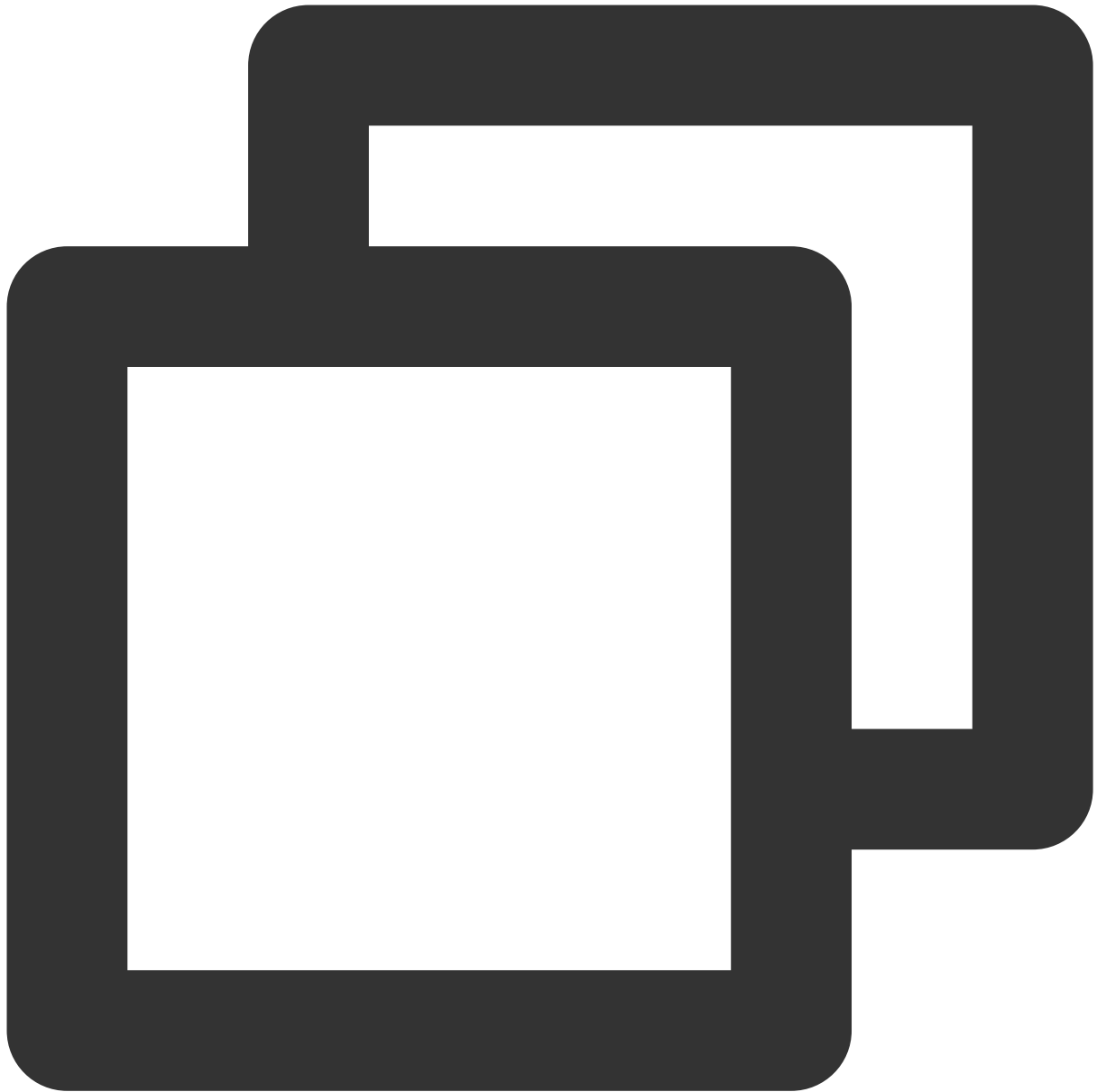
```
#修改容器内ssh默认端口为2222，防止与host冲突  
sed -i 's/#Port 22/Port 2222/' /etc/ssh/sshd_config  
service ssh restart && netstat -tulpn
```



```
#设置root passwd  
passwd root
```



```
#产生ssh key  
ssh-keygen
```



```
#配置ssh使其默认使用2222端口
#创建 ~/.ssh/config, 并添加以下内容, 保存退出
#注意: 这里使用的ip是两台机器`ifconfig eth0`显示的ip
Host gpu1
  hostname 10.0.2.8
  port 2222
Host gpu2
  hostname 10.0.2.9
  port 2222
```



```
#两台机器互相免密，同时本机对自己做免密  
ssh-copy-id gpu1  
ssh-copy-id gpu2
```





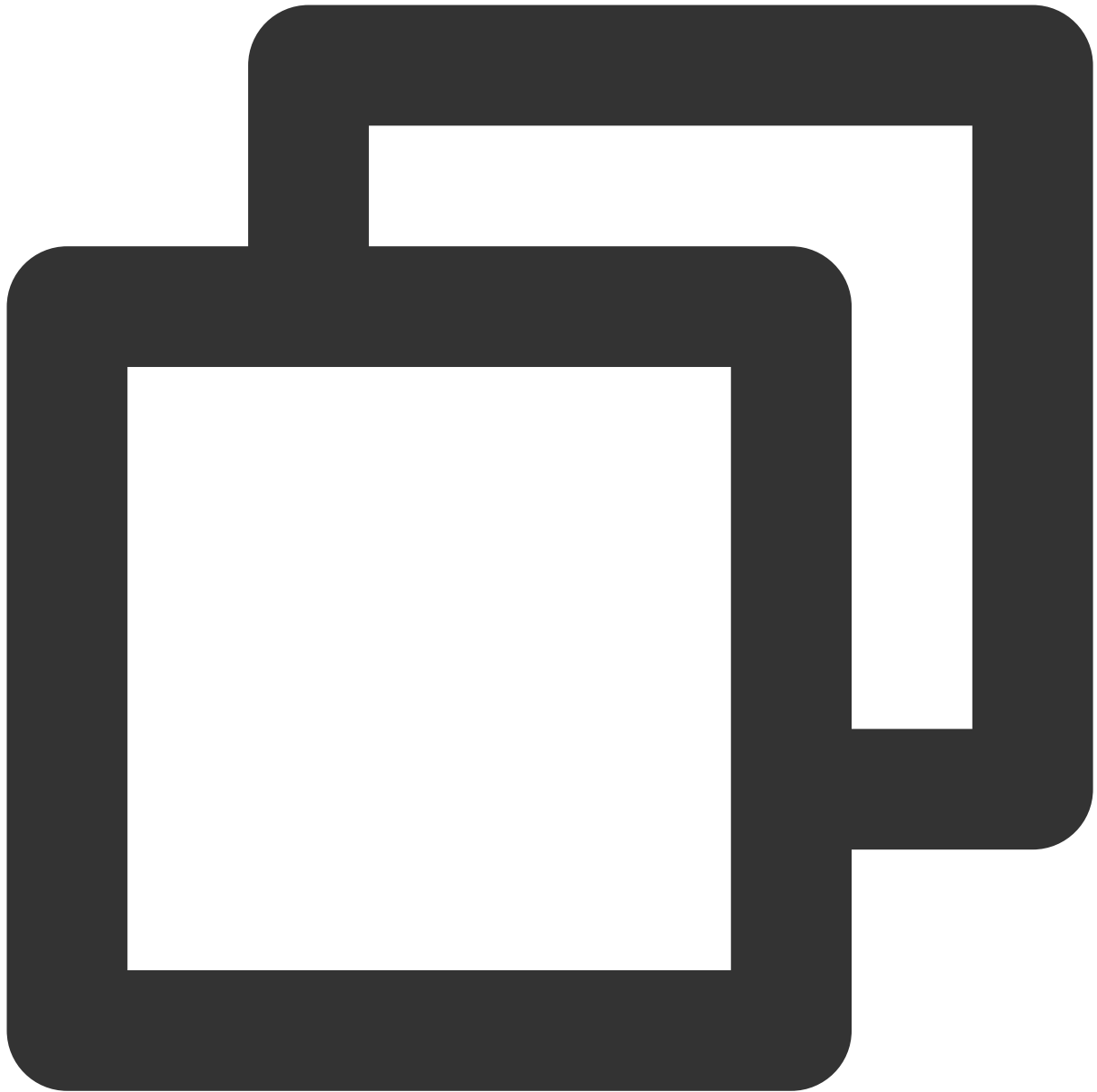
```
#测试是否建立成功免密  
ssh gpu1  
ssh gpu2
```

3. 执行以下命令，下载 Horovod 的 benchmark 脚本。



```
wget https://raw.githubusercontent.com/horovod/horovod/master/examples/tensorflow/t
```

4. 执行以下命令，开始 ResNet50 的多机训练 benchmark。



```
/usr/local/openmpi/bin/mpirun -np 16 -H gpu1:8,gpu2:8 --allow-run-as-root -bind-to
```

此处命令参数针对八卡机型，如果有其他配置则需修改 `-np` 和 `-H` 参数。其中主要参数说明如下：

`NCCL_ALGO=RING`：选择 NCCL 中的通信算法为 ring 算法。

`NCCL_DEBUG=INFO`：开启 NCCL 中的 debug 输出。开启后，HARP 将产生如下图所示的输出：

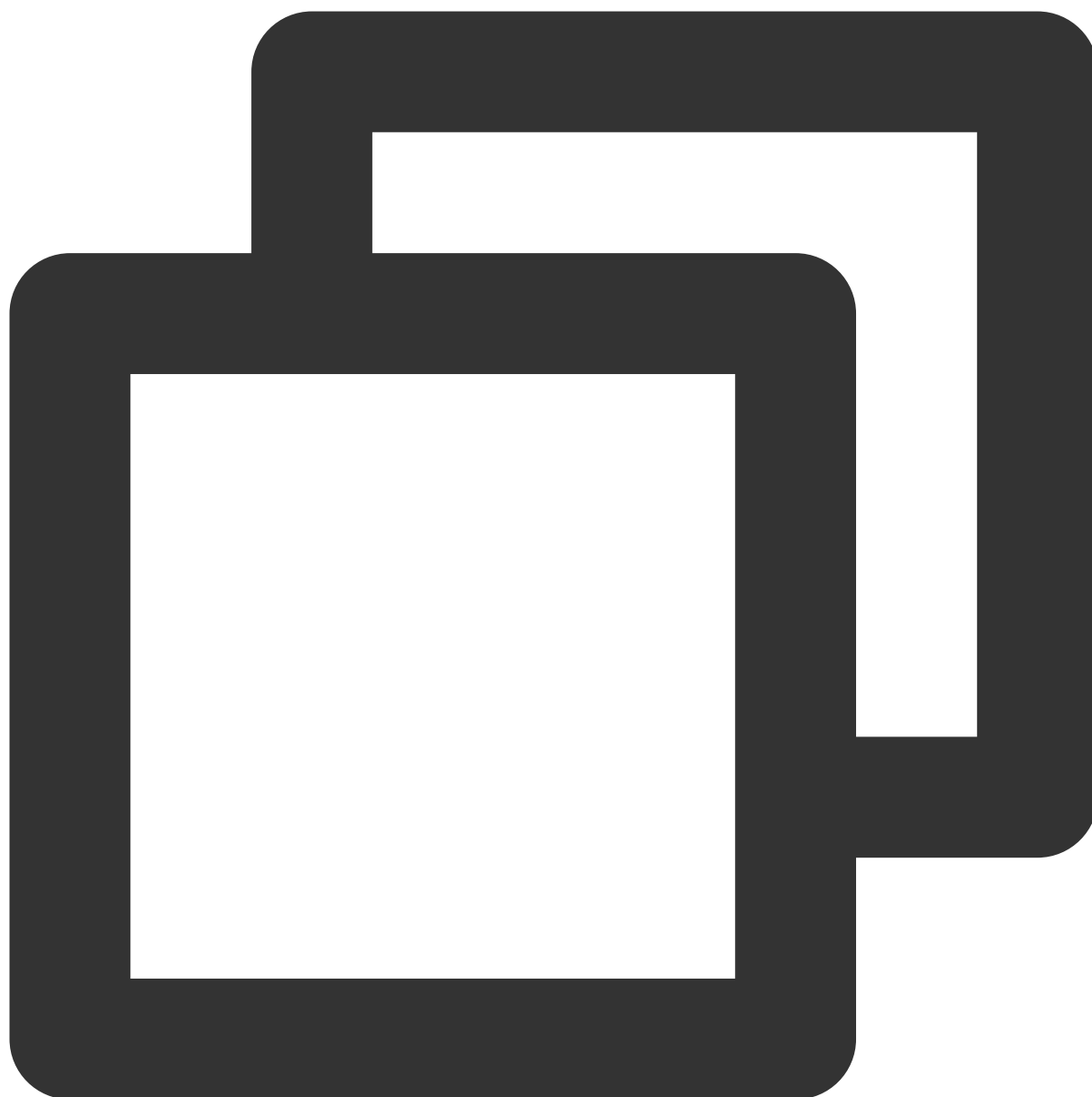
```
[0] NCCL INFO Channel 00 : 1[b010] -> 0[b010] [send] via NET/Tencent
[0] NCCL INFO Channel 01 : 1[b010] -> 0[b010] [send] via NET/Tencent
```

`-mca btl_tcp_if_include eth0` : 选择 eth0 设备作为 MPI 多机通信的网络设备，在有多个网卡时需要特殊指定。有些网卡无法通信，不指定 MPI 选到无法通信的网卡会产生报错。

NCCL 初始化完成后，可查看如下图所示的网络输出：

```
[0] NCCL INFO Using network Tencent_zTCP
```

5. HARP 通过 Plugin 的方式集成到 NCCL 中，无需用户做任何配置，自动启用。如需关闭 HARP，请在容器中执行如下命令：



```
mv /usr/lib/x86_64-linux-gnu/libnccl-net.so /usr/lib/x86_64-linux-gnu/libnccl-net.s
```