Cloud GPU Service

AI Optimization

Product Documentation

Contents

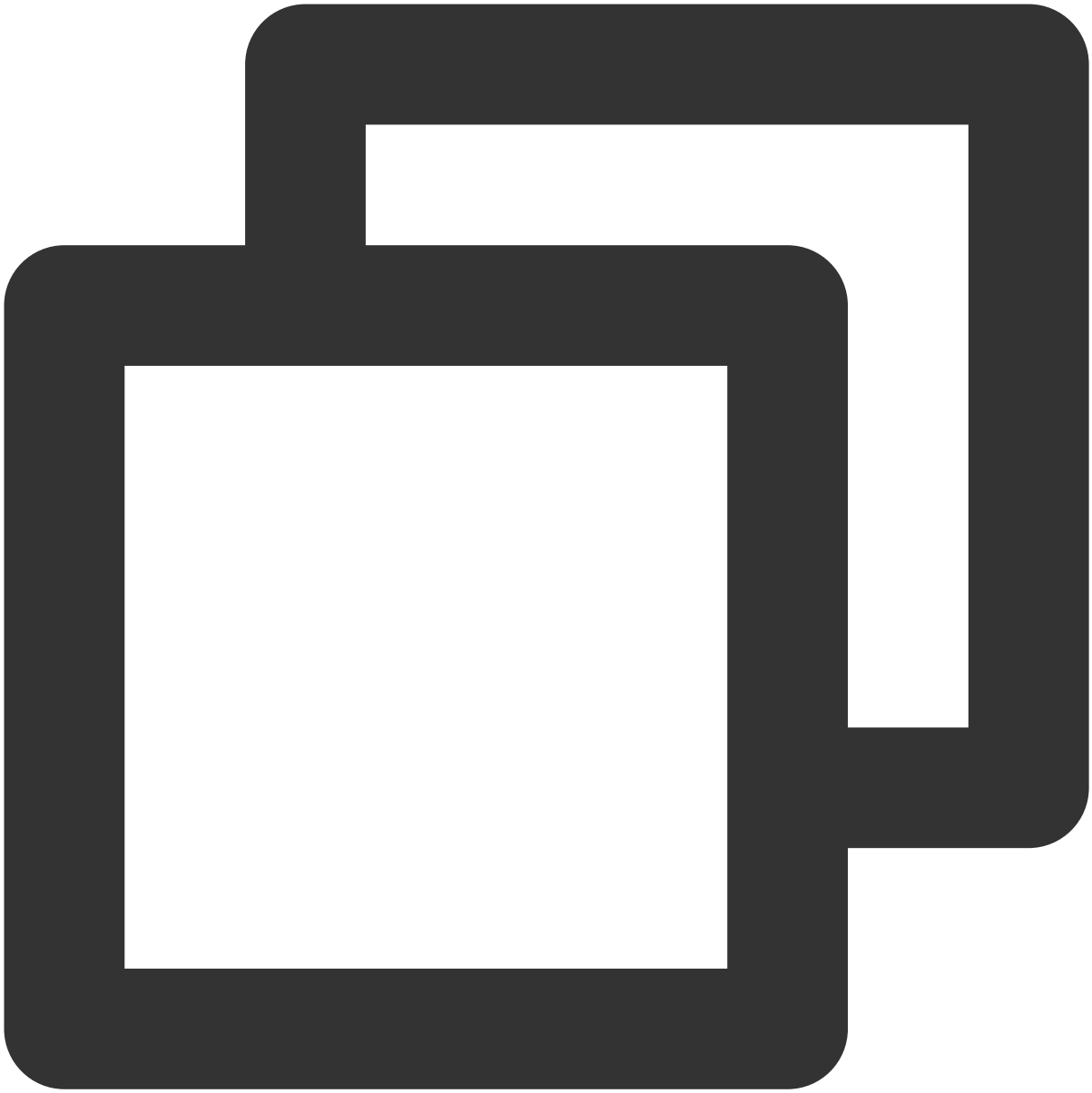
AI Optimization
Training Acceleration Engine TACO Train Deployment and Practices

Last updated：2024-01-11 17:11:13

This document describes how to deploy and use TACO-Training on GPU instances.

Notes

Currently, TACO-Training is supported only by Cloud GPU Service.
Currently, the three acceleration components of TACO-Training have been integrated into the same Docker image, which can be pulled at the following address:



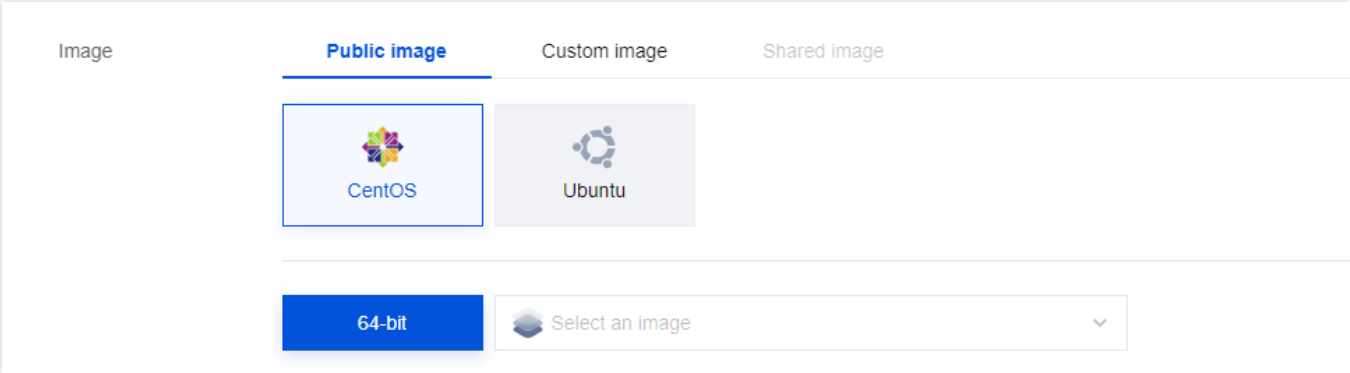
```
ccr.ccs.tencentyun.com/qcloud/taco-training:cu112-cudnn81-py3-0.3.2
```

Directions

Preparing the instance environment

1. Create at least two instances meeting the following requirements as instructed in Purchasing NVIDIA GPU Instance:

Instance: We recommend you select the Computing GT4 GT4.41XLARGE948 8-card model.
Image: Select CentOS 7.8 or Ubuntu 18.04 or later and select **Automatically install GPU driver on the backend** to use the automatic installation feature to install the GPU driver.
Automatic installation of CUDA and cuDNN is not required for this deployment, and you can choose as needed.



System disk: We recommend you configure a system disk of 100 GB or above in size to store the Docker image and intermediate state files generated during training.
2. Install Docker as instructed in the corresponding document based on the operating system type of your instance:

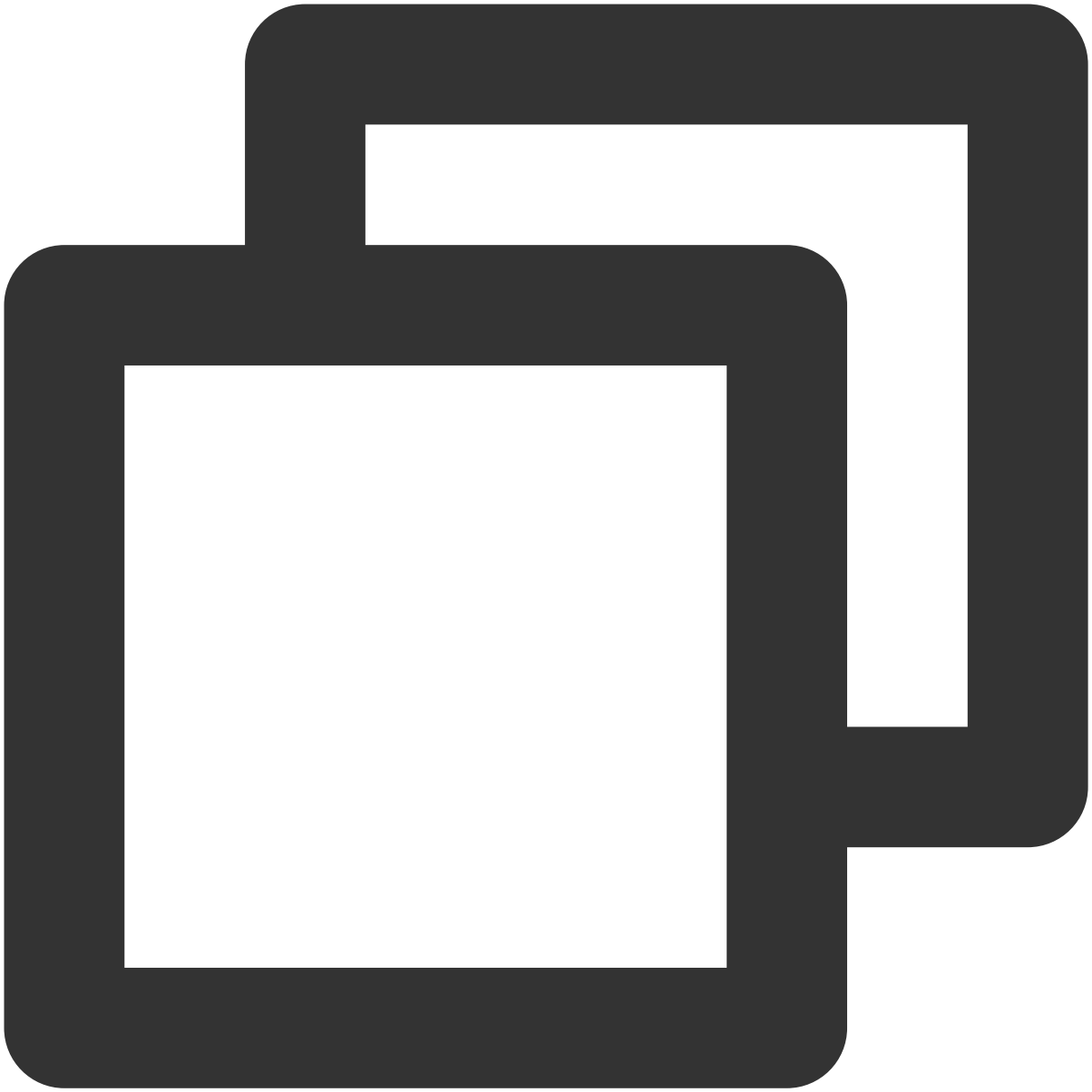
OS	Description
CentOS	For detailed directions, see Install Docker Engine on CentOS.
Ubuntu	For detailed directions, see Install Docker Engine on Ubuntu.

3. Install nvidia-docker as instructed in Docker.

Using TTF

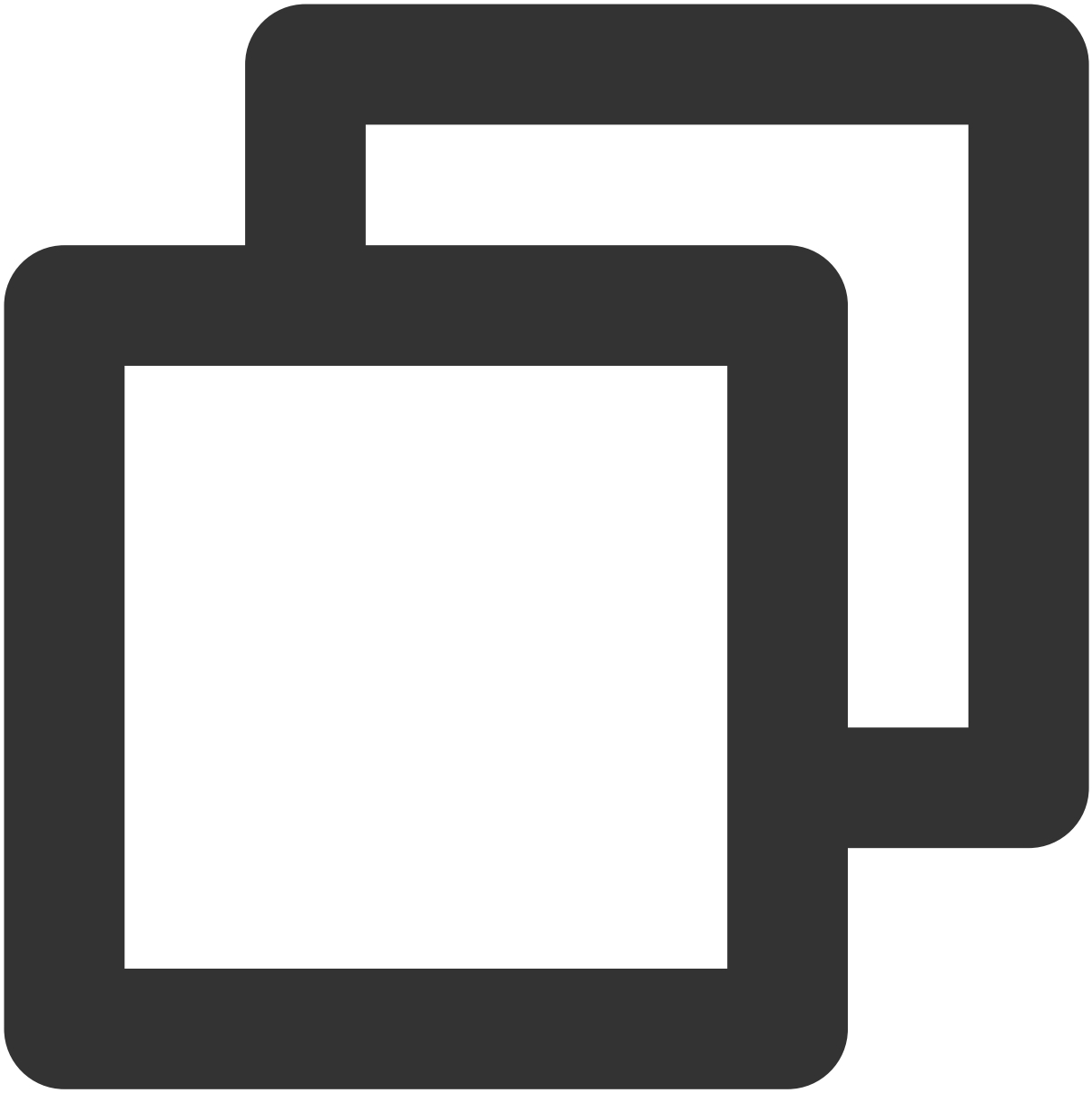
Installation

1. Run the following command as the root user and install TTF through a Docker image:



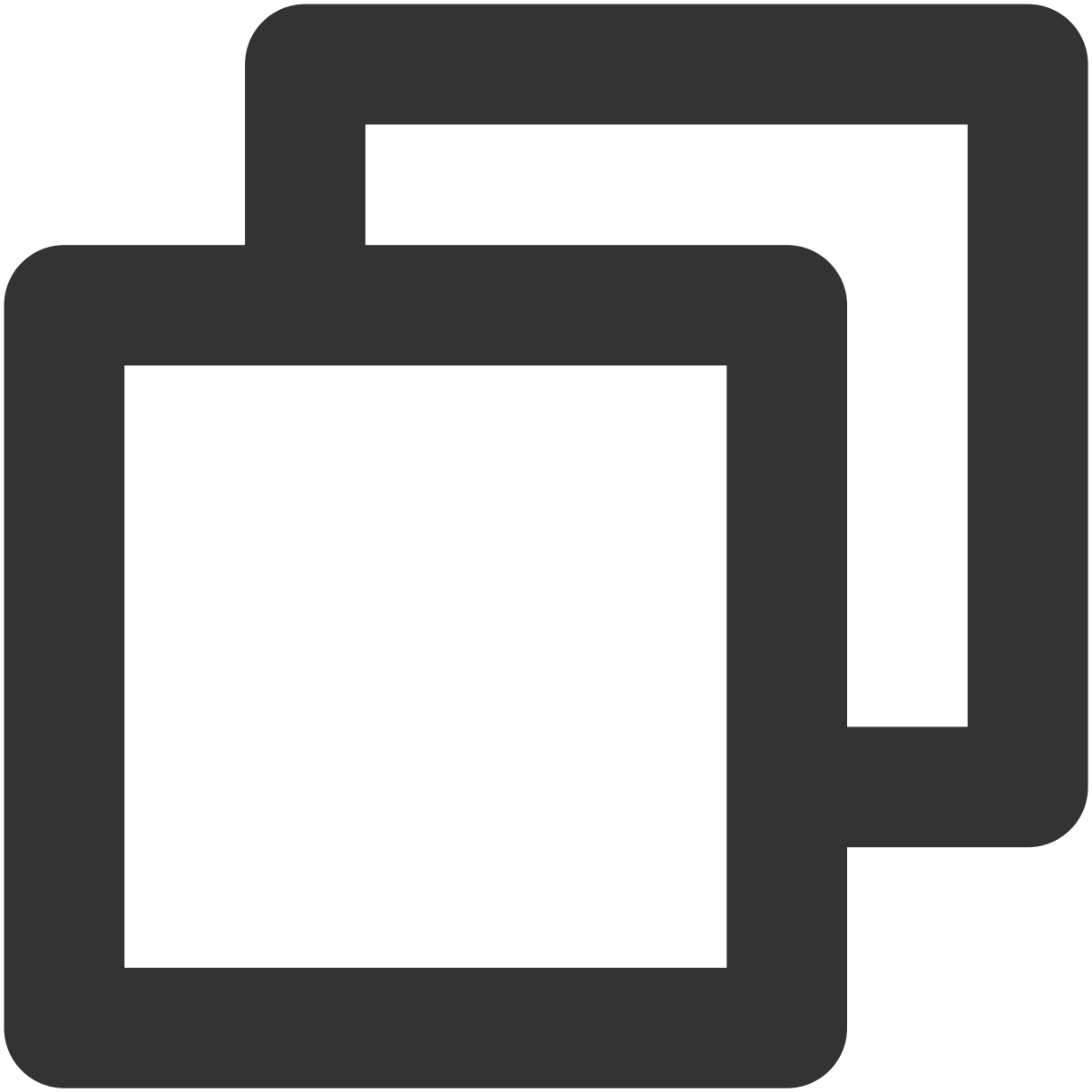
```
docker pull ccr.ccs.tencentyun.com/qcloud/taco-training:cu112-cudnn81-py3-0.3.2
```

2. Run the following command to start Docker:



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

3. Run the following command to view the TTF version:



```
pip show ttensorflow
```

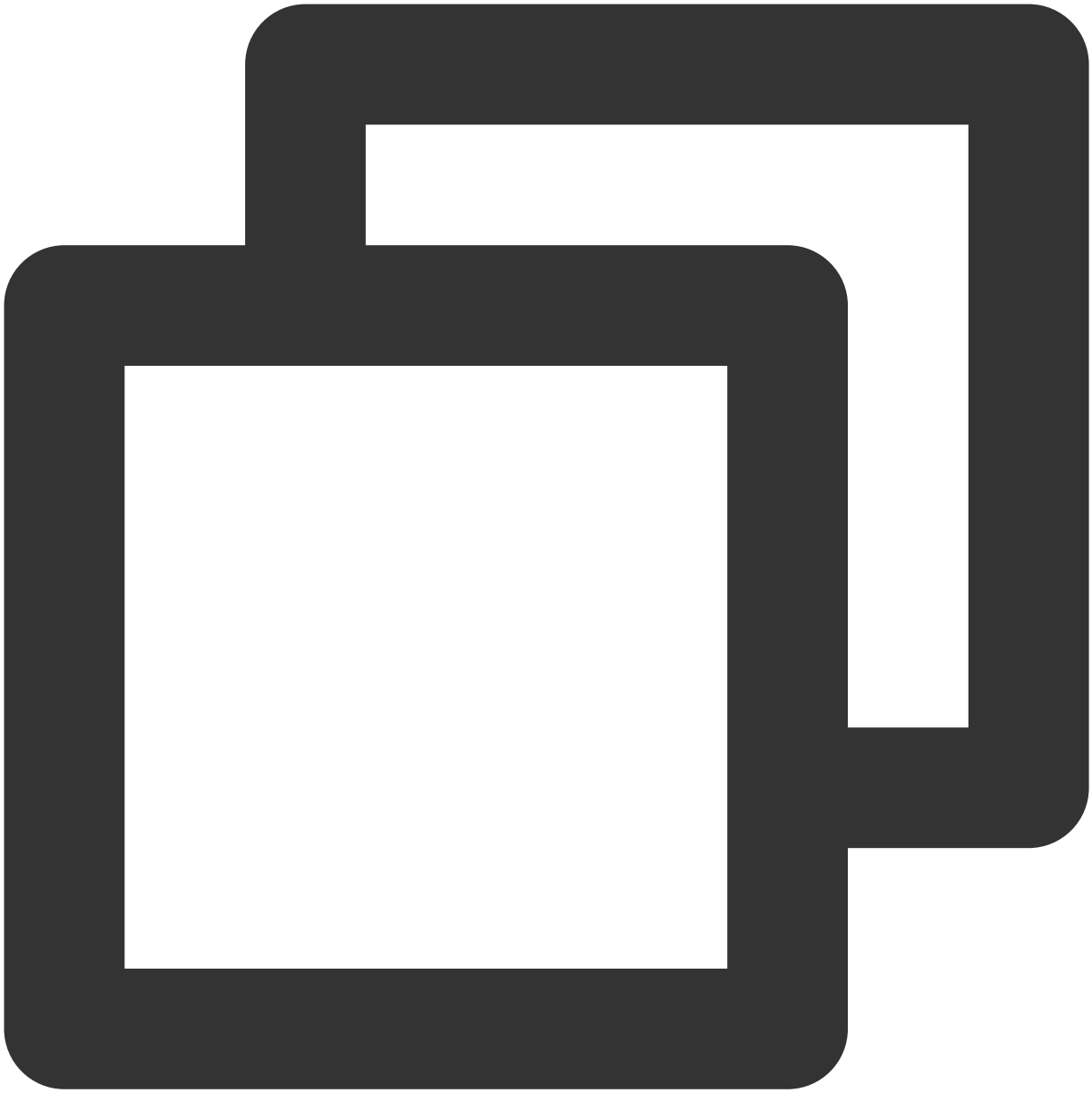
Model adaptation

Dynamic embedding

Below is the code for the native static embedding of TF and dynamic embedding of TTF:

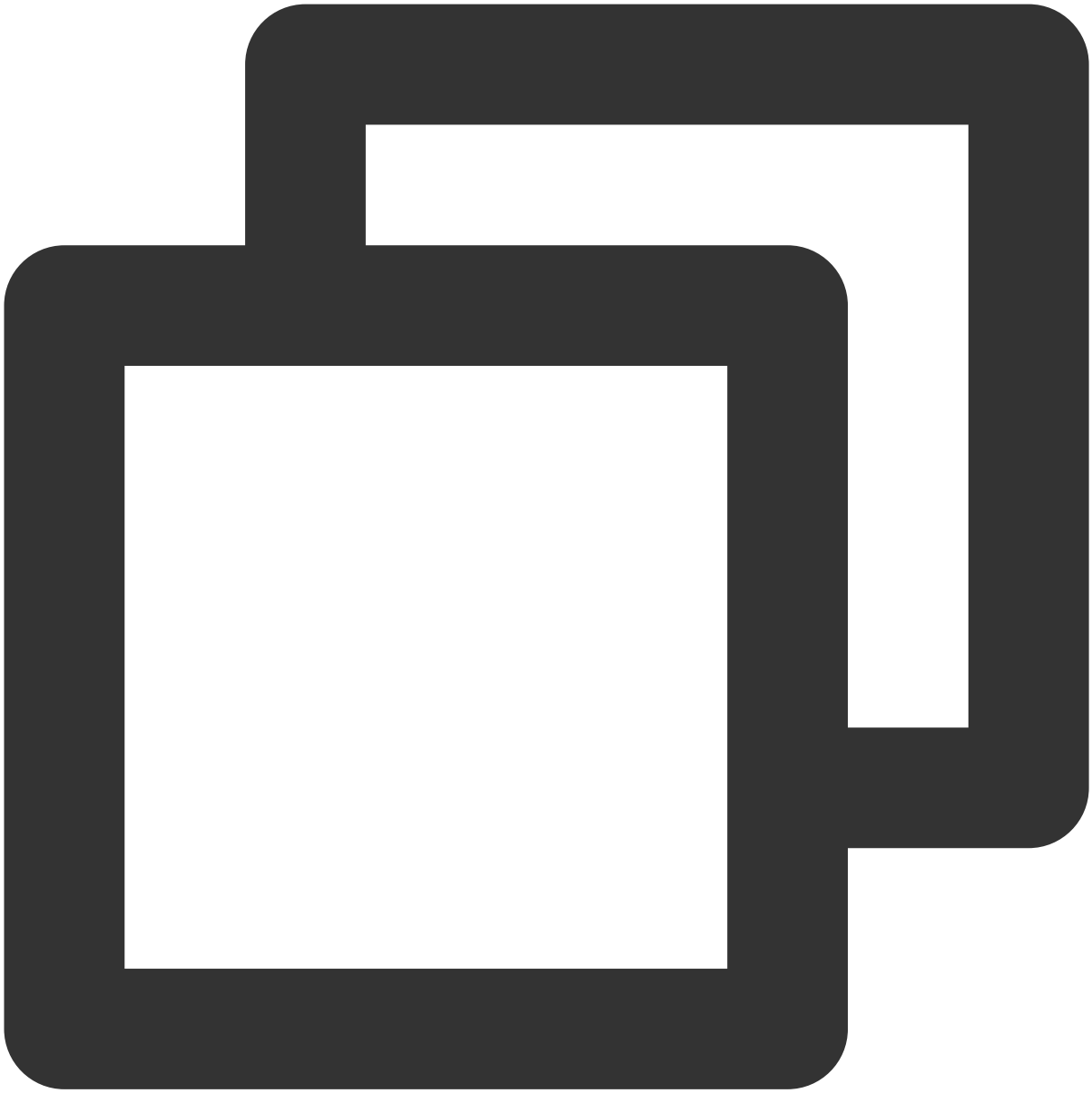
Native static embedding of TF

Dynamic embedding of TTF



```
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])
```

By comparing the code, you can see that TTF mainly modifies the following two parts:

Embedding uses `tf.dynamic_embedding.get_variable` . For more information, see tfra.dynamic_embedding.get_variable.

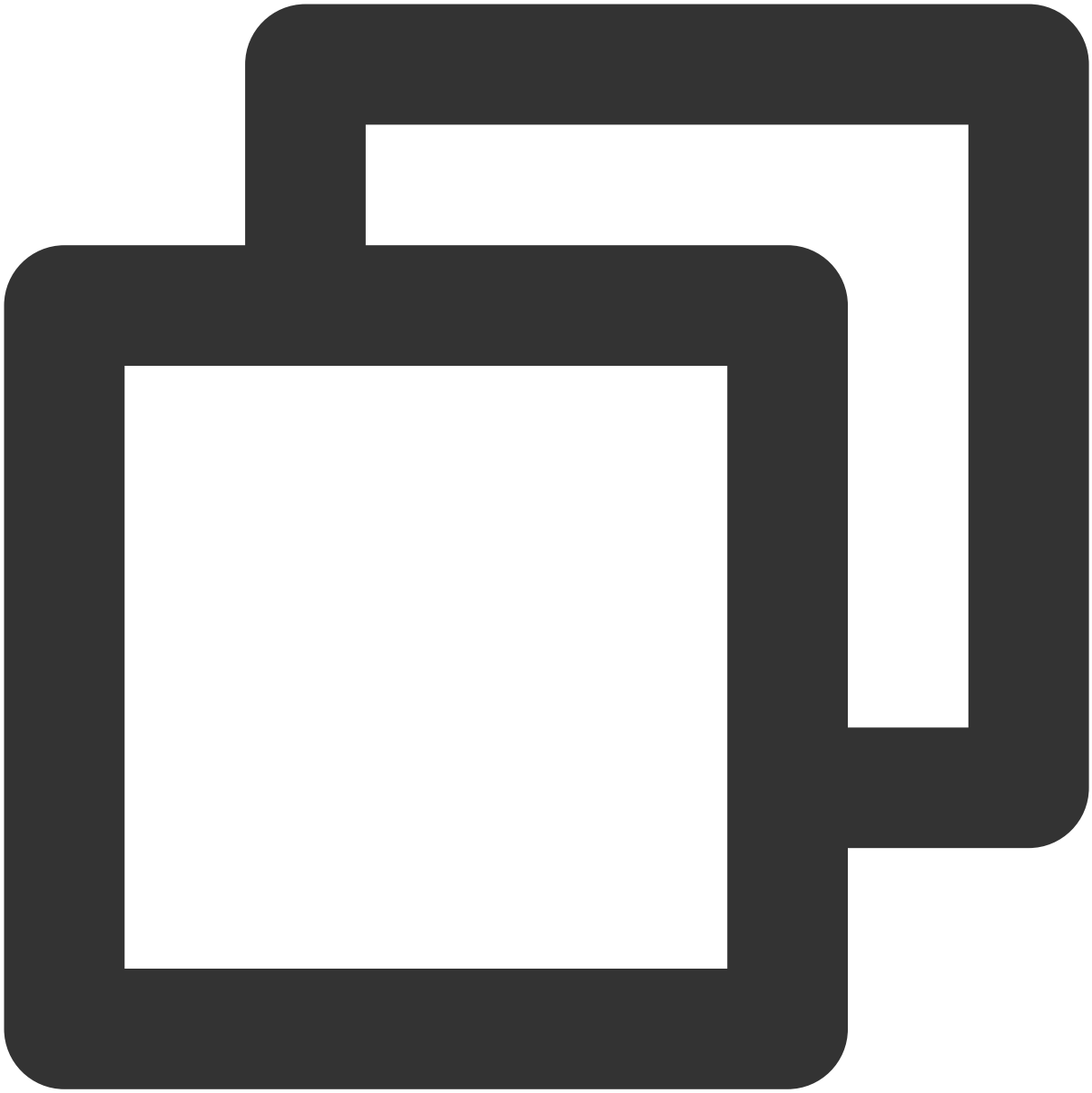
Lookup uses `tf.dynamic_embedding.embedding_lookup` . For more information, see tfra.dynamic_embedding.embedding_lookup.

For the detailed API documentation, see Module: tfra.dynamic_embedding.

Mixed precision

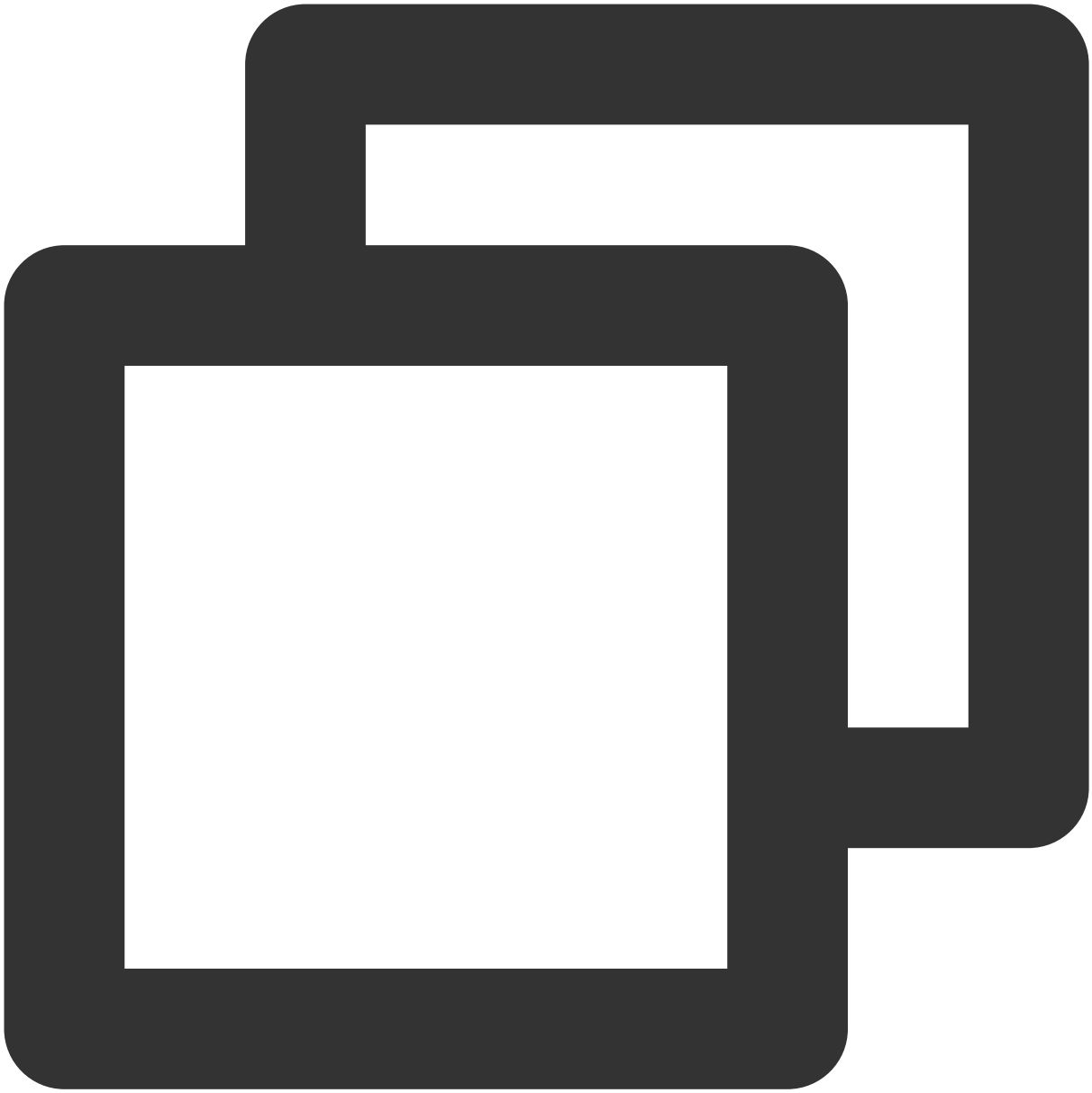
Mixed precision can be implemented by rewriting the code of the optimizer or modifying environment variables:

Through code modification:



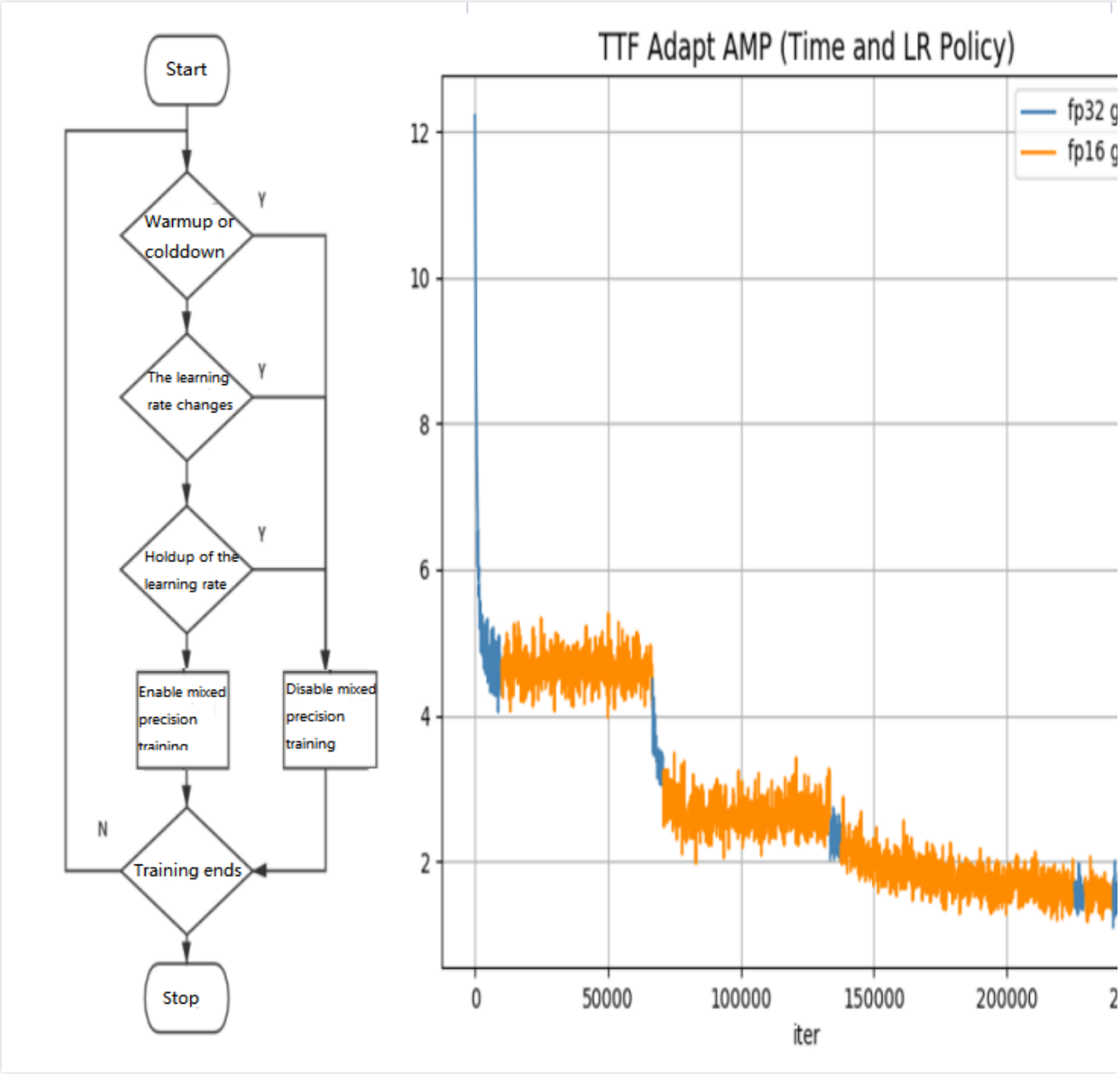
```
opt = tf.train.experimental.enable_mixed_precision_graph_rewrite(opt)
```

Through environment variable modification:



```
export TF_ENABLE_AUTO_MIXED_PRECISION=1
```

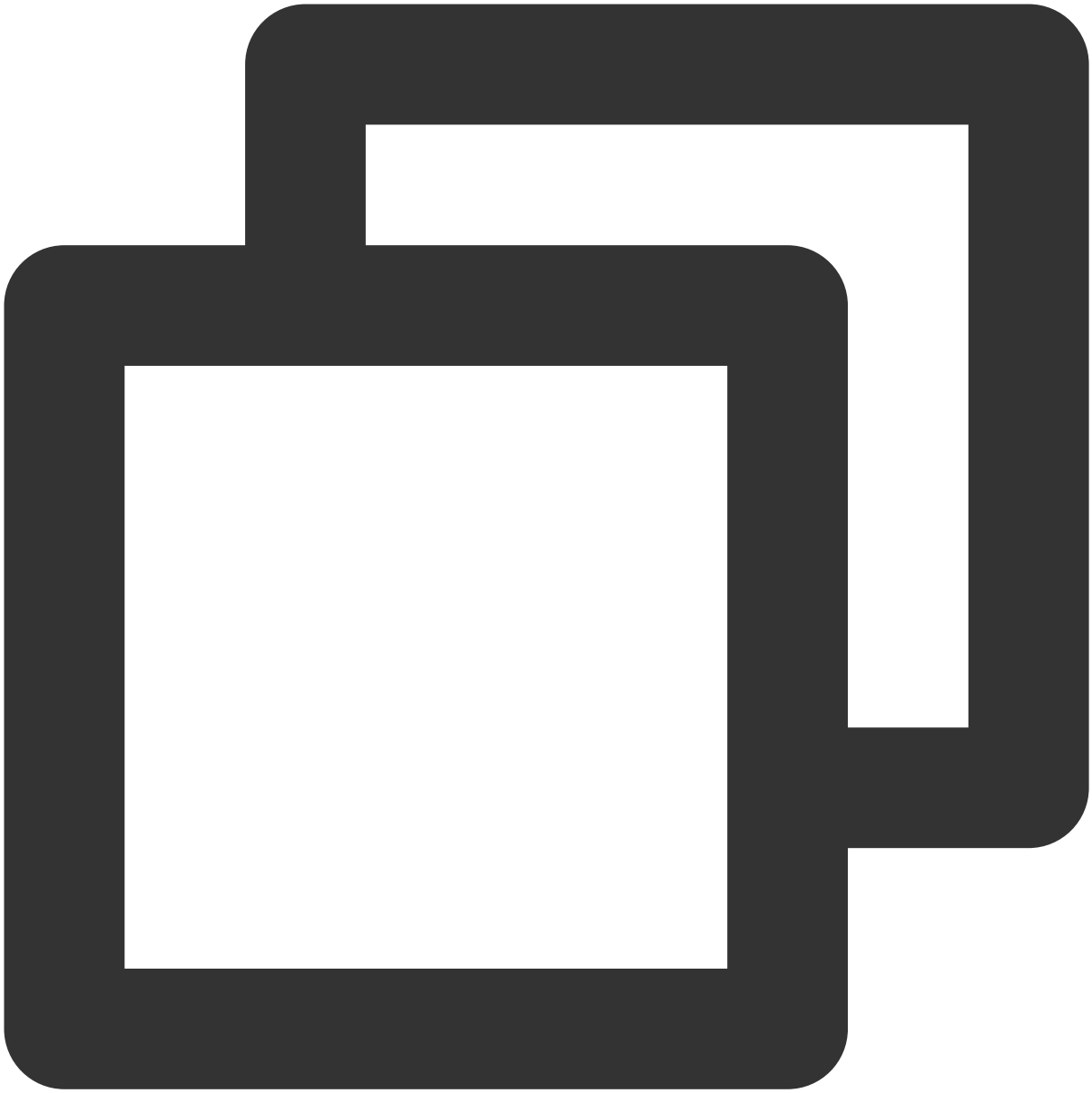
The loss change curve of resnet50 training with the ImageNet dataset by TTF at mixed precisions is as follows:



XLA

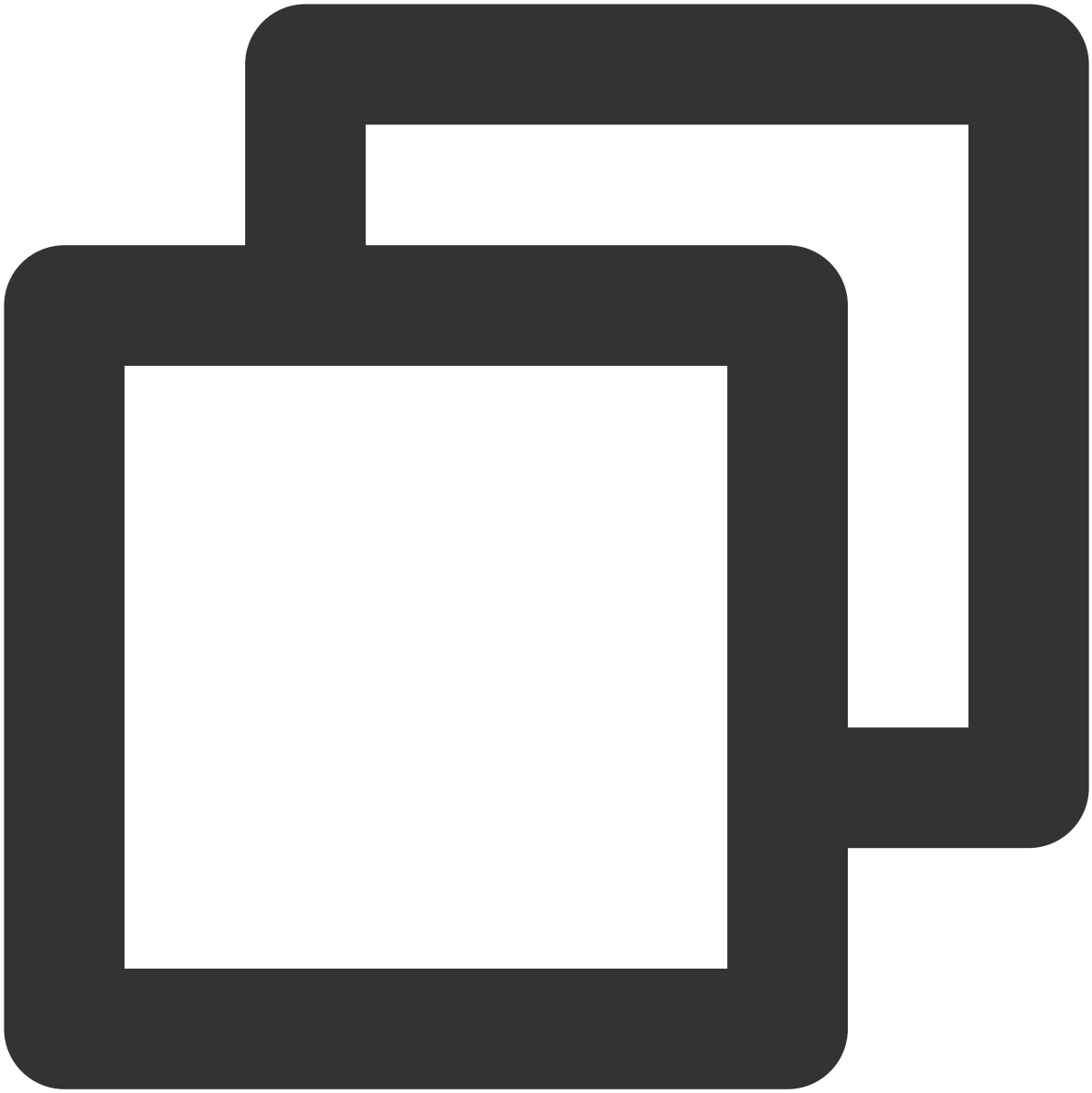
XLA can be configured through the code or environment variables:

Through code modification:



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

Through environment variable modification:

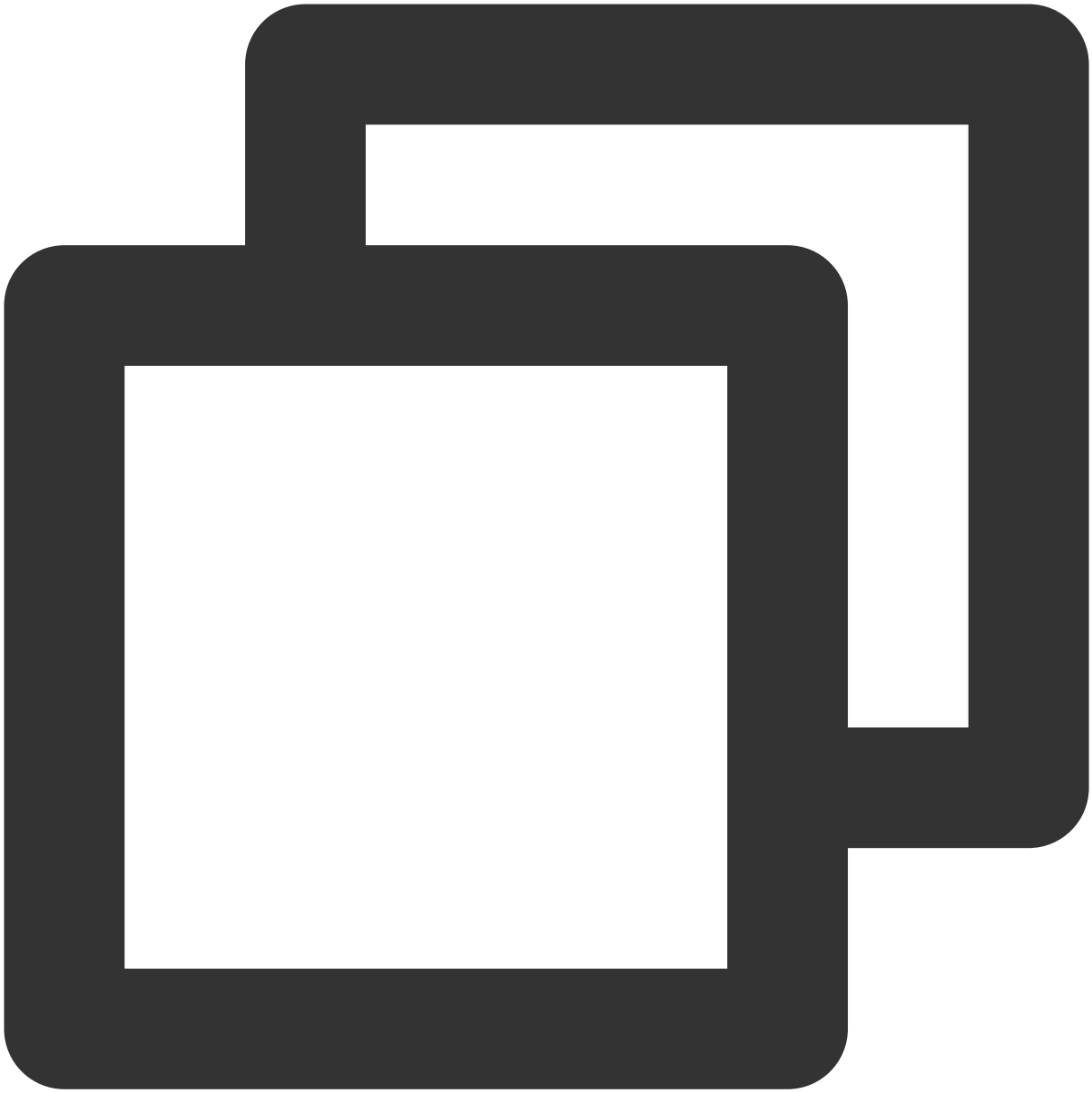


```
TF_XLA_FLAGS=--tf_xla_auto_jit=1
```

Demo

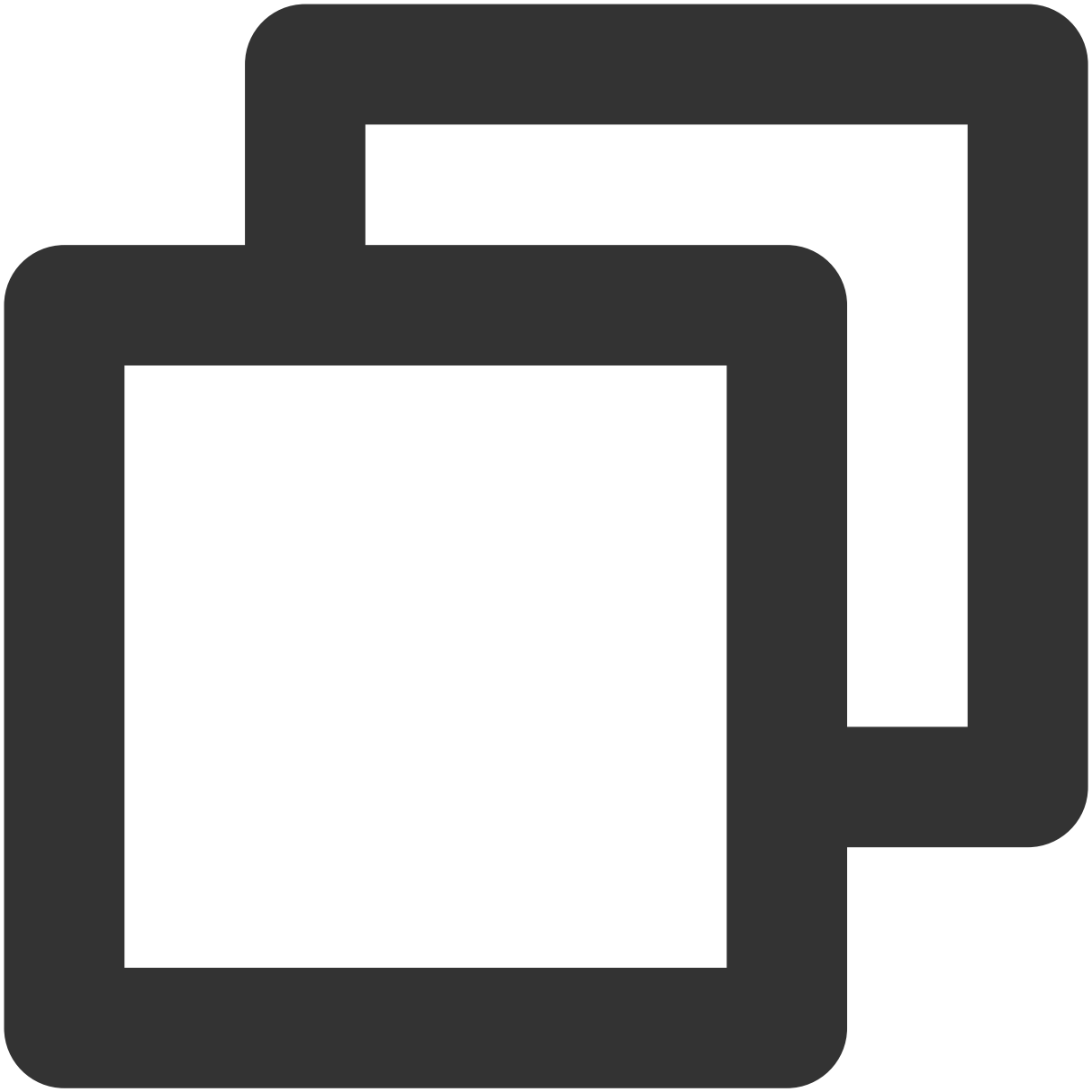
Before running the demo:

1. Run the following command to enter the `demo` directory:



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

2. Run the following command to download the dataset:

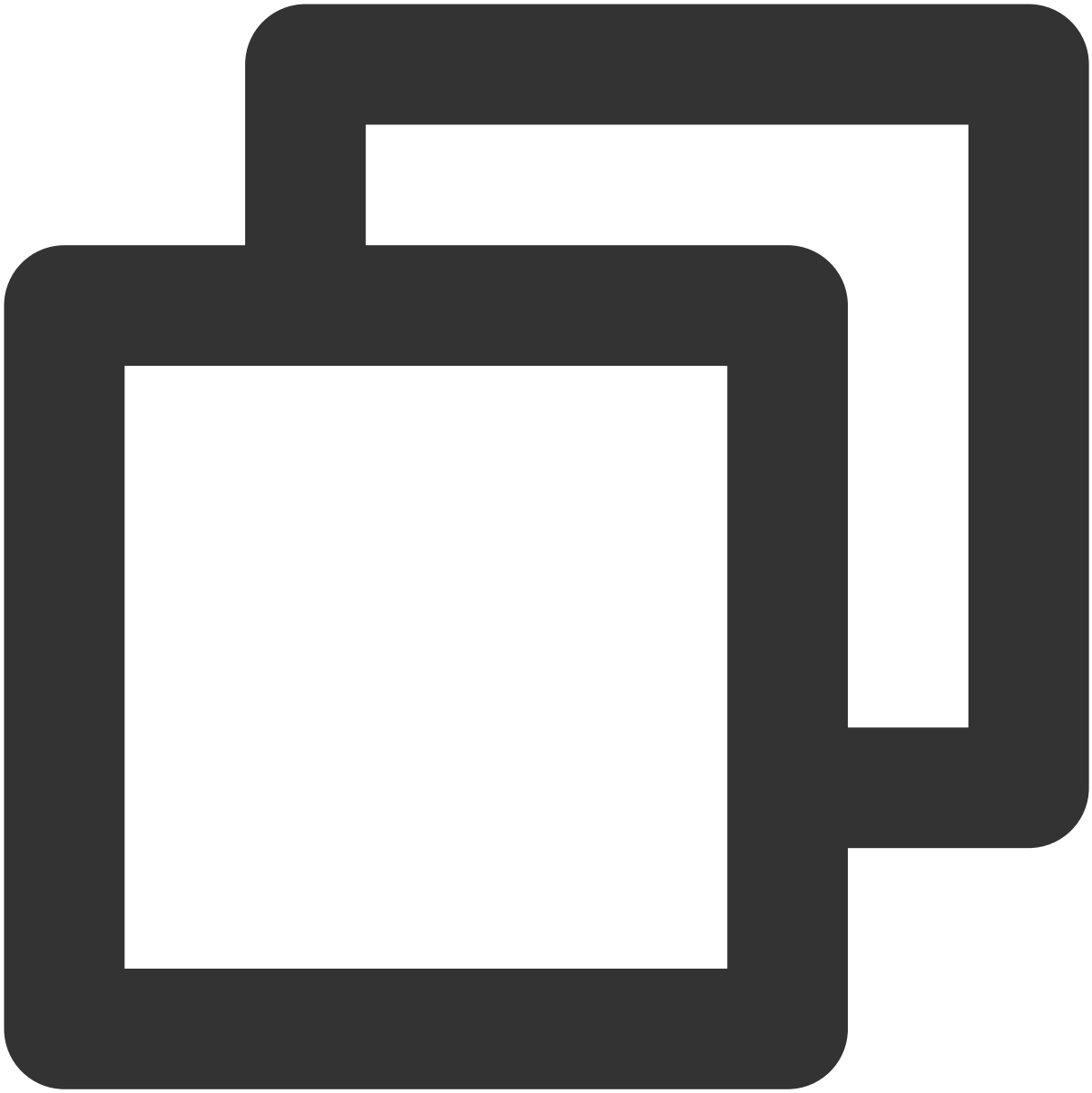


```
bash download_dataset.sh
```

You can get started with TTF quick by using the following demo:

benchmark

This demo is used to compare and test the performance of the dynamic embedding and the native static embedding:

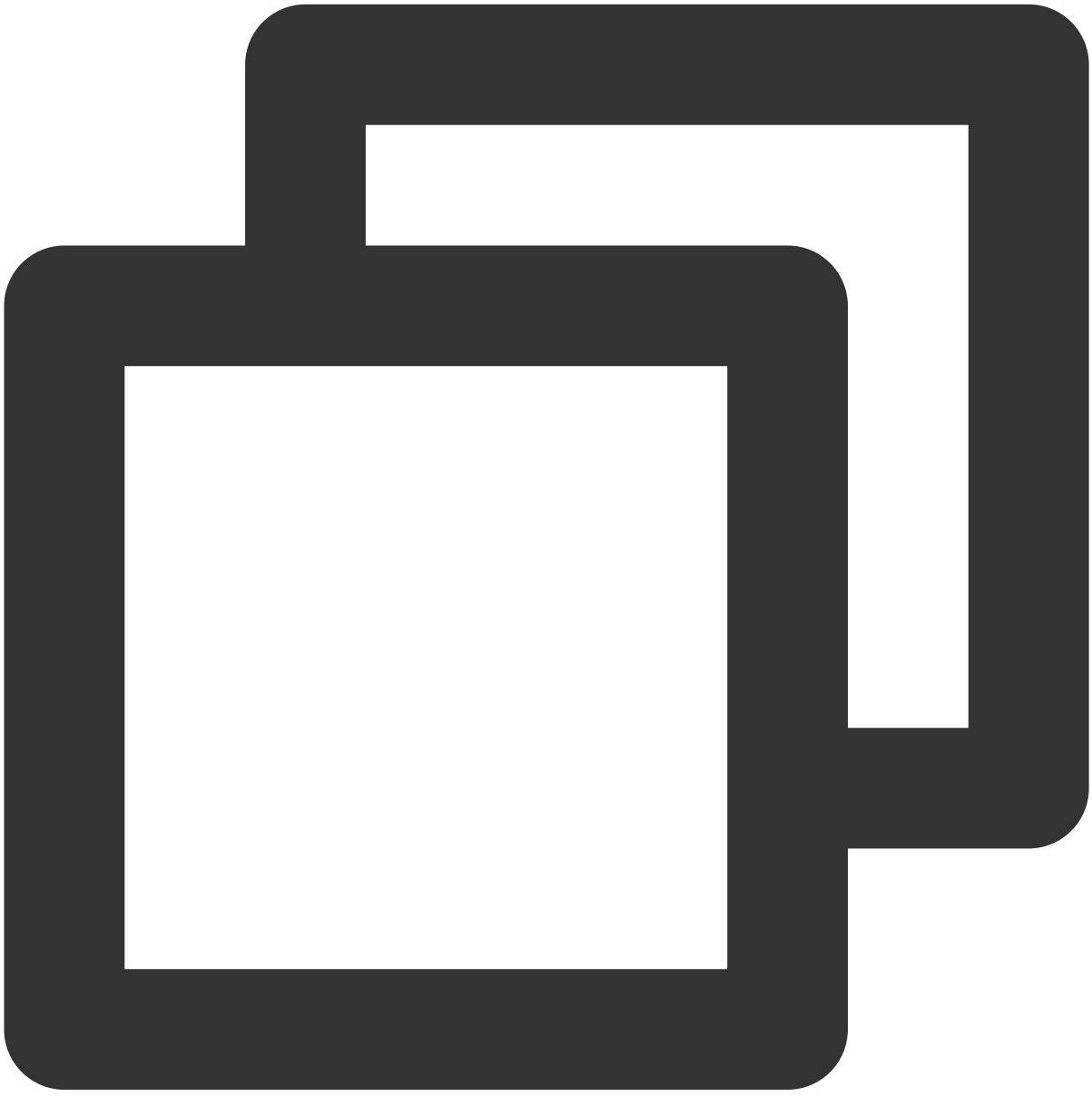


```
# Enter the `benchmark` directory
cd benchmark
# Run the demo according to the default configuration
python train.py

# You need to delete the local dataset cache files every time you modify `batch siz
rm -f .index .data-00000-of-00001
python train.py --batch_size=16384

# Use the static embedding and dynamic embedding to train the DeepFM model respecti
python train.py --batch_size=16384 --is_dynamic=False
```

```
python train.py --batch_size=16384 --is_dynamic=True

# Adjust the number of fully connected layers of the deep part
python train.py --batch_size=16384 --dnn_layer_num=12
```

ps

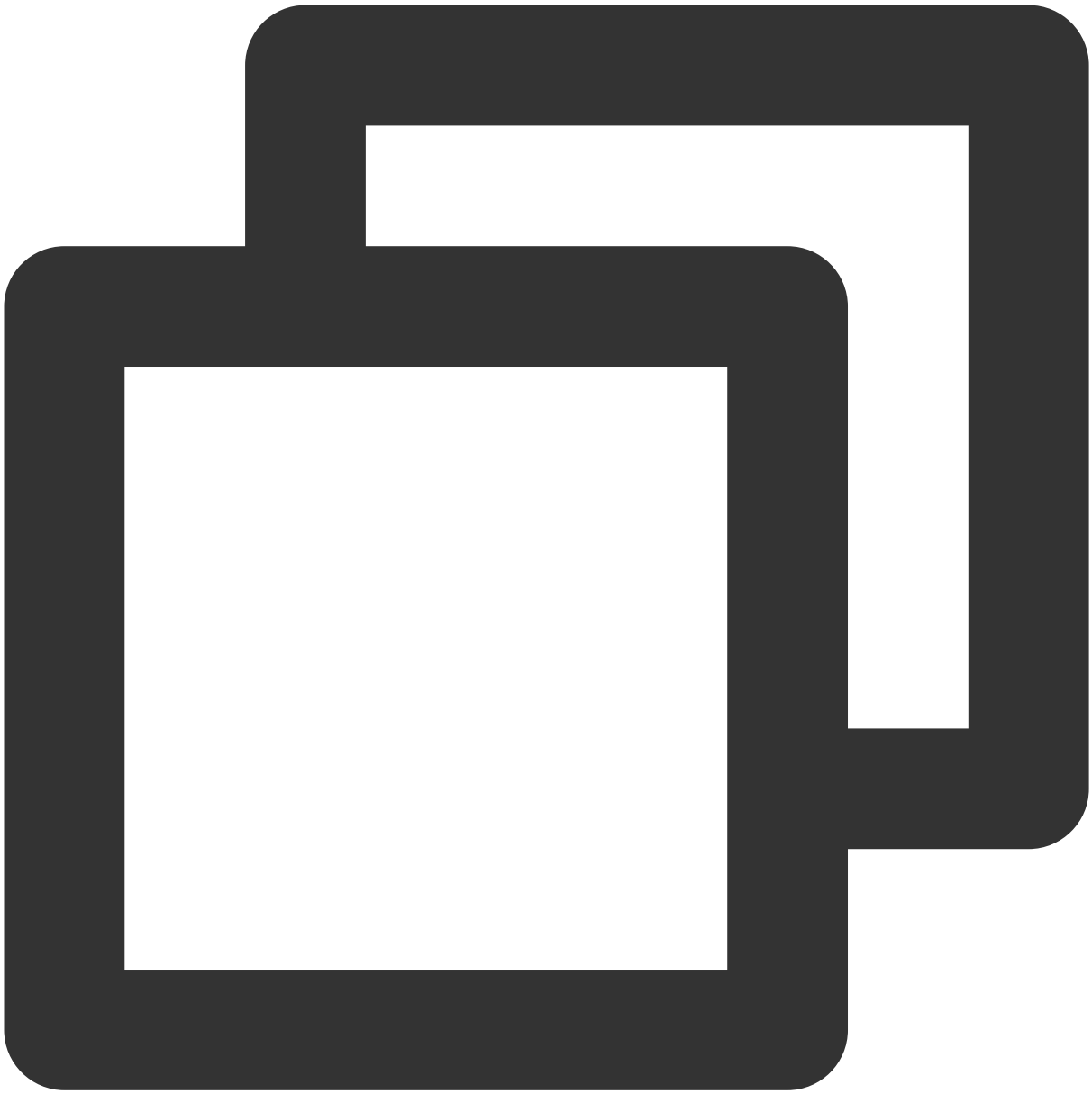
This demo shows how to use the dynamic embedding in `ps` mode.



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

Estimator

This demo shows how to use the dynamic embedding in estimator mode.

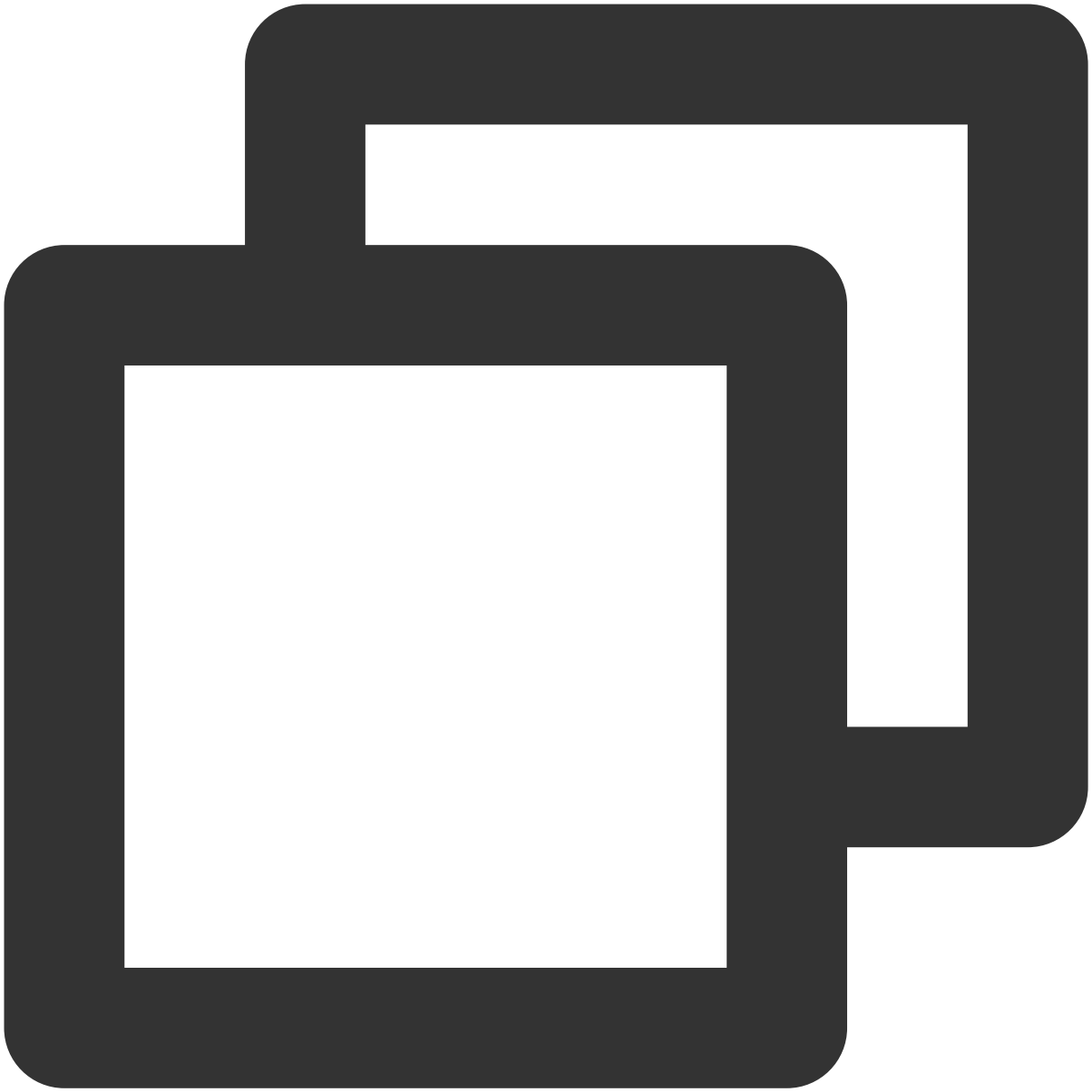


```
cd estimator && bash start.sh
```

Using LightCC

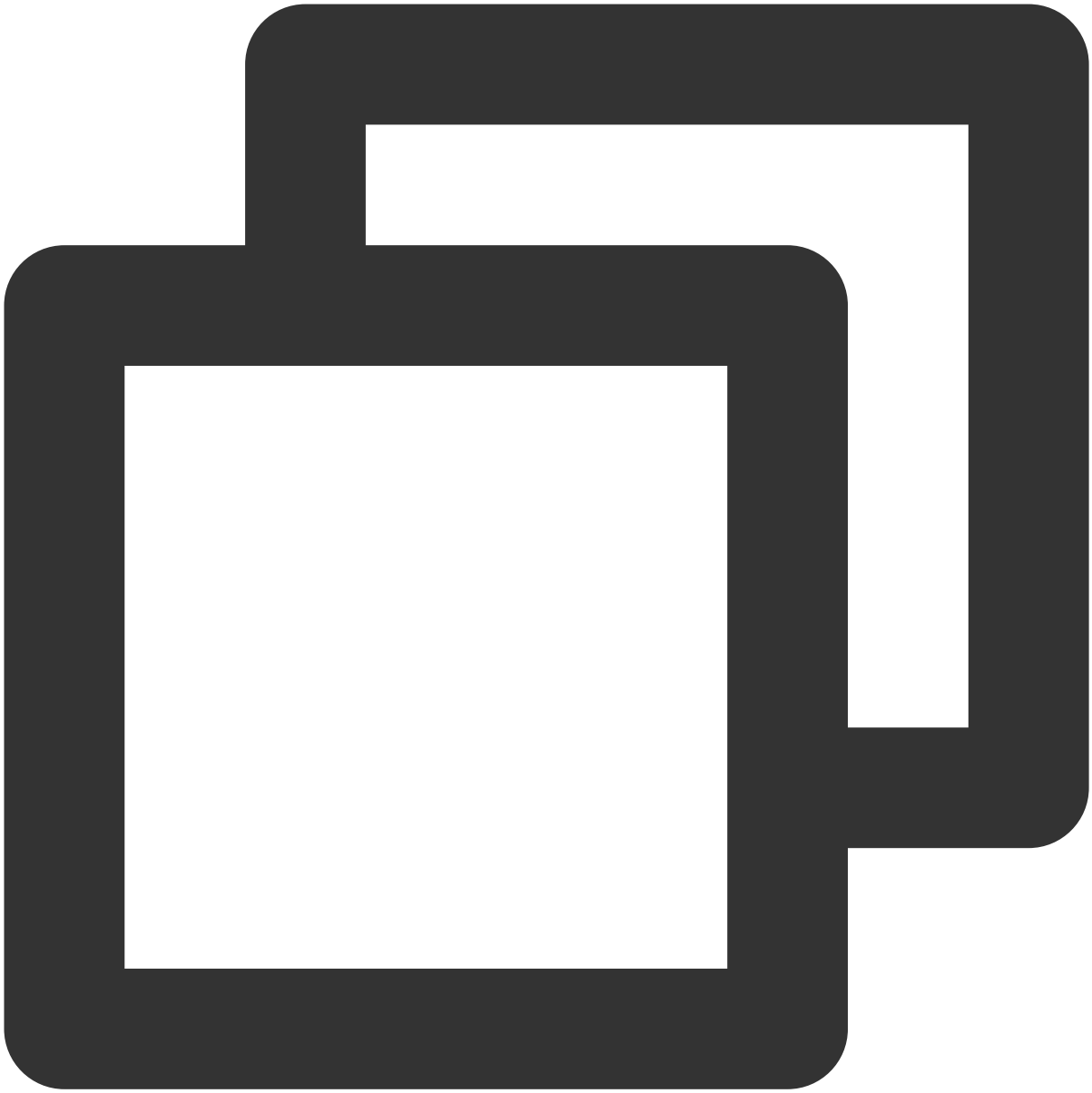
Installation

1. Run the following command as the root user and install LightCC through a Docker image:



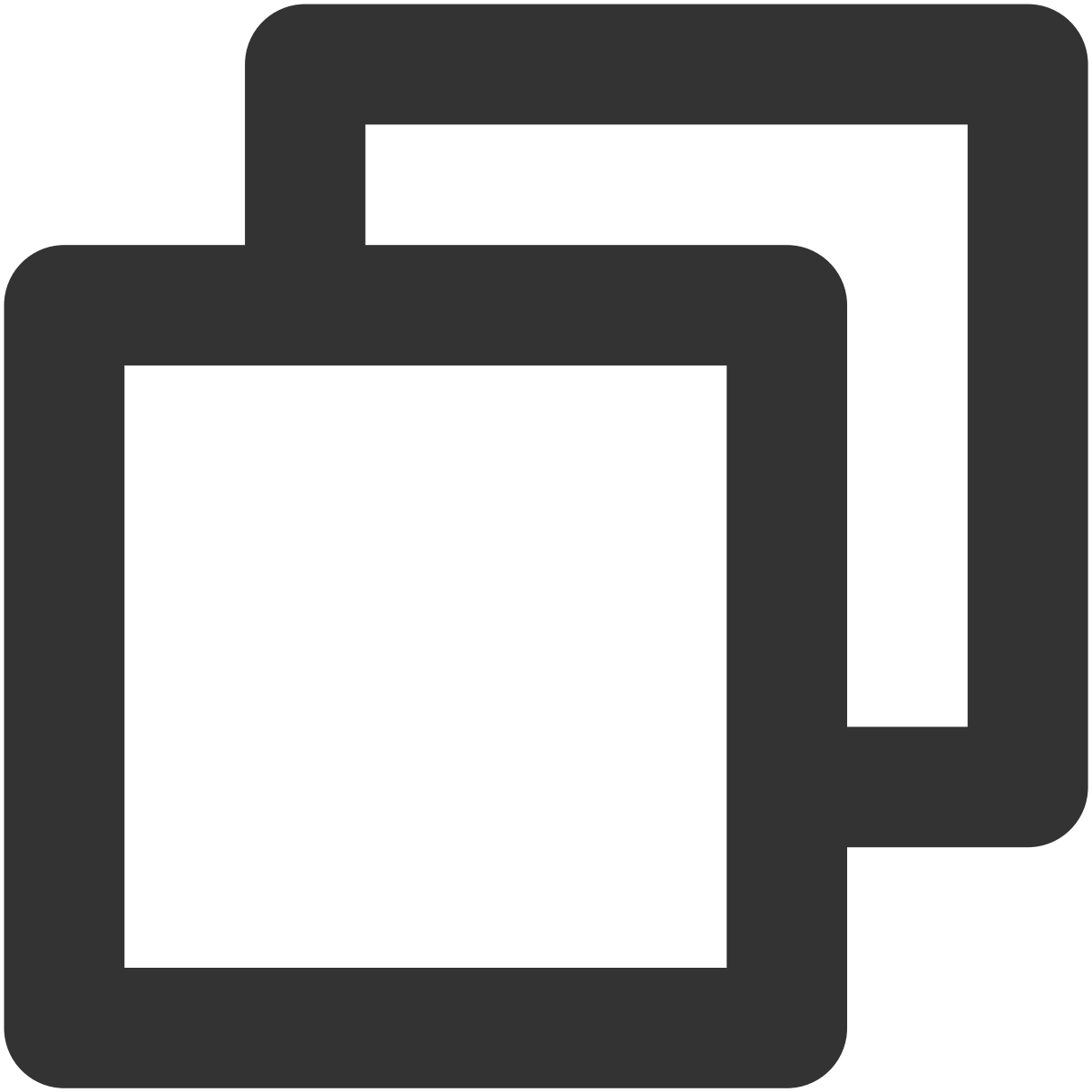
```
docker pull ccr.ccs.tencentyun.com/qcloud/taco-training:cu112-cudnn81-py3-0.3.2
```

2. Run the following command to start Docker:



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

3. Run the following command to view the LightCC version:



```
pip show light-horovod
```

Note:

When the kernel protocol stack is used for NCCL network communication, or if the runtime environment of the HARP protocol stack is not configured, you need to move the `/usr/lib/x86_64-linux-gnu/libnccl-net.so` file in the image to a path other than the system `lib` directory, such as the `/root` directory, as the system will check whether the HARP configuration file exists in a certain directory in the `lib` directory during `init` and will report an error if the file doesn't exist.

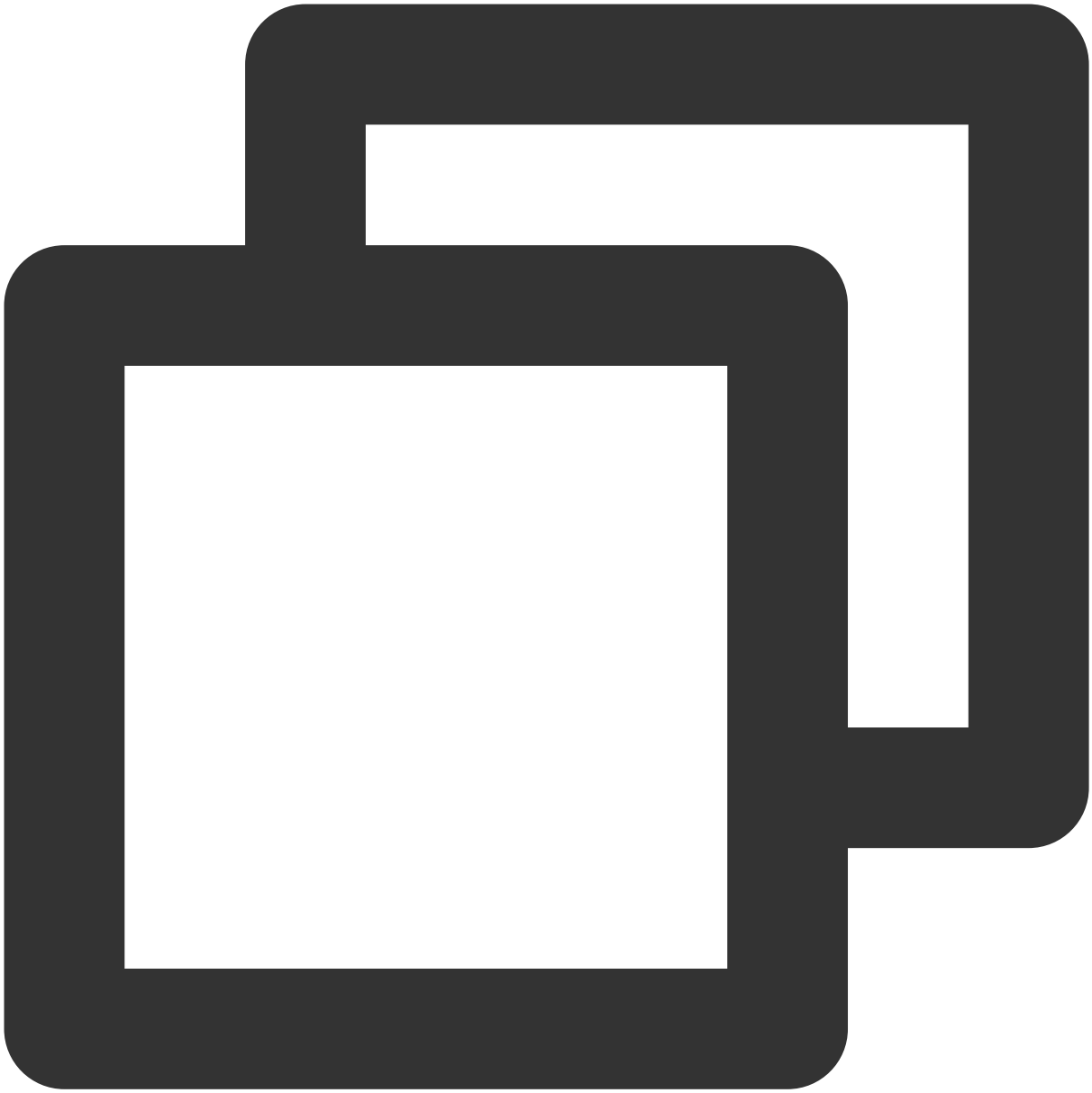
Environment variable configuration

LightCC environment variables are as detailed below, which can be configured as needed:

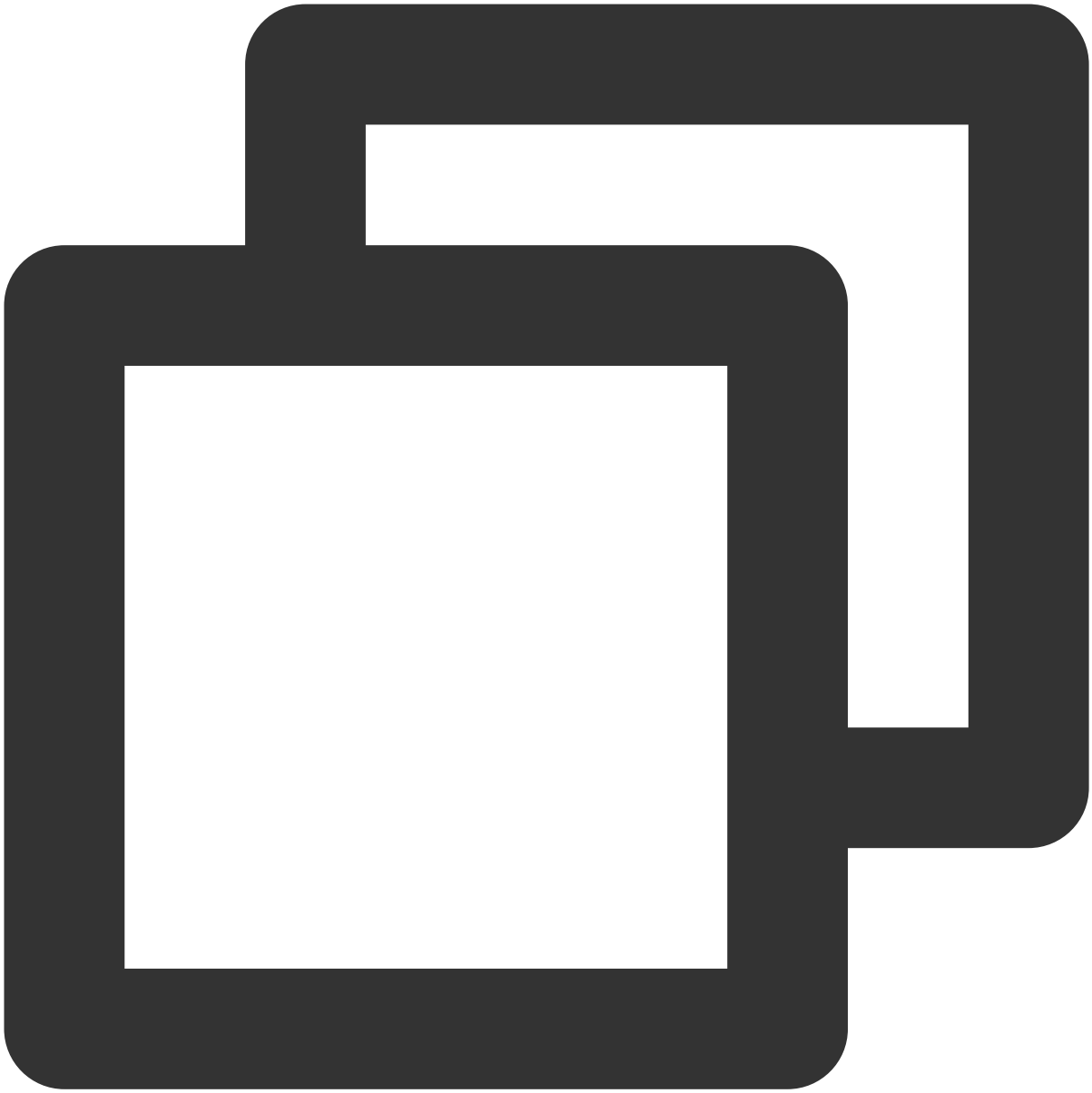
Environment Variable	Default Value	Description
LIGHT_2D_ALLREDUCE	0	Whether to use the 2D-Allreduce algorithm
LIGHT_INTRA_SIZE	8	Number of GPUs in a 2D-Allreduce group
LIGHT_HIERARCHICAL_THRESHOLD	1073741824	Threshold for 2D-Allreduce in bytes. Only data of a size less than this threshold can use 2D-Allreduce.
LIGHT_TOPK_ALLREDUCE	0	Whether to use TOPK to compress the communication data
LIGHT_TOPK_RATIO	0.01	Compression ratio of TOPK
LIGHT_TOPK_THRESHOLD	1048576	Threshold for TOPK compression in bytes. Only communication data of a size greater than or equal to this threshold can be compressed through TOPK.
LIGHT_TOPK_FP16	0	Whether to convert the values of the compressed communication data to FP16

Demo

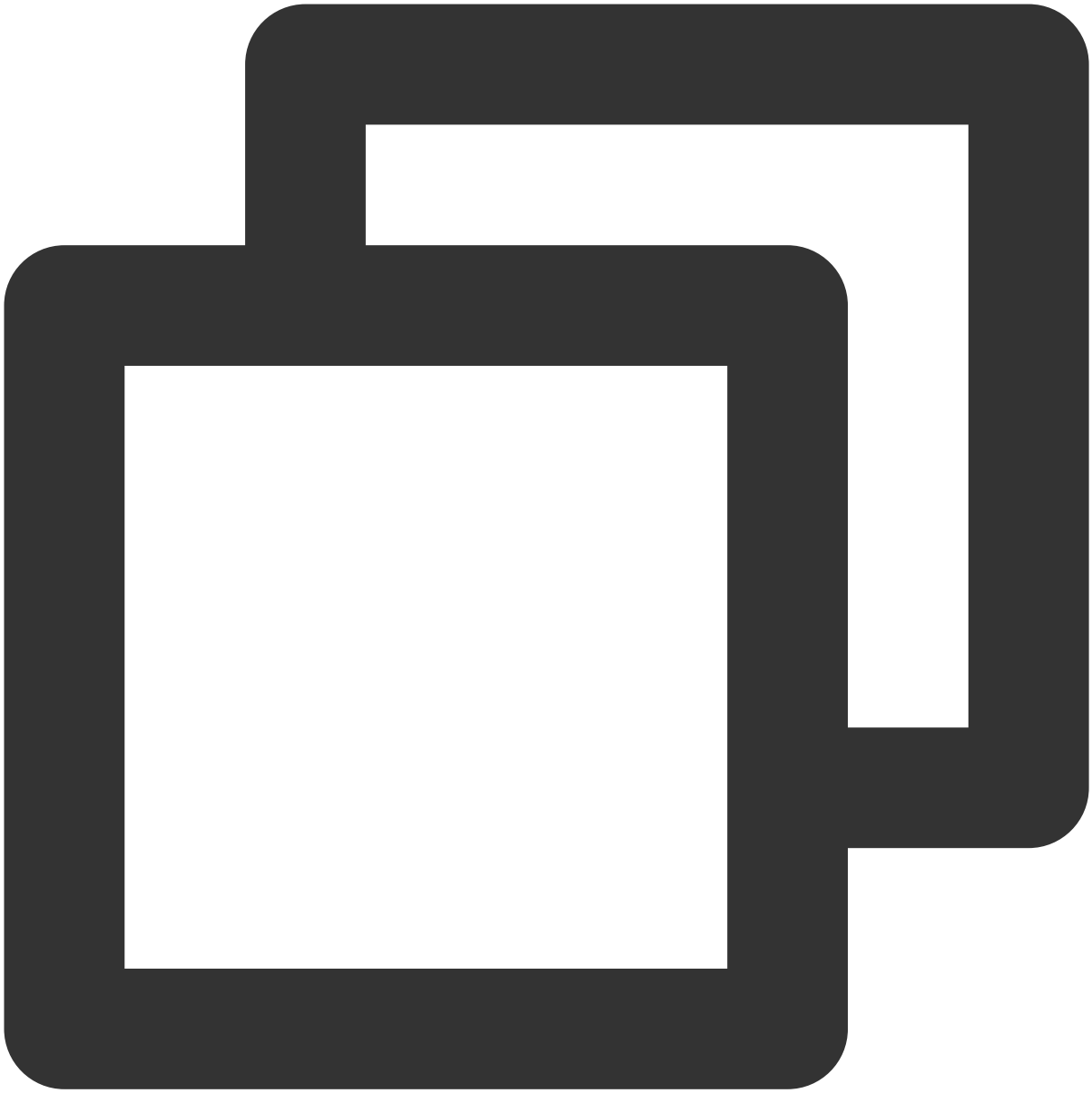
1. After creating two GPU instances, install LightCC and configure environment variables in the above steps.
2. Run the following commands in the container to configure passwordless SSH login for the instances:



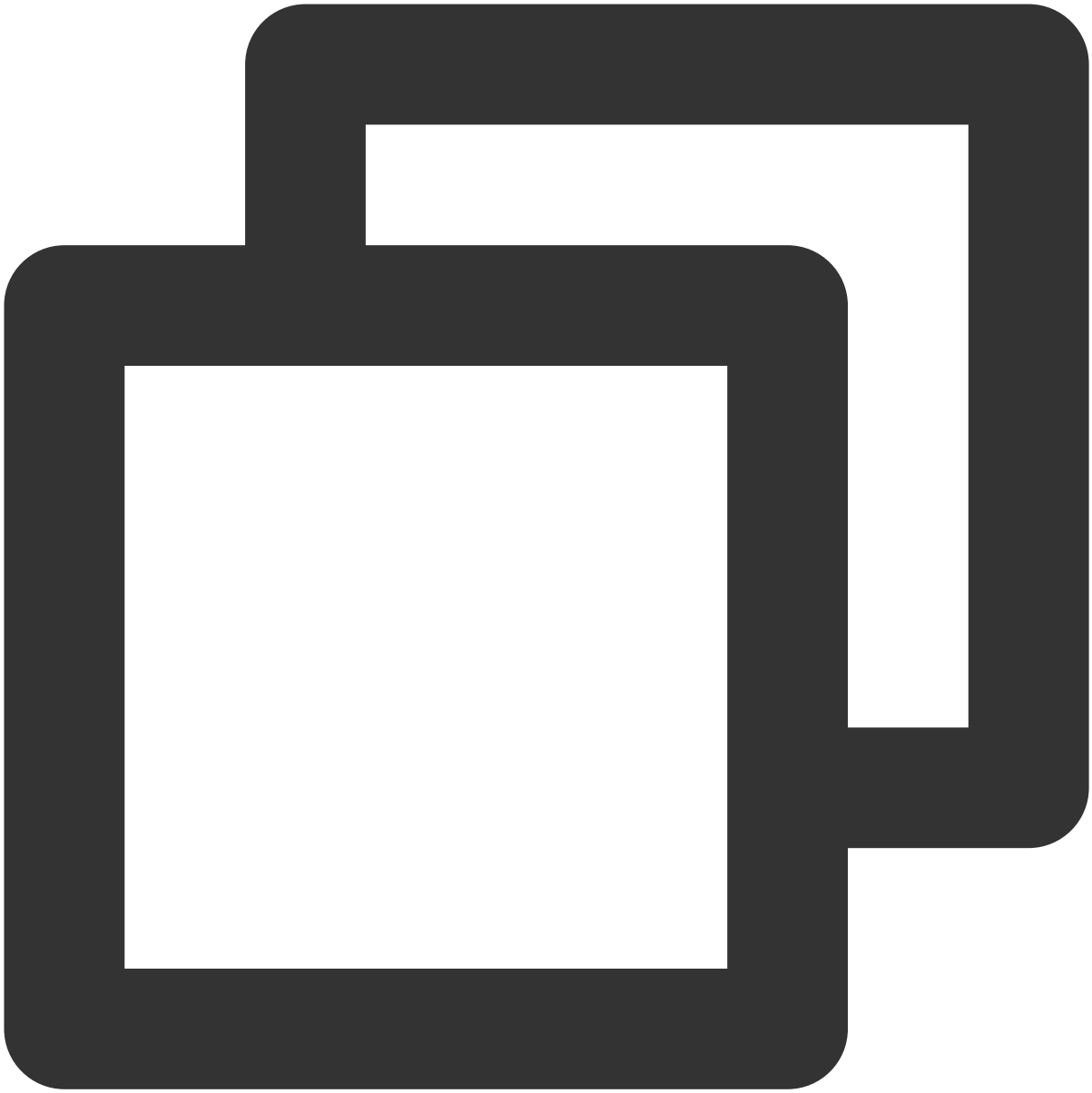
```
# Allow the root user to use the SSH service and start the service (default port: 2
sed -i 's/#PermitRootLogin prohibit-password/PermitRootLogin yes/' /etc/ssh/sshd_co
service ssh start && netstat -tulpn
```



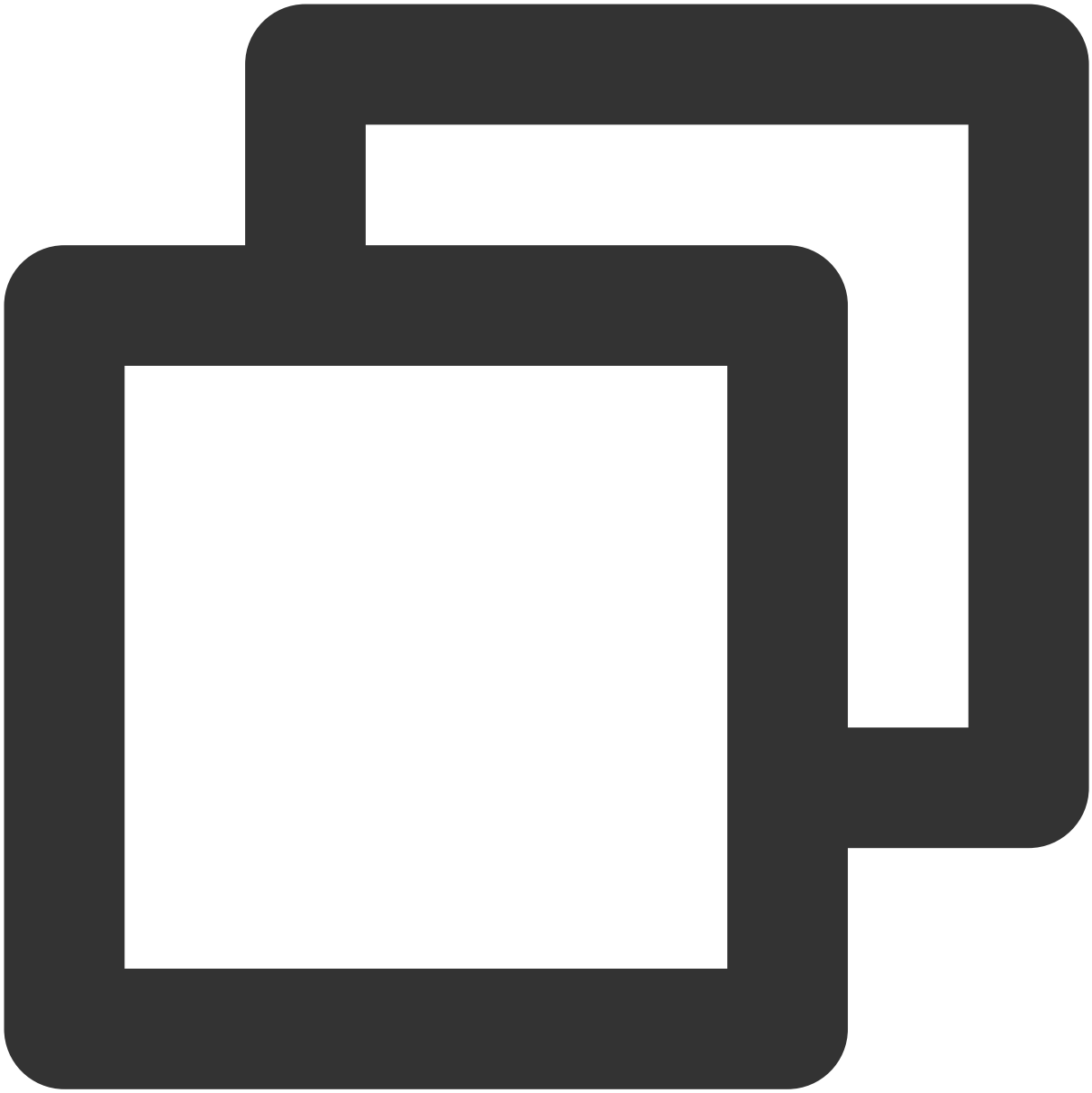
```
# Change the default SSH port in the container to 2222 to avoid conflicts with the
sed -i 's/#Port 22/Port 2222/' /etc/ssh/sshd_config
service ssh restart && netstat -tulpn
```



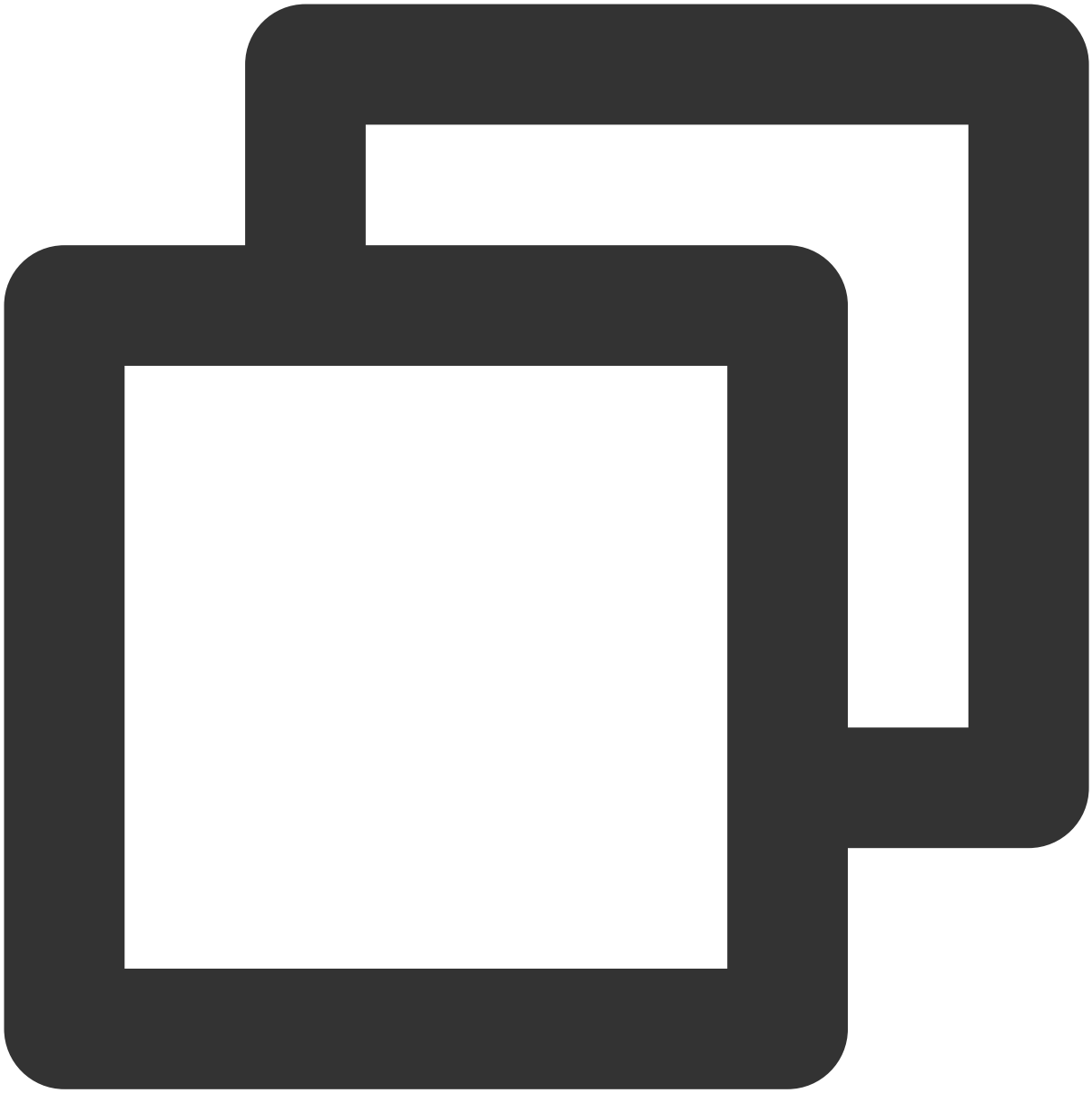
```
# Set `root passwd`
passwd root
```



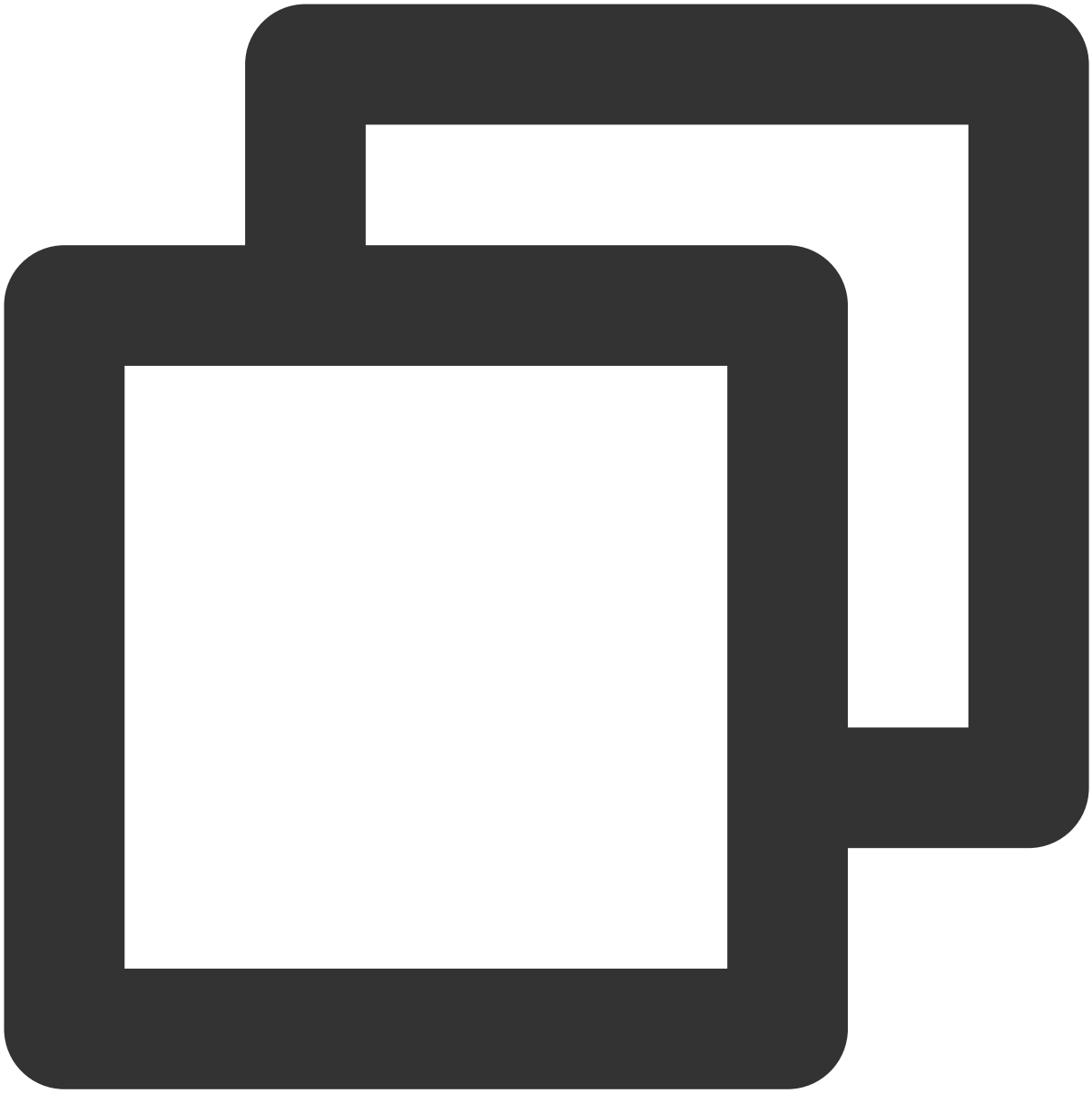
```
# Generate an SSH key
ssh-keygen
```



```
# Configure SSH to use port 2222 by default
# Create `~/.ssh/config`, add the following content, and save and exit the file:
# Note: The IP used here is the IP displayed in `ifconfig eth0` of the two instance
Host gpu1
 hostname 10.0.2.8
 port 2222
Host gpu2
 hostname 10.0.2.9
 port 2222
```

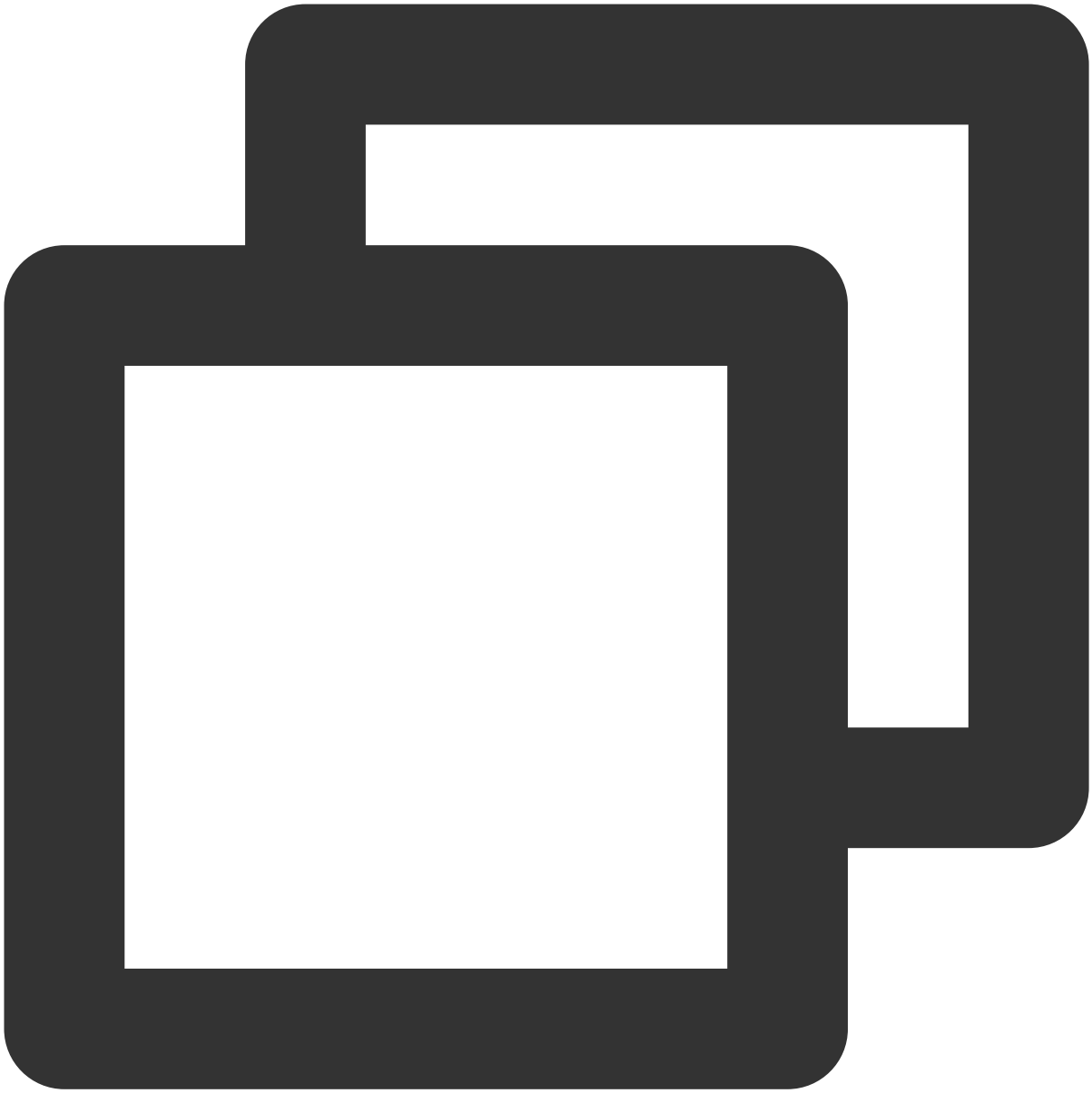


```
# Configure mutual passwordless login for the instances and local passwordless logi
ssh-copy-id gpu1
ssh-copy-id gpu2
```



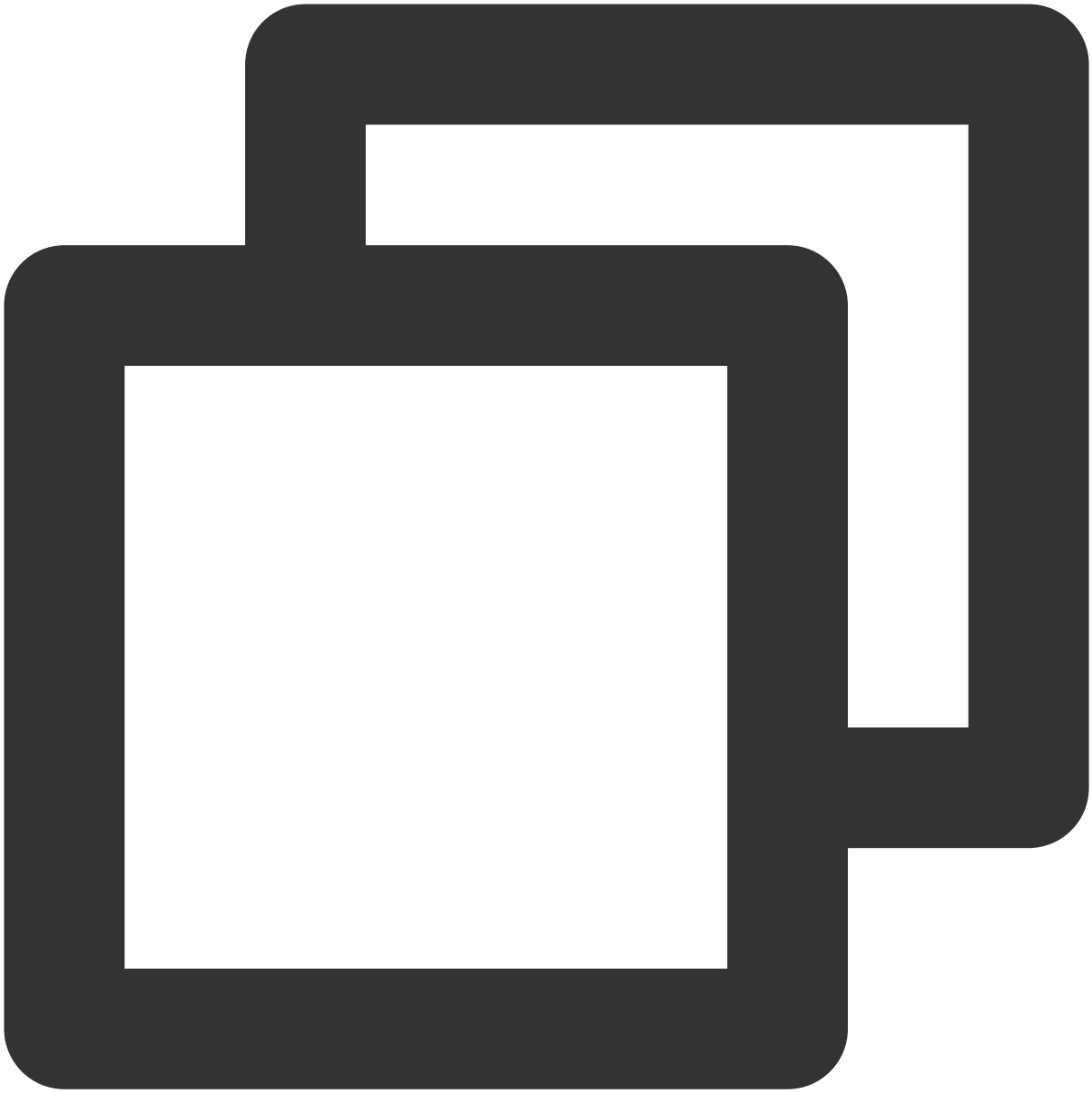
```
# Test whether passwordless login is configured successfully
ssh gpu1
ssh gpu2
```

3. Run the following command to download the benchmark test script of Horovod:



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

4. Run the following command to start multi-server training benchmark based on ResNet-50:



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

Here, the command parameters are used for an 8-card model. To configure another model, modify the `-np` and `-H` parameters. Other parameters are as detailed below:

`NCCL_ALGO=RING` : Select the ring algorithm as the communication algorithm in NCCL.

`NCCL_DEBUG=INFO` : Enable debugging output in NCCL.

`-mca btl_tcp_if_include eth0` : Select the eth0 device as network device for MPI multi-server communication. As some ENIs cannot communicate, you need to specify the ENI if there are multiple ones; otherwise, an error will occur if MPI chooses an ENI that cannot communicate.

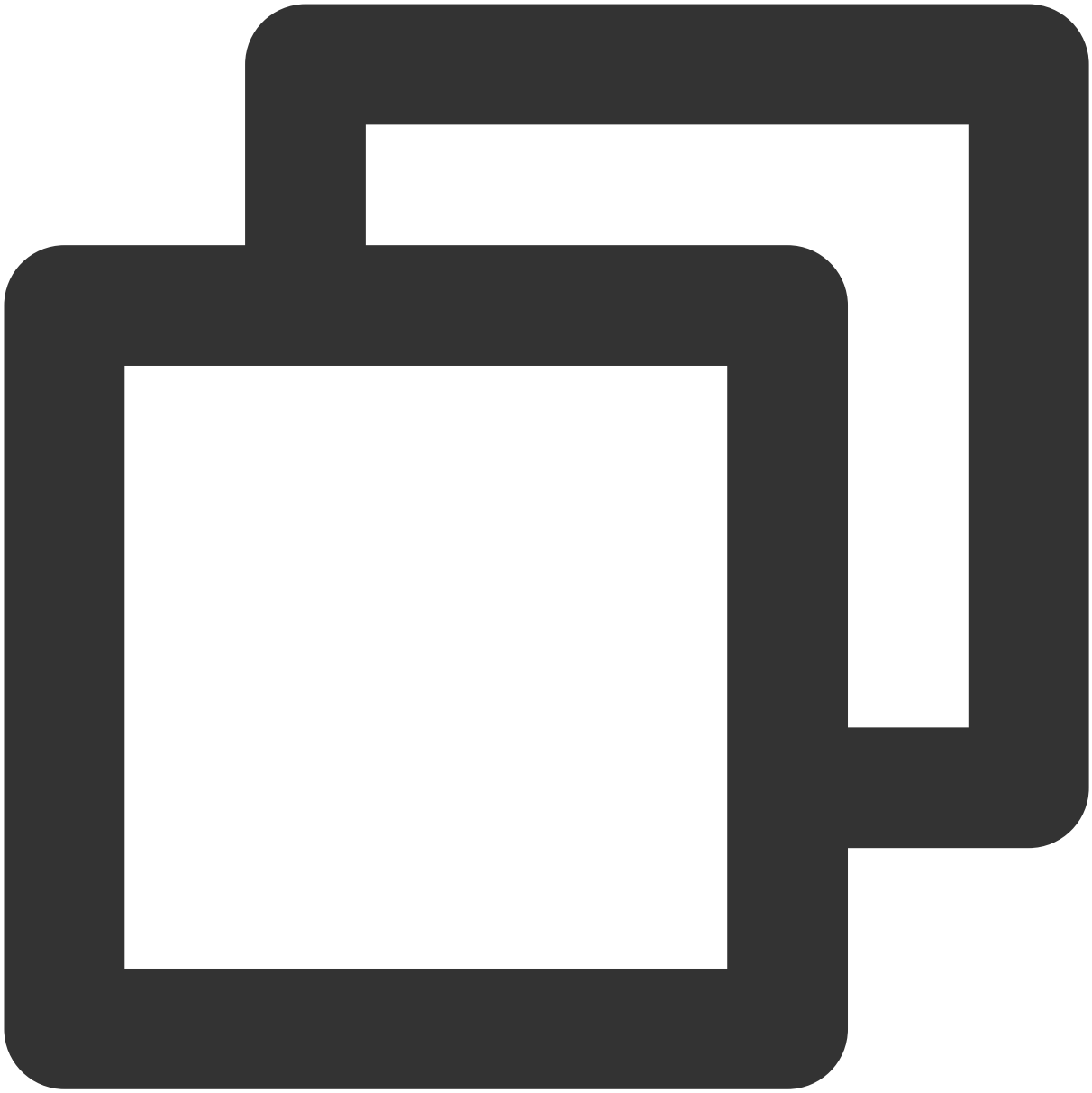
The reference throughput rates of LightCC multi-server training benchmark in two GT4.41XLARGE948 instances are as detailed below:

Model: CVM GT4.41XLARGE948 (A100 * 8 + 50G VPC) GPU driver: 460.27.04 Container: 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

Using HARP

Preparing the instance environment

1. Run the following command as the root user to modify `cmdline` of the kernel and configure a 50 GB huge page memory.



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

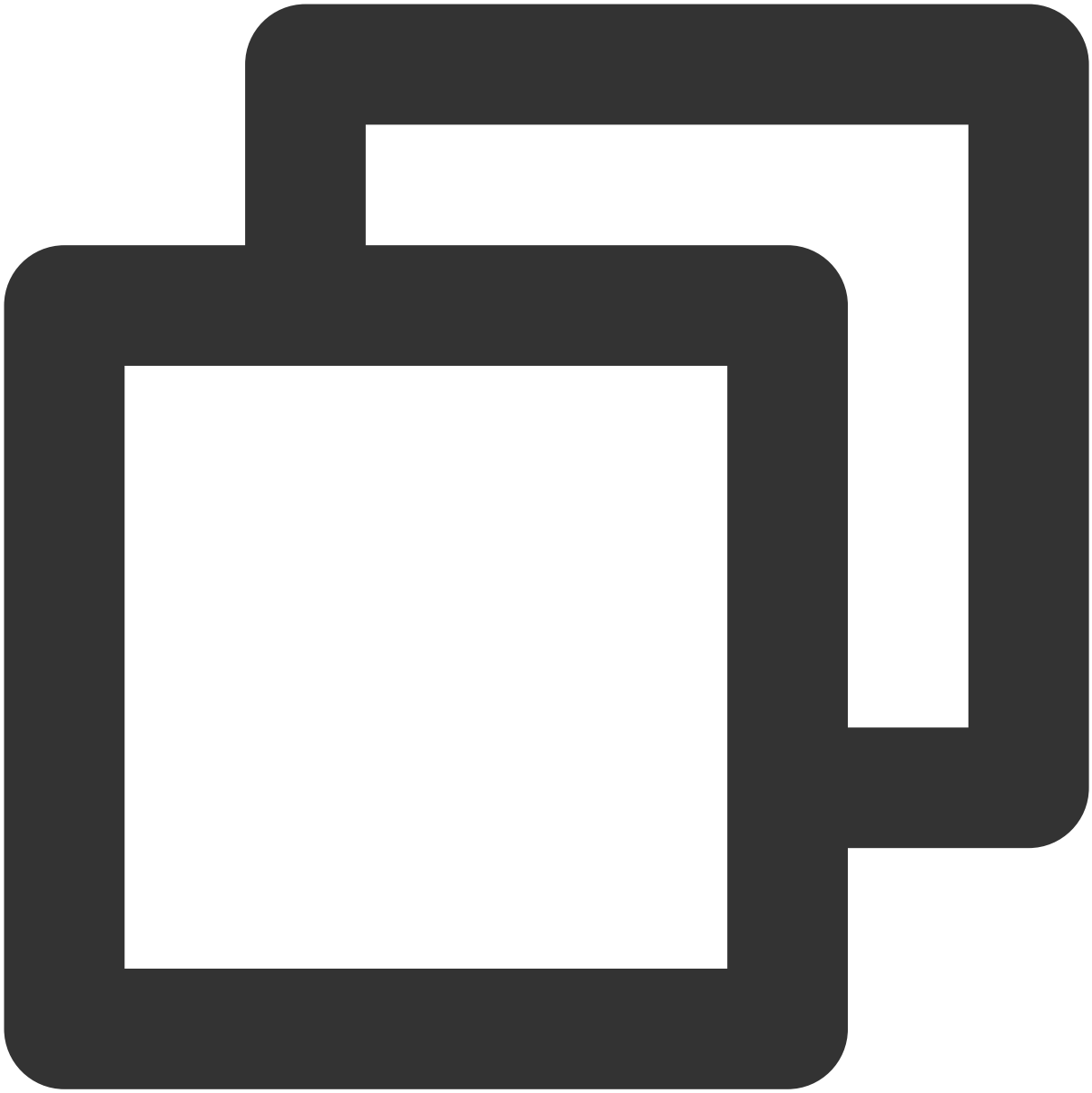
Note:

You can configure `hugepages=50` for an 8-card instance or `hugepages= (number of GPU cards * 5 + 10)` for other models.

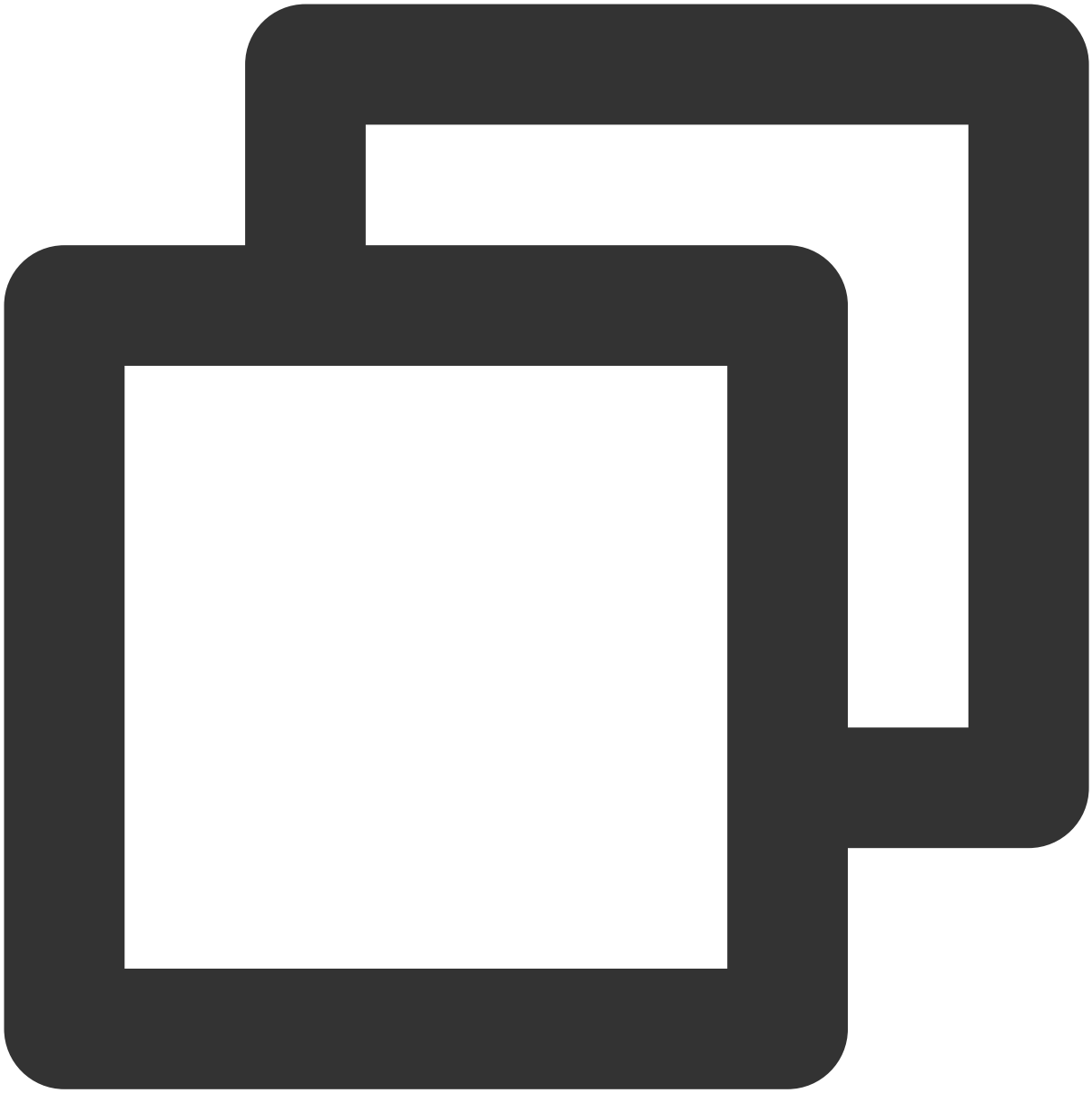
2. Run the following command based on your operating system version to make the configuration take effect and restart the instance:

Ubuntu

CentOS or TencentOS



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



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

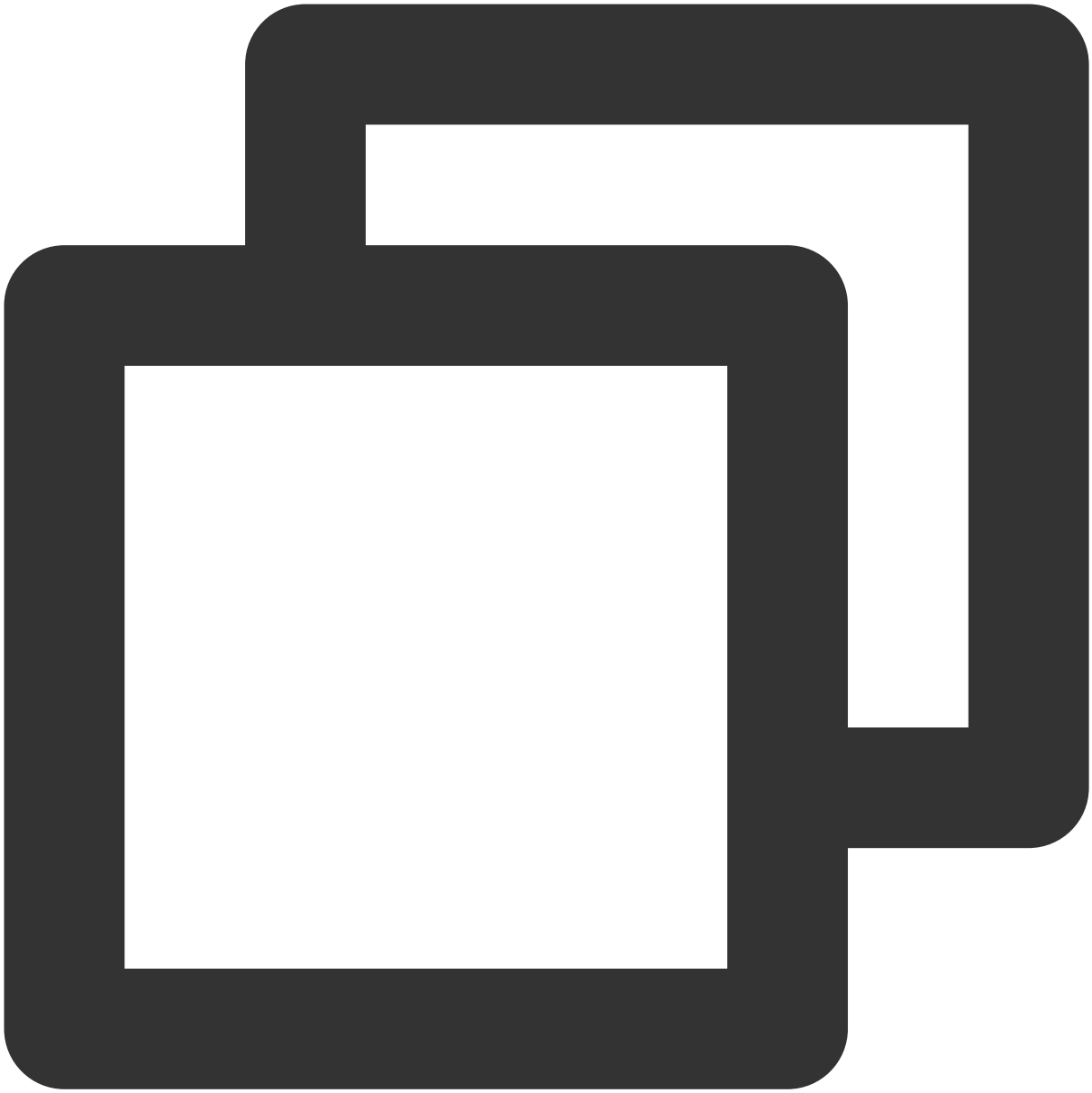
3. Bind ENIs as follows. The number of ENIs should be the same as the number of GPU cards.

3.1 Log in to the CVM console, select **More** > **IP/ENI** > **Bind ENI** on the right of the target instance.

3.2 In the **Bind ENI** pop-up window, select created ENIs or create new ones and bind them as needed.

3.3 Click **OK**.

4. Run the following command to initialize the configuration information.

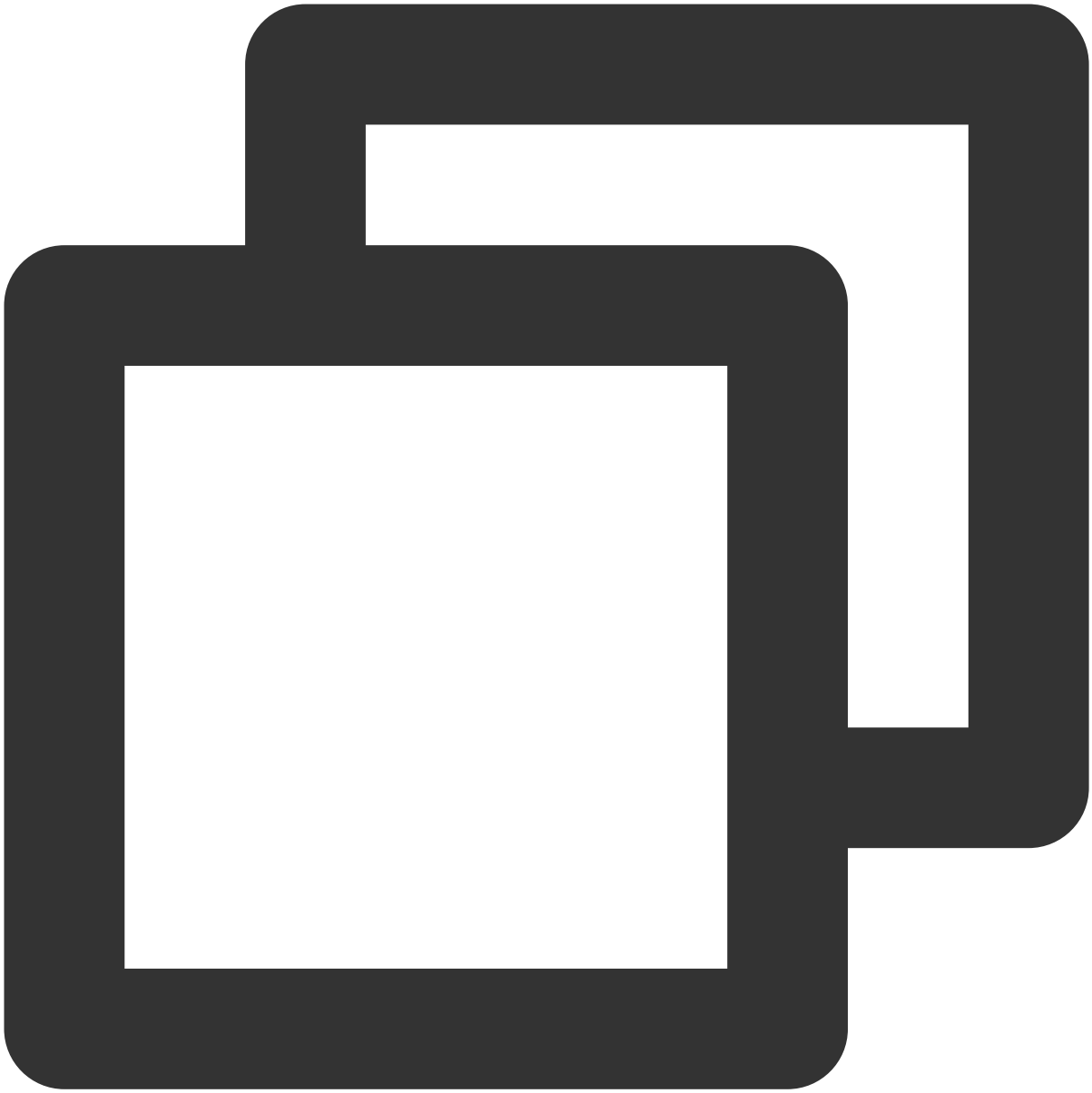


```
curl -s -L http://mirrors.tencent.com/install/GPU/taco/harp_setup.sh | bash
```

If the input result contains "Set up HARP successfully", and the `ztcp*.conf` configuration file is generated in the `/usr/local/tfabric/tools/config` directory, the configuration has been completed successfully.

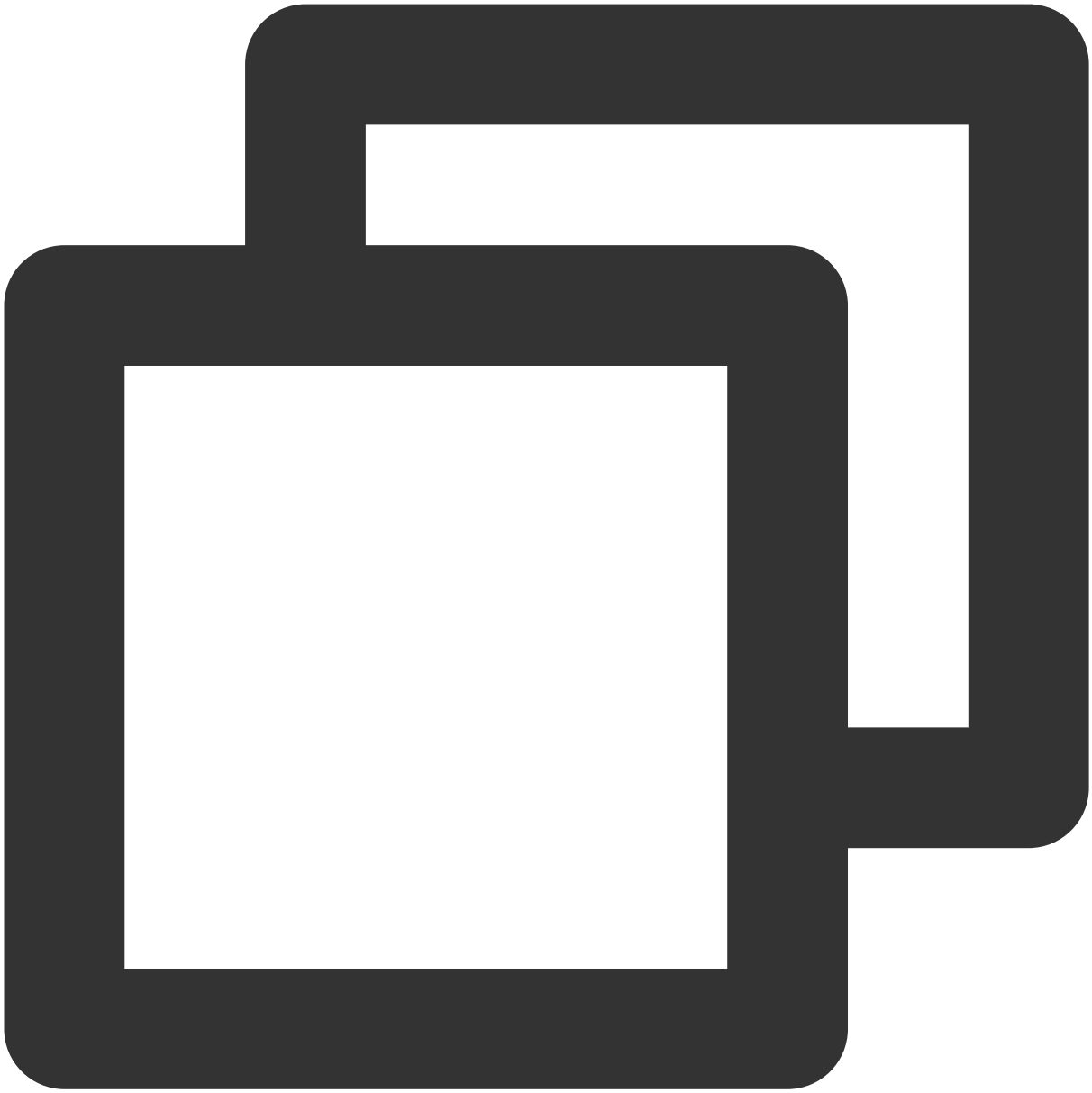
Installation

1. Run the following command as the root user and install HARP through a Docker image:



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

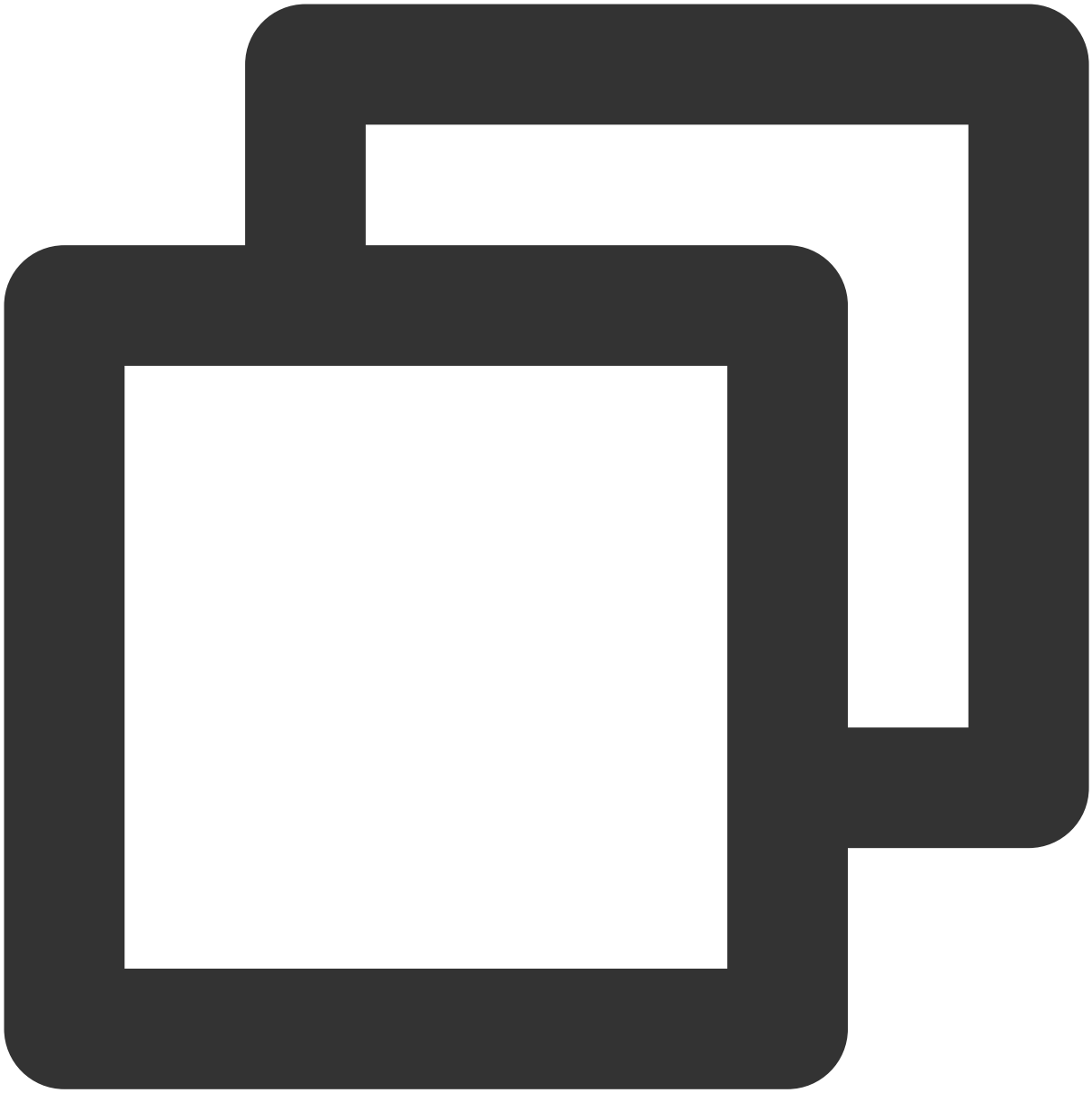
2. Run the following command to start Docker:



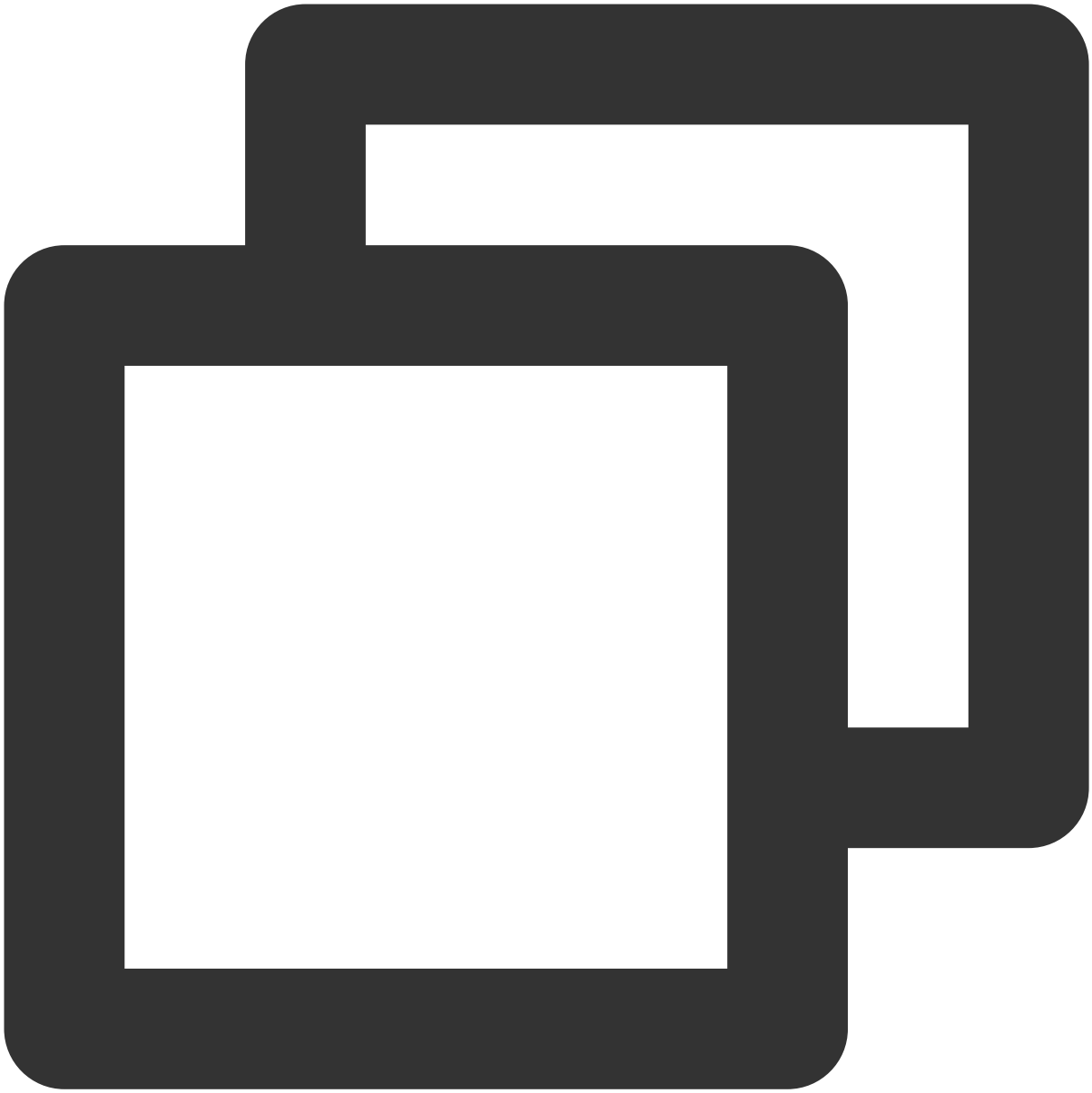
```
docker run -it --rm --gpus all --privileged --net=host -v /sys:/sys -v /dev/hugepag
```

Demo

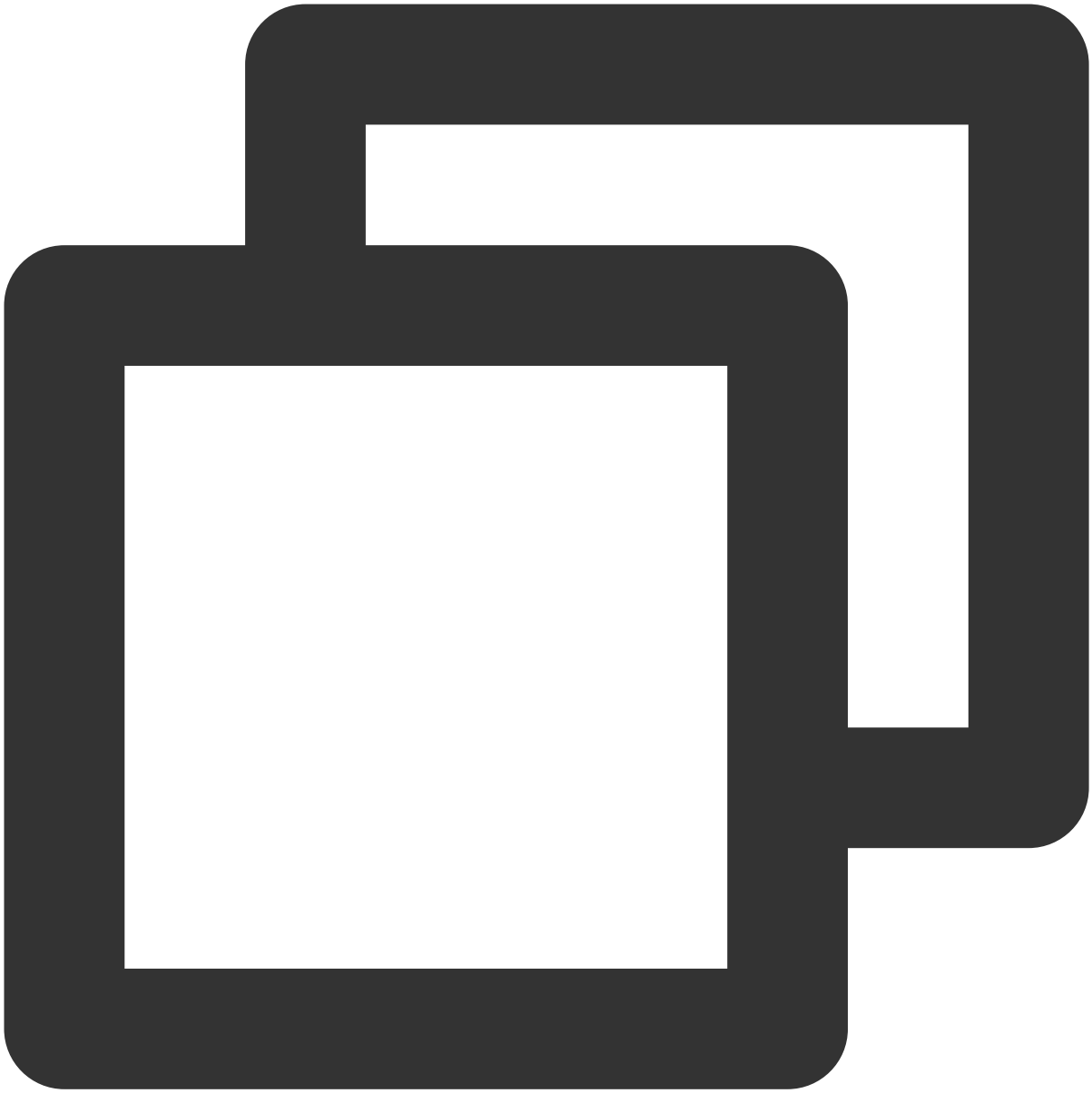
1. After creating two GPU instances, configure the instance environment and install HARP in the above steps.
2. Run the following commands in Docker to configure passwordless SSH login for the instances:



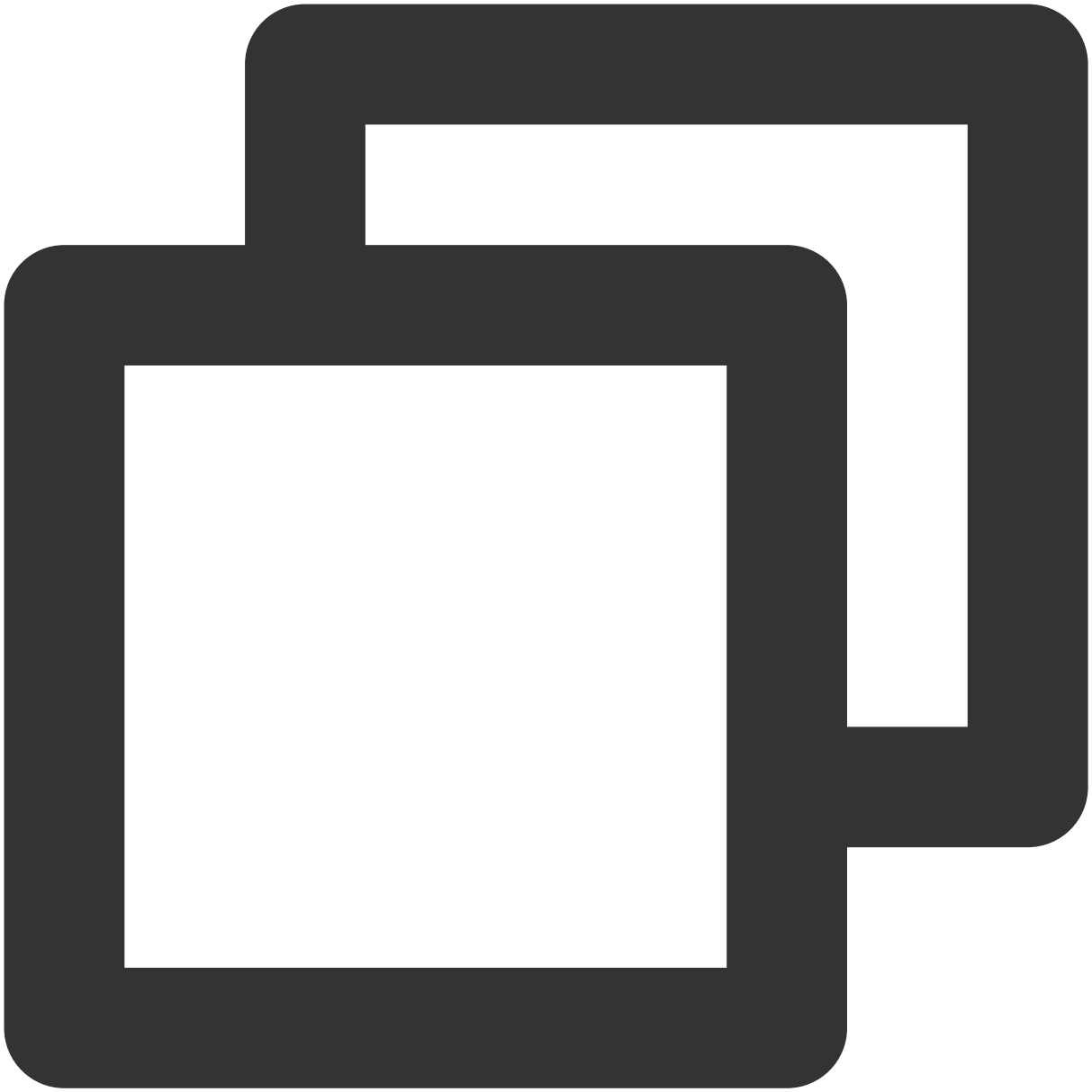
```
# Allow the root user to use the SSH service and start the service (default port: 2
sed -i 's/#PermitRootLogin prohibit-password/PermitRootLogin yes/' /etc/ssh/sshd_co
service ssh start && netstat -tulpn
```



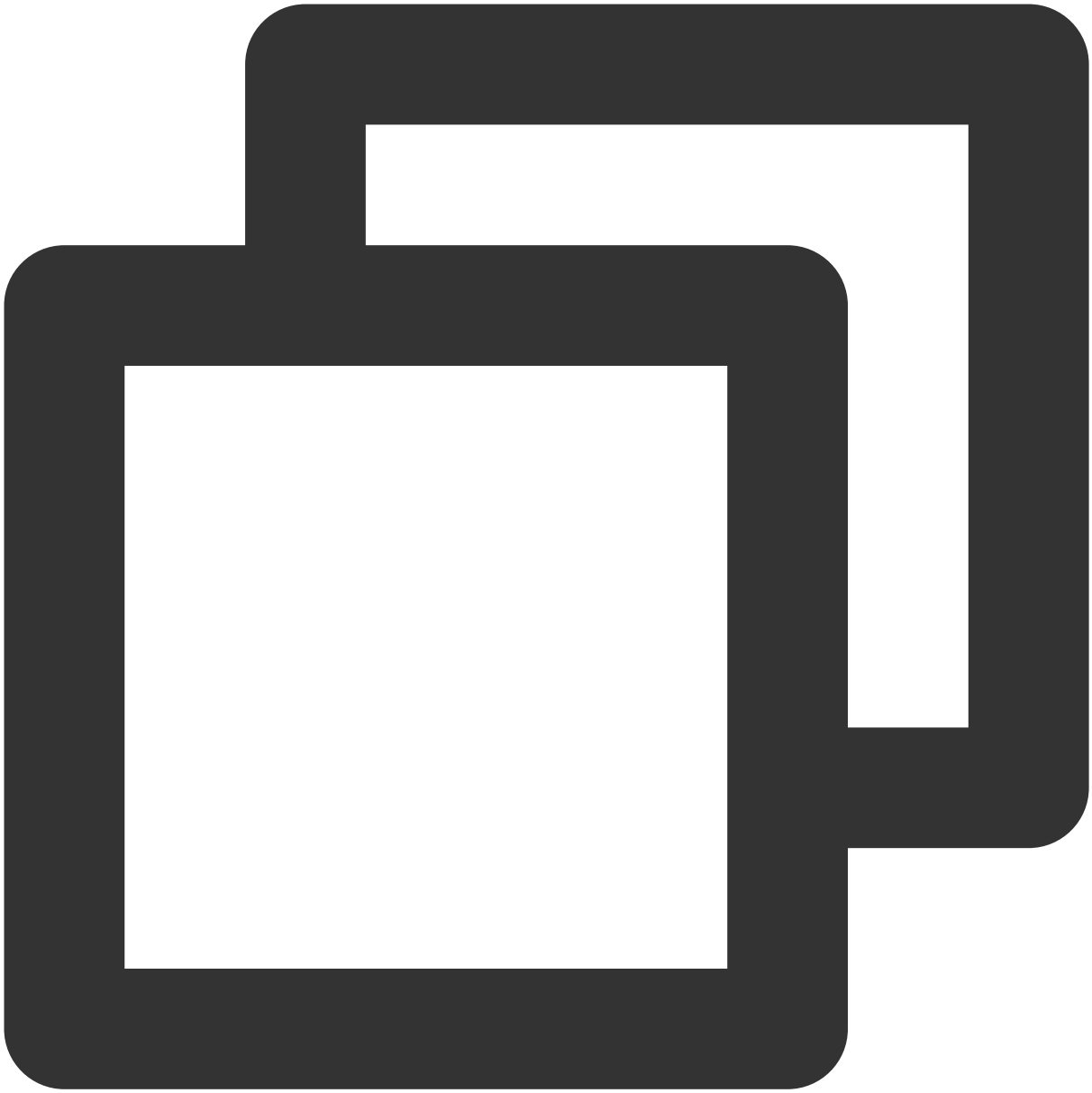
```
# Change the default SSH port in the container to 2222 to avoid conflicts with the
sed -i 's/#Port 22/Port 2222/' /etc/ssh/sshd_config
service ssh restart && netstat -tulpn
```



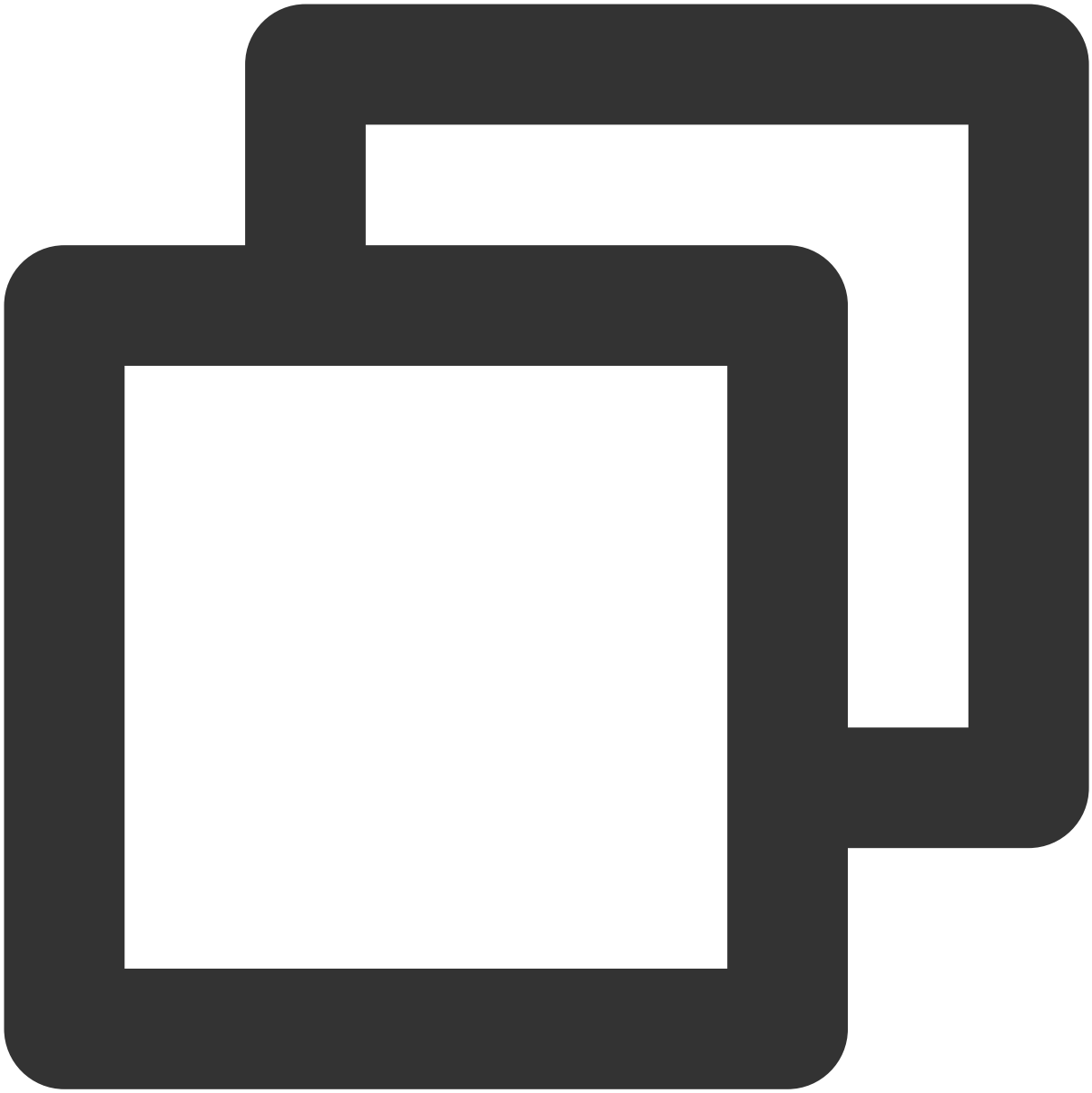
```
# Set `root passwd`
passwd root
```



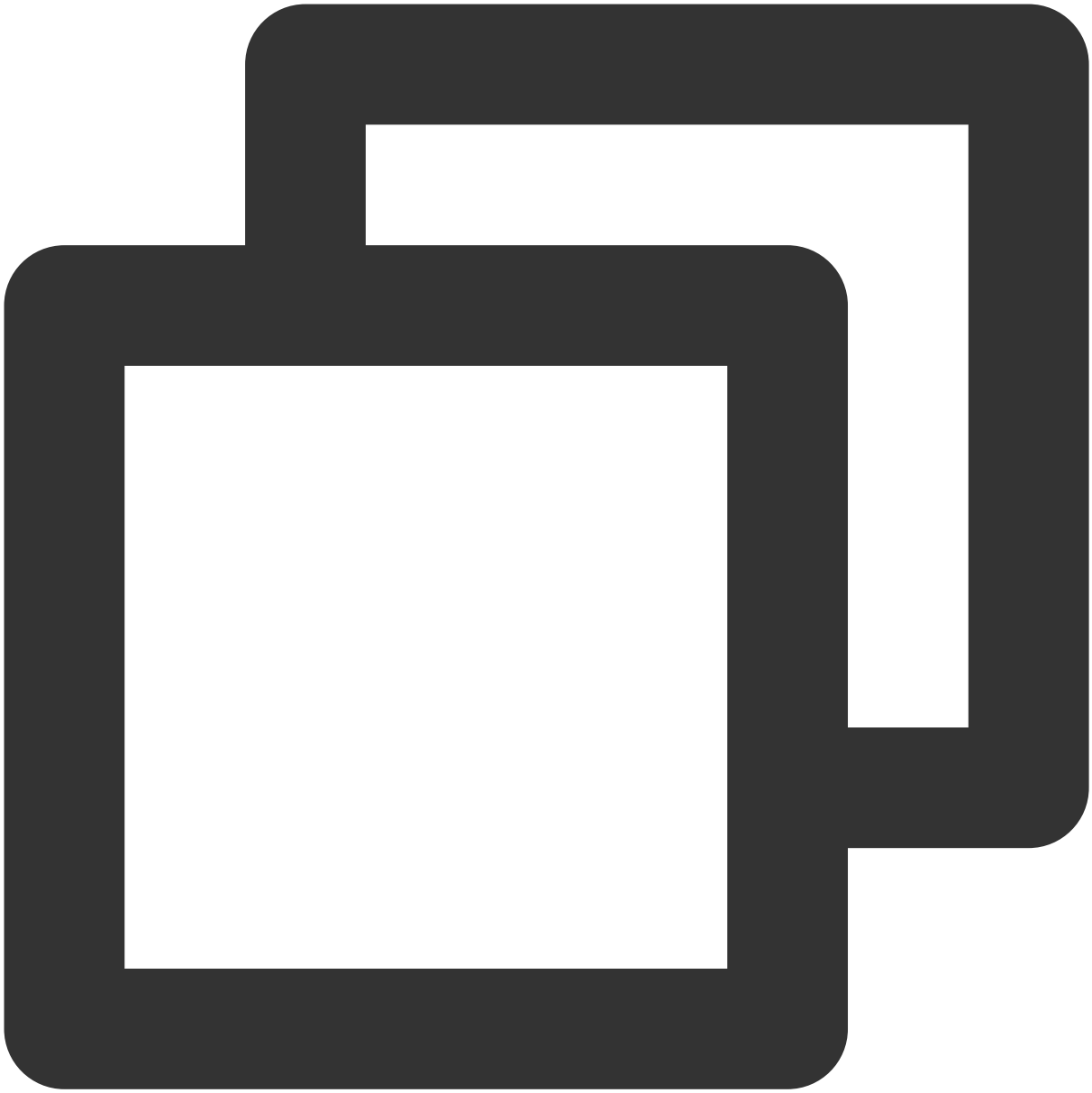
```
# Generate an SSH key
ssh-keygen
```



```
# Configure SSH to use port 2222 by default
# Create `~/.ssh/config`, add the following content, and save and exit the file:
# Note: The IP used here is the IP displayed in `ifconfig eth0` of the two instance
Host gpu1
 hostname 10.0.2.8
 port 2222
Host gpu2
 hostname 10.0.2.9
 port 2222
```

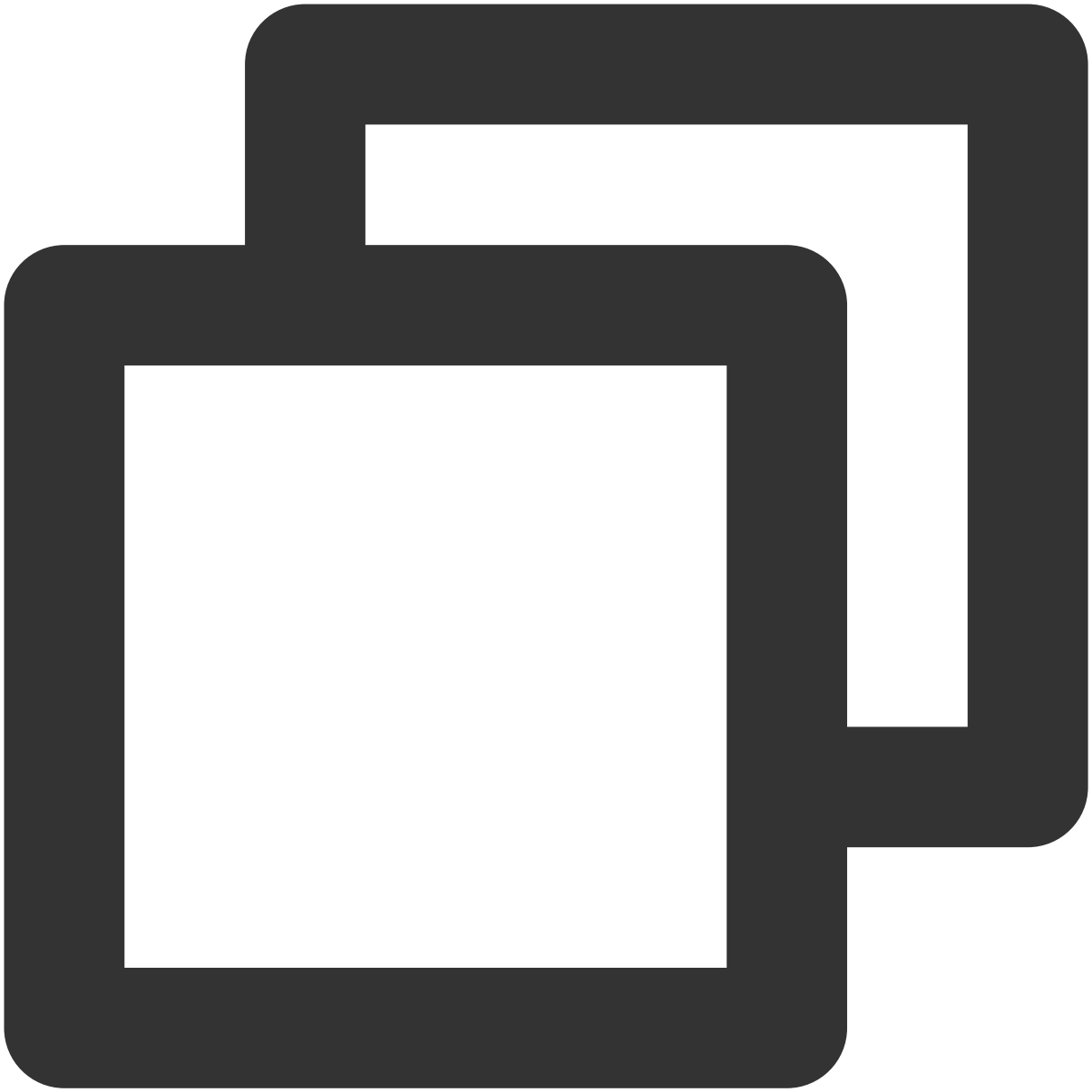


```
# Configure mutual passwordless login for the instances and local passwordless logi
ssh-copy-id gpu1
ssh-copy-id gpu2
```



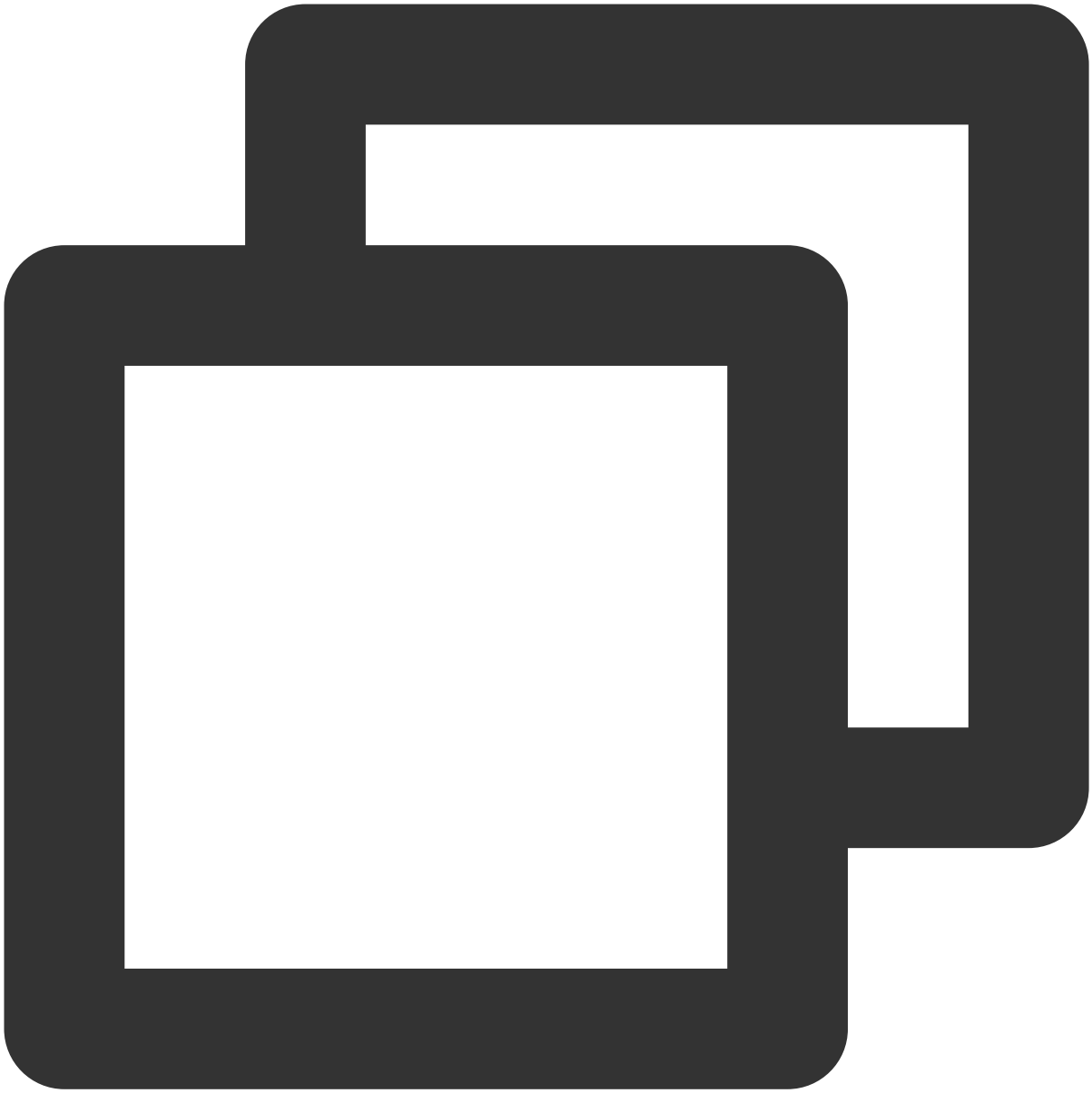
```
# Test whether passwordless login is configured successfully
ssh gpu1
ssh gpu2
```

3. Run the following command to download the benchmark script of Horovod:



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

4. Run the following command to start multi-server training benchmark based on ResNet-50:



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

Here, the command parameters are used for an 8-card model. To configure another model, modify the `-np` and `-H` parameters. Other parameters are as detailed below:

`NCCL_ALGO=RING` : Select the ring algorithm as the communication algorithm in NCCL.

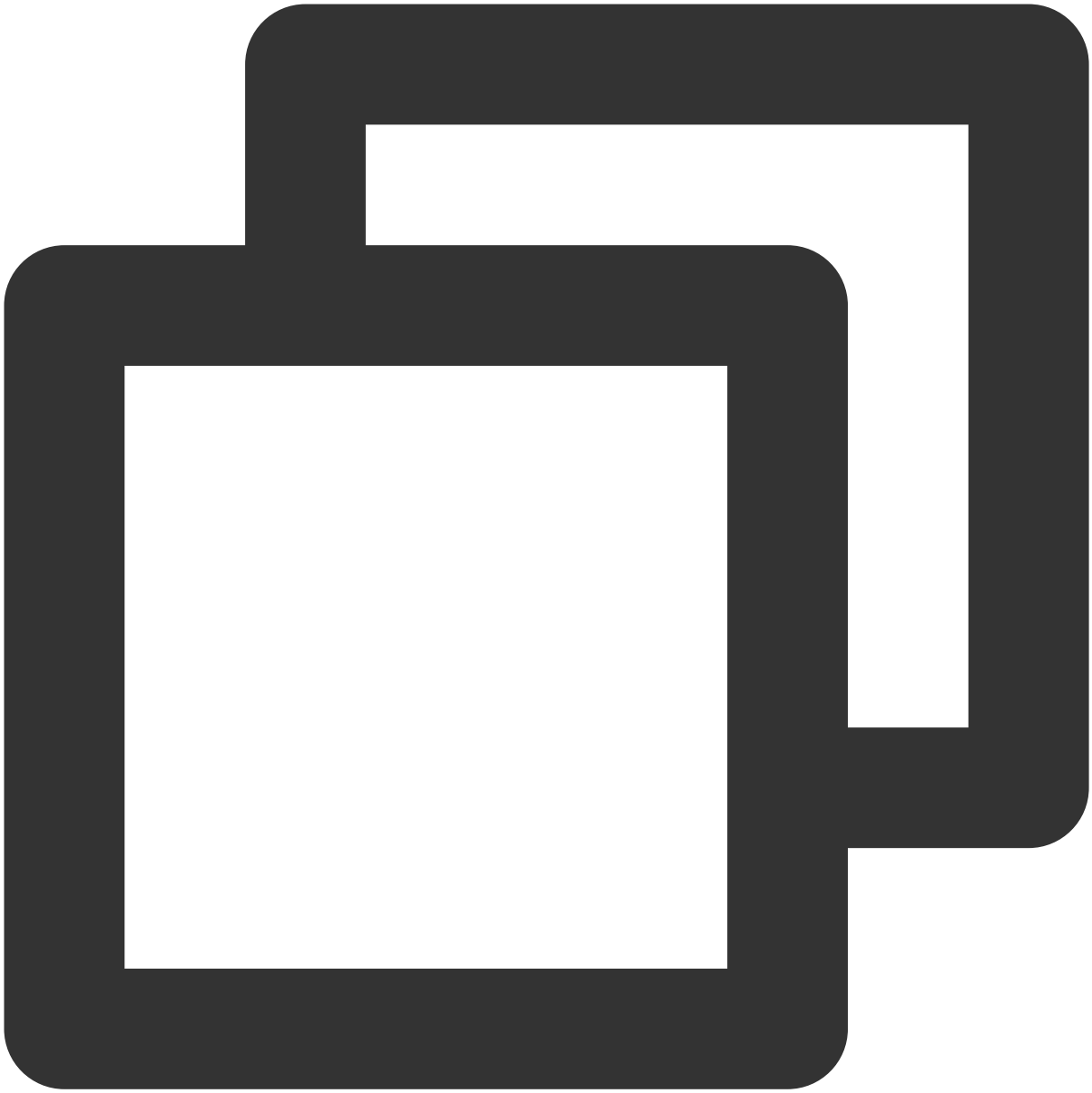
`NCCL_DEBUG=INFO` : Enable debugging output in NCCL. After it is enabled, HARP will output the following content:

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

`-mca btl_tcp_if_include eth0` : Select the eth0 device as network device for MPI multi-server communication. As some ENIs cannot communicate, you need to specify the ENI if there are multiple ones; otherwise, an error will occur if MPI chooses an ENI that cannot communicate.
After NCCL initialization, you can view the network output:

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

5. HARP is integrated to NCCL as a plugin and is enabled automatically without any configuration required. To disable HARP, run the following command in the container:



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