



CoCoPIE
MAKING UBIQUITOUS AI SWEET AS PIE

XGen Users Guide

v1.4.1

© 2026 CoCoPIE Inc. All Rights Reserved.

About This Manual

This manual introduces the CoCoPIE XGen solution by CoCoPIE and how to use it to optimize AI models for end devices.

Current Version: v1.4.1

Contact Information

- Email: info@cocopie.ai
- Phone: +18336851115
- Address: 2479 E Bayshore Rd, Palo Alto, CA 94303.

Copyright

© 2023 CoCoPIE Inc. All Rights Reserved.

CoCoPIE Inc., its logo and certain names, product and service names reference herein may be registered trademarks, trademarks, trade names or service marks of CoCoPIE Inc. in certain jurisdictions.

The material contained herein is proprietary, privileged and confidential and owned by CoCoPIE Inc. or its third-party licensors. The information herein is provided only to be person or entity to which it is addressed, for its own use and evaluation; therefore, no disclosure of the content of this manual will be made to any third parties without specific written permission from CoCoPIE Inc.. The content herein is subject to change without further notice. Limitation of Liability – CoCoPIE Inc. shall not be liable.

All other trademarks are the property of their respective owners. Other company and brand products and service names are trademarks or registered trademarks of their respective holders.

Limitation of Liability

CoCoPIE Inc. shall not be liable.

Chapter 1. Introduction

About

CoCoPIE XGen is a tool dedicated to optimization of AI models for end devices. The creation of XGen was prompted by a problem often encountered by AI application developers. When they try to deploy a trained AI model on devices, the developers often find that the model is too slow, too large, or too power-demanding for practical needs. The problem is most serious on mobile and embedded devices, such as smartphones, Raspberry Pi, and IoT devices. This version focuses the support on smartphones that run Android systems and iOS systems. For other kinds of devices (e.g., Raspberry Pi, RISC-V, etc.) and non-16-bit precisions, please contact CoCoPIE Inc.

CoCoPIE XGen solves the AI deployment problem by offering an easy-to-use tool that automatically optimizes AI models. With it, an input AI model can turn into directly deployable code as much as tens of times smaller and faster with a still satisfying accuracy. With XGen removing the key barrier and making more AI tasks possible to achieve on end devices, developers can capitalize on the huge market of mobile AI that had been beyond the reach.

CoCoPIE now provides XGen as an on-premise software tool, installable on users' machines, avoiding concerns on data/model security. This particular version is a single-node version, running on one single machine. For the distributed version that runs on computer clusters, please contact CoCoPIE Inc.

Benefits from XGen

- Shortened time-to-market of AI solutions

With the automated AI optimization by XGen, the process of making an AI model deployable in mobile apps can be shortened from several months to several days.

- Lower cost

The dramatically enhanced productivity can substantially reduce the cost in deploying an AI solution as well as its maintenance and upgrades. In addition, as its superior optimizations make some cloud-based AI tasks possible to run on end devices, XGen can help lower the cloud cost for businesses.

- Superior performance and compact models

CoCoPIE's world-leading co-optimization technology makes XGen generate AI models and code several times smaller, faster, and more power efficient than those by existing tools while maintaining the same level of accuracy.

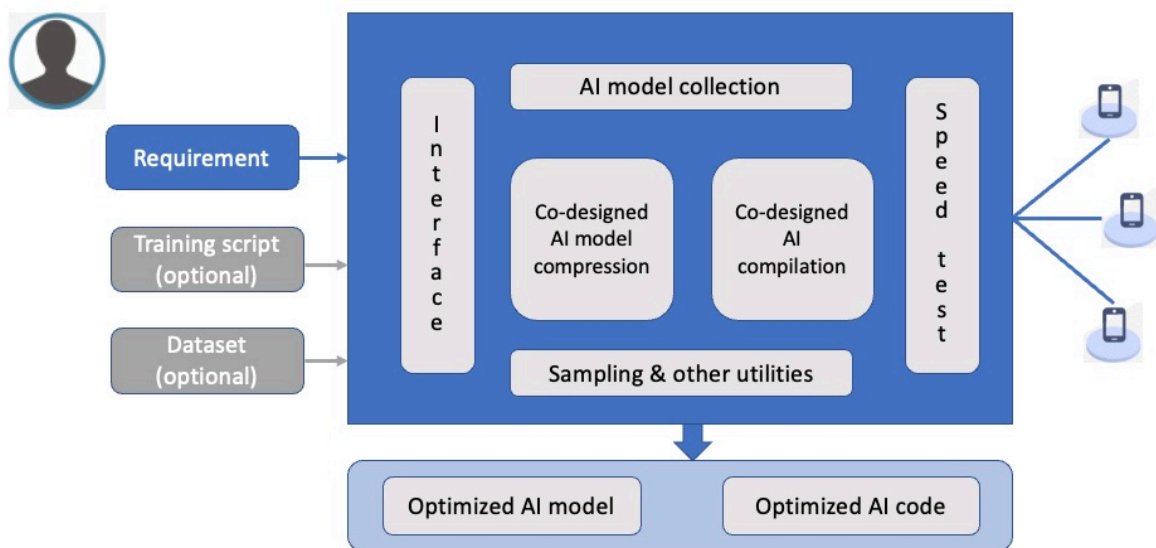
- Expanded business opportunities

By making many AI tasks achieve realtime performance on end devices, XGen can be used to exploit many new business opportunities blocked by the lack of device AI capabilities.

Where XGen Fits in the AI Workflow

Users can use XGen to easily obtain a DNN model and code that meet their AI needs. XGen supports the generation of efficient AI models on either public datasets or users' own datasets.

The high-level view of XGen is shown as below. This version of XGen runs on one single computer connected with one or more real devices. It receives users' requirements and other inputs, optimizes the AI model of interest, and outputs the optimized model and code. During the optimization, it includes on-device speed testing into the loop to guarantee the actual performance of the generated model. It has some predefined AI models but also works on users' own AI models. It reuses the models produced in previous runs to save time, but does not depend on them to function.



Specifically, there are two main usage scenarios as follows.

- Scenario I: The user needs a common AI capability that meets certain requirements (e.g., a certain speed on some common devices with a certain accuracy and size), and one of the

base AI models already included in XGen suits the general task, but is too large or too slow, and needs to be made work well on users' data.

Usage I: From the model set included in XGen, the users select the base model already included in XGen that fits their needed AI capability, and inputs their requirements; XGen optimizes the model and generates the model and code that meet the requirements.

- Scenario II: The user needs an AI capability but the base AI models already included in XGen do not fit that types of task. The users have their own model and the training script, but the model is too large or too slow.

Usage II: The users follow the guidelines in the [Usage](#) to make their training script compatible with XGen's requirement, give the training script and other inputs to XGen, and XGen automatically generates a model that satisfies users requirements.

Besides these main usage scenarios, XGen can also be used for some smaller tasks, such as training an AI model, assessing the quality and speed of an AI model, testing the compatibility of an AI training model with XGen, managing AI models, and so on.

General Features

- Support a broad range of AI models (e.g., CNN, RNN, Transformers) for various AI tasks (CV, NLP, etc.)
- Free of security and privacy concerns as an on-premise tool
- Superior speed & accuracy
- Single-click solution as well as fully-customizable usage
- Flexible support of users' DNN models and data
- Co-optimizing DNN models and code through the world-leading technology
- Proprietary cutting-edge DNN pruning methods
- Novel effective AI compilation and code generation
- Reliable quality and speed assessment with actual devices in the loop
- Flexible power control APIs to ensure power-efficient AI
- Generated code works on all mainstream Android and iOS devices; support to other devices can be added separately

Features Specific to this Version

- This XGen is a single-node version; it runs on a single GNU/Linux machine. It can use one or more GPUs equipped in the machine.
- The generated AI model is 16-bit in precision, working on Android 9 and iOS 11 and later versions.
- When used for optimizing customer's AI models, a training script in PyTorch must be provided.
- It supports multiple users, allowing the launch of multiple XGen instance containers on a machine

For the support of special processors or other kinds of devices (e.g., iOS, Raspberry Pi, RISC-V, etc.), or non-16-bit precisions (e.g., via quantization), please contact CoCoPIE Inc.

Chapter 2. Requirements, Installation, Upgrade, Uninstallation

Overview

In previous versions, physical devices such as phones needed to be connected to the host machine where XGen was running. The workflow involved training with XGen and then executing tasks on devices connected to the local machine. After task completion, the process returned to XGen for further training, and this cycle repeated.

Subsequent practical experience revealed several drawbacks. Firstly, a limited number of test physical devices could be connected to a single machine, and these physical devices couldn't be used for other tests simultaneously. Secondly, this approach was inconvenient, particularly for users whose training machines were located in server rooms or data centers, making it challenging or impossible to connect physical devices directly to these machines.

To address these issues, the current version introduces the concept of a "Device Lab," which abstracts and decouples the process of conducting test tasks on physical devices within the XGen workflow. Essentially, the process involves XGen sending a task to a physical device, the device executing the task, and then sending the results back to XGen, enabling the continuous workflow.

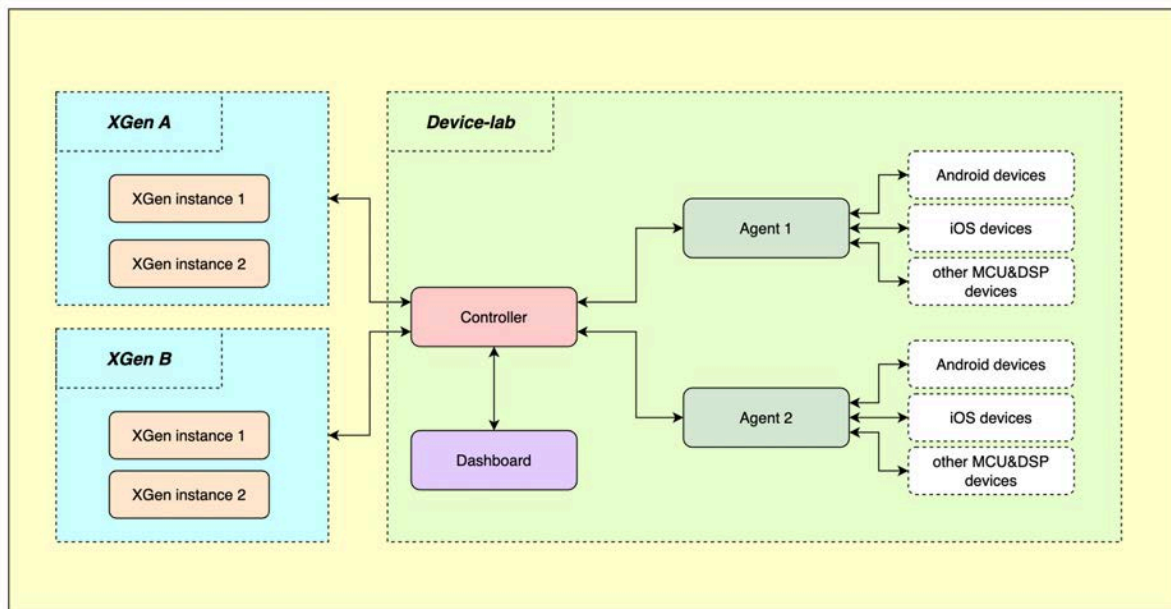
The Device Lab consists of two parts: a controller and one or more agents. The controller acts as a dispatcher or router, while the agent serves as a docking station for physical devices, which are connected to the agent.

Agents are connected to the controller, and XGen is also connected to the controller. Communication between the three occurs through RPC.

The benefits of this approach are substantial decoupling between XGen and the target devices running tasks. Physical devices visible to XGen could come from multiple agents located in different places. For instance, one agent in Beijing might have 10 physical devices such as phones, while another agent in Boston has 20 physical devices. XGen doesn't need to be concerned about how the devices are connected and where they are located. The user of XGen just needs to choose the target devices she wants XGen to run tasks on; XGen will then send the tasks to the controller, which then distributes them to the corresponding agents. The agents then forward the tasks to the physical devices. After task completion, the physical devices send the results back to the agents,

which in turn send the results to the controller. The controller forwards the results to XGen, and the process continues.

It's clear that this is a highly loosely-coupled distributed system. Apart from the controller (only one controller is needed), everything else can be horizontally scaled. Multiple agents, XGen instances, and physical devices can be added as needed. They can all be independently deployed as well. For example, agents could be deployed in phone data centers, XGen could be in a computing center, and the controller could be in the cloud. This flexibility allows deployment and scaling according to specific requirements. Of course, as you might have guessed, everything could also be deployed on a single machine, similar to the previous version of XGen.



In brief, Device Lab offers benchmark services on physical devices for XGen. XGen submits benchmark jobs to Device Lab Controller then Controller will dispatch the job to corresponding Device Lab Agent, which is responsible for running the job on the device. The benchmark results will be sent back to Controller; after gathering all benchmark results from all agents, the Controller sends the results to XGen. The included Device Lab Dashboard offers a web-based tool for monitoring the benchmark jobs, which is basically another user-friendly interface for Device Lab Controller.

Thus, the installation process of XGen is a little bit different from the previous version. For more information about Device Lab, please refer to [Device Lab Introduction](#). For properly making things work, please follow the instructions below to install XGen and Device Lab.

Device Lab should be installed and configured before installing XGen. The 3 parts should be installed and configured in order: Controller (with Dashboard), Device Agent, and XGen. That is, XGen should be installed after everything is settled for Controller and Agent.

If there is only one host machine to install XGen and Device Lab, `All-in-One` installation script is recommended. This scenario is described below as Scenario 3.

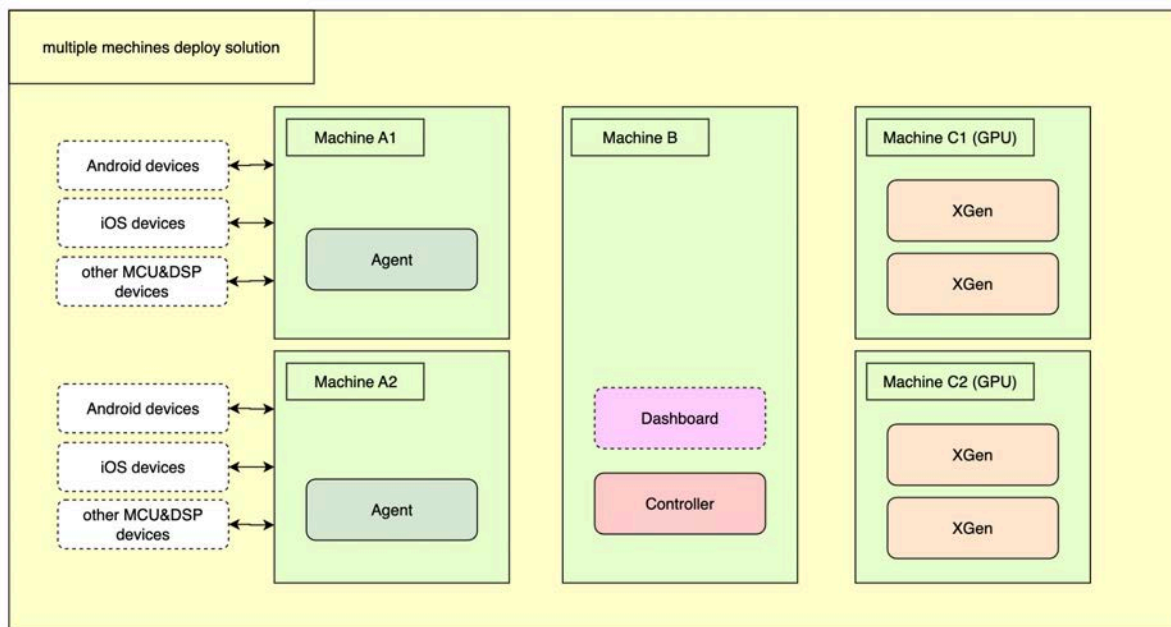
Installation scenarios

If deployment on multiple machines is required, it is preferable that the machines hosting xgen, controller, and agent are all on the same network. If they are not, then please install the controller on a machine with a fixed public IP address. This way, other machines with xgen and agent installed can establish communication by connecting to the machine with the installed controller. Of course, the ports `80`, `19000`, `28000`, and `55672` need to be opened, as communication between xgen, controller, and agent relies on these ports.

If the installation is being done on a single machine, the above steps are not necessary.

Scenario 1: Multiple Machines

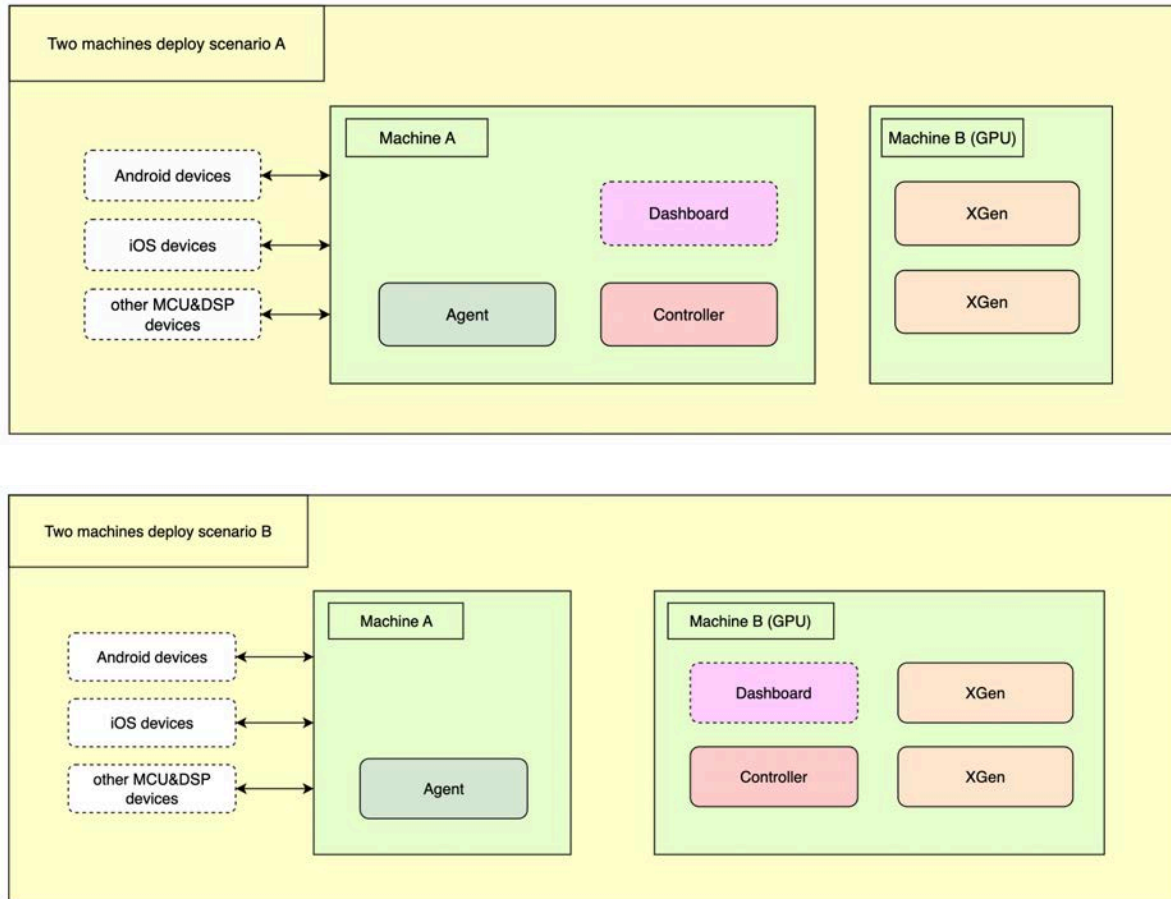
If there are multiple machines to deploy, please note that there should be only **ONE** Device Lab Controller as the center node.



Scenario 2: Two Machines

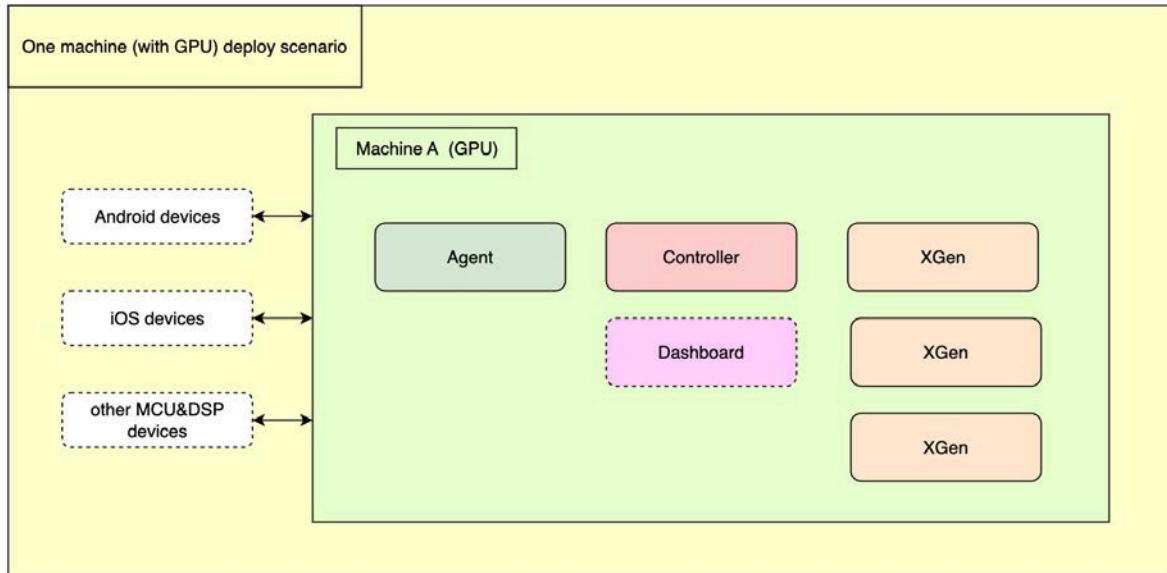
If there are two machines to deploy, XGen should be deployed on the machine with GPU, and Device Lab Controller should be deployed on the machine that has an external IP visible to other

machines (in the scenarios below, the `scenario A` is recommended).



Scenario 3: One Machine

If there is only **ONE** machine to deploy, the user can install XGen, Device Lab Controller and Device Lab Agent on the same machine. This machine must be equipped with GPU.



Services and Container Introduction

Of course, whether the XGen, Controller, and Agent services are installed separately on multiple machines or on a single machine, they all ultimately manifest as 4 Docker containers (other containers provide basic services such as Redis, etc.).

The Docker container corresponding to XGen is named `xgen_<version>_<username>`, which is the core component for user interaction.

The Controller service is supported by two Docker containers: `xgen_controller` and `xgen_dashboard`. As their names suggest, the `xgen_dashboard` mainly provides web functionality, while the main logic is centralized within `xgen_controller`.

The Docker container for the Agent service is named `xgen_agent`.

For these core containers, namely `xgen_<version>_<username>`, `xgen_controller`, and `xgen_agent`, convenient commands have been developed for starting or entering the containers.

To enter a container, use the command `run_<name>`. For instance, to enter the controller container, execute the command `run_controller`. If the `xgen_controller` container does not exist on the machine at that point, the command will first create and start the container before entering it. If the `xgen_controller` container already exists on the machine, running `run_controller` will simply enter the container, equivalent to the Docker command:

```
docker exec -it xgen_controller bash
```

Similarly, for `run_agent`, the logic is the same as `run_controller`.

However, `run_xgen` is slightly different due to its support for multiple concurrent users, each running an independent container. Therefore, the `--name` option needs to be added, for example:

```
run_xgen --name bar
```

Exiting the containers is the same for all of them – running the `exit` command within the container will leave it.

```
exit
```

The difference lies in the behavior after exiting. For Controller and Agent containers, exiting does not mean stopping the container; the container will continue to run in the background. However, for the XGen container, exiting means stopping the container.

In the aforementioned containers, specific commands have been developed to check the status of running services. `xgectl status` can be used to view the current status of services within the container. If issues arise, `xgectl restart` can be employed to restart the services.

Both XGen and Agent containers feature the `xgen_devices` command, but the parameters slightly differ. Detailed differences can be viewed using the `--help` parameter.

Installation

Based on the information provided above, you should now have an understanding of your deployment scenario.

If you need to install XGen and Device Lab on a single machine, please refer to the [deployment documentation for a single machine](#).

If XGen and Device Lab's Controller and Agent are being installed on separate machines, please consult the [deployment documentation for multiple machines](#).

How to Install XGen and Device Lab on a Single Machine

If the user would like to install everything on a local machine, this part explains the process. It can be done simply through the `all-in-one` script of XGen.

Requirements

- GNU/Linux x86_64 with kernel version > 3.10 (Ubuntu or CentOS or WSL2)
- Network (Recommend): The controller needs to be accessible from the agent and XGen. Port **80** should be open and reserved for the controller. There are many ports are used internally by XGen components.

If your firewall is enabled, please ensure following ports are allowed in the firewall rules: 22 80 28000 38000 19000 12379 6379 55672 45672 4369 5671 5672 15671 25672 27017 19001-19200 Please ensure following domains are also accessible: *.cocopie.ai *.cocopie.io *.bitbucket.org *.aliyuncs.com *.amazoncs.com

- Docker
- Memory size: 16GB or more
- Devices: The agent needs to be able to connect to the devices via USB. The default suggested devices are 64-bit devices (phones/tablets) that run **Android 9** and **iOS 11** or above. Support of other devices can be added separately.
- NVIDIA GPU with Architecture >= Kepler (or compute capability 3.0)
- Disk size: 500GB or more. XGen by itself takes less than 20GB. But more space would be needed to hold AI models and datasets. The amount depends on the models and datasets the user uses. The AI models included in XGen's common AI are 500MB or less. The sizes of the default datasets included in XGen's common AI are as follows (downloadable on demand):
 - ImageNet: 144.02 GB
 - UCF101: 75.74 GB
 - CoCo2017: 25.01 GB

The requirements on the devices:

- A feature of XGen is that it can ensure accurate assessment of the quality and speed of the generated AI model and code by including real devices in the optimization loop. To use this feature, the machine that runs XGen needs to connect to Device Lab. XGen uses the devices for speed measurement.
- When no devices are connected, XGen can still be used to optimize AI models as explained in the [Model Optimization Alone Without Running on Devices](#); the code generation functionality of XGen and on-device benchmarking are not usable in that mode though.

Step 1: Install Nvidia-Driver, Docker and Nvidia Container Toolkit

If the XGen machine is running the Ubuntu Linux operating system:

1. Please make sure that nvidia-driver (version ≥ 510) has been installed. Instructions: [Nvidia-Driver Installation Guide](#) (CUDA installation is not mandatory on the host, because the Nvidia Container Toolkit in the later step includes CUDA support.) There is another solution as well if there is no specific requirement for the driver:

```
# install ubuntu-devices-common to check devices version
sudo apt install ubuntu-devices-common -y
# check devices version and determine CUDA driver version needed
sudo ubuntu-drivers devices # this command will list all available drivers and
related apt package name
# output example:
vendor    : NVIDIA Corporation
driver    : nvidia-driver-470 - distro non-free
driver    : nvidia-driver-535-open - distro non-free
driver    : nvidia-driver-525 - distro non-free
driver    : nvidia-driver-535 - distro non-free recommended
driver    : nvidia-driver-525-server - distro non-free
driver    : nvidia-driver-535-server - distro non-free
driver    : nvidia-driver-535-server-open - distro non-free
driver    : nvidia-driver-470-server - distro non-free
driver    : nvidia-driver-525-open - distro non-free
driver    : xserver-xorg-video-nouveau - distro free builtin

# install the recommended driver or the driver you need
sudo apt install nvidia-driver-535 -y
```

2. Please make sure that Docker Engine has been installed. Instructions: [Install Docker Engine](#)

```
curl -fsSL https://get.docker.com -o get-docker.sh
sudo bash ./get-docker.sh
```

3. Please make sure that Nvidia Container Toolkit is installed. Instructions: [Nvidia Container Toolkit Installation Guide](#)

After completing the above steps, the user can verify whether the deployment requirements of XGen Docker Image are met by running the following command in Linux:

```
docker run --rm --gpus all nvidia/cuda:11.1.1-base-ubuntu20.04 nvidia-smi
```

When success, the NVIDIA GPU information will be output to the screen.

Hint: to run docker command without sudo authorization, the user needs to be granted the docker permission. The admin can do that through the following steps:

```
sudo groupadd docker # if docker group does not exist
sudo usermod -aG docker $USER
```

Important: re-login to make the change take effect.

```
docker ps
```

If the above command runs successfully, the user would have the docker permission.

Install the following commands if they are not installed yet:

```
sudo apt install unzip curl git bc
```

Before installation, please ensure that `usbmuxd` service is **not** running on the host machine. `usbmuxd` is used by XGen in communications with iOS devices, similar to how `adb-server` is used for Android devices. In the previous version of XGen, `usbmuxd` was installed as a system service on the host machine. In the current version, we have included it within the Docker container. Therefore, if `usbmuxd` is present on your machine, it will cause the agent container to fail to start. Therefore, when running the agent container, please stop the `usbmuxd` service and remove it to avoid conflicts.

How to stop `usbmuxd` service on UBUNTU20.04:

```
sudo systemctl stop usbmuxd
```

And how to delete `usbmuxd` service on UBUNTU20.04:

```
sudo apt remove usbmuxd -y
```

Step 2: Installation

The installation script has many installation options. The user can run the following command to see the options:

```
bash xgen_aio_deploy.sh -h

Usage: bash xgen_aio_deploy.sh [OPTIONS]
A script for fetching and setting up the docker images of CoCoPIE XGen.
Options:
  -h, --help                Show this help message.
  --region string           US or HK (default US; HK is faster for mainland
                           China).
  --memory bytes            Memory limit for XGen (default 70% of System
                           Memory).
  --xgen-path string        Home directory of XGen (default $HOME/XGen).
  --xgen-data string        Path in the host to store dataset (default $xgen-
                           path/Data).
  --xgen-output string      Output path of XGen (default $xgen-path/Output).
  --controller-path string  Home directory of Controller (default
                           $HOME/XGenController).
  --agent-path string       Home directory of Agent (default
                           $HOME/XGenAgent).

Examples:
  bash xgen_aio_deploy.sh
  bash xgen_aio_deploy.sh --region HK
  bash xgen_aio_deploy.sh --xgen-path ~/XGen
  bash xgen_aio_deploy.sh --xgen-path ~/XGen --xgen-data ~/MLDataSets --
  controller-path ~/XGenController --agent-path ~/XGenAgent
```

For users in mainland China, the following command usually runs faster; for other users, use '--region US'.

```
bash xgen_aio_deploy.sh --region HK
```

After installation, XGen and Device Lab are installed into the configured path. And commands `run_xgen`, `run_controller`, `run_agent`, `upgrade_xgen`, `upgrade_controller`, `upgrade_agent` are written into `$HOME/.bashrc`.

Update the shell environment:

```
source ~/.bashrc
```

The user can now type the following command to enter XGen shell:

```
run_xgen --name <username>
```

`run_xgen` : This command will start all containers including Device Lab Controller with Dashboard, Device Lab Agent as well as XGen. Then will enter into the XGen instance. Each XGen container instance needs to be assigned a unique name with param `--name` to distinguish it from other instances. If the XGen container named with the `--name` parameter is already running, enter that container. Otherwise, start a new container with the specified name.

`run_agent`, `run_controller` : These two commands will start Device Lab Controller with Dashboard and Device Lab Agent and enter into the corresponding container. In `All-in-One` installation, these two commands are usually not needed to run; run `run_xgen` instead to start everything. Similar to the `run_xgen` command, if the `xgen_controller` container or `xgen_agent` container already exists, running the `run_agent` or `run_controller` command will enter the corresponding container.

`upgrade_xgen`, `upgrade_controller`, `upgrade_agent` : Executing the `upgrade_xgen` command will upgrade all containers, including the Device Lab Controller with Dashboard, Device Lab Agent, as well as XGen. However, you can also choose to upgrade the Device Lab components individually using the `upgrade_controller` and `upgrade_agent` commands.

`uninstall_xgen`, `uninstall_controller`, `uninstall_agent` : These commands are used to remove the corresponding container environments and images. The difference lies in the fact that `uninstall_xgen` encapsulates the logic of `uninstall_controller` and `uninstall_agent`. This is because, when installed on the same machine, the uninstallation process can be made more convenient and efficient.

Before connecting any devices, please read the [How to connect Devices](#) part below for more information. Connect your devices via USB to the host machine, and run the following command in **XGen container** to check the status of the devices:

```
xgen_devices
```

Platform Connection	UUID Status	Brand	Model	Task Status
-----	-----	-----	-----	-----
Android available	3131612233002SC	vivo	V2072A	idle
Android available	6c74106e	Xiaomi	MI 9	idle
iOS available	00008101-0009245134F9001E	Apple	iPhone 12 16.3.1	idle

Step 3: Configuration

All components in `All-in-One` script are already well **configured**. There is no more configuration required. To access the Device Lab Dashboard, simply visit http://deployed_machine_ip from your machine.

tips: If there are firewalls or other network restrictions, you can use an SSH tunnel to map port 80 on the deployed XGen machine to a port on your local machine, such as 8080. This way, you can access the remote machine's dashboard by visiting <http://127.0.0.1:8080>.

SSH tunnel port mapping example: `ssh -L 8080:127.0.0.1:80 <username>@<host_ip>`

At the first time, you'll be redirected to the login page. **The default account and password are both `admin`**. Then you'll be redirected to change password page to change the default password for security.

For more usage about Device Lab, please refer to [Device Lab Usage](#).

How to Start XGen

To start XGen, the first step is to start its docker container by running the following command, which resides in the `bin` folder of XGen.

The underlying operational logic is as follows: start the controller and agent along with their related dependent containers. If these containers are not already started, they will be initiated. Once they are up and running, an XGen container will be launched and accessed.

Of course, you can also individually start the controller container and agent container using the `run_controller` and `run_agent` commands before launching the XGen container. The `run_xgen` command, however, simplifies this entire process for you:

```
run_xgen --name <username>
```

The command creates and starts an XGen container named `xgen_<version>_<username>`. Meanwhile, it creates a folder `Projects` under the XGen home directory. It is for holding all AI models and optimization results; users' own to-be-optimized models should be put there as well. That folder is mapped to the `/root/Projects` folder inside the XGen container; the files in that folder are hence accessible both inside and outside the XGen container.

After `run_xgen --name <username>`, XGen's shell will be started.

XGen is proprietary software. During the initial login, the user needs to execute `xgenc1 status` and adhere to the instructions for acquiring a license.

The license file must be named `license` and situated within the `/etc/xgen/license` directory. Users can verify the license status by executing `xgenc1 status`.

After configuring the license, type the following to start XGen:

```
XGen
```

Because the XGen container is a docker container, users can use the standard docker commands to manage the container, such as stopping it, checking its status, and so on. **But users should avoid using docker commands to directly create XGen containers or rename the created XGen container. Always use `run_xgen --name <username>` to create XGen containers.** Otherwise, there could be unexpected data overwritten and file corruptions. For more detailed info on XGen command usage, please see [XGen Usage](#).

Sometimes, during an XGen session, users might experience a disconnection from the machine running XGen. Here are some helpful [hints](#).

Exiting the XGen container is the same as all containers – running the `exit` command within the container will leave it.

```
exit
```

The difference lies in the behavior after exiting. For Controller and Agent containers, exiting does not mean stopping the container; the container will continue to run in the background as services. However, for the XGen container, exiting means stopping the container.

`xgen_controller`, `xgen_agent`, and the underlying foundational data storage containers operate as services, thus running persistently in the background even if a user executes the `exit` command within the container. If there is a need to halt these services, the following Docker commands can be employed:

```
docker stop xgen_controller xgen_dashboard xgen_agent
```

As for XGen, it supports concurrent usage by multiple users. In most cases, an XGen container functions as a client, with each user employing an individual container to accomplish their tasks. Upon completion, users exit the container, causing it to stop. This approach optimizes computational resources for other users' utilization.

How to Upgrade XGen

For users who already have version v1.1.0 installed, direct upgrading to the latest v1.3.0 version using the `upgrade_xgen` command **is not supported**. A fresh installation is required for v1.3.0 since there have been architectural and module organization optimizations.

For new users installing version v1.3.0, for future upgrades, simply use the `upgrade_xgen` command to directly upgrade.

For more details about v1.3.0, please see [What's New](#).

After re-installation, the user can start XGen as before through `run_xgen --name <username>`.

In future version upgrades, the user can run the `upgrade_xgen` command. In an all-in-one installation of the XGen environment, `upgrade_xgen` will upgrade all container images, including the controller, dashboard, and agent. The user can upgrade the controller or the agent individually by simply running `upgrade_controller` or `upgrade_agent`.

How to Uninstall XGen

The user may uninstall XGen by running the `uninstall_xgen` command in the `bin` folder of XGen:

```
uninstall_xgen
```

The command will present users with options to choose the components (such as previously generated models) to retain. Similar to the `upgrade_xgen` command, the `uninstall_xgen` command will uninstall all container images, including the controller, dashboard, and agent.

How to Connect Devices

XGen measures the speed of the optimized models directly on connected devices (e.g., Android phones). The devices should be connected to the machine that runs an XGen Agent. This part explains how to connect devices.

Before explaining the detail, it is worth noting that regardless of the method of connection, the following apply:

- For Android devices, the user should turn on the "Developer mode", "USB debugging". Please see [instructions](#).

- On the devices, at the first connection to the machine, an authorization window pops up; please authorize the machine to access this device and trust the machine always.
- The USB cable used for connection must be a data cable rather than a charging cable.
- Running command `adb devices` inside the agent shell would show the list of Android devices that are accessible by agent.
- Running command `sib devices` inside the agent shell would show the list of iOS devices that are accessible by agent.

Connect Android devices via USB

Connect the devices with the agent machine via a USB *data* cable. Some USB cables are only for charging, which won't work. (When the phone is connected with the computer via a data cable, usually the computer prompts to select connection mode, select `Transfer Files`.) The phone needs to be configured as stated above.

ADB has been installed in **Agent container**. So after completing the devices connection, then run the following command to check if the devices are connected to the agent container machine:

```
# enter
run_agent
# check if your android device is connected
xgen_devices list -nm
```

Connect iOS devices

1. Install `CoCoBench` app in `TestFlight`. For installation, the user needs to have `TestFlight` installed on the iOS device to be used with XGen, and follow the invitation link in the XGen welcome email to install `CoCoBench` via `TestFlight`, and then launch `CoCoBench` on the iOS device. To guarantee the app runs correctly, here are some tips:

- Make sure that `CoCoBench` of the latest version is installed.
- Make sure that `CoCoBench` is running foreground.
- Make sure that the screen of the iOS device is always on, which means that the screen is not locked and the device does not enter the sleep mode.
- Make sure that there is no password set for the iOS device.
- Make sure that the auto update of `CoCoBench` in `TestFlight` is turned on; otherwise, manually check it to keep the app up to date.

- By default, `CoCoBench` expires **90** days after the **publish date** due to the limitation of `TestFlight`. The user must update `CoCoBench` to the latest version in `TestFlight` before the expiration date; enabling auto update would simplify this.
2. Connect the iOS device to the host machine via a USB data cable (not a charging cable). Then the host machine might pop up a window to ask the user to trust the host machine. Click `Trust`.
 3. Enter **Agent container**, then run the following command to check if the iOS device is connected to the agent container machine:

```
# enter
run_agent
# check if your iOS device is connected
xgen_devices list -nm
```

4. If the iOS device falls offline due to some reason (e.g., unplugged and replug), the user needs to restart the `usbmuxd` services as shown below:

```
xgenctl restart -s usbmuxd
Service usbmuxd restarted.
```

Alternatively, a simpler approach is to restart all services.

```
xgenctl restart
```

After that, reconnect the iOS device to the host machine.

Hints

Avoid issues from disconnection

Sometimes in the middle of an XGen session, the user may get disconnected from the machine that runs XGen. To ensure that the session is not interrupted by the disconnection and the user can get back to the session after reconnection, the user can use `tmux`:

- Install `tmux` by, for instance, the following command (on Ubuntu): `sudo apt-get install tmux`
- In the host terminal, run the following to start a new session: `tmux new -s Some-Session-Name`
- In the `tmux` window, start XGen as before: `run_xgen --name <username>`

- The session will continue running even if a disconnection happens. If the user is disconnected and then reconnected to the machine, run the following command will bring back the session:
`tmux attach -t Some-Session-Name .`

How to Install Device Lab and XGen on Multiple Machines

A typical scenario is that XGen runs on a remote server (e.g., a machine in a data center) and the testing devices are local. XGen and device lab can be installed on different machines. This part explains the installation in such a scenario.

Network & firewall configuration before installation

There are many ports are used internally by XGen components. Use command below to check if your firewall is enabled:

```
sudo ufw status
```

If your firewall is enabled, please ensure following ports are allowed in the firewall rules: 22 80 28000 38000 19000 12379 6379 55672 45672 4369 5671 5672 15671 25672 27017 19001-19200 . And please ensure these domains are also accessible *.cocopie.ai *.cocopie.io *.bitbucket.org *.aliyuncs.com *.amazoncs.com .

How to Install Device Lab Controller

Requirements

- GNU/Linux x86_64 with kernel version > 3.10 (Ubuntu or CentOS or WSL2)
- Docker
- Memory size: 16GB or more
- Disk size: 100GB or more
- Network: **Important**. The controller needs to be accessible from the agent and XGen. Ports **80, 19000, 28000, 55672** should be open and reserved for the controller. If you want to use Device Lab across the networks, please configure your firewall or security group to allow traffic on these ports.

Step 1: Install Docker

Please follow the official Docker instruction to make sure that Docker Engine has been installed.

Instructions: [Install Docker Engine](#)

Make sure that the current user is in the docker group, if not, please run the following command:

```
sudo groupadd docker # if docker group does not exist
sudo usermod -aG docker $USER
```

Important: re-login to make the change take effect.

```
docker ps
```

if the above command can run successfully, the user has been granted the docker permission.

Install the following commands if they are not installed yet:

```
sudo apt install unzip curl git bc
```

Step 2: Install Device Lab Controller

Install Device Lab Controller with Dashboard with the install script.

```
bash xgen_controller_deploy.sh

Usage: bash xgen_controller_deploy.sh [OPTIONS]
A script for fetching and setting up the docker image of CoCoPIE XGen controller.
Options:
  -h, --help                Show this help message.
  --region string            US or HK (default US; HK is faster for mainland
                             China).
  --path string              Home directory of XGen Controller (default
                             $HOME/XGenController).
Examples:
  bash xgen_controller_deploy.sh
  bash xgen_controller_deploy.sh --region HK
  bash xgen_controller_deploy.sh --region HK --path ~/XGenController
```

For most of the cases, the default options are good enough. So just run the following command:

```
bash xgen_controller_deploy.sh --region HK

# or

bash xgen_controller_deploy.sh --region US
```


If following output is shown, then the installation is successful.

```
Congratulations! XGen Controller has been successfully installed in the directory /home/ubuntu/XGenController (In the current example, the user is using Ubuntu. If your user is not using Ubuntu, that is okay. In any case, the installation will be performed in the directory specified by the $HOME variable)..
```

```
Next, you can access the XGen Controller by typing the command "run_controller". This will take you to the controller container environment.
```

```
For the initial Controller setup, kindly consult the documentation and configure the Controller's public IP address on the dashboard webpage.
```

After the installation, command `run_controller` will be written into `.bashrc`. Please run command `source ~/.bashrc` to make this command available in your shell.

Now for configuring the controller, please run command `run_controller` to start controller container and dependencies.

Output like this will be presented:

```
Kindly consult the documentation (https://xgen.cocopie.ai) and set up the public IP address of the controller on the dashboard webpage. Once the configuration is finalized, proceed to execute the run_controller command once more.
```

- By default, the URL of dashboard is the address of your controller host machine and listening port is `80` and opened to `127.0.0.1`. You can visit the dashboard by <http://127.0.0.1>
- If the user wants to visit dashboard in other machine instead of controller, the user should visit the dashboard by `LAN` (internal) IP address of controller host machine.
- If the firewall has been configured correctly, the user can also visit dashboard by `WAN` (public) IP address of controller host machine.

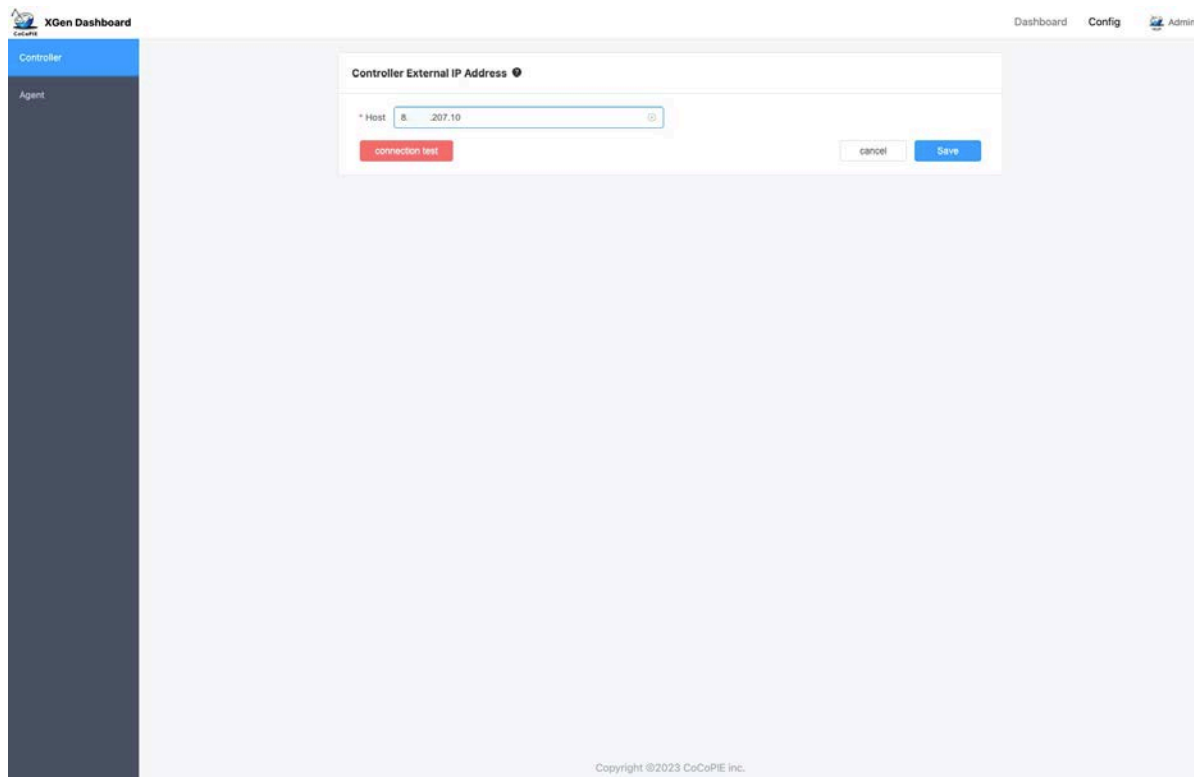
tips: If there are firewalls or other network restrictions, you can use an SSH tunnel to map port 80 on the deployed XGen machine to a port on your local machine, such as 8080. This way, you can access the remote machine's dashboard by visiting <http://127.0.0.1:8080>.

SSH tunnel port mapping example: `ssh -L 8080:127.0.0.1:80 <username>@<host_ip>`

At the first time, you'll be redirected to the login page. **The default account and password are both `admin`**. Then you'll be redirected to change password page to change the default password for security.

After changing your password, go to `Config` page by clicking the `Config` button on the top right corner. Then go to `Controller` tab to configure **controller's external IP address**. This address is used by other components (e.g., XGen, agents) to connect to the controller.

Please note, the IP address should be the `LAN` IP address of controller host machine. If the firewall is right configured, it can also be the `WAN` IP address of the controller hosting machine. Which address to use depends on the network environment and the way for the Device Lab to offer the service.



Click `save` to save the changes.

Then go back to the controller, enter the controller by running `run_controller`. (Alternatively, you can use the `docker exec -it xgen_controller bash` command, which has the same effect.) Then run `xgenc1 status` to check the status of controller and its dependencies.

```
xgenc1 status

Settings check passed.
Redis check passed.
MQ check passed.
Supervisor services check passed.
```

If any error is shown, please fix it before going to next step by following the error message.

You can use the following command to exit the Controller container, but it will not stop the Controller service:

```
$ exit
```

How to Upgrade Device Lab Controller

Just simply run `upgrade_controller`. Controller and dashboard will be upgraded to the latest version.

How to Uninstall Device Lab Controller

Just simply run `uninstall_controller`. The controller, dashboard, and related containers, images will be uninstalled.

How to Install Device Lab Agent

Agent should be installed **after** the controller is installed and configured.

Requirements

- GNU/Linux x86_64 with kernel version > 3.10 (Ubuntu or CentOS or WSL2)
- Docker
- Memory size: 16GB or more
- Disk size: 100GB or more
- Network: The agent needs to be accessible from the controller.
- Devices: The agent needs to be able to connect to the devices via USB. The default suggested devices are 64-bit devices (phones/tablets) that run **Android 9** and **iOS 11** or above. Support of other devices can be added separately.

Step 1: Install Docker

Please follow the official Docker instruction to make sure that Docker Engine has been installed.

Instructions: [Install Docker Engine](#)

Make sure that the user is in the docker group, if not, please run the following command:

```
sudo groupadd docker # if docker group does not exist
sudo usermod -aG docker $USER
```

Important: re-login to make the change take effect.

```
docker ps
```

if the above command can run successfully, the user has been granted the docker permission.

Install the following commands if they are not installed yet:

```
sudo apt install unzip curl git bc
```

Step 2: Install Device Lab Agent

Install Device Lab Agent with the install script.

Before installation, please ensure that `usbmuxd` service is **not** running on the host machine. Here we need to explain that we use `usbmuxd` for communication with iOS devices, similar to how `adb-server` works for Android devices. In the previous version, `usbmuxd` was installed as a system service on the host machine. In the current version, we have included it within the Docker container. Therefore, if `usbmuxd` is present on your machine, it will cause the agent container to fail to start. Therefore, when running the agent container, please stop the `usbmuxd` service and remove it to avoid conflicts.

How to stop `usbmuxd` service on UBUNTU20.04:

```
sudo systemctl stop usbmuxd
```

And how to delete `usbmuxd` service on UBUNTU20.04:

```
sudo apt remove usbmuxd -y
```

Install Device Lab Agent with the agent deploy script.

```
bash xgen_agent_deploy.sh
```

Usage: `bash xgen_agent_deploy.sh [OPTIONS]`

A script `for` fetching and setting up the docker image of CoCoPIE XGen agent.

Options:

`-h, --help`

Show this `help` message.

`--region string`

China).

US or HK (default US; HK is faster `for` mainland

```
--path string           Home directory of XGen (default $HOME/XGenAgent).
Examples:
bash xgen_agent_deploy.sh
bash xgen_agent_deploy.sh --region HK
bash xgen_agent_deploy.sh --region HK --path ~/XGenAgent
```

For most of the cases, the default options are good enough. So just run the following command:

```
bash xgen_agent_deploy.sh --region HK

# or

bash xgen_agent_deploy.sh --region US
```

Output like this will be presented:

```

Congratulations! XGen Agent is successfully installed under directory
/home/ubuntu/XGenAgent.(If your user is not using Ubuntu, that is okay. In any
case, the installation will be performed in the directory specified by the $HOME
variable).
Next, you can access XGen Agent by typing command 'run_agent'. It will lead you
to the Agent container environment.

For the initial execution of the Agent, please access the xgen_agent container
and provide the xgen-controller connection details in the configuration file
'/root/.xgen_agent/agent.conf'.
Once the agent.conf is configured, do not forget to run 'xgenctl restart' to
apply the settings, and use 'xgenctl status' to verify the proper functioning of
all services.
```

After the installation, command `run_agent` will be written into `.bashrc`. Please run command `source ~/.bashrc` to make this command available in your shell.

Running command `run_agent` to enter the agent. To make the agent accessible from the controller, there are some configurations to be done.

- Configure the `/root/.xgen_agent/agent.conf` file of the agent container like the following

```
xgen_controller_host=<your_controller_ip>
xgen_controller_port=28000 # do not change this
```

The 'xgen_controller_host' filled in here should be the public IP address of the controller, which is the same as the one entered in the dashboard configuration page during the controller installation.

- Then restart all background services by running the following command:

```
xgencctl restart
```

To check the status of the agent, run the following command:

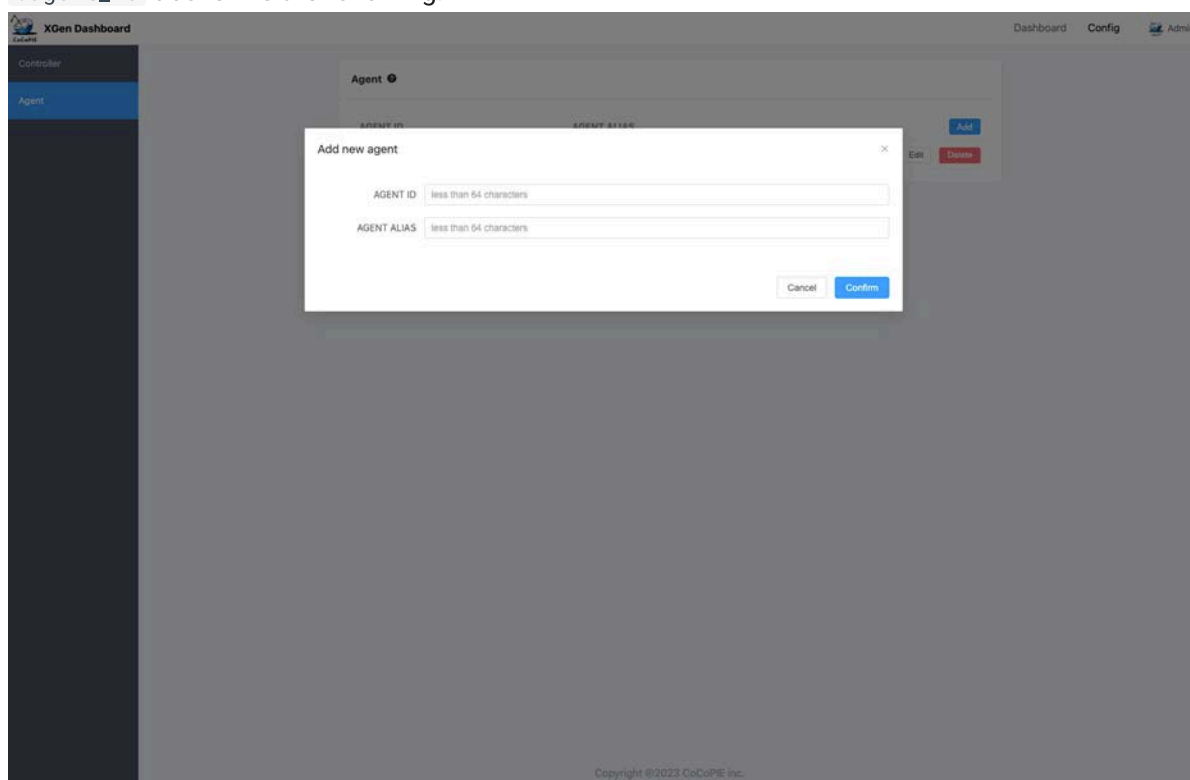
```
xgencctl status

Settings check passed.
Redis check passed.
MQ check passed.
Supervisor services check passed.
```

- One more thing. Check out the `.agent_id` at `/root/.xgen_agent/.agent_id` of the agent container

```
cat /root/.xgen_agent/.agent_id
```

Log into the dashboard, and then go to the "config -> agent" page. Add the agent with the `.agent_id` above like the following:



The purpose of this step is to add the agent ID to the whitelist of the controller, informing the controller that this is a legitimate agent.

- Before connecting any devices, please read the [How to connect Devices](#) part below for more information.
- Connect your devices via USB to the agent host machine, and run the following command to check the status of the devices:

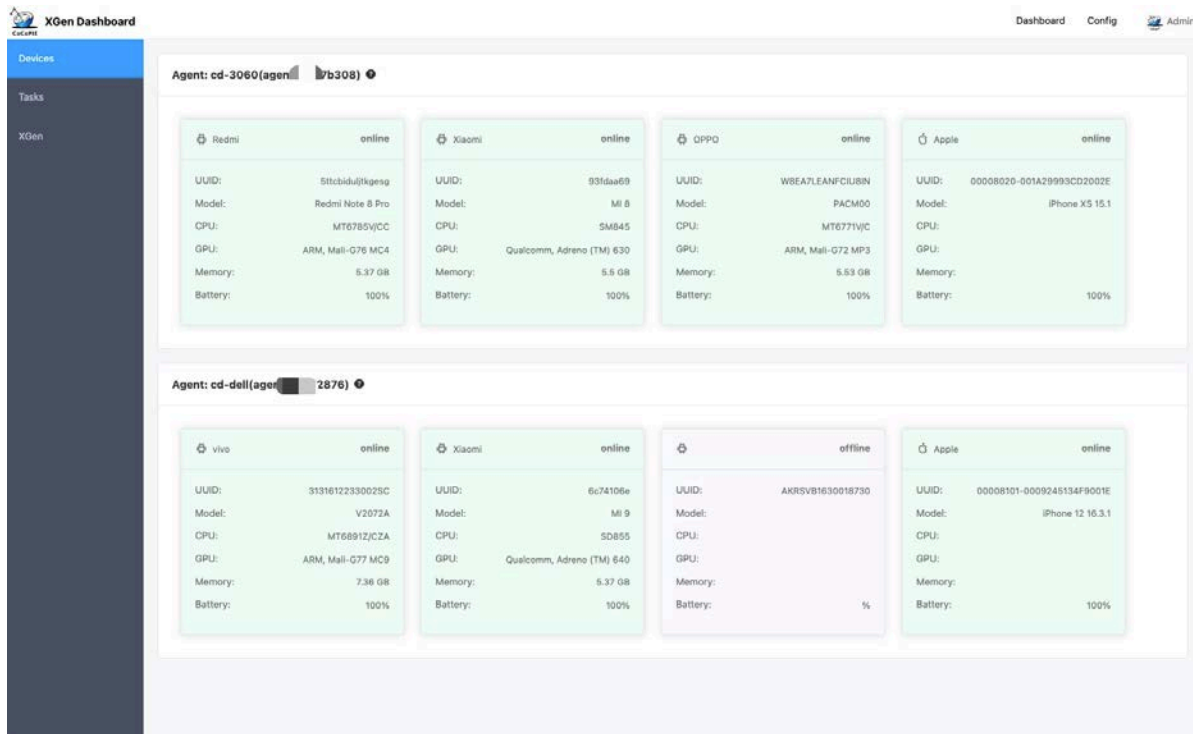
```
xgen_devices list
```

Platform	UUID	Brand	Model	Task Status
Connection Status				
-----	-----	-----	-----	-----
-	-	-	-	-
Android available	3131612233002SC	vivo	V2072A	idle
Android available	6c74106e	Xiaomi	MI 9	idle
iOS available	00008101-0009245134F9001E	Apple	iPhone 12 16.3.1	idle

- You can use the following command to exit agent container, but it will not stop the agent service:

```
$ exit
```

The user may connect multiple agents to a controller by simply repeating this process. If everything is OK, the agents will show in the dashboard:



How to Upgrade Device Lab Agent

Just simply run `upgrade_agent`

How to Uninstall Device Lab Agent

Just simply run `uninstall_agent`. The Agent and related containers, images will be uninstalled.

How to Install XGen

Requirements

- GNU/Linux x86_64 with kernel version > 3.10 (Ubuntu or CentOS or WSL2)
- NVIDIA GPU with Architecture >= Kepler (or compute capability 3.0)
- Memory size: 16GB or more
- Disk size: 500GB or more XGen by itself takes less than 20GB. But more space would be needed to hold AI models and datasets. The amount depends on the models and datasets the user uses. The AI models included in XGen's common AI are 500MB or less. The sizes of the default datasets included in XGen's common AI are as follows (downloadable on demand):

- ImageNet: 144.02 GB
- UCF101: 75.74 GB
- CoCo2017: 25.01 GB

Step 1: Install Nvidia-Driver, Docker and Nvidia Container Toolkit

If the XGen machine is running the Ubuntu Linux operating system:

1. Please make sure that nvidia-driver (version ≥ 510) has been installed. Instructions: [Nvidia-Driver Installation Guide](#) (CUDA installation is not mandatory on the host, because the Nvidia Container Toolkit in the later step includes CUDA support.) There is another solution as well if there is no specific requirement for the driver:

```
# install ubuntu-devices-common to check devices version
sudo apt install ubuntu-devices-common -y
# check devices version and determine CUDA driver version needed
sudo ubuntu-drivers devices # this command will list all available drivers and
related apt package name
# output example:
vendor    : NVIDIA Corporation
driver    : nvidia-driver-470 - distro non-free
driver    : nvidia-driver-535-open - distro non-free
driver    : nvidia-driver-525 - distro non-free
driver    : nvidia-driver-535 - distro non-free recommended
driver    : nvidia-driver-525-server - distro non-free
driver    : nvidia-driver-535-server - distro non-free
driver    : nvidia-driver-535-server-open - distro non-free
driver    : nvidia-driver-470-server - distro non-free
driver    : nvidia-driver-525-open - distro non-free
driver    : xserver-xorg-video-nouveau - distro free builtin

# install the recommended driver or the driver you need
sudo apt install nvidia-driver-535 -y
```

2. Please make sure that Docker Engine has been installed. Instructions: [Install Docker Engine](#)

```
curl -fsSL https://get.docker.com -o get-docker.sh
sudo bash ./get-docker.sh
```

3. Please make sure Nvidia Container Toolkit is installed. Instructions: [Nvidia Container Toolkit Installation Guide](#)

After completing the above steps, the user can verify whether the deployment requirements of XGen Docker Image are met by running the following command in Linux:

```
docker run --rm --gpus all nvidia/cuda:11.1.1-base-ubuntu20.04 nvidia-smi
```

When success, the NVIDIA GPU information will be output to the screen.

Hint: to run docker command without sudo authorization, the user needs to be granted the docker permission. The admin can do that through the following steps:

```
sudo groupadd docker # if docker group does not exist
sudo usermod -aG docker $USER
```

Important: re-login to make the change take effect.

```
docker ps
```

if the above command can run successfully, the user has been granted the docker permission.

Install the following commands if they are not installed yet:

```
sudo apt install unzip curl git bc
```

Run command below to avoid memory issue:

```
sudo sysctl vm.overcommit_memory=1
```

Step 2: Install XGen

XGen installation script has many installation options. The user can run the following command to see the options:

```
bash xgen_deploy.sh -h

Usage: bash xgen_deploy.sh [OPTIONS]
This script is designed to fetch and set up the Docker image for CoCoPIE XGen.
Options:
  -h, --help                Display this help message.
  --region string            Specify the region (default is US; HK is faster
  for mainland China).
  --memory bytes            Set the memory limit for XGen (default is 70% of
  System Memory).
  --xgen-path string        Define the home directory for XGen (default is
  $HOME/XGen).
```

```
--data string          Specify a location on the host to store the
dataset (default is $xgen-path/Data).
--output string        Define the output directory for XGen (default is
$xgen-path/Output).
Examples:
bash xgen_deploy.sh
bash xgen_deploy.sh --region HK
bash xgen_deploy.sh --xgen-path ~/ML/XGen
bash xgen_deploy.sh --xgen-path ~/ML/XGen --data ~/MLDataSets
```

For instance, the following command installs the XGen Docker image with the default setting. For users in mainland China, the following command usually runs faster, otherwise, use '--region US'.

```
bash xgen_deploy.sh --region HK
```

After the command, a folder `XGen` is created under the user `$HOME` directory as XGen's home directory, and a command `run_xgen` is put into the directory. The user can define a folder as XGen's home directory by adding the option `--xgen-path /path/to` to the command. (If the user has specified a system path as XGen's home directory, the user needs to run install script with `sudo`; it is however not recommended to install XGen in a system directory).

Update the shell environment:

```
source ~/.bashrc
```

The user can now type the following command to enter XGen shell:

```
run_xgen --name <username>
```

Step3: Configure XGen

XGen is proprietary software. If a user lacks a license, execute `xgenctl status` and adhere to the instructions for acquiring a license.

The license file must be named `license` and situated within the `/etc/xgen/license` directory. Users can verify the license status by executing `xgenctl status`.

Configure the file `/root/.xgen/xgen.conf` to your device lab controller IP address and port. like this:

```
xgen_controller_host = <controller_ip>
xgen_controller_port = 28000
```

Then run `xgenctl restart` to restart the daemon services. For checking the status of XGen, run `xgenctl status`.

```
xgenctl status

Your license has 959 days left.
License check passed.
Settings check passed.
Redis check passed.
Controller connection check passed.
```

If there is any problem reported by `xgenctl status`, please follow the error message to fix it.

Listing Devices

In the XGen container, we also provide a tool to list the devices connected to Device Lab by command `xgen_devices`. It works for both Android and iOS devices. The usage is as follows:

```
# enter into the XGen container
run_xgen --name <username>
# to see the usage of xgen_devices
xgen_devices --help

Usage: xgen_devices [OPTIONS]

    List all devices connected to XGen.

Options:
  -avl, --available-only      Only show available devices which are
                              reachable by XGen
  -p, --platform [android|ios|all]
                              Filter devices by platform
  -v, --verbose               Verbose mode
  -w, --watch                 Watch mode
  --help                     Show this message and exit.

# list all the connected devices
xgen_devices
```

The command returns a list of devices like the following:

Platform	UUID	Brand	Model	Task Status
Connection	Status			
-----	-----	-----	-----	-----
Android	3131612233002SC	vivo	V2072A	idle
available				
Android	6c74106e	Xiaomi	MI 9	idle
available				

Android available	5ttcbiduljtkgesg	Redmi	Redmi Note 8 Pro	idle
Android available	93fdaa69	Xiaomi	MI 8	idle
Android available	W8EA7LEANFCIU8IN	OPPO	PACM00	idle
iOS available	00008101-0009245134F9001E	Apple	iPhone 12 16.3.1	idle
iOS available	00008020-001A29993CD2002E	Apple	iPhone XS 15.1	idle

- `Platform` is the platform of the device, either `Android` or `iOS`.
- `UUID` is the device ID.
- `Brand` is the brand of the device.
- `Model` is the model of the device.
- `Task Status` is the status of the device, either `Idle` or `Busy`.
- `Connection Status` is the status of the device, either `Available` or `Unavailable`.

How to Start XGen

To start XGen, the first step is to start its docker container by running the following command, which resides in the `bin` folder of XGen:

```
run_xgen --name <username>
```

The command creates and starts an XGen container named `xgen_<version>_<username>`. Meanwhile, it creates a folder `Projects` under the XGen home directory. It is for holding all AI models and optimization results, including users' own models to be optimized. That folder is mapped to `/root/Projects` folder inside the XGen container; the files in that folder are hence accessible both inside and outside the XGen container.

After `run_xgen --name <username>`, XGen's shell will be started. In that shell, type the following to start XGen:

```
XGen
```

Please follow the instruction to get the license.

Because the XGen container is a docker container, users can use the standard docker commands to manage the container, such as stopping it, checking its status, and so on. **But users should avoid using docker commands to directly create XGen containers or rename the created XGen**

container. Always use `run_xgen --name <username>` to create XGen containers. Otherwise, there could be unexpected data overwritten and file corruptions. more detail XGen command usage please following [XGen Usage](#).

Sometimes, during an XGen session, users might experience a disconnection from the machine running XGen. Here are some helpful [hints](#).

How to Upgrade XGen

For users who already have version v1.1.0 installed, direct upgrading to the latest v1.3.0 version using the `upgrade_xgen` command **is not supported**. A fresh installation is required for v1.3.0 since there have been architectural and module organization optimizations.

For new users installing version v1.3.0, for future upgrades, simply use the `upgrade_xgen` command to directly upgrade.

For more details about v1.3.0, please see [What's New](#).

After re-installation, the user can start XGen as before through `run_xgen --name <username>`.

How to Uninstall XGen

The user may uninstall XGen by running the `uninstall_xgen` command in the `bin` folder of XGen:

```
uninstall_xgen
```

The command will prompt options for users to choose the components (e.g., previously generated models) to keep.

How to Connect Devices

XGen measures the speed of the optimized models directly on connected devices (e.g., Android phones). The devices should be connected to the machine that runs an XGen Agent. This part explains how to connect devices.

Before explaining the detail, it is worth noting that regardless of the method of connection, the following apply:

- For Android devices, the user should turn on the "Developer mode", "USB debugging". Please see [instructions](#).

- On the devices, at the first connection to the machine, an authorization window pops up; please authorize the machine to access this device and trust the machine always.
- The USB cable used for connection must be a data cable rather than a charging cable.
- Running command `adb devices` inside the agent shell would show the list of Android devices that are accessible by agent.
- Running command `sib devices` inside the agent shell would show the list of iOS devices that are accessible by agent.

Connect Android devices via USB

Connect the devices with the agent machine via a USB *data* cable. Some USB cables are only for charging, which won't work. (When the phone is connected with the computer via a data cable, usually the computer prompts to select connection mode, select `Transfer Files`.) The phone needs to be configured as stated above.

ADB has been installed in **Agent container**. So after completing the devices connection, then run the following command to check if the devices are connected to the agent container machine:

```
# enter
run_agent
# check if your android device is connected
xgen_devices list -nm
```

Connect iOS devices

1. Install `CoCoBench` app in `TestFlight`. For installation, the user needs to have `TestFlight` installed on the iOS device to be used with XGen, and follow the invitation link in the XGen welcome email to install `CoCoBench` via `TestFlight`, and then launch `CoCoBench` on the iOS device. To guarantee the app runs correctly, here are some tips:

- Make sure that `CoCoBench` of the latest version is installed.
- Make sure that `CoCoBench` is running foreground.
- Make sure that the screen of the iOS device is always on, which means that the screen is not locked and the device does not enter the sleep mode.
- Make sure that there is no password set for the iOS device.
- Make sure that the auto update of `CoCoBench` in `TestFlight` is turned on; otherwise, manually check it to keep the app up to date.

- By default, `CoCoBench` expires 90 days after the **publish date** due to the limitation of `TestFlight`. The user must update `CoCoBench` to the latest version in `TestFlight` before the expiration date; enabling auto update would simplify this.
2. Connect the iOS device to the host machine via a USB data cable (not a charging cable). Then the host machine might pop up a window to ask the user to trust the host machine. Click `Trust`.
 3. Enter **Agent container**, then run the following command to check if the iOS device is connected to the agent container machine:

```
# enter
run_agent
# check if your iOS device is connected
xgen_devices list -nm
```

4. If the iOS device falls offline due to some reason (e.g., unplugged and replug), the user needs to restart the `usbmuxd` services as shown below:

```
xgenctl restart -s usbmuxd
Service usbmuxd restarted.
```

Alternatively, a simpler approach is to restart all services.

```
xgenctl restart
```

After that, reconnect the iOS device to the host machine.

Hints

Avoid issues from disconnection Sometimes in the middle of an XGen session, the user may get disconnected from the machine that runs XGen. To ensure that the session is not interrupted by the disconnection and the user can get back to the session after reconnection, the user can use `tmux`:

- Install `tmux` by, for instance, the following command (on Ubuntu): `sudo apt-get install tmux`
- In the host terminal, run the following to start a new session: `tmux new -s Some-Session-Name`
- In the `tmux` window, start XGen as before: `run_xgen --name <username>`

- The session will continue running even if a disconnection happens. If the user is disconnected and then reconnected to the machine, run the following command will bring back the session:
`tmux attach -t Some-Session-Name .`

Chapter 3. Quick Start

1. Install XGen

- Follow the [install](#) instructions

2. Start XGen

Run the following command on an XGen machine to start XGen shell:

```
run_xgen --name <username>
```

Shell mode

In XGen shell, run the following command to start XGen:

```
XGen
```

Webserver mode

In XGen shell, type the following command to launch the web server (more information [XGen webserver usage](#)):

```
XGen webserver
```

3. Input requests

In XGen shell, can input requirements through interactive shell like below:

```
A DNN optimizer built on compression-compilation co-design.
It makes AI deployment sweet as pie by simultaneously
optimizing model size, accuracy, speed, and more.

© 2023 CoCoPIE Inc. All Rights Reserved.

Third Party Software License Disclaimer In:
/root/third_party_license.txt

Your license has 928 days left.

Which platform device do you want?
> Android
  iOS
  Simulation

Please choose your device(s):
[ ] Android -- AMFU6R1616020634 model: JLH-AN00
> [*] Android -- RFCN70JC9RL model: SM-N9810
Press <space> or <tab> for multi-selection, <up> and <down> or letter key <j> and <k> to move, <enter> to accept.

Please choose the base model (and the default dataset) to start with:
> Image Classification 0: DigitNet (MNIST)
  Image classification I: EfficientNet (default: ImageNet (high resolution))
  Image classification II: ResNet (default: ImageNet (high resolution))
  Image classification III: MobileNet (default: ImageNet (high resolution))
  Image classification IV: ViT (default: ImageNet (high resolution))
  Object detection: YOLOv6 (CoCo2017)
  Object detection: YOLOX (CoCo2017)
  Object detection: YOLOv8 (CoCo2017)
  Segmentation: UNet (ISBI-2012)
  Video classification: R2+1d (UCF101)
  Natural language processing: MobileBERT (Squad1.1)
  Your own model

Will you use the default dataset: MNIST?
> Yes
  No

What do you want to do? (Pruning recommended):
  Compatibility test
> Pruning
  Scaling
  PaS
  Customized operation

Please choose your primary interest
> Latency
  Quality score

[Desired inference latency (e.g., 10ms, 0.5s, or just use the default): 1.6ms
[Desired quality score (e.g., accuracy): MAX

Which GPU(s) do you want to use:
> [*] GPU 0: NVIDIA GeForce RTX 3090 (UUID: GPU-647eedd7-0986-f7aa-0a83-8974ba1fff95)
Press <space> or <tab> for multi-selection, <up> and <down> or letter key <j> and <k> to move, <enter> to accept.

[What is the batch size per GPU: 128
[What is the learning rate: 0.01
[Number of training or fine-tuning epochs: 3

XGen pretrained model absolute path (optional):
No value detected, confirm to continue?
> Yes
  No
```

Inx XGen webserver, can input requirements through XGen Web UI like below:



[START](#) [Home](#) [Sessions](#) [Models](#) [Datasets](#) [Tools](#) [Docs](#)

← Sessions

Which platform device do you want? ⓘ

☐ Mobile GPU/CPU (Android)

☐ Mobile GPU/CPU (iOS)

☒ Server GPU (NVIDIA)

☐ Server GPU (AMD)

☐ Server CPU (Intel, AMD)

☐ Embedded CPU (ARM MCU, Raspberry Pi)

Please choose the base model to start with

Image Classification 0: DigitNet

Choose the dataset

☒ MNIST (11.06 MB Training this dataset for 10 epochs using a single GTX 3090 would take approximately 10 minutes.)

☐ Choose Your Dataset

☐ Upload Your Dataset Now

Run Compatibility test or Choose optimizations method

Compatibility test

☒ Compatibility test

A quick check of whether the model and setup are compatible with XGen. Such a test is much quicker than optimizations (pruning or scaling).

Optimization Methods

☐ Pruning (Recommended ⓘ)

One of the ways that XGen uses to compress a model and generate efficient code. Pruning is suitable for networks with a significant number of 3x3 convolutions, as well as situations where 3D convolutions are present in the network.

☐ Scaling

One of the ways XGen uses to compress a model, which scales down (or up) the width of DNN layers. Scaling is a general method.

☐ PaS

One of the methods employed by XGen to compress a model involves reducing the width of DNN layers. For a detailed explanation, You can refer to the paper at <https://arxiv.org/abs/2206.01198>

☐ Customized operation

Users can change the XGen configuration of a model to enable special operations. (path: /root/Projects/[model_name_folder]/[model_name]_config/xgen.json)
More detail usage, please visit: [XGen Usage](#)

Please choose your GPU(s)

☒ GPU 0: NVIDIA GeForce RTX 3090 (UUID: GPU-5a90993f-966d-31af-980e-b8a7d23ba8f9)

What is the batch size per GPU:

128

Back

Reset

Submit

© 2023 CoCoPIE Inc. All Rights Reserved.

4. Wait for the result

* XGen will automatically compress the model and optimize the code to meet the required latency while maximizing the accuracy. The result comes out in several minutes and is stored in `/root/output/DigitNet_MNIST/``. The info about the final model and paths is shown on the screen:

In XGen shell:

Quality Score	Latency(ms)	Model Size	Non-zero model parameters
98.92	1.67	1.196M	320.458K

Closest Result:

Model-speed(latency): 1.67

Model Quality Score: 98.92

Model Size: 1.196M

Compiled File Path:

`/root/output/DigitNet_MNIST/20221013024128/164379d8-8d52-4ccompiled_file`

Onnx Model Path:

`/root/output/ DigitNet_MNIST /20221013024128/164379d8-8d52-4c.onnx`

Pytorch Model Path:

`/root/output/ DigitNet_MNIST /20221013024128/164379d8-8d52-4c.pth`

Please find the info of all existing models here: `/root/output/DigitNet_MNIST /20221013024128/summarized_data.csv`

In XGen Web UI:



Industry-First DNN Model-Code Co-Optimization Tool for Device AI
(built on compression-compilation co-design)

[← Sessions](#)[Output Log](#)[Parameter](#)[Output Product](#)

Successful

The training results are ready for you. You can download them and integrate them into your app or other applications.

The downloaded content represents the results that are closest to user requirements.

If you wish to reference other results, please refer to the table below.

Download

# Parameter	Value 1	Value 2
config_path	/root/output/DigitNet_MNIST/20231113082759/696d6dd9-be10-47args_ai.json	/root/output/DigitNet_MNIST/20231113082759/36c26208-59fb-4aargs_ai.json
work_place	/root/output/DigitNet_MNIST/20231113082759	/root/output/DigitNet_MNIST/20231113082759
onnx_path	/root/output/DigitNet_MNIST/20231113082759/696d6dd9-be10-47.onnx	/root/output/DigitNet_MNIST/20231113082759/36c26208-59fb-4a.onnx
pth_path	/root/output/DigitNet_MNIST/20231113082759/696d6dd9-be10-47.pth	/root/output/DigitNet_MNIST/20231113082759/36c26208-59fb-4a.pth
quality score	99.12	98.73
model_size	1.196M	1.196M
total_nz_parameters	1.196M	320.458K
flops	456.433M	5.668M
file_name	696d6dd9-be10-47.pth	36c26208-59fb-4a.pth
uuid	696d6dd9-be10-47	36c26208-59fb-4a
prune_rates	0	0.99
scaling_factor	0	0
trained_yn	1	1
pruning		slice
global_sparsity	0	0.732125273134286
global_sparsity_prune_layer	0	0.9899359809027776
quantized	0	0
pas_ratio	0	0
express_path	1	1

Chapter 4. Usage of XGen

XGen offers two ways to help users obtain desirable AI models and code:

- **Common AI: [First option to consider]** This approach uses the already included AI models in XGen as the base for its optimizations. XGen includes a set of AI models for common AI tasks. Users may choose to let XGen optimize the models on the default or their own datasets. This approach should be considered first as it is simple and can meet most needs.
- **Custom AI: [Only if Common AI is not applicable]** This approach allows users to invoke XGen to optimize their own AI models. This approach is more flexible but also more complicated, requiring more work from the users. To optimize their own models, the users must have a workable training script of their AI model, written in PyTorch (other front-end support will be added in the future).

In addition to optimizing AI, XGen has several other functionalities that may sometimes come in handy, such as AI model training, AI model/code quality assessment, and so on. This chapter explains those functionalities after describing the two main ways of using XGen for AI optimizations.

Overview of XGen Modes and Executions

XGen has a commandline interface. When users start XGen in a Shell terminal, XGen gets users' requirements and other inputs through a series of prompts. After that, XGen starts its execution. XGen provides four modes:

- **Compatibility testing:** to test whether the to-be-optimized AI model and its training script are compatible with XGen. The execution of this mode usually takes a much shorter time than other modes do.
- **Pruning:** to figure out what weights to prune from the to-be-optimized AI model and prune them such that the model becomes smaller and faster. To apply it to a custom AI model, the user needs to specify the pruning range and other parameters (detailed [later](#)). For now, this mode is opened only for **Android** devices.
- **Scaling:** to figure out the width of each layer and scale the network accordingly such that the model becomes smaller and faster. It is an alternative to pruning for model compression and

optimization. To apply it to a custom AI model, the user needs to embed a search variable in their model and specify some parameters (detailed [later](#)).

- **PaS:** to figure out the width of each layer through a model's computational graph and scale the network accordingly such that the model becomes smaller and faster. It is an alternative to pruning for model compression and optimization. To apply it to a custom AI model, the user needs to specify the pas ratio_list and other parameters (detailed [later](#)).
- **Customization:** to allow customized usage of XGen, which can be pruning an AI model in a specific way or training a model with a specific configuration.

The `pruning` and `scaling` modes conduct automatic search of the optimization space and generate the optimization results that best meet users' requirements. The `customization` mode does not conduct search, but executes training or pruning in a specific way defined by the user. It is worth noting that XGen also offers `distillation`, a way to enhance model accuracy; it can be used in both `pruning` and `scaling` modes. All the four modes can be used in both common AI and custom AI, explained next.

Common AI

XGen includes a set of base DNN models for some common types of AI tasks, listed in the following table, along with the default datasets. DigitNet is included mainly for demo purpose as it completes in several minutes. Please note that these models are only some samples of the many models XGen supports. Most AI models are supported by XGen thanks to its broad coverage of DNN operators as detailed in [Appendix I](#).

Category	Model	Dataset	Note
Image classification	DigitNet	MNIST	10 classes
Image classification	EfficientNet	ImageNet (auto-download)	1000 classes
Image classification	ResNet	ImageNet (auto-download)	1000 classes
Image classification	MobileNet	ImageNet (auto-download)	1000 classes

Category	Model	Dataset	Note
Image classification	ViT	ImageNet (auto-download)	1000 classes
Segmentation	UNet	ISBI-2012 (included)	–
Video classification	R2+1d	UCF101 (auto-download)	101 classes
Natural Language Processing (Question Answer)	MobileBERT	Squad1.1 (auto-download)	–
Object detection	YOLOv6	CoCo (auto-download)	80 classes
Object detection	YOLOX	CoCo (auto-download)	80 classes
Object detection	YOLOv8	CoCo (auto-download)	80 classes

If the AI task needed by a user is in one of the covered categories, common AI is the way to go. Through XGen's interface, users may pick one of the models that suite their task, and input their desired latency or accuracy and the dataset of interest. XGen will then transform the base model and generate code that meets the users' needs. The specific procedure is as follows.

Step 1: Start XGen

```
XGen # assuming that the XGen container was started already (Chapter 2)
```

Step 2: Input the Requirements

After XGen is started, it will collect the user's needs through a series of questions.

- Target Platform: Choose `Android` or `iOS` as the target platform. XGen will generate the optimized AI model and code for the selected platform. If no devices are available, the user can choose `Simulation` as the platform, in which case, XGen will still optimize the AI model to

find the smallest model that meets the desired accuracy, even though the reported latencies are imprecise; what it produces will be optimized model files rather than generated model code for a particular platform.

- Target device(s): The wizard lists the devices of the chosen platform currently connected to this machine. The user may select zero or more devices of interest. XGen will try to generate the AI model and code that best meet the requirements on all of the selected devices. If no device is selected (or there are no device connected/listed), XGen will behave as if `Simulation` was chosen as the platform: It uses pseudo-latency in its optimizations; the exact latency value would be imprecise, but the accuracy in the optimization process is still accurate; what it produces will be optimized model files rather than generated model code for a particular platform.
- Pick the base model: XGen then lists the already included base AI models (along with their default datasets). Users pick the model that best fits their needs. If none of the models fit the nature of the task, the user may consider [Custom AI](#) of XGen.
- Specify datasets: If users want to use datasets other than the default datasets associated with the picked AI model, the users need to specify the datasets of interest for training and evaluation. In that case, the users have to make sure that the datasets are organized and formatted in the same way as the default dataset of the model (detailed in [Appendix II](#)). Otherwise, the users would need to revise the data loader in the default training script of the model. If a default dataset is chosen but the dataset is not included in XGen, XGen will automatically download the dataset.
- Specify other model parameters: If non-default datasets are used, beside the dataset paths, some model parameters may need to be specified as well, such as the number of classes for classification models.
- Pick the task for XGen to perform. XGen lists four options as follows:
 - Compatibility test: a quick check of whether the model and setup are compatible with XGen. Such a test is much quicker than optimizations (pruning or scaling). So it is a good practice to run this test first before starting an optimization to find out possible problems sooner.
 - Pruning: one of the ways that XGen uses to compress a model and generate efficient code. It uses the world-leading proprietary pruning technology of CoCoPIE to prune DNN weights.
 - Scaling: one of the ways XGen uses to compress a model, which scales down (or up) the width of DNN layers.
 - PaS: one of the ways XGen uses to compress a model, which scales down the width of DNN layers.

- Customization: customized operation. Users can change the XGen configuration of a model to enable special operations, detailed later in [Advanced Common AI](#).

The three compression methods, *pruning*, *scaling* and *pas*, suit different AI models differently. Their differences are detailed later in [Custom AI](#). For a model in Common AI, the XGen prompt already specifies the recommended choice, as shown in the following example:

Please choose the base model (and the default dataset) to start with:

```
> Image Classification 0: DigitNet (MNIST)
Image classification I: EfficientNet (default: ImageNet (high resolution))
Image classification II: ResNet (default: ImageNet (high resolution))
Image classification III: MobileNet (default: ImageNet (high resolution))
Image classification IV: ViT (default: ImageNet (high resolution))
Object detection: YOLOv6 (CoCo2017)
Object detection: YOLOX (CoCo2017)
Object detection: YOLOv8 (CoCo2017)
Segmentation: UNet (ISBI-2012)
Video classification: R2+1d (UCF101)
Natural language processing: MobileBERT (Squad1.1)
Your own model
```

Will you use the default dataset: MNIST?

```
> Yes
No
```

What do you want to do? (Pruning recommended):

```
Compatibility test
> Pruning
Scaling
PaS
Customized operation
```

When some of the common AI models are selected, XGen may show only one of `pruning` and `scaling` in the option list because the other mode does not apply to the model at all.

- If one of `Image classification I: EfficientNet (high resolution, default: ImageNet)`, `Image classification II: ResNet (high resolution, default: ImageNet)` or `Image classification III: MobileNet (high resolution, default: ImageNet)` is chosen as the base model, user also can optionally optimize the model with `distillation`, detailed later in [config for distillation](#); XGen will prompt several relevant questions:

```
```bash
user can do distillation with your own model
Do you want to use distillation?
> Yes
No

Will you use the default distillation model?
> Yes
```

```

No
...
of course, user can also use their own distillation model

Do you want to use distillation?
> Yes
No

Will you use the default distillation model?
Yes
> No

Teacher model name: regnety_160
Teacher model path: /root/Projects/classif/root/Projects/classification-multi-
model-efficientnet/timm/teacher_models/regnety_160-a5fe301d.pth
Number of classes: 1000
...

```

- XGen then asks for the optimization objectives (except in the `Compatibility test` mode). Users may specify either "Quality score" or "Latency" as the primary interest, and the other as the secondary interest. XGen will try to meet the primary interest first and then try to meet the secondary interest. The acceptable specifications are as follows:
  - Latency: format is *NumberUnit*, where *Number* can be either an integer or a decimal number, *Unit* can be "ms" (milliseconds), "s" (seconds), "m" (minutes) or "hr" (hours). If latency is the secondary interest, users may also input `min` to indicate the intent to minimize the latency while meeting the primary goal.
  - Quality score: format is *op number*, where *op* can be `>`, `<`, `>=`, `<=`, and *number* can be either an integer or a decimal number. *Quality score* corresponds to the score used to measure the quality of the AI model. It can be accuracy for classification models, PSNR for image enhancement, loss for some other tasks, and so on. No matter what the actual measure is, XGen represents it uniformly in *Quality score* and handles it in a unified manner. If quality score is the secondary interest, users may also input `min` or `max` to indicate the intent to minimize or maximize the quality score while meeting the latency goal.
- XGen then asks which GPU(s) the user would like XGen to use. The user may select one or multiple GPUs; XGen will try to conduct training on all the selected GPUs if the training script of the model supports the use of multiple GPU devices.
- XGen then asks the users to input their desired training parameters of the AI model (e.g., batch size, learning rate, fine-tuning epoches).
- XGen then asks for the path of a pretrained model if there is one (except in the `Compatibility test` mode). If none, XGen will create a pretrained model before doing the model optimizations.

- XGen will automatically determine the level of optimization based on the user's selections in earlier questions. Depending on the chosen models and other selections, XGen will default to using Extensive Optimization.
  - Extensive Optimization (Taking several rounds of training time): Users don't need to make any choices, as this is the default optimization method provided by XGen. This option will be executed by default when users select Latency as the target. The model will first search within XGen's built-in models to see if there's one that satisfies user requirements. If a suitable model is found, it will use that built-in model to optimize the corresponding data. If no suitable model is found within the built-in models, it will resort to the optimization method previously chosen by the user to obtain an optimized model.
- Finally, XGen asks for *xgen-workplace*. It is a directory where the user wants to store models and data. If the user skips this question, XGen will create a unique folder and inform the user.

If the directory already contains the results of a previous run of XGen, XGen will check whether the XGen configuration used in the previous run is identical to the configuration in this current run. If so, XGen would ask the user whether to continue its work from where the previous run stopped to avoid unnecessary repeated work.

### Step 3: Wait for the Results

After inputting the requirements, XGen will start compressing and optimizing the base AI model. The process may include a series of search and one or several times of model trainings. The time it takes depends on how long the model training takes. For the seven models listed above, the time ranges from several hours to several days. For reference, below is a list showing the training time of each of the models on their default datasets:

```
* DigitNet (5-10 minutes on a 1-card titanX)
* EfficientNet [2 days on 4-card v100]
* ResNet-ImageNet [3 days on 4-card v100]
* MobileNet [7 days a 4-card 3090]
* ViT [7 days a 4-card v100]
* UNet [1 hour on 1-card 3090ti]
* R2+1d [2-3 days 8-card titanX]
* YOLOv6 [7 days on 2-car 3090]
* YOLOX [7 days on 2-car 3090]
* YOLOv8 [5 days on 2-car 3090]
```

During the execution of XGen, the status will be printed on the screen in real time, including the actual latency, quality score and model size of the generated models by XGen. At the end, the screen will show the path where the final results are stored. Please see [XGen Results](#) for the detailed explanation of the final results and how to use the results.

## Use Non-Default Datasets

In Common AI, each AI model comes with a default dataset. XGen supports the use of other datasets. To take that option, the users need to ensure that the datasets are in a format and structure compatible with the expectation of XGen. Please refer to [Appendix II](#) for details.

## Advanced Common AI

The standard common AI can already produce much more competitive results than other existing tools do. Advanced common AI may allow XGen to achieve even better results. To use it, the user needs to adjust the content of the XGen configuration file associated with the AI model, and then follow the same procedure as mentioned earlier.

The XGen configuration file determines many aspects of the optimizations by XGen. Each of the included AI models has one. Users can create the configuration for their own AI models as well. The following list provides the locations of the configurations of the included AI models and some other info.

- Image Classification 0: DigitNet (MNIST)
  - XGen config path: `/root/Projects/classification-digitnet/xgen.json`
  - Training script path: `/root/Projects/classification-digitnet/`
  - Recommend mode: `pruning`
- Image classification I: EfficientNet (ImageNet)
  - XGen config path: `/root/Projects/classification-multi-model-efficientnet/timm/efficientnet_config/xgen.json`
  - training script path: `/root/Projects/classification-multi-model-efficientnet/timm`
  - Recommend mode: `scaling`
- Image classification II: ResNet (ImageNet)
  - XGen config path: `/root/Projects/classification-multi-model-resnet/timm/resnet_config/xgen.json`
  - Training script path: `/root/Projects/classification-multi-model-resnet/timm`
  - Recommend mode: `pruning`
- Image classification III: MobileNet (ImageNet)
  - XGen config path: `/root/Projects/classification-mobilenet/timm/mobilenet_config/xgen.json`

- training script path: `/root/Projects/classification-mobilenet/timm/`
- Recommend mode: `scaling`
- Image classification IV: ViT (ImageNet)
  - XGen config path: `/root/Projects/classification-vit/timm/vit_config/xgen.json`
  - training script path: `/root/Projects/classification-vit/timm/`
  - Recommend mode: `scaling`
- Segmentation: UNet
  - XGen config path: `/root/Projects/segmentation-unet/unet_config/xgen.json`
  - training script path: `/root/Projects/segmentation-unet/`
  - Recommend mode: `scaling`
- Video classification: R2+1D
  - XGen config path: `/root/Projects/video-classification-s2-1d/s2+1d_config/xgen.json`
  - training script path: `/root/Projects/video-classification-s2-1d/`
  - Recommend mode: `pruning`
- Natural language processing: MobileBERT (Squad1.1)
  - XGen config path: `/root/Projects/nlp-mobilebert/mobilebert_config/xgen.json`
  - training script path: `/root/Projects/nlp-mobilebert/`
  - Recommend mode: `scaling`
- Object detection: YOLOv6 (CoCo2017)
  - XGen config path: `/root/Projects/object-detection-yolov6/yolov6_xgen/yolov6_config/xgen.json`
  - training script path: `/root/Projects/object-detection-yolov6/yolov6_xgen`
  - Recommend mode: `scaling`
- Object detection: YOLOX (CoCo2017)
  - XGen config path: `/root/Projects/object-detection-yolox/yolox_xgen/yolox_config/xgen.json`
  - training script path: `/root/Projects/object-detection-yolox/yolox_xgen`
  - Recommend mode: `scaling`
- Object detection: YOLOv8 (CoCo2017)

- XGen config path: `/root/Projects/object-detection-yolov8/yolov8_xgen/yolov8_config/xgen.json`
- training script path: `/root/Projects/object-detection-yolov8/yolov8_xgen`
- Recommend mode: `scaling`

Users can change the values of the parameters specified in the configuration files to change how XGen works with the model. Next is the explanation of the content of a XGen model configuration. *Hint: Before changing the default configuration of a common AI model, users are suggested to create a backup copy of the configuration file first for easy resetting when necessary.*

## Model Configuration for XGen

This part explains the essential entries in an XGen configuration by drawing on the following example.

```
{
 # general parameters of XGen.
 "general": {
 "work_place": null, # the folder for XGen to store results
 "pretrained_model_path": null, # pretrained model location
 "enable_ddp": true, # enable DDP training
 "CUDA_VISIBLE_DEVICES": "0,1,2,3", # which GPUs are used
 },

 # parameters of the original training script of the AI model can be put into
 this section
 # XGen will pass them into the training script
 "training_script_args": {
 "common_train_epochs": 30, # the number of epochs in one training
 # Optional: Any argument of the training_main script can appear here
 "video-scale": 2,
 "learning-rate": 0.01,
 "batch_size": 16, # the batch size per GPU
 },

 "user_requirements": {
 # Auto generated options from XGen interface. Do not change.
 "accuracy": null,
 "latency": 30,
 "use_distillation": true,
 "use_default_distillation_model": false,
 # Other options that are manually changeable.
 "margin": 2, # Acceptable error margin for the termination of XGen
 optimization
 "max_sampling": 3 # The max sampling number in the pruning or scaling
 mode
 }
}
```



```

If latency is primary: suggested value is 20;
If accuracy is primary: suggested value is at least
3, no more than 5
distillation default model options
"teacher_model": "efficientnet_b4", # default model name
"teacher_path": "/model_path_to/efficientnet_b4.pth", # default model weights
path
},

parameters defining the pruning of XGen
"pruning" : {
 "prune_method": "magnitude" # "magnitude" (for magnitude-based pruning;
default) or "admm" (for ADMM pruning) or "null" (no pruning)
 "prune_weight_ranking": global # "global" or "intra-layer"
 "prune_rate": 0.5, # uniform pruning rate (take effect only in the
customization mode)
 "prune_layers": null, # the list of layers to prune and their respective
pruning rates
 # If "null", XGen automatically decides the layers
to prune

 "admm_lr": 0.01, # admm special learning rate
 "admm_update_epoch": 5, # how frequently we update z and u in admm
pruning
 "admm_rho": 0.001, # (a hyperparameter in admm loss function)
},

parameters defining the scaling of XGen
"scaling" : {
 "searching_variable": null, # Variable to search; must be a variable in the
"pruning" or "training_script_args" section
 "searching_range": null, # A range (e.g., [0,1])
 "searching_granularity": null, # a number determines the smallest unit in the
search space; null means continuous space
},
"pas": {
 "pas_ratio": 0, # uniform pas scaling rate (take effect only in the
customization mode)
 "pas": false , # using pas slection
 "limit_loss_weights": 5.0, # pas limit loss weight which can limit model size
 "use_limit_loss": true, # using limit loss or not
 "pas_debug": false , # If ture while show loss item
 "pas_rebuild": false , # If true while rebuild model for pas_model
 "pas_finetune_epoch": 200, # number of epochs for fine-tune model for
pas_model
 "pas_pretrained_weight_path": null # pretrained pas_model weight path
},
distillation key args
"distillation": {
 "distillation_method" : "classic_distillation", # "null" (no distillation;
default) or "classic_distillation" (for classic distillation;)

 # classic_distillation config start

```

```

general setting
"original_loss_weights":0.1, # loss weights for original loss which define by
user
Tag loss setting:
"tag_loss_weights": 0.9, # tag loss weights
"tag_loss": 'kl', # None will ignore tag loss, only support mse loss and kl
loss for now
"tag_temperature": 4, # tag temperature
"tag_loss_combination_method": "avg", # if there are multiple teacher, how we
combined them, we support "avg" and "random" method for now
feature loss setting
"feature_loss_weights":0.9, # feature loss weights
"feature_default_temperature":1, # feature default temperature
"advance_feature_mapping": {}, # teacher_model_name: [student_layer_name,
teacher_layer_name,loss_type,weights,{adv_config}]
regularization loss
"regularization_loss_weights": 1, # regularization loss weight
"discriminator_lr":1e-4 # tag_discriminator learning rate
classic_distillation config end
},

"compiler": {
 "input_shape": "(1,3,360,640)", # The shape of the input tensor of the AI
model
 "opset_version": 11, # The version of ONNX (11 or 9); default 11
}
}

```

The configuration is a JSON file, consisting of seven sections. The inputs from the XGen interface automatically updates the corresponding entries in this JSON file. Users may also directly modify these entries. Most entries are self-explained in the example. Several notes are worth mentioning.

### Configure Pruning

- XGen has two pruning methods, magnitude-based pruning and ADMM-based pruning. The former prunes the weights with the least magnitudes, the latter uses a more advanced method to select the weights to prune. Parameter `prune_weight_ranking` determines whether the ranking of weights for pruning is within each individual layer or across all layers.
- In the `pruning` mode, parameters `prune_method` and `prune_rate` are ignored, and XGen uses magnitude-based pruning and automatically determines the appropriate pruning rates for the layers listed in `prune_layers`. If `prune_layers` is null, XGen automatically determines the layers to prune. The preset pruning rates in `prune_layers` are ignored in this mode.
- In the `customization` mode, XGen conducts either pruning (if `prune_method` is not null) or training of the AI model (otherwise); the parameter `prune_method` determines the pruning method, and the parameter `prune_layers` determines the layers to prune and their pruning rates. If the `prune_rate` parameter is not null, it takes a higher priority over the pruning rates

in `prune_layers`. In the `customization` mode, if `prune_method` is not null, `prune_rate` and `prune_layers` cannot be both null.

- The `prune_layers` entry is a JSON string consisting a list of layers in the AI model and their pruning rates (i.e., the percentage of weights to be removed). An example is as follows.

```
'{"prune_rates": {"layer1.0.conv1.weight": 0.2, "layer1.0.conv2.weight": 0.4,
"layer1.1.conv1.weight": 0.5, "layer1.1.conv2.weight": 0.5,
"layer2.0.conv1.weight": 0.5,
"layer2.0.conv2.weight": 0.5, "layer2.0.shortcut.0.weight": 0.5,
"layer2.1.conv1.weight": 0.5,
"layer2.1.conv2.weight": 0.5, "layer3.0.conv1.weight": 0.5,
"layer3.0.conv2.weight": 0.5,
"layer3.0.shortcut.0.weight": 0.5, "layer3.1.conv1.weight": 0.4,
"layer3.1.conv2.weight": 0.2}}'
```

- Users can also create a separate yaml file to specify the layers and the pruning rate of each layer, and then provide the file path as the value of "prune\_layers"; XGen will get the value from the file content. To help users set `prune_layers`, XGen provides a utility function for users to get the list of names of the layers in an AI model amenable for pruning. Users just need to add `from co_lib.utils import export_pruning_layer` to the beginning of the training script of the AI model and use `export_pruning_layer(the-AI-model,output-yaml-file-name)` in the training script to export the model's prunable layers into a file. The pruning rates in the exported list are all set to 0.5 by default. Below is an example of such a yaml file:

```
prune_ratios:
 module.layer1.0.conv1.0.0.weight:
 0.6
 module.layer2.0.conv2.0.3.weight:
 0.8
```

- To use ADMM-based pruning, the `customization` mode has to be used. Two runs of XGen are needed. The first run must set `prune_method` to `admm` and the second run `magnitude`. The two runs must use the same `prune_rate` and `prune_layers`.

## Configure Scaling

**IMPORTANT:** *To use scaling for a custom AI model, it is necessary for the user to add a search variable into the AI model and configure the scaling in the configuration file. Without that, `scaling` would not function and would give only the original model.*

- In the `scaling` mode, XGen has a generic design. It changes the value of the `searching_variable` while assessing the quality of the AI model. So in principle, the user may express any aspect of the AI model with the `searching_variable`, and then use `scaling` of

XGen to search for the best value. A typical use of this feature is to express, in the training script, the widths or depths of the layers of the AI model as expressions of the `searching_variable`. Then, XGen will automatically scale down (or up) those layers towards the objectives. For instance, in the training script, the user may set the first three layer's widths to `100*w`, `200*w`, `300*w`. If in the configuration file `w` is set as the `searching_variable` and the search range is `[0.5, 1]` with a step size 0.1, XGen will investigate six variants of the AI model with different widths. Users can also use this feature to change other aspects of the AI model. Note that the `searching_variable` must be a variable in the `training_script_args` section in the configuration file.

Below is an example showing how scaling is configured in UNet:

```
Inside the configuration file of UNet (config.json),
nch_ker is set as the searching variable.
"searching_variable": "nch_ker",

Inside the model construction file of UNet, `nch_ker` is used as a variable
to control
the width of each layer. Notice that different factors are used with
`nch_ker` at different
layers.

self.enc1_1 = CNR2d(1 * self.nch_in, 1 * self.nch_ker, kernel_size=3,
stride=1, norm=self.norm, relu=0.0)
self.enc1_2 = CNR2d(1 * self.nch_ker, 1 * self.nch_ker, kernel_size=3,
stride=1, norm=self.norm, relu=0.0)

self.pool1 = Pooling2d(pool=2, type='max')

self.enc2_1 = CNR2d(1 * self.nch_ker, 2 * self.nch_ker, kernel_size=3,
stride=1, norm=self.norm, relu=0.0)
self.enc2_2 = CNR2d(2 * self.nch_ker, 2 * self.nch_ker, kernel_size=3,
stride=1, norm=self.norm, relu=0.0)

self.pool2 = Pooling2d(pool=2, type='max')

self.enc3_1 = CNR2d(2 * self.nch_ker, 4 * self.nch_ker, kernel_size=3,
stride=1, norm=self.norm, relu=0.0)
self.enc3_2 = CNR2d(4 * self.nch_ker, 4 * self.nch_ker, kernel_size=3,
stride=1, norm=self.norm, relu=0.0)

self.pool3 = Pooling2d(pool=2, type='max')

self.enc4_1 = CNR2d(4 * self.nch_ker, 8 * self.nch_ker, kernel_size=3,
stride=1, norm=self.norm, relu=0.0)
self.enc4_2 = CNR2d(8 * self.nch_ker, 8 * self.nch_ker, kernel_size=3,
stride=1, norm=self.norm, relu=0.0)

self.pool4 = Pooling2d(pool=2, type='max')
```

```
self.enc5_1 = CNR2d(8 * self.nch_ker, 2 * 8 * self.nch_ker, kernel_size=3,
stride=1, norm=self.norm, relu=0.0)
```

- Model accuracy is sensitive to the shapes of the stem (i.e. input) layer and the output head layer. So when setting up for scaling, try to avoid scaling down those layers much. There are various online resources on how to best do AI model scaling.
- In the `customization` mode, XGen does not do scaling; the *scaling* section in the configuration file is ignored.

## Configure Distillation

Distillation is a way to improve the accuracy of a pruned or scaled model.

To use distillation, users just need to answer `Yes` when being asked in the XGen interactive interface. In the XGen configuration file (`xgen.json`), there are several options in the `user_requirements` section about distillation and a separate `distillation` section that control the behavior of distillation, as shown in the example configuration file in the [Model Configuration for XGen](#) part. Please see the comments in that example configuration file for the explanations of those options.

## Configure PaS

PaS (Pruning as Search) is a novel method of structured pruning that searches through a model's computational graph to find a smaller and more optimal model. To use the PaS algorithm, the user must select the `PaS` option when being asked in the XGen interactive interface, or select the customization mode while having parameter `pas` set in the configuration file.

The main PaS configurations include `pas_ratio`, which indicates the degree of compression you want for your model; `pas_finetune_epoch`, which specifies the number of training epochs required for fine-tuning your final model. See the example configuration above for other parameters.

Initially, a larger PaS-generated model is trained to identify more suitable weights, and then, using the fine-tune epochs specified by the user, the previously-generated model is pruned to the desired size. Notably, this method cannot adequately handle dynamic computational graphs and requires the user's model to have a static graph, with no conditional statements in the forward function, among other requirements. In experimental comparisons, the use of `torch.batchnorm` may significantly impact the accuracy. If using the PaS algorithm, it is recommended to replace `torch.batchnorm` with `nn.BatchNorm`.

## Hints

Here are some hints on using the various options provided in the XGen configuration.

- The `pretrained_model_path` entry is for holding the location of the pretrained model if there is any. The format of the pretrained model can be one of the following:
  - result saved by `torch.save()`, such as `torch.save({'net': net.state_dict(), 'acc': acc, 'epoch': epoch, }, file)`, or `torch.save(net.state_dict(), file)`
  - a pickle file created by, for instance, `pickle.dump(quant_sim_model, file)`
- An optimization done by XGen (`pruning` or `scaling`) involves the trainings of multiple AI models, and may hence take hours or days to complete. Usually fewer trainings are needed if "latency" rather than "quality score" is chosen as the primary interest.
- If the original model weights were obtained by distillation (or transfer learning) from some other networks, the `pruning` mode is often a better option than `scaling`.
- Pruning is applicable to a layer only if the sizes of the kernels in that layer are multiples of 8, such as 4x4.
- When users are not sure of what configurations work best, they can try to run XGen multiple times with a different configuration used each time.
- The `pruning` and `scaling` modes both involve an iterative search process, finding the appropriate pruning rates or scaling variable values. The `customization` mode does not involve a search process. XGen simply trains the model based on the provided configuration; if `pruning_method` is not null, XGen conducts a one-time pruning of the AI model before training it.
- XGen supports data parallel training on multiple GPUs in a machine. To use that support, the user just needs to enable parallel training in the training script of the AI model. This version of XGen does not support training one model on multiple machines.

## Example

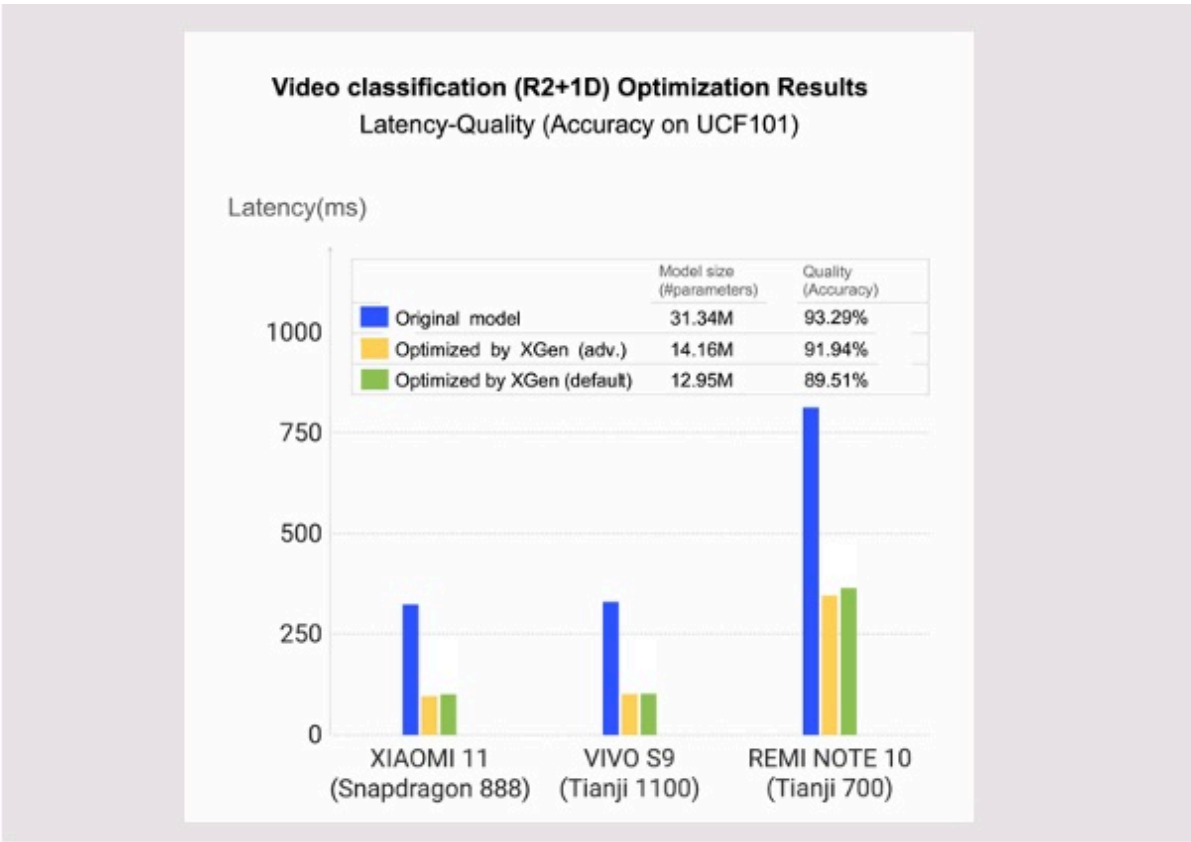
As an example, the following table shows the differences between the default configuration of `R2+1d` model and modified configurations that achieve better speed-quality tradeoffs. The modified configurations use the `customization` mode, including two runs of XGen. The first run uses the ADMM method to identify the weights that are good to prune and make them small in magnitude, and the second run uses magnitude-based pruning to set those weights to zero and retrain the model.

Parameter	Default configuration	Modified configurations
mode	pruning	customization

Parameter	Default configuration	Modified configurations
prune_method	magnitude	admm (run-1), magnitude (run-2)
common_train_epochs	35	50 (run-1), 130 (run-2)
learning_rate	0.005	0.0005
admm_update_epoch	5	10
admm_rho	0.001	0.0001
prune_layers	null	{module.layer1.0.conv1.0.0.weight: 0.8
		module.layer1.0.conv1.0.3.weight: 0.8
		module.layer1.0.conv2.0.0.weight: 0.8
		module.layer1.0.conv2.0.3.weight: 0.8
		module.layer1.1.conv1.0.0.weight: 0.8
		module.layer1.1.conv1.0.3.weight: 0.8
		module.layer1.1.conv2.0.0.weight: 0.8
		module.layer1.1.conv2.0.3.weight: 0.8
		module.layer2.0.conv1.0.0.weight: 0.7
		module.layer2.0.conv1.0.3.weight: 0.7
		module.layer2.0.conv2.0.0.weight: 0.7
		module.layer2.0.conv2.0.3.weight: 0.7
		module.layer2.1.conv1.0.0.weight: 0.7

Parameter	Default configuration	Modified configurations
		module.layer2.1.conv1.0.3.weight: 0.7
		module.layer2.1.conv2.0.0.weight: 0.7
		module.layer2.1.conv2.0.3.weight: 0.7}

The customization mode leads to both a higher accuracy and a lower latency on three devices, as shown as follows.



## Custom AI

If users have their own model to optimize, custom AI of XGen can be the way to go. But before using this option, the users are recommended to first consider common AI as custom AI involves more efforts on the user's end.



To use custom AI, the user must already have a workable training script of their own AI model. *The script must be written in PyTorch.* (Support of other front ends—such as TensorFlow, Keras—will be added in the future.)

The usage of custom AI is similar to common AI except that the user would need to pick the "Your own model" option when selecting the base model in the XGen interface. The rest of the procedure is self-explained. There are however three steps to go through before XGen can start working on a custom AI:

- Checking the model applicability;
- Preparing the running environment;
- Preparing the training script. An example is included inside XGen ( `/root/examples` inside XGen container).

### Step I: Checking Model Applicability

Although XGen has a broad coverage of DNN operators, some AI models may still contain operators beyond XGen's coverage and cause XGen to fail. This first step is to run such a check.

#### 1. Export the AI model to an ONNX file

The user may use the PyTorch function `torch.onnx.export()` to export the model to an ONNX file. There are three requirements for making the call:

- The second argument, model input, **must have one as the batch size of the input** (i.e., the first dimension of the tensor must be 1);
- the `do_constant_folding` argument must be `True`;
- the `opset_version` argument must be **11** or **9**. Here is an example:
- do **not** set `dynamic_axes` argument.

```
torch.onnx.export(
 model, # model being run
 x, # model input
 onnx_save_path, # where to save the model
 do_constant_folding=True,
 input_names=['input'], # the model's input names (an
arbitrary string)
 output_names=['output'], # the model's output names (an
arbitrary string)
 opset_version=11 # XGen supports 11 or 9
)
```

Sometimes users' training script may already has a customized function to export ONNX format of the model. In that case, the user should use that function instead of the default

PyTorch function.

## 2. Run the check

Use `XGen onnx_latency_benchmark` to test the support status of generated onnx file, if the benchmark succeeded, it means that current onnx file is supported. Otherwise, the onnx file is not supported.

The supported operators (listed in the [Appendix I](#)).

## Step II: Prepare Running Environment

Because XGen involves trainings of the AI model inside its docker container, the users need to install into the docker container the dependencies required by the training of the AI model. It is better to ensure that the training script can run properly in the docker container before moving to the next step.

## Step III: Prepare Training Scripts

To allow XGen to optimize a custom AI model, the training script of the model has to be revised so that XGen can interact with the training script during the optimization process. This part explains the needed changes.

Descriptions of the APIs used below can be found in [Appedix III](#)

1. Rename the main training script to `train_script_main.py`, and ensure that (i) at the end of the script, the script invokes the main training function; (ii) the main training function to be named `training_main`; (iii) this renamed script `train_script_main.py` reside in the top-level folder of the custom AI model (e.g., at the end of the script, the script invokes the main training function. (XGen currently requires the main training function to be named `training_main`; this requirement will become obsolete in the next version.)

```
if __name__ == '__main__':
 training_main()
```

2. Import the XGen modules into the training script by adding the following lines to the beginning of the training script:

```
from co_lib import Co_Lib as CL
from xgen_tools import *
```

3. Initialize XGen, and connect the control variables in the training script with the internal variables inside XGen so that XGen can control the training process. Some of the internal variables of XGen to be connected with are listed as follows. For the full list, please see the

`training_script_args` field in xgen config json file (e.g., `/root/Projects/object-detection-yolox/yolox_xgen/yolox_config/xgen.json`).

```
common_train_epochs: control the number of training epochs
learning_rate: control the learning rate
batch_size: control the batch size
device_num: the number of GPUs to use
num_classes: the number of classes if this model is a classification model

train_data_path: path of the training data
eval_data_path: path of the eval data
train_label_path: path of the train label
eval_label_path: path of the eval label
pretrain_model_weights_path: the path of the pretrained model weights
```

XGen sets these internal variable values based on the values it attains during the interactions with the user. So for the training script to follow these controls, the user would need to ensure that the corresponding variables inside the training script take their values from these XGen internal variables.

XGen offers an API `xgen_init(user_args, args_ai=None, map=COCOPIE_MAP)` to facilitate the process. The call, as shown as follows, returns two values, with the first (`originalArgs` in the example) holding the original arguments in the training script, and the second (`args_ai`) holding the internal variables of XGen.

```
user_args represents a user variable holding arguments in the training_main
function
the returned args_ai is the same as the argument `args_ai`
the return type of originalArgs is identical to the type of user_args (e.g.,
type argparse.Namespace)
originalArgs, args_ai = xgen_init(user_args, args_ai=None, map=COCOPIE_MAP)
```

Specifically, the variables that xgen actually controls for the training process are in `args_ai['training_script_args']` field. So for reference conveniences, the user may add this line:

```
xgen_control_args = args_ai['training_script_args']
```

Note, that call to `xgen_init` should be put after the parsing of arguments in the original training script, where `user_args` should be replaced with the variable in the original training script that holds the arguments parsing results (of type `argparse.Namespace` or `dict`).

User can then connect the variables in the `originalArgs` with the corresponding variables in `args_ai`. For instance, if the original training script uses `batchSize` variable for the batch size of training, the following code will connect it with the internal variable of XGen.

```
originalArgs.batchSize = xgen_control_args['batch_size']
```

If the original training script uses other ways to organize the control variables of training, the modifications would just need to ensure that the connection is done properly. For example, the training script of Yolox passes the training parameters into the training script through a Python file (`config.py`) and then uses a variable `exp` of type `ABCMeta` to hold them. The corresponding code changes would need to connect the fields in `exp` with the corresponding internal variables of XGen:

```
XGen: invoke xgen_init
If args is not provided, it can be set as an empty dictionary{}.
If args is {}, then args will be consistent with the args_ai generated by
initialization in xgen_init
user_args, args_ai = xgen_init(args, args_ai=None)
XGEN: The variable that xgen actually controls the training process is in
args_ai['training_script_args'],
which is a dictionary, so you can use it in the following ways.
Note: The args_ai['training_script_args'] field is actually mapped by the
'training_script_args' field in xgen config json
xgen_control_args = args_ai['training_script_args']
XGen: connect variables with XGen internal variables
exp.max_epoch = xgen_control_args['common_train_epochs']
exp.data_dir = xgen_control_args['train_data_path']
```

It is worth mentioning that it is not necessary to connect XGen's every internal variable. If the training script has no use of some of the controls, those variables do not need to be connected with the user's variables. Normally, at the minimum, the `common_train_epochs` should be connected with the user's variable.

For conveniences, XGen provides a builtin mechanism for forming the connection. But **note** that this mechanism applies only if the original variables are directly passed into the training function through arguments. Rather than writing those assignment statements, the user may use a dictionary `COCOPIE_MAP`, as illustrated as follows. The variables in the original training script are put as the keys and the XGen internal variables as the values. Put this map at the beginning of the training script. Then, after the call to `user_args, args_ai = xgen_init(args, args_ai=None)`, the `user_args` will contain all the original variables in the `COCOPIE_MAP`, and they will be all connected with the internal variables of XGen.

```
from xgen_tools import XgenArgs
COCOPIE_MAP = {
 'training epochs variable' : 'common_train_epochs',
 'your_lr': 'learning_rate', # assign `learning_rate` in xgen
configuration to `your_lr`
 'your_batch_size': 'batch_size', # assign `batch_size` in xgen
configuration to `your_batch_size`
```

```

 'your_train_data_path': 'train_data_path', # assign `train_data_path` in
xgen configuration to `your_train_data_path`
 'your_val_data_path': 'eval_data_path', # assign `val_data_path` in xgen
configuration to `your_val_data_path`
 'your_train_label_path': 'train_label_path', # assign `train_label_path`
in xgen configuration to `your_train_label_path`
 'your_eval_label_path': 'eval_label_path', # assign `eval_label_path` in
xgen configuration to `your_eval_label_path`
 'your_pretrain_model_weights_path': 'pretrain_model_weights_path', #
assign `pretrain_model_weights_path` in xgen configuration to
`your_pretrain_model_weights_path`,
 }

```

After the connection is done, the user would need to ensure that the rest of the training script uses those control variables to control the training process.

4. Insert the following line after the line where the AI model architecture is assigned to a variable ( `AImodel` in the example). This line should replace the origin model weight loading function in the script. It loads the model weights from the file specified by the `pretrained_model_path` entry in the XGen configuration file. If that entry is `null`, the model will be initialized by the default model initialization in the training script.

```

AImodel represents a variable holding the model architecture
xgen_load(AImodel, args_ai=args_ai)

```

5. Insert the following line after the DNN model is loaded, but before the training starts. `AImodel`, `dataLoader`, and `AIoptimizer` represent variables in the training script: `AImodel` is the holder of the DNN model, `dataLoader` is the training data loader, and `AIoptimizer` is the optimizer of the training process.

```

cl_model = CL.init(args=args_ai, model=AImodel:torch.nn.Module,
optimizer=AIoptimizer:torch.optim,
 data_loader=dataLoader:torch.utils.data.data_loader.DataLoader)
if cl_model:
 AImodel = cl_model

```

6. Insert the following line at the beginning of each training epoch (inside the training epoch loop), where `epochCount` is the counter of epochs:

```

CL.before_each_train_epoch(epoch=epochCount)

```

7. Insert the following line after the changes to the learning rate in the training loop, where `epochCount` is the counter of epochs. Skip this step if the training loop has no changes to the learning rate.

```
CL.after_scheduler_step(epoch=epochCount)
```

8. Insert the following line after the loss function, where `loss` is the loss tensor calculated by the loss function in the training script. This line allows XGen to adjust the loss for its optimization purpose.

```
loss = CL.update_loss(loss)
```

9. Inside the training loop, insert the following line right after an evaluation of the current model, where `AImodel` and `epochCount` have been explained earlier, and `evaluationResult` represents a variable for the evaluation result, such as the accuracy, average PSNR (for super resolution), and so on. The evaluation can be invoked for every one or more epochs. If there are no evaluations in the training loop, the user would need to add them.

```
xgen_record(args_ai, AImodel, evaluationResult, epoch=epochCount)
```

- Note:

- If the training script uses EMA, DataParallel or DistributedDataParallel, the actual model is in a field `module` of the model (i.e., `AImodel.module`), in which case, the line should be `xgen_record(args_ai, AImodel.module, evaluationResult, epoch=epochCount)`
- Sometimes the training script already has a customized function to export the model in ONNX format. In that case, the user should use that customized function to export the model to an ONNX file. The user may then pass the `onnx_file_path` to `xgen_record` as follows: `xgen_record(args_ai, AImodel, evaluationResult, epoch=epochCount, onnx_file_path= the onnx file path );`

10. At the end of the `training_main` function, insert the following. If the training script has no evaluation function, use `-float('inf')` for `evaluationResult`

```
xgen_record(args_ai, AImodel, evaluationResult, epoch=-1)
```

11. If the user wants to use the "Scaling" mode of XGen to compress the model, the user needs to modify the training script of the AI model to give XGen the control of the scale (e.g., width, depth) of the AI model. Please see [Configure Scaling for XGen](#) part for details.
12. XGen has a builtin order check that inspects the training script to identify obvious api call errors. The api order check is enabled by default. If the user wants to disable it, she can add the following line into the training script:

```
from xgen_tools import helper
helper.ENABLE_HELPER = False
```

## Step IV: Prepare Config file

The next step is to prepare a configuration file which determines how XGen applies optimizations to the model. Please see [Model Configuration](#) for details.

### EXAMPLE DOCUMENTS

You can refer to our documentation on YOLOX model modification to understand this process.

[YOLOX example](#)

### EXAMPLE CODE

Following is an example of a training script after being revised for XGen. The changed parts are surrounded by `xgen_tools START` and `xgen_tools END` comments.

```
xgen_tools START
#####
from co_lib import Co_Lib as CL
from xgen_tools import *
COCOPIE_MAP = {'train_epochs' : 'common_train_epochs'}
xgen_tools END
#####

def training_main():

 originalArgs = ... # parsing users' parameters (including "train_epochs" in
 this example)

 # xgen_tools START
 #####
 user_args,args_ai = xgen_init(originalArgs,map = COCOPIE_MAP)
 # xgen_tools END
 #####

 # define dataloader
 trainDataLoader = torch.utils.data.DataLoader(...)
 testDataLoader = torch.utils.data.DataLoader(...)

 # define DNN model
 model = ...

 # define loss function
 criterion = ...

 # define optimizer
```

```
optimizer = torch.optim.Adam(
 model.parameters(),
 lr=args.learning_rate,
 betas=(0.9, 0.999),
 eps=1e-08,
 weight_decay=args.decay_rate
)
scheduler = torch.optim.lr_scheduler.StepLR(optimizer, step_size=30,
gamma=0.7)

xgen_tools START
#####
#load model
xgen_load(model,args_ai=args_ai)
cl_model = CL.init(args=args_ai, model=model, optimizer=optimizer,
data_loader=trainDataLoader)
if cl_model:
 model = cl_model
xgen_tools END
#####

for epochCount in range(0,user_args.train_epochs):

 # xgen_tools START
 #####
 CL.before_each_train_epoch(epoch=epochCount)
 # xgen_tools END
 #####

 scheduler.step()

 # xgen_tools START
 #####
 CL.after_scheduler_step(epoch = epochCount)
 # xgen_tools END
 #####

 for batch_id, data in enumerate(trainDataLoader):

 model.train()

 output = model(input)

 loss = criterion(...) # regular loss, i.e., cross-entropy, mse, ...

 # xgen_tools START
 #####
 loss = CL.update_loss(loss)
 # xgen_tools END
 #####

 loss.backward()
 optimizer.step()
```



```

 if epoch % user_args.eval_epochs == 0:
 accuracy = eval(model)
 # save the model
 save_path = 'path_name.pth'
 state = {
 'epoch': epochCount,
 'model_state_dict': model.state_dict(),
 'optimizer_state_dict': optimizer.state_dict(),
 }
 # xgen_tools START
 #####
 #if users want to save the intermediate models and log info
 (optional)
 xgen_record(args_ai,model,accuracy,epoch=epochCount)
 # xgen_tools END
 #####

 accuracy = eval(model) # needed only if the total of the training epochs is
less than one
 # xgen_tools START
 #####
 xgen_record(args_ai,model,accuracy,epoch=-1)

if __name__ == '__main__':
 # don't forget to add this line to make sure it will start the main function
 training_main()

```

## Customized Operation

It is common when user wants to include a new model to XGen, user may need to add some customized operations, want to control the pruning rate / pruning method or just want to run the model without optimized by XGen. User'll need `customized operation` mode to do that. Before start, please make sure the model has been already correctly adapted to XGen due to [Advanced Common AI](#).

Here, let's take a look at how to use `customized operation` mode.

### 1. Choose DigitNet (MNIST) for example.

```

Please choose the base model (and the default dataset) to start with:
> Image Classification 0: DigitNet (MNIST)
Image classification I: EfficientNet (high resolution, default: ImageNet)
Image classification II: ResNet (high resolution, default: ImageNet)
Image classification III: MobileNet (high resolution, default: ImageNet)
Image classification IV: ViT (default: ImageNet (high resolution))
Segmentation: UNet (ISBI-2012)
Object detection: YOLOv6 (CoCo2017)
Object detection: YOLOX (CoCo2017)

```

```
Object detection: YOLOv8 (CoCo2017)
Video classification: R2+1d (UCF101)
Natural language processing: MobileBERT (Squad1.1)
Your own model
```

2. In the mode choice, choose `customized operation mode`.

```
What do you want to do? (Pruning recommended):
Compatibility test
Pruning
Scaling
PaS
> Customized operation
```

3. Choose or input any further question.

```
Which GPU(s) do you want to use:
> [*] GPU 0: NVIDIA GeForce RTX 3090 (UUID: GPU-e75cbd31-7281-3c03-xxxx-
xxxxxx)
 [] GPU 1: NVIDIA GeForce RTX 3090 (UUID: GPU-647eedd7-0986-f7aa-xxxx-
xxxxxx)
Press <space> or <tab> for multi-selection, <up> and <down> or letter key
<j> and <k> to move, <enter> to accept.

What is the batch size per GPU: 128
What is the learning rate: 0.01
How many epochs: 3

XGen pretrained model absolute path (optional):
No value detected, confirm to continue?
> Yes
 No

Your workplace for this session (if you don't input a path, XGen will make
one for you):
This is your default workplace for this session:
/root/output/DigitNet_MNIST/20221219121928
```

In the output, you'll see there is only **one** stage in XGen, because XGen won't do any other optimizing operations like pruning or scaling in `customized operation mode`. And the `latency` and `accuracy` of the original model is measured by XGen. That is to say, if you want to know the model latency or accuracy before being optimized by XGen, user can use `customized operation mode`.

Another use case is that you want to control the pruning rate or pruning method. user can also use `customized operation mode` to do that. There is a little bit more work to do. Just simply edit the `pruning` field of `xgen.json` config file in your model project. For `DigitNet`, the file is located at

~/Projects/classification-digitnet/xgen.json . user can change the `pruning` field to the following:

```
{
 ...
 "pruning": {
 "prune_method": "magnitude",
 "prune_layers": null,
 "prune_weight_ranking": "intra-layer",
 "prune_rate": 0.1,
 "admm_update_epoch": 5,
 "admm_rho": 0.001,
 "admm_sparsity_type": "block_punched",
 "admm_lr": 0.01,
 "admm_block": "(8,4)",
 "prune_threshold": -1.0
 },
 ...
}
```

There are many choices for value of `pruning` field. Please refer to the `pruning` field in [Model Config](#) for more details. Now when you run XGen in `customized operation` mode, user can see the pruning rate is `0.1` and the pruning method is `magnitude` . That is to say, XGen won't do any pruning operation in this case.

## Other Usage

At the center of XGen are three components, the AI model optimizer for model compression and optimization, the compiler for code optimization and generation, and the model testing component for measuring the speed of the models. The usage of XGen mentioned so far automatically invokes all of those components. But those components can also be used separately. One of the benefits this flexibility brings is that even if there are no devices connected, XGen can still be used to optimize an AI model via the AI model optimizer (with some limitations); the other example flexibility is to use XGen to skip the model optimization step, and quickly generate executable code for a model and test it out or profile its execution.

## Model Optimization Alone Without Running on Devices

This option is for the case where the user wants to find a model with a certain accuracy and minimum latency but does not have or want to use any connected device. The usage of XGen is the same as in the default case except that in the step of platform selection, select `Other Device` , or in the step of device selection, select no device.

In this case, XGen does not run the AI models on the physical devices to measure the speed. It instead uses a simple oracle to output a pseudo-latency. The oracle only ensures that the pseudo-latencies it gives out follow the rule that the smaller the AI model is, the faster it runs. The pseudo-latency values can be far away from the real latencies. But because the rule it follows is roughly consistent with actual observations, despite the imprecision of latency, XGen can still find satisfying AI models when the users' requirement is to meet an accuracy while minimizing the latency. (The accuracy it gets is still correct.) If the users' requirement is to find a model with a certain latency, this option is not applicable.

## Using the Compiler and Testing Components Only

This usage allows the compiler and the testing components to be invoked without the AI optimizer. It can come handy if the users have collected one or more AI models and would like to measure their speeds (in a batch).

Users can do that through the following command:

```
XGen onnx-latency-benchmark --model-path [onnx-model-path] --output-path [output-path] --devices device_id1 device_id2 ... --verbose
```

Note: The standalone `onnx-latency-benchmark` command (without `XGen`) is deprecated. Please use `XGen onnx-latency-benchmark` instead.

The script will measure the speed of every ONNX model in the `onnx-model-path` and output one single csv file in the output path. The csv file reports the inference latency of each of the models along with some other detailed information, such as the place where the code generated by the compiler resides ( `output_dir` ), the number of parameters in the model ( `params` ), the amount of computation in the model ( `MACs` ).

Parameter `devices` is optional, if you don't specify it, you will enter an interactive interface to select the devices. Parameter `verbose` is also optional, if you specify it, the script will print out the detailed information.

```
XGen onnx-latency-benchmark --benchmark --model-path [onnx-model-path] --output-path [output-path] --devices device_id1 device_id2 ... --verbose
```

Parameter `benchmark` is optional. If specified, this tool will record the running time of the model and the detailed running information of each operator.

```
I deepvan/run/statistics.cc:359] -----


```

```

I deepvan/run/statistics.cc:359]
Sort by Run Order
I deepvan/run/statistics.cc:359] -----

I deepvan/run/statistics.cc:359] | Op Type | Avg(ms) | % | GMACPS |
MACs(Dense) | Stride | Pad | Filter Shape | Output Shape | Dilation |
name |
I deepvan/run/statistics.cc:359] -----

I deepvan/run/statistics.cc:359] | BufferTransform | 0.046 | 0.873 | 0.000 |
0.00000 | | | | [1,224,224,3] | |
deepvan_node_input.1_mem_type_2 |
I deepvan/run/statistics.cc:359] | Conv2D | 0.076 | 1.427 | 53.477 |
4.06426 | [2,2] | [2,2] | [12,3,3,3] | [1,112,112,12] | [1,1] |
Clip_1 |
I deepvan/run/statistics.cc:359] | DepthwiseConv2d | 0.071 | 1.329 | 19.135 |
1.35475 | [1,1] | [2,2] | [1,12,3,3] | [1,112,112,12] | [1,1] |
Clip_3 |
I deepvan/run/statistics.cc:359] | Conv2D | 0.037 | 0.690 | 49.152 |
1.80634 | [1,1] | [0,0] | [12,12,1,1] | [1,112,112,12] | [1,1] |
Conv_4 |
I deepvan/run/statistics.cc:359] | Eltwise | 0.029 | 0.540 | 0.000 |
0.00000 | | | | [1,112,112,12] | |
Add_5 |
I deepvan/run/statistics.cc:359] | Conv2D | 0.071 | 1.331 | 76.432 |
5.41901 | [1,1] | [0,0] | [36,12,1,1] | [1,112,112,36] | [1,1] |
Clip_7 |
I deepvan/run/statistics.cc:359] | DepthwiseConv2d | 0.106 | 1.999 | 9.541 |
1.01606 | [2,2] | [2,2] | [1,36,3,3] | [1,56,56,36] | [1,1] |
Clip_9 |
I deepvan/run/statistics.cc:359] | Conv2D | 0.037 | 0.697 | 48.623 |
1.80634 | [1,1] | [0,0] | [16,36,1,1] | [1,56,56,16] | [1,1] |
Conv_10 |
I deepvan/run/statistics.cc:359] | Conv2D | 0.035 | 0.652 | 57.757 |
2.00704 | [1,1] | [0,0] | [40,16,1,1] | [1,56,56,40] | [1,1] |
Clip_12 |
I deepvan/run/statistics.cc:359] | DepthwiseConv2d | 0.072 | 1.361 | 15.572 |
1.12896 | [1,1] | [2,2] | [1,40,3,3] | [1,56,56,40] | [1,1] |
Clip_14 |
I deepvan/run/statistics.cc:359] | Conv2D | 0.037 | 0.701 | 53.736 |
2.00704 | [1,1] | [0,0] | [16,40,1,1] | [1,56,56,16] | [1,1] |
Conv_15 |
I deepvan/run/statistics.cc:359] | Eltwise | 0.017 | 0.316 | 0.000 |
0.00000 | | | | [1,56,56,16] | |
Add_16 |
I deepvan/run/statistics.cc:359] | Conv2D | 0.046 | 0.871 | 90.836 |
4.21478 | [1,1] | [0,0] | [84,16,1,1] | [1,56,56,84] | [1,1] |
Clip_18 |
I deepvan/run/statistics.cc:359] | DepthwiseConv2d | 0.077 | 1.454 | 7.653 |
0.59270 | [2,2] | [2,2] | [1,84,3,3] | [1,28,28,84] | [1,1] |
Clip_20 |

```

```

I deepvan/run/statistics.cc:359] | Conv2D | 0.057 | 1.075 | 32.209 |
1.84397 | [1,1] | [0,0] | [28,84,1,1] | [1,28,28,28] | [1,1] |
Conv_21 |
I deepvan/run/statistics.cc:359] | Conv2D | 0.028 | 0.532 | 55.751 |
1.58054 | [1,1] | [0,0] | [72,28,1,1] | [1,28,28,72] | [1,1] |
Clip_23 |
I deepvan/run/statistics.cc:359] | DepthwiseConv2d | 0.041 | 0.775 | 12.301 |
0.50803 | [1,1] | [2,2] | [1,72,3,3] | [1,28,28,72] | [1,1] |
Clip_25 |
I deepvan/run/statistics.cc:359] | Conv2D | 0.053 | 0.992 | 29.906 |
1.58054 | [1,1] | [0,0] | [28,72,1,1] | [1,28,28,28] | [1,1] |
Conv_26 |
I deepvan/run/statistics.cc:359] | Eltwise | 0.011 | 0.207 | 0.000 |
0.00000 | | | | [1,28,28,28] | |
Add_27 |
I deepvan/run/statistics.cc:359] | Conv2D | 0.028 | 0.533 | 58.745 |
1.66835 | [1,1] | [0,0] | [76,28,1,1] | [1,28,28,76] | [1,1] |
Clip_29 |
I deepvan/run/statistics.cc:359] | DepthwiseConv2d | 0.041 | 0.775 | 12.984 |
0.53626 | [1,1] | [2,2] | [1,76,3,3] | [1,28,28,76] | [1,1] |
Clip_31 |
I deepvan/run/statistics.cc:359] | Conv2D | 0.056 | 1.052 | 29.765 |
1.66835 | [1,1] | [0,0] | [28,76,1,1] | [1,28,28,28] | [1,1] |
Conv_32 |
I deepvan/run/statistics.cc:359] | Eltwise | 0.011 | 0.206 | 0.000 |
0.00000 | | | | [1,28,28,28] | |
Add_33 |
I deepvan/run/statistics.cc:359] | Conv2D | 0.047 | 0.890 | 81.510 |
3.86355 | [1,1] | [0,0] | [176,28,1,1] | [1,28,28,176] | [1,1] |
Clip_35 |
I deepvan/run/statistics.cc:359] | DepthwiseConv2d | 0.044 | 0.819 | 7.121 |
0.31046 | [2,2] | [2,2] | [1,176,3,3] | [1,14,14,176] | [1,1] |
Clip_37 |
I deepvan/run/statistics.cc:359] | Conv2D | 0.059 | 1.105 | 32.825 |
1.93178 | [1,1] | [0,0] | [56,176,1,1] | [1,14,14,56] | [1,1] |
Conv_38 |
I deepvan/run/statistics.cc:359] | Conv2D | 0.036 | 0.681 | 48.379 |
1.75616 | [1,1] | [0,0] | [160,56,1,1] | [1,14,14,160] | [1,1] |
Clip_40 |
I deepvan/run/statistics.cc:359] | DepthwiseConv2d | 0.027 | 0.510 | 10.396 |
0.28224 | [1,1] | [2,2] | [1,160,3,3] | [1,14,14,160] | [1,1] |
Clip_42 |
I deepvan/run/statistics.cc:359] | Conv2D | 0.053 | 0.998 | 33.042 |
1.75616 | [1,1] | [0,0] | [56,160,1,1] | [1,14,14,56] | [1,1] |
Conv_43 |
I deepvan/run/statistics.cc:359] | Eltwise | 0.009 | 0.169 | 0.000 |
0.00000 | | | | [1,14,14,56] | |
Add_44 |
I deepvan/run/statistics.cc:359] | Conv2D | 0.034 | 0.635 | 58.452 |
1.97568 | [1,1] | [0,0] | [180,56,1,1] | [1,14,14,180] | [1,1] |
Clip_46 |
I deepvan/run/statistics.cc:359] | DepthwiseConv2d | 0.030 | 0.556 | 10.727 |
0.31752 | [1,1] | [2,2] | [1,180,3,3] | [1,14,14,180] | [1,1] |

```

```

Clip_48 |
I deepvan/run/statistics.cc:359] | Conv2D | 0.061 | 1.140 | 32.548 |
1.97568 | [1,1] | [0,0] | [56,180,1,1] | [1,14,14,56] | [1,1] |
Conv_49 |
I deepvan/run/statistics.cc:359] | Eltwise | 0.009 | 0.173 | 0.000 |
0.00000 | | | [1,14,14,56] | | |
Add_50 |
I deepvan/run/statistics.cc:359] | Conv2D | 0.036 | 0.681 | 58.135 |
2.10739 | [1,1] | [0,0] | [192,56,1,1] | [1,14,14,192] | [1,1] |
Clip_52 |
I deepvan/run/statistics.cc:359] | DepthwiseConv2d | 0.032 | 0.596 | 10.667 |
0.33869 | [1,1] | [2,2] | [1,192,3,3] | [1,14,14,192] | [1,1] |
Clip_54 |
I deepvan/run/statistics.cc:359] | Conv2D | 0.065 | 1.211 | 32.673 |
2.10739 | [1,1] | [0,0] | [56,192,1,1] | [1,14,14,56] | [1,1] |
Conv_55 |
I deepvan/run/statistics.cc:359] | Eltwise | 0.009 | 0.166 | 0.000 |
0.00000 | | | [1,14,14,56] | | |
Add_56 |
I deepvan/run/statistics.cc:359] | Conv2D | 0.044 | 0.835 | 77.942 |
3.46842 | [1,1] | [0,0] | [316,56,1,1] | [1,14,14,316] | [1,1] |
Clip_58 |
I deepvan/run/statistics.cc:359] | DepthwiseConv2d | 0.041 | 0.776 | 13.481 |
0.55742 | [1,1] | [2,2] | [1,316,3,3] | [1,14,14,316] | [1,1] |
Clip_60 |
I deepvan/run/statistics.cc:359] | Conv2D | 0.097 | 1.817 | 46.068 |
4.45939 | [1,1] | [0,0] | [72,316,1,1] | [1,14,14,72] | [1,1] |
Conv_61 |
I deepvan/run/statistics.cc:359] | Conv2D | 0.048 | 0.908 | 75.887 |
3.66912 | [1,1] | [0,0] | [260,72,1,1] | [1,14,14,260] | [1,1] |
Clip_63 |
I deepvan/run/statistics.cc:359] | DepthwiseConv2d | 0.037 | 0.692 | 12.446 |
0.45864 | [1,1] | [2,2] | [1,260,3,3] | [1,14,14,260] | [1,1] |
Clip_65 |
I deepvan/run/statistics.cc:359] | Conv2D | 0.082 | 1.533 | 44.937 |
3.66912 | [1,1] | [0,0] | [72,260,1,1] | [1,14,14,72] | [1,1] |
Conv_66 |
I deepvan/run/statistics.cc:359] | Eltwise | 0.010 | 0.184 | 0.000 |
0.00000 | | | [1,14,14,72] | | |
Add_67 |
I deepvan/run/statistics.cc:359] | Conv2D | 0.049 | 0.915 | 82.211 |
4.00781 | [1,1] | [0,0] | [284,72,1,1] | [1,14,14,284] | [1,1] |
Clip_69 |
I deepvan/run/statistics.cc:359] | DepthwiseConv2d | 0.036 | 0.679 | 13.858 |
0.50098 | [1,1] | [2,2] | [1,284,3,3] | [1,14,14,284] | [1,1] |
Clip_71 |
I deepvan/run/statistics.cc:359] | Conv2D | 0.088 | 1.649 | 45.621 |
4.00781 | [1,1] | [0,0] | [72,284,1,1] | [1,14,14,72] | [1,1] |
Conv_72 |
I deepvan/run/statistics.cc:359] | Eltwise | 0.010 | 0.185 | 0.000 |
0.00000 | | | [1,14,14,72] | | |
Add_73 |
I deepvan/run/statistics.cc:359] | Conv2D | 0.086 | 1.606 | 107.551 |

```

```

9.20102 | [1,1] | [0,0] | [652,72,1,1] | [1,14,14,652] | [1,1] |
Clip_75 |
I deepvan/run/statistics.cc:359] | DepthwiseConv2d | 0.049 | 0.920 | 5.868 |
0.28753 | [2,2] | [2,2] | [1,652,3,3] | [1,7,7,652] | [1,1] |
Clip_77 |
I deepvan/run/statistics.cc:359] | Conv2D | 0.161 | 3.025 | 38.064 |
6.13402 | [1,1] | [0,0] | [192,652,1,1] | [1,7,7,192] | [1,1] |
Conv_78 |
I deepvan/run/statistics.cc:359] | Conv2D | 0.100 | 1.882 | 76.578 |
7.67693 | [1,1] | [0,0] | [816,192,1,1] | [1,7,7,816] | [1,1] |
Clip_80 |
I deepvan/run/statistics.cc:359] | DepthwiseConv2d | 0.042 | 0.783 | 8.630 |
0.35986 | [1,1] | [2,2] | [1,816,3,3] | [1,7,7,816] | [1,1] |
Clip_82 |
I deepvan/run/statistics.cc:359] | Conv2D | 0.202 | 3.798 | 37.948 |
7.67693 | [1,1] | [0,0] | [192,816,1,1] | [1,7,7,192] | [1,1] |
Conv_83 |
I deepvan/run/statistics.cc:359] | Eltwise | 0.008 | 0.149 | 0.000 |
0.00000 | | | | [1,7,7,192] | |
Add_84 |
I deepvan/run/statistics.cc:359] | Conv2D | 0.102 | 1.905 | 81.196 |
8.24141 | [1,1] | [0,0] | [876,192,1,1] | [1,7,7,876] | [1,1] |
Clip_86 |
I deepvan/run/statistics.cc:359] | DepthwiseConv2d | 0.044 | 0.824 | 8.800 |
0.38632 | [1,1] | [2,2] | [1,876,3,3] | [1,7,7,876] | [1,1] |
Clip_88 |
I deepvan/run/statistics.cc:359] | Conv2D | 0.211 | 3.959 | 39.077 |
8.24141 | [1,1] | [0,0] | [192,876,1,1] | [1,7,7,192] | [1,1] |
Conv_89 |
I deepvan/run/statistics.cc:359] | Eltwise | 0.009 | 0.162 | 0.000 |
0.00000 | | | | [1,7,7,192] | |
Add_90 |
I deepvan/run/statistics.cc:359] | Conv2D | 0.108 | 2.031 | 98.428 |
10.64986 | [1,1] | [0,0] | [1132,192,1,1] | [1,7,7,1132] | [1,1] |
Clip_92 |
I deepvan/run/statistics.cc:359] | DepthwiseConv2d | 0.061 | 1.142 | 8.204 |
0.49921 | [1,1] | [2,2] | [1,1132,3,3] | [1,7,7,1132] | [1,1] |
Clip_94 |
I deepvan/run/statistics.cc:359] | Conv2D | 0.297 | 5.579 | 61.216 |
18.19350 | [1,1] | [0,0] | [328,1132,1,1] | [1,7,7,328] | [1,1] |
Conv_95 |
I deepvan/run/statistics.cc:359] | Conv2D | 0.207 | 3.895 | 110.606 |
22.95082 | [1,1] | [0,0] | [1428,328,1,1] | [1,7,7,1428] | [1,1] |
Clip_97 |
I deepvan/run/statistics.cc:359] | Pooling | 0.030 | 0.573 | 0.000 |
0.00000 | [7,7] | [0,0] | [7,7] | [1,1,1,1428] | |
AveragePool_99 |
I deepvan/run/statistics.cc:359] | FullyConnected | 0.692 | 12.998 | 2.063 |
1.42800 | | | [1000,1428,1,1] | [1,1,1,1000] | |
Conv_100 |
I deepvan/run/statistics.cc:359] | BufferTransform | 0.011 | 0.216 | 0.000 |
0.00000 | | | | [1,1000,1,1] | |
deepvan_node_469_mem_type_0 |

```



```

I deepvan/run/statistics.cc:359] | Reshape | 0.005 | 0.096 | 0.000 |
0.00000 | | | | [1,1000] | |
Reshape_101 |
I deepvan/run/statistics.cc:359] | FullyConnected | 0.694 | 13.027 | 2.058 |
1.42800 | | | [1000,1428,1,1] | [1,1,1,1000] | |
Conv_102 |
I deepvan/run/statistics.cc:359] | BufferTransform | 0.012 | 0.223 | 0.000 |
0.00000 | | | | [1,1000,1,1] | | |
deepvan_node_476_mem_type_0 |
I deepvan/run/statistics.cc:359] | Reshape | 0.005 | 0.095 | 0.000 |
0.00000 | | | | [1,1000] | | |
Reshape_103 |
I deepvan/run/statistics.cc:359] | Eltwise | 0.006 | 0.108 | 0.000 |
0.00000 | | | | [1,1000] | | |
Add_104 |
I deepvan/run/statistics.cc:359] | Eltwise | 0.006 | 0.112 | 0.000 |
0.00000 | | | | [1,1000] | | |
Div_105 |

```

*Hint:* Users would need to first put the ONNX model files into the XGen container. There are two approaches:

- They may put the ONNX model files under the `Projects/` folder of XGen on the host machine. Because that folder is mapped to `/root/Projects` folder inside the XGen container, the files will be directly accessible to XGen.
- They may use `docker cp`. Example: `docker cp ./onnxModels`  
`xgen_<version>_<username>:/root/speedTests/`. The `onnxModels` will be copied into `/root/speedTests/` inside the XGen container.

## Using XGen in Non-Interactive Mode

XGen can be used in a non-interactive mode. The user needs to provide a task-description file. The config file is a json file that contains all the information needed to run XGen. The command to use in this mode is as follows:

```

XGen non-interactive --config-file /path/to/config.json --platform
[android|ios|simulation] --devices device_id_a device_id_b ...

```

The argument `--platform` is optional; if you don't specify it, the script will use the `device_platform` config of the config file. The argument `devices` is one or more UUIDs of devices. It is optional, but is required if the `--platform` argument is `android` or `ios`.

The config file also has `device_platform` and `devices` fields corresponding to the `platform` and `devices` fields in the command line. The command line values override those in the config file.

The non-interactive mode of XGen allows the use of the models included in XGen (Common AI) or customized AI models, with default or customized datasets.

XGen includes some sample config files (in `/root/noninteractive_config_sample`) for users' reference.

Following is an example config file of efficientnet on imageNet dataset:

```
{
 "description": "this is a sample of common ai config file in non-interactive mode",
 "device_platform_options": [
 "android",
 "ios",
 "simulation"
],
 "device_platform": "simulation",
 "devices": [],
 "xgen_model_options": {
 "description": "different platforms support different models, 'simulation' platform supports all models",
 "android_support": [
 "DigitNet_MNIST",
 "EfficientNet_ImageNet",
 "ResNet_ImageNet",
 "MobileNet_ImageNet",
 "ViT_ImageNet",
 "YOLOv6_CoCo2017",
 "UNet_ISBI-2012",
 "R2+1d_UCF101",
 "MobileBERT_Squad1.1"
],
 "ios_support": [
 "DigitNet_MNIST",
 "EfficientNet_ImageNet",
 "MobileNet_ImageNet",
 "YOLOv6_CoCo2017",
 "UNet_ISBI-2012",
 "MobileBERT_Squad1.1"
]
 },
 "select_model": "EfficientNet_ImageNet",
 "use_default_dataset": false,
 "custom_dataset_config": {
 "description": "if use_default_dataset == false, config the section, section title is the same as your select_model",
 "dataset": {
 "train_data_path": {
 "required": true,
 "value": "/data/jenkins_common_ai/data/imagenet_subsets/train"
 },
 },
 },
}
```

```

 "train_label_path": {
 "required": false,
 "value": ""
 },
 "eval_data_path": {
 "required": true,
 "value": "/data/jenkins_common_ai/data/imagenet_subsets/val"
 },
 "eval_label_path": {
 "required": false,
 "value": ""
 },
 "customized": {
 "description": "if using customized distillation, please make sure
num_classes as the same as this section",
 "num_classes": {
 "type": "int",
 "required": true,
 "path": 50
 }
 }
 },
 "use_distillation": false,
 "use_default_distillation_config": true,
 "custom_distillation_config": {
 "teacher_model": {
 "description": "if using customized dataset, please make sure num_classes
as the same as this section",
 "required": true,
 "name": "efficientnet_b4",
 "file_path":
"/data/jenkins_common_ai/teacher_model/efficientnet_b4_best.pth.tar",
 "num_classes": 50
 }
 },
 "finetune_teacher_model_batch_size": 64,
 "xgen_mode_options": {
 "description": "different platforms support different xgen mode, 'simulation'
platform supports all models",
 "android_support": [
 "compatible_testing",
 "scaling",
 "pas",
 "customization"
],
 "ios_support": [
 "compatible_testing",
 "scaling",
 "pas",
 "customization"
]
 }
},

```

```

"xgen_mode_recommended": "scaling",
"xgen_mode": "scaling",
"xgen_gpus": "0",
"xgen_learning_rate": 0.048,
"xgen_batch_size": 192,
"xgen_epochs": 1,
"pas_finetune_epoch": 200,
"primary_interest_options": [
 "latency",
 "quality_score"
],
"primary_interest": "latency",
"latency": "21ms",
"quality_score": ">=73",
"resume_workplace_path": "",
"custom_workplace_path": "",
"pretrained_model_path": "",
"express_path_config": {
 "express_mode_description": "In the case of using default datasets and custom
datasets, different operation modes support different express path optimization
methods. The specific options are shown in the \"express_mode_options\" field.",
 "express_mode_options": {
 "compatible_testing": {
 "default_dataset": {
 },
 "customized_dataset": {
 }
 },
 "pruning": {
 "default_dataset": {
 "Express Optimization (Quickest, likely no training needed)": 0,
 "Extensive Optimization (Taking several rounds of training time)": 2
 },
 "customized_dataset": {
 "Fine Optimization (Taking about one round of training time)": 1,
 "Extensive Optimization (Taking several rounds of training time)": 2
 }
 },
 "scaling": {
 "default_dataset": {
 "Express Optimization (Quickest, likely no training needed)": 0,
 "Extensive Optimization (Taking several rounds of training time)": 2
 },
 "customized_dataset": {
 "Fine Optimization (Taking about one round of training time)": 1,
 "Extensive Optimization (Taking several rounds of training time)": 2
 }
 },
 "pas": {
 "default_dataset": {
 "Express Optimization (Quickest, likely no training needed)": 0,
 "Extensive Optimization (Taking several rounds of training time)": 2
 },

```

```

 "customized_dataset": {
 "Fine Optimization (Taking about one round of training time)": 1,
 "Extensive Optimization (Taking several rounds of training time)": 2
 },
 "customization": {
 "default_dataset": {
 "Express Optimization (Quickest, likely no training needed)": 0,
 "Extensive Optimization (Taking several rounds of training time)": 2
 },
 "customized_dataset": {
 "Fine Optimization (Taking about one round of training time)": 1,
 "Extensive Optimization (Taking several rounds of training time)": 2
 }
 },
 "express_mode": 0
 }
}

```

Following is an example config file for running XGen on custom models (Custom AI).

```

{
 "description": "this is a sample of customized ai config file in non-interactive mode",
 "device_platform_options": [
 "android",
 "ios",
 "simulation"
],
 "device_platform": "simulation",
 "devices": [],
 "xgen_config_file": "/path/to/xgen.json",
 "training_script_folder": "/path/to/training/script/folder",
 "xgen_mode_options": {
 "description": "different platforms support different xgen mode, 'simulation' platform supports all models",
 "android_support": [
 "compatible_testing",
 "pruning",
 "scaling",
 "pas",
 "customization"
],
 "ios_support": [
 "compatible_testing",
 "scaling",
 "pas",
 "customization"
]
 },
 "xgen_mode": "scaling",
}

```

```

"xgen_gpus": "0,1",
"xgen_learning_rate": 0.01,
"xgen_batch_size": 128,
"xgen_epochs": 3,
"pas_finetune_epoch": 200,
"primary_interest_options": [
 "latency",
 "quality_score"
],
"primary_interest": "latency",
"latency": "30ms",
"quality_score": ">=85",
"resume_workplace_path": "",
"custom_workplace_path": "",
"pretrained_model_path": "",
"use_distillation": false,
"use_default_distillation_config": false,
"custom_distillation_config": {
 "teacher_model": {
 "description": "if using customized dataset, please make sure num_classes
as the same as this section",
 "required": true,
 "name": "EfficientNet_ImageNet",
 "file_path": "/data/jenkins_common_ai/data/teacher_model/efficientnet-
b0.pth",
 "num_classes": 50
 }
},
"finetune_teacher_model_batch_size": 16,
"express_path_config": {
 "express_mode_description": "In the case of using default datasets and custom
datasets, different operation modes support different express path optimization
methods. The specific options are shown in the \"express_mode_options\" field.",
 "express_mode_options": {
 "compatible_testing": {
 "default_dataset": {
 },
 "customized_dataset": {
 }
 },
 "pruning": {
 "default_dataset": {
 },
 "customized_dataset": {
 }
 },
 "scaling": {
 "default_dataset": {
 },
 "customized_dataset": {
 }
 }
 },
 "pas": {

```

```

 "default_dataset": {
 },
 "customized_dataset": {
 }
 },
 "customization": {
 "default_dataset": {
 },
 "customized_dataset": {
 }
 }
},
"express_mode": 0
}

```

### Parameter description for task-description file

- `description`

Description of the task, which is used to describe the task in the non-interactive mode.

- `device_platform_options`

It is a list of platforms that the user could choose from when filling the field `device_platform`. There are currently three options: `android`, `ios`, `simulation`.

- `device_platform`

Users can pick one platform from the list in `device_platform_options` for this field.

- `devices`

A list of IDs of the device that the user wants to use. The user can run command `xgen_devices` to see the ids of all the connected devices. If `Simulation` is chose and the platform, this field will be ignored.

- `xgen_config_file`

The path of the xgen.json file. Its content is about the specifications on the model and dataset to use, as in the interactive mode.

- `training_script_folder`

The path of the training script folder. The training script in the folder should be named as `train_script_main.py`.

- `xgen_model_options`

A list of AI models that the user may choose from to fill the next field `select_model`. This list depends on the `device_platform` field as some models may work well on only some platforms. The user can refer to the sub-fields `android_support` and `ios_support` to find the supported models.

- `select_model`

Pick one AI model in the list of `xgen_model_options`.

- `use_default_dataset`

If the user wants to use the default dataset for the selected model, set it to `true`, otherwise, set it to `false`, and config the `custom_dataset_config` with the information of the custom dataset.

- `custom_dataset_config`

If the user wants to use custom dataset, set the field `use_default_dataset` to `false`, and set the datasets detail information in this field of the config file (a sample is in `/root/noninteractive_config_sample` folder).

There are 5 fields in the sub-section, `train_data_path`, `train_label_path`, `eval_data_path`, `eval_label_path`, `customized`. The users can set their own dataset paths in these fields, the `required` field is a tip. `customized` is a field for customized parameters, the user can set their own parameters in this field. but need to notice, the `type` field must be set, and the `path` field must be set to the value of the parameter.

- `use_distillation`

If the user wants to use distillation, set it to `true`, otherwise, set it to `false`. (So far, only the `EfficientNet_ImageNet`, `ResNet_ImageNet`, `MobileNet_ImageNet` models support distillation.)

- `use_default_distillation_config`

If the user wants to use the default distillation config, set it to `true`, otherwise, set it to `false` and config `custom_distillation_config` with the information for the custom distillation.

- `custom_distillation_config`

If the users want to use custom distillation, they need to set the field `use_default_distillation_config` to `false`, and set the distillation detail information (teacher model details) in this field (sample config files are in `/root/noninteractive_config_sample` folder).

- `xgen_mode_options`



This is a list of XGen modes that the user can choose from to fill the field `xgen_mode`. It depends on the `device_platform` field. The user can refer to the sub-fields `android_support` and `ios_support` to find the modes that can work on the platform selected in the `device_platform` field.

- `xgen_mode_recommended`

It is the recommended mode to use for AI model compression ( `pruning` or `scaling` ). Note, this field is meaningful only for the Android platform because at this point only `scaling` mode applies to iOS devices.

- `xgen_mode`

Pick one xgen mode from `xgen_mode_options` to use in this run.

- `xgen_gpus`

It is a list of GPUs to use. Running `nvidia-smi -L` can list all the GPUs on your machine. The above example is to use the first 8 GPUs.

- `xgen_batch_size`

It is the per-GPU batch size to use. It is an integer.

- `xgen_learning_rate`

It is the learning rate to use at the beginning of training. It is a floating-point number.

- `xgen_epochs`

It is the number of epochs to use in training. It is an integer.

- `primary_interest_options`

It is a list of interests a user can choose from to fill in `primary_interest`. For now, XGen supports `latency` and `quality_score` only.

- `primary_interest`

Pick one primary interest from `primary_interest_options`.

- `latency`

It is the target latency. It is a string (e.g., "30ms").

- `quality_score`

It is the quality score that you want to use. It is a string (e.g., ">=85").

- `resume_workplace_path`

If the user wants to resume the previous work, the user needs to set the path of the previous work in this field. Leave it empty otherwise. If `resume_workplace_path` is filled, XGen will check whether the `select_model`, `xgen_mode` and other parameters are the same as the previous work; if not, it will raise an error.

- `custom_workplace_path`

It is the workplace path to use. It is a string (e.g., "/path/to/output", if the path does not exist, XGen will create it). As in the interactive mode, it is optional; XGen will create a unique folder and inform the user on the screen if it is not specified. (If `resume_workplace_path` is filled, `custom_workplace_path` will be ignored.). If the path contains train data, it will raise an error.

- `pretrained_model_path`

It is the pretrained model path to use. It is a string (e.g., "/path/to/pretrained\_model\_path"). As in the interactive mode, it is optional.

- `finetune_teacher_model_batch_size`

It is the batch size to use in finetuning the teacher model. It is an integer.

- `express_mode_options`

It is a list of express modes that the user can choose from to fill the field `express_mode`. It depends on the `device_platform` field. The user can refer to the sub-fields `android_support` and `ios_support` to find the modes that can work on the platform selected in the `device_platform` field.

- `express_mode`

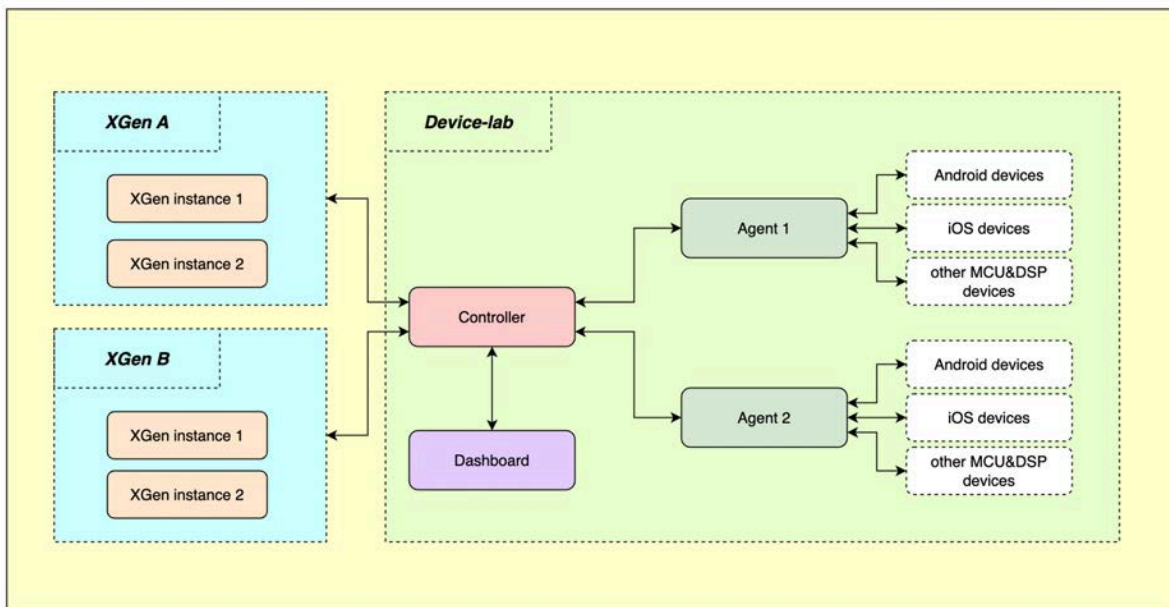
Pick one express mode from `express_mode_options` to use in this run.

## Device Lab Introduction

The Device Lab avoids the need to connect mobile devices directly to the training machine. In many scenarios, the training machine is located in a server room or data center, making it impractical to connect physical mobile devices. Now, with the ability of Device Lab, you can keep your devices local without plugging them in the training machines.

## Overview

### Device Lab architecture



As shown in the above diagram, the current version of XGen consists of two parts: the XGen client and the Device Lab module. The Device Lab module further includes two components: a Controller and one or more agents.

**Device Lab Controller** is the core of the device lab. The primary responsibility of the controller is to establish connections between XGen and the Agent, forwarding and logging messages between XGen and Agent. If these three components are not within the same network environment, the controller needs to expose its public IP address to the outside, in order for XGen and Agent to locate and connect to it.

Alongside the controller, there is also a **Dashboard** installed. The purpose of the **Dashboard** is to track tasks forwarded by the controller and to handle a portion of the configuration work related to the connections between XGen and the agent. In short, **Dashboard** is a web application that provides a user interface for monitoring devices and tasks.

**Device Lab Agent** is a daemon running on a host machine which can connect with many testing devices, like your phones. It is responsible for communicating with the controller and executing tasks.

**XGen Client** is a tool named `xgend` (serving as a service within the XGen container) that is utilized to interact with the controller. It is used to submit tasks, check task status, and download task results.

## Installation

Please follow the steps below to install Device Lab. First install the controller and dashboard, then install the agent, and finally install the XGen client. For more details, please refer to [Device Lab Installation](#).

## Usage

About XGen usage, please refer to [XGen Usage](#). The following content is about Device Lab usage.

### Controller Usage

#### Controller service

After installation, the controller service will be started automatically. You can also use the following command to start the controller and enter controller container. Controller container is a docker container which runs controller service.

```
$ run_controller
```

In most cases, you do not need to run `run_controller` command manually. if the 'xgen\_controller' container already exists, running the `run_controller` command will enter the corresponding container.

You can use the following command to check controller service status:

```
$ xgenctl status
```

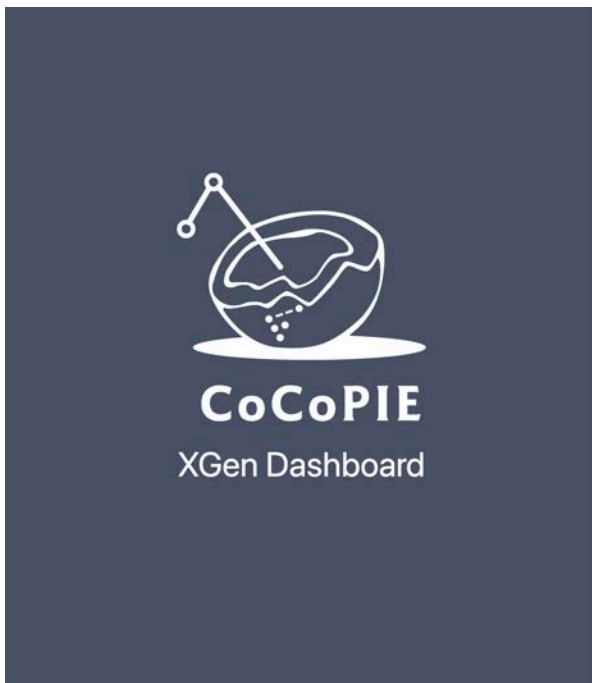
You can use the following command to exit controller container, but it will not stop the controller container:

```
$ exit
```

## Dashboard Usage

After Device Lab Controller installation, you can visit the web page of dashboard by `http://<controller-ip>` in your browser.

### Login page



**Sign in**

Username

Password  
 

At the first time you visit the dashboard, you need to change the default password. The default username is `admin`, and the default password is `admin`. The web page will redirect to the change password page automatically.

After changing password, it will return to the home page.

In the top right corner of the home page, there are 3 buttons, it indicates that the web system includes 3 modules, `Dashboard`, `Config`, `Admin`. You can click the `Dashboard` button to go back to the home page. Additionally, you can click the `Config` button to access the configuration page, where you can configure controller public IP and the agent. Furthermore, the Admin button allows you to access three sub-pages: the document page, the password change page, and the system logout. Here is the structure:

- Dashboard Page Root
  - Dashboard
    - Devices
    - Tasks
    - XGen
  - Config
    - Controller IP
    - Agent
  - Admin (logo icon)
    - Document
    - Change Password
    - Logout

## Devices page

The screenshot displays the XGen Dashboard interface. On the left is a sidebar with navigation links: 'Devices' (highlighted), 'Tasks', and 'XGen'. The top right corner shows navigation links: 'Dashboard', 'Config', and 'Admin'. The main content area is divided into two sections based on agents.

**Agent: cd-3060(agen-15308)**

Device	Status	UUID	Model	CPU	GPU	Memory	Battery
Redmi	online	5fcbiduljkgesg	Redmi Note 8 Pro	MT6785VCC	ARM, Mali-G76 MC4	5.37 GB	100%
Xiaomi	online	93fdaa69	Mi 8	SM845	Qualcomm, Adreno (TM) 630	5.5 GB	100%
OPPO	online	W8EA7LEAFClUBN	PACM00	MT6771VJC	ARM, Mali-G72 MP3	5.53 GB	100%
Apple	online	00008020-001A29993CD2002E	iPhone X5 15.1				100%

**Agent: cd-dell(agen-2876)**

Device	Status	UUID	Model	CPU	GPU	Memory	Battery
vivo	online	31316122330025C	V2072A	MT6891Z/CZA	ARM, Mali-G77 MC9	7.36 GB	100%
Xiaomi	online	6c74106e	Mi 9	SD855	Qualcomm, Adreno (TM) 640	5.37 GB	100%
	offline	AKRSVB1630018730					%
Apple	online	000080101-0009245134F9001E	iPhone 12 16.3.1				100%

On the home page (the default home page is the Devices page), you can view a list of devices grouped by agent. The status of each device is indicated by the color of its card. Devices can have

the status of online (displayed as a light green card) or offline (displayed as a light gray card). The device status is updated automatically.

In each device card, you can find details such as the device name, device model, device OS version, and more.

## Tasks page

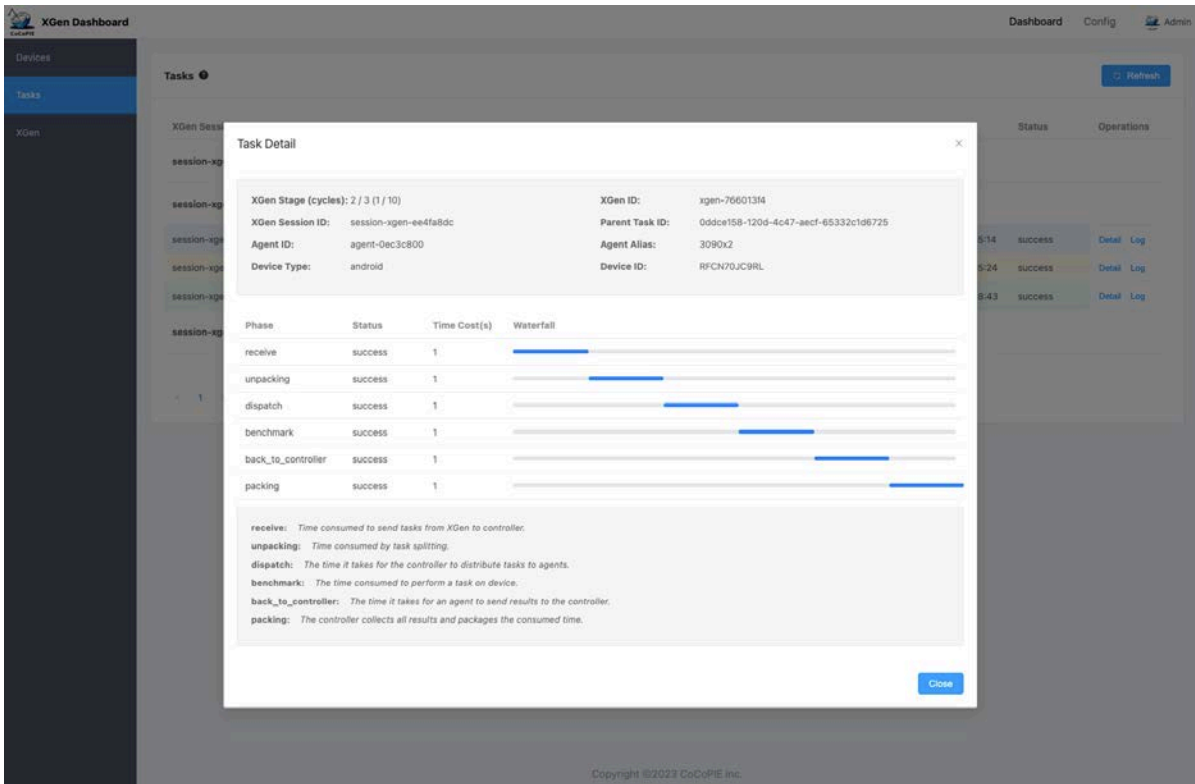
XGen Session ID	XGen HOST ID	XGen ID	Stage (cycles)	Device ID	Agent	Launch Time	Status	Operations
session-xgen-567c59ba		xgen-cef9f25b						
session-xgen-567c59ba	xgen-host-8cb874ac	xgen-cef9f25b	0 / 4 (1 / 1)	6c74106e	cd-agent	2023-08-10 16:02:00	success	<a href="#">Detail</a> <a href="#">Log</a>
session-xgen-567c59ba	xgen-host-8cb874ac	xgen-cef9f25b	0 / 4 (1 / 1)	6c74106e	cd-agent	2023-08-10 16:02:38	success	<a href="#">Detail</a> <a href="#">Log</a>
session-xgen-567c59ba	xgen-host-8cb874ac	xgen-cef9f25b	0 / 4 (1 / 1)	6c74106e	cd-agent	2023-08-10 16:03:20	success	<a href="#">Detail</a> <a href="#">Log</a>
session-xgen-567c59ba	xgen-host-8cb874ac	xgen-cef9f25b	1 / 4 (1 / 1)	6c74106e	cd-agent	2023-08-10 16:04:01	success	<a href="#">Detail</a> <a href="#">Log</a>
session-xgen-567c59ba	xgen-host-8cb874ac	xgen-cef9f25b	2 / 4 (1 / 20)	6c74106e	cd-agent	2023-08-10 16:04:45	success	<a href="#">Detail</a> <a href="#">Log</a>
session-xgen-567c59ba	xgen-host-8cb874ac	xgen-cef9f25b	2 / 4 (2 / 20)	6c74106e	cd-agent	2023-08-10 16:05:26	success	<a href="#">Detail</a> <a href="#">Log</a>
session-xgen-567c59ba	xgen-host-8cb874ac	xgen-cef9f25b	2 / 4 (3 / 20)	6c74106e	cd-agent	2023-08-10 16:06:08	success	<a href="#">Detail</a> <a href="#">Log</a>
session-xgen-567c59ba	xgen-host-8cb874ac	xgen-cef9f25b	4 / 4 (1 / 1)	6c74106e	cd-agent	2023-08-10 16:07:27	success	<a href="#">Detail</a> <a href="#">Log</a>
session-xgen-1b12f940		xgen-cef9f25b						
session-xgen-dca05dc6		xgen-cef9f25b						
session-xgen-18de073d		xgen-cef9f25b						
session-xgen-cd00c3c1		xgen-cef9f25b						

In the tasks page, you can see the task list, grouped by xgen-session. Different stages of the task are shown by the color of the task rows.

An XGen session refers to one-time execution of XGen or one-time execution of onnx\_latency\_benchmark.

For more information about a task, you can click the detail button to see the task details. and also you can click the Log button to see the task's runtime log.

Task Details:



Each task consists of multiple phases, and each phase requires a certain amount of time to complete.

On the task details page, you can access detailed information about the task, including its status, task result, and the duration of each phase. This allows you to identify any bottlenecks in the task. Additionally, a description is provided for each phase at the bottom of the page.

XGen page



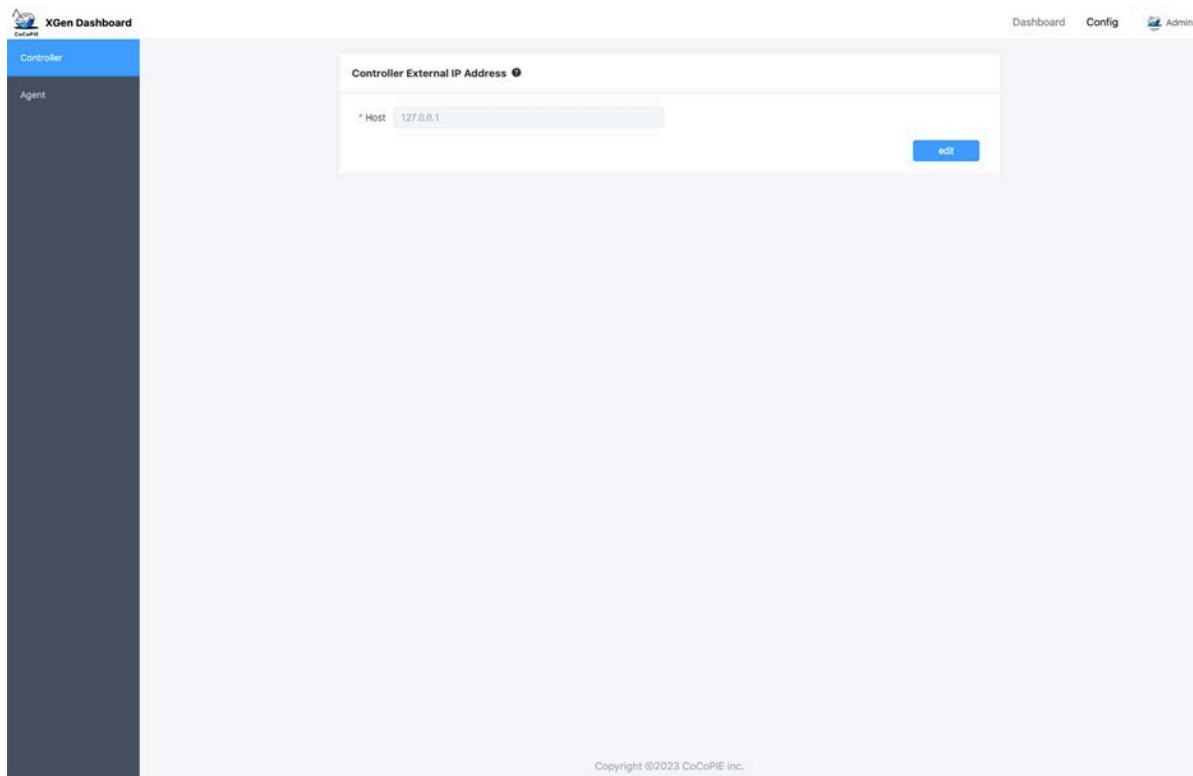
In the XGen page, all connected XGen instances are listed here, the status and some statistic information are show in the list.

Config Controller IP



If your agent, XGen, and controller are not within the same local network, then you need to configure the public IP of the controller so that the agent and XGen can connect to the controller via the internet. This is where you configure the external access IP for the controller.

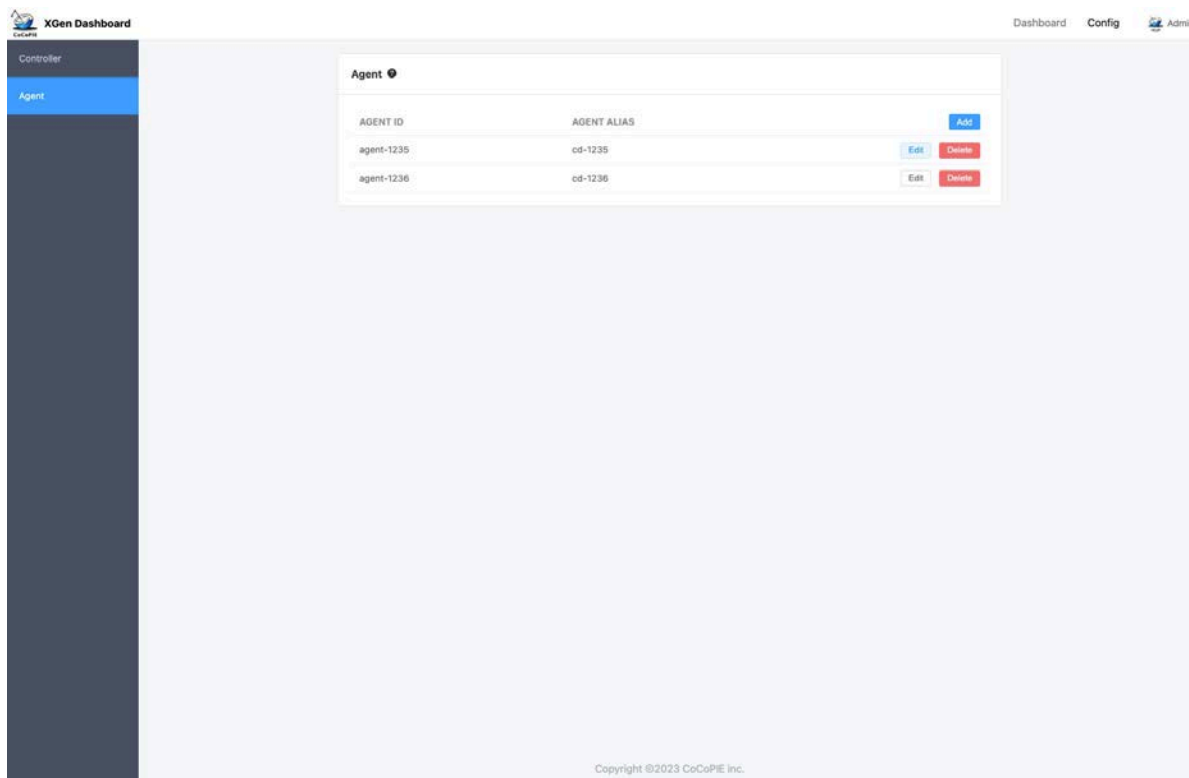
You can configure Controller public IP in the config part of dashboard by clicking on the `Config` button in the top right corner of the page.



By clicking `edit`, you can update the configuration. Please note that **update Controller IP configuration will cause all agents to disconnect and all tasks to stop.**

### Config Agent Info

Adding agent here is intended to control the connection of legitimate agents. You can configure the agent in the config page. You can `add` or `edit` agent configuration. This page controls which agents can connect to the controller. Please note that **update agent configuration may cause agents to disconnect and stop current tasks.**



If you meet any problem, please refer to [Troubleshooting](#) for help.

### Reset dashboard web page password

Enter into the dashboard container by running command:

```
$ docker exec -it xgen_dashoard bash
```

and in the dashboard container, run the following command to reset the password:

```
$ reset-web-passwd -p <new_password>
```

## Agent Usage

### Check devices

`xgen_devices` is a powerful command to check devices.

```
$ xgen_devices --help
```

```
Usage: xgen_devices [OPTIONS] [COMMAND [ARGS]]...
```

```
Tool for XGen Device Lab Agent
```

```
Options:
 --help Show this message and exit.

Commands:
 list List all devices
 update Update device info.
```

To list all devices, just simply run `xgen_devices list`:

```
$ xgen_devices list
```

Platform	UUID	Brand	Model	Task Status	Connection Status
Android	eb24e4c4	Xiaomi	M2011K2C	idle	available

To see more information about devices

```
$ xgen_devices list -v
```

Platform	UUID	Brand	Model	CPU	GPU	Memory	Battery %	Task Status	Task Num	Connection Status
Android	eb24e4c4	Xiaomi	M2011K2C	Venus based on Qualcomm Technologies, Inc SM8350	Qualcomm, Adreno (TM) 660	7.13 GB	99	idle		available

To see devices without network connection

```
$ xgen_devices list -nm
```

By default, Agent will try to fetch device information by all available methods. However, missing some information is normal. Users can update device information by `xgen_devices update`:

```
xgen_devices update --help

Usage: xgen_devices update [OPTIONS]

Update device info. Update device info may cause unexpected error, Please
Use With Caution.

Options:
 -d, --device TEXT Device ID of the device you want to update
 [required]
 -f, --field [gpu|cpu|brand|model]
```

<code>-v, --value TEXT</code>	Field you want to update [required] Value you want to update. ( No more than 64 characters ) [required]
<code>--help</code>	Show this message and exit.

## Update agent configuration

If you need to update the agent configuration, the user may edit the configuration in `/root/.xgen_agent/agent.conf` and restart the agent by running command `xgenctl restart`.

## Daemon services

The agent runs as a daemon service. After updating the corresponding configuration, you need to restart the service to make the configuration take effect. You can use the following command to restart the service:

```
xgenctl restart
```

You can use the following command to check agent service status:

```
$ xgenctl status
```

You can use the following command to exit agent container, but it will not stop the agent container:

```
$ exit
```

If you meet any problem, please refer to [Troubleshooting](#) for help.

## Troubleshooting

As Device Lab is a distributed system and many daemons are running in the background, problems may happen. Logs are useful. Please check the logs of controller, dashboard, and agent.

---

Controller logs are located in `/root/.xgen_controller/logs`.

Controller log structure:

```
controllerd device pack_monitor queue tasks unpack xgenctl
```

- `controllerd` directory records the logs of controllerd service, which is the main controller daemon and responsible for doing initialization and starting other daemons.

- `device` directory records the logs of the daemon which is responsible for transferring devices information from agents to all connected XGen instances.
  - `pack_monitor` directory records the logs of the daemon which is responsible for packing all benchmark results from different agents and send them to the corresponding XGen instance.
  - `queue` directory records the logs of the daemon which is responsible for managing queue creating and deleting.
  - `tasks` directory records logs of all benchmark tasks.
  - `unpack` directory records the logs of the daemon which is responsible for unpacking benchmark requests for different devices from XGen instances.
  - `xgenctl` directory records the logs of the daemon which is responsible for communicating with other daemons.
- 

Dashboard logs are located in `/root/.xgen_dashboard/logs`.

Dashboard log structure:

```
service xgenctl
```

- `service` directory records the logs of the daemon which is responsible for offering web service and API services.
  - `xgenctl` directory records the logs of the daemon which is responsible for communicating with other daemons.
- 

Agent logs are located in `/root/.xgen_agent/logs`.

Agent log structure:

```
xgen_devices agent_status agentd task_manager xgenctl
```

- `xgen_devices` directory records the logs of the daemon which is responsible for checking devices status in real-time.
- `agent_status` directory records the logs of the daemon which is responsible for sending agent devices status to controller.
- `agentd` directory records the logs of the daemon which is responsible for doing initialization and starting other daemons.

- `task_manager` directory records the logs of the daemon which is responsible for managing benchmark tasks.
- `xgenctl` directory records the logs of the daemon which is responsible for communicating with other daemons.

## Health Check

For controller, dashboard and agent, just simply run `xgenctl status` for a health check.

If the controller is running normally, the output will be like this:

```
Settings check passed.
Redis check passed.
MQ check passed.
Supervisor services check passed.
```

If the dashboard is running normally, the output will be like this:

```
Redis check passed.
Supervisor services check passed.
```

If the agent is running normally, the output will be like this:

```
Settings check passed.
Redis check passed.
MQ check passed.
Supervisor services check passed.
```

If there are any problems reported by the health check, please follow the presented instructions to fix the problem.

## Chapter 4-3. Usage of XGen Web UI

XGen now is shipped with a new interface in Web UI. For quick start the webui, please refer to [Quick Start](#). Basically, just simply run command below in your XGen container:

```
XGen webserver

or specify a port

XGen webserver --port 8080
```

By default, XGen will randomly choose free port from 19001 to 19200 . Please ensure the port is not occupied by other process and accessible from your browser.

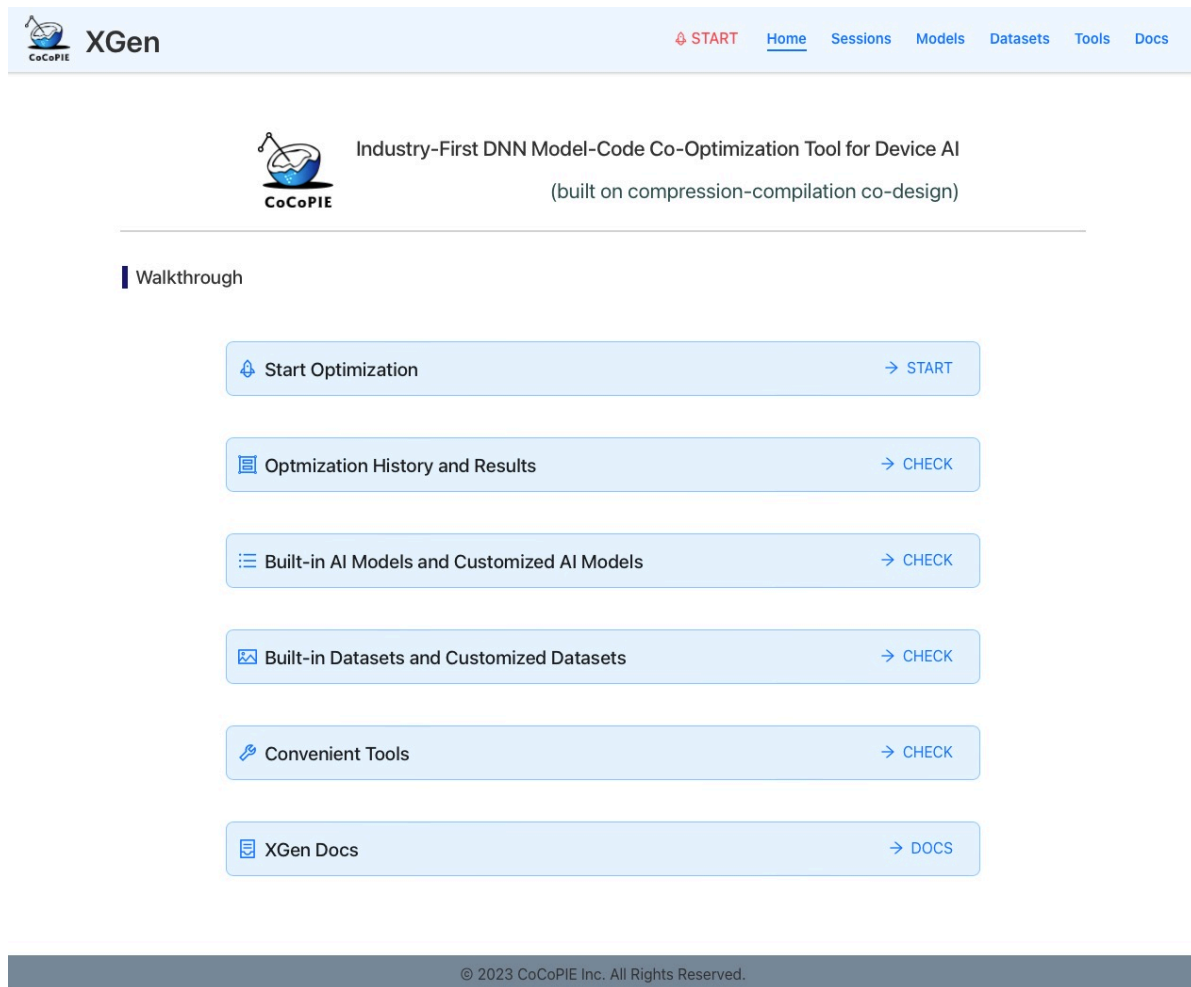
If logged with ssh shell, please remember to add `-L` option to forward the port to local machine. Because the webui is running in container, it is not accessible from outside of the container. For example, if you want to forward port 8080 to local machine, run command below:

```
ssh -L 8080:localhost:8080 user@host
the command format is
ssh -L <localport>:<remotehost>:<remoteport> user@host
```

Then open your browser and visit `http://localhost:8080` to access the webui.

Home

Home page offers shortcuts to navigate to other pages.



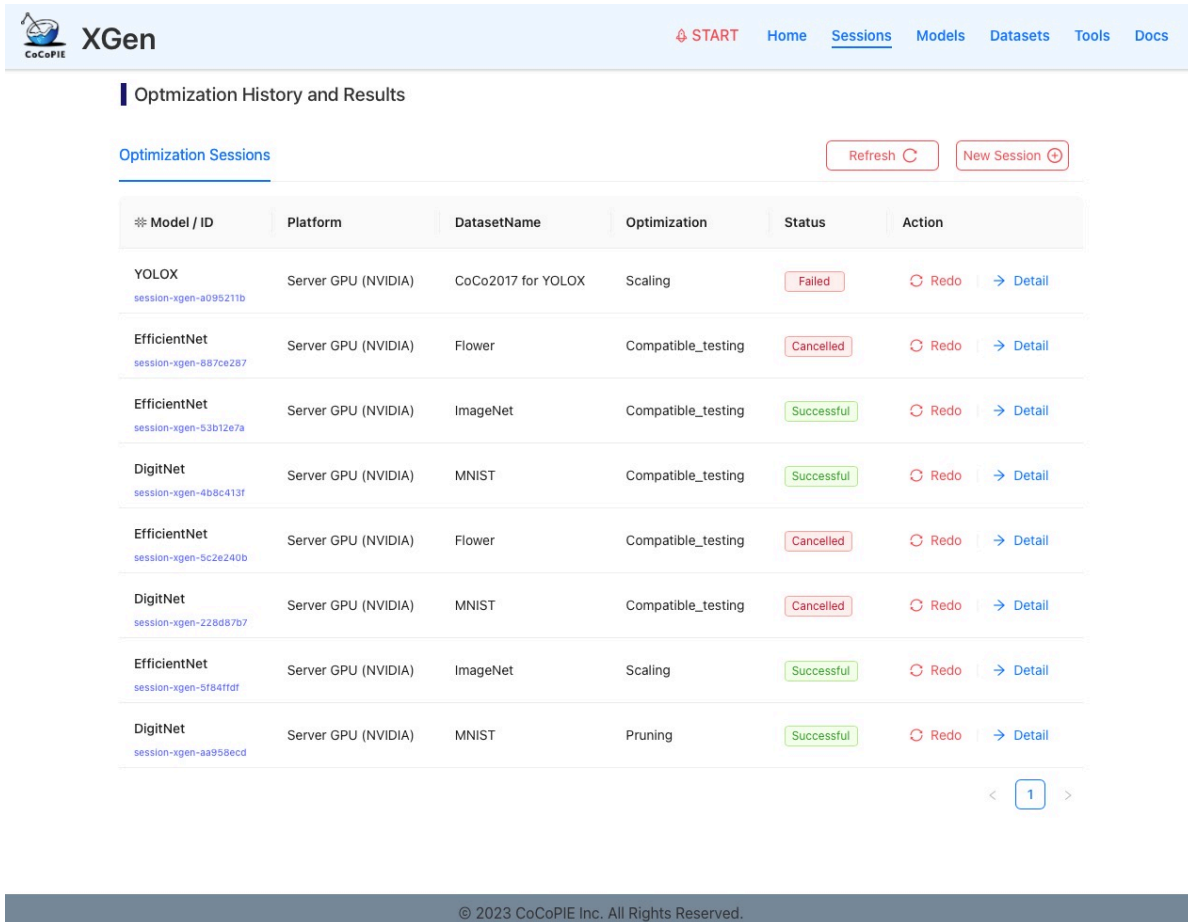
- Start Optimization: You can select an AI model and click the “Start Optimization” button to begin the optimization process.
- Optimization History and Results: View previous optimization history and the results of the optimization.
- Built-in AI Models and Customized AI Models: View the built-in models provided by XGen. You can select a model and click the “Download” button to download the model’s code maintained by XGen. You can also upload your own model and use it for optimization.
- Built-in Datasets and Customized Datasets: View the built-in datasets provided by XGen. You can also upload your own dataset for optimization.
- Convenient Tools: Some convenient tools provided by XGen, such as the benchmark tool. We will include more convenient tools in the feature.
- XGen Docs: Link to the XGen documentation.

Click `check` button to go to corresponding page.



## Sessions

Go to this page by clicking **Optimization History and Results** button in home page or **Sessions** button in the navigation bar on top of the site. This page shows the history of the optimization process. You can see the status of each optimization process. If the optimization process is finished, you can see the final result of the process.



**XGen** START Home Sessions Models Datasets Tools Docs

**Optimization History and Results**

[Optimization Sessions](#) Refresh New Session

Model / ID	Platform	DatasetName	Optimization	Status	Action
YOLOX <a href="#">session-xgen-a095211b</a>	Server GPU (NVIDIA)	CoCo2017 for YOLOX	Scaling	Failed	<span>Redo</span> <span>Detail</span>
EfficientNet <a href="#">session-xgen-687ce287</a>	Server GPU (NVIDIA)	Flower	Compatible_testing	Cancelled	<span>Redo</span> <span>Detail</span>
EfficientNet <a href="#">session-xgen-53b12e7a</a>	Server GPU (NVIDIA)	ImageNet	Compatible_testing	Successful	<span>Redo</span> <span>Detail</span>
DigitNet <a href="#">session-xgen-4b8c413f</a>	Server GPU (NVIDIA)	MNIST	Compatible_testing	Successful	<span>Redo</span> <span>Detail</span>
EfficientNet <a href="#">session-xgen-5c2e240b</a>	Server GPU (NVIDIA)	Flower	Compatible_testing	Cancelled	<span>Redo</span> <span>Detail</span>
DigitNet <a href="#">session-xgen-228d87b7</a>	Server GPU (NVIDIA)	MNIST	Compatible_testing	Cancelled	<span>Redo</span> <span>Detail</span>
EfficientNet <a href="#">session-xgen-5f84fddf</a>	Server GPU (NVIDIA)	ImageNet	Scaling	Successful	<span>Redo</span> <span>Detail</span>
DigitNet <a href="#">session-xgen-aa958ecd</a>	Server GPU (NVIDIA)	MNIST	Pruning	Successful	<span>Redo</span> <span>Detail</span>

< 1 >

© 2023 CoCoPIE Inc. All Rights Reserved.

There are several status of each optimization.

- Pending** : The optimization process is waiting for resource to start.
- Successful** : The optimization process is finished successfully.
- Training** : The optimization process is running.
- Failed** : The optimization process is failed.

You can click the link under each **Model / ID** then will be navigated to detail page of this optimization.

## Optimization Page

Go to this page by clicking `Start Optimization` button in home page or `Start` button in the navigation bar on top of the site. This page is used to start a new optimization process.



[START](#)
[Home](#)
[Sessions](#)
[Models](#)
[Datasets](#)
[Tools](#)
[Docs](#)

[←](#) Sessions

Which platform device do you want? ⓘ

☐ Mobile GPU/CPU (Android)  
☐ Mobile GPU/CPU (iOS)  
☒ Server GPU (NVIDIA)  
☐ Server GPU (AMD)  
☐ Server CPU (Intel, AMD)  
☐ Embedded CPU (ARM MCU, Raspberry Pi)

Please choose the base model to start with

Image Classification 0: DigitNet

Choose the dataset

☒ MNIST (11.06 MB Training this dataset for 10 epochs using a single GTX 3090 would take approximately 10 minutes.)  
☐ Choose Your Dataset  
☐ Upload Your Dataset Now

Run Compatibility test or Choose optimizations method

Compatibility test

☒ Compatibility test  
 A quick check of whether the model and setup are compatible with XGen. Such a test is much quicker than optimizations (pruning or scaling).

Optimization Methods

☐ Pruning (Recommended ⓘ)  
 One of the ways that XGen uses to compress a model and generate efficient code. Pruning is suitable for networks with a significant number of 3x3 convolutions, as well as situations where 3D convolutions are present in the network.

☐ Scaling  
 One of the ways XGen uses to compress a model, which scales down (or up) the width of DNN layers. Scaling is a general method.

☐ PaS  
 One of the methods employed by XGen to compress a model involves reducing the width of DNN layers. For a detailed explanation, You can refer to the paper at <https://arxiv.org/abs/2206.01198>

☐ Customized operation  
 Users can change the XGen configuration of a model to enable special operations. (path: /root/Projects/[model\_name\_folder]/[model\_name]\_config/xgen.json)  
 More detail usage, please visit: [XGen Usage](#)

Please choose your GPU(s)

☒ GPU 0: NVIDIA GeForce RTX 3090 (UUID: GPU-5a90993f-966d-31af-980e-b8a7d23ba8f9)

What is the batch size per GPU:

128

[Back](#)
[Reset](#)
[Submit](#)

© 2023 CoCoPIE Inc. All Rights Reserved.

This page is a large dynamic form. Answer of each question will take effect on questions below. Some of the questions are marked with an 'i' icon, which means there is a tooltip for this question.

Hover on the icon to see the tooltip.

 **XGen**

[START](#) [Home](#) [Sessions](#) [Models](#) [Datasets](#) [Tools](#) [Docs](#)

Run Compatibility test or Choose optimizations method

Compatibility test

ⓘ Why is Pruning Recommended

The numerous 3x3 convolution kernels make this model a good candidate for sparsification and acceleration through pruning.

OK

☒ Pruning (Recommended ⓘ)

One of the ways that XGen uses to compress a model and generate efficient code. Pruning is suitable for networks with a significant number of 3x3 convolutions, as well as situations where 3D convolutions are present in the network.

☐ Scaling

One of the ways XGen uses to compress a model, which scales down (or up) the width of DNN layers; Scaling is a general method.

☐ PaS

One of the methods employed by XGen to compress a model involves reducing the width of DNN layers. For a detailed explanation, You can refer to the paper at <https://arxiv.org/abs/2206.01198>

☐ Customized operation

Users can change the XGen configuration of a model to enable special operations. (path: /root/Projects/[model\_name\_folder]/[model\_name]\_config/xgen.json)  
More detail usage, please visit: [XGen Usage](#)

\* Please choose your GPU(s)

☐ GPU 0: NVIDIA GeForce RTX 3090 (UUID: GPU-5a90993f-966d-31af-980e-b8a7d23ba8f9)

\* What is the batch size per GPU:

128

⬆ ⬇ ⬆

Back

Reset

Submit

© 2023 CoCoPIE Inc. All Rights Reserved.

Just simply full fill the form as needed. Below is an example

[https://xgen.cocopie.ai/v1.4.1/12\\_XGen\\_WebUI\\_Usage/](https://xgen.cocopie.ai/v1.4.1/12_XGen_WebUI_Usage/)

6/21

## ← Sessions

### \* Which platform device do you want? ⓘ

- ☐ Mobile GPU/CPU (Android)
- ☐ Mobile GPU/CPU (iOS)
- ☒ Server GPU (NVIDIA)
- ☐ Server GPU (AMD)
- ☐ Server CPU (Intel, AMD)
- ☐ Embedded CPU (ARM MCU, Raspberry Pi)

### Please choose the base model to start with

Image Classification 0: DigitNet

### \* Choose the dataset

- ☒ MNIST (11.06 MB Training this dataset for 10 epochs using a single GTX 3090 would take approximately 10 minutes.)
- ☐ Choose Your Dataset
- ☐ Upload Your Dataset Now

### Run Compatibility test or Choose optimizations method

#### Compatibility test

- ☐ Compatibility test

A quick check of whether the model and setup are compatible with XGen. Such a test is much quicker than optimizations (pruning or scaling).

#### Optimization Methods

- ☒ Pruning (Recommended ⓘ)

One of the ways that XGen uses to compress a model and generate efficient code. Pruning is suitable for networks with a significant number of 3x3 convolutions, as well as situations where 3D convolutions are present in the network.

- ☐ Scaling

One of the ways XGen uses to compress a model, which scales down (or up) the width of DNN layers. Scaling is a general method.

- ☐ PaS

One of the methods employed by XGen to compress a model involves reducing the width of DNN layers. For a detailed explanation, You can refer to the paper at <https://arxiv.org/abs/2206.01198>

- ☐ Customized operation

Users can change the XGen configuration of a model to enable special operations. (path: /root/Projects/[model\_name\_folder]/[model\_name]\_config/xgen.json)  
More detail usage, please visit: [XGen Usage](#)

### \* Please choose your primary interest

Latency and accuracy form a tradeoff for XGen optimizations. XGen will try to meet the primary interest first, and then consider the other interest.

**NOTE: accuracy as the primary interest will take XGen several times longer to optimize than latency as the primary interest.**

- ☒ Latency ☐ Quality score

### \* Desired inference latency (e.g., 10ms, 0.5s):

Hint: You can find out the latency of the original model first for your reference. To do that, select 'Compatibility Test'; the latency will appear at the end of the test report.

0.05ms

### \* Desired quality score (e.g., accuracy):

MAX

### \* Please choose your GPU(s)

- ☒ GPU 0: NVIDIA GeForce RTX 3090 (UUID: GPU-5a90993f-966d-31af-980e-b8a7d23ba8f9)

### \* What is the batch size per GPU:

128

### \* What is the learning rate:

0.01

Number of training or fine-tuning epochs:

3

Pretrained model absolute path (Optional) ⓘ

Select Path

Back Reset Submit

Then click **Submit** button to submit the form. If the not using customized dataset or customized model and compatibility testing, a modal will show up to present the estimated time to complete the optimization. Click **OK** to start the optimization process. Or click **Cancel** to cancel the optimization process and update the form due to suggestions given in the modal.

The screenshot shows the XGen web interface with a configuration form and a modal dialog. The form includes fields for 'Desired quality', 'Please choose', 'GPU 0: NVIDIA', 'What is the', 'What is the learning rate:', 'Number of training or fine-tuning epochs:', 'Pretrained model absolute path (Optional)', and 'Would you like to measure quality score of the original model?'. The modal dialog, titled 'Estimated Training Time', provides information about the estimated running time and suggestions for optimization.

**Estimated Training Time**

Depending on the configuration you choose, the estimated running time is between 17 days, 20 hours and 22 days, 5 hours

Suggestions

- Submit pretrained\_model\_path, stage1 will not be trained
- Do not measure the quality score of the original model, which will not affect the final result in the scaling mode you choose.
- Use smaller data sets from the built-in

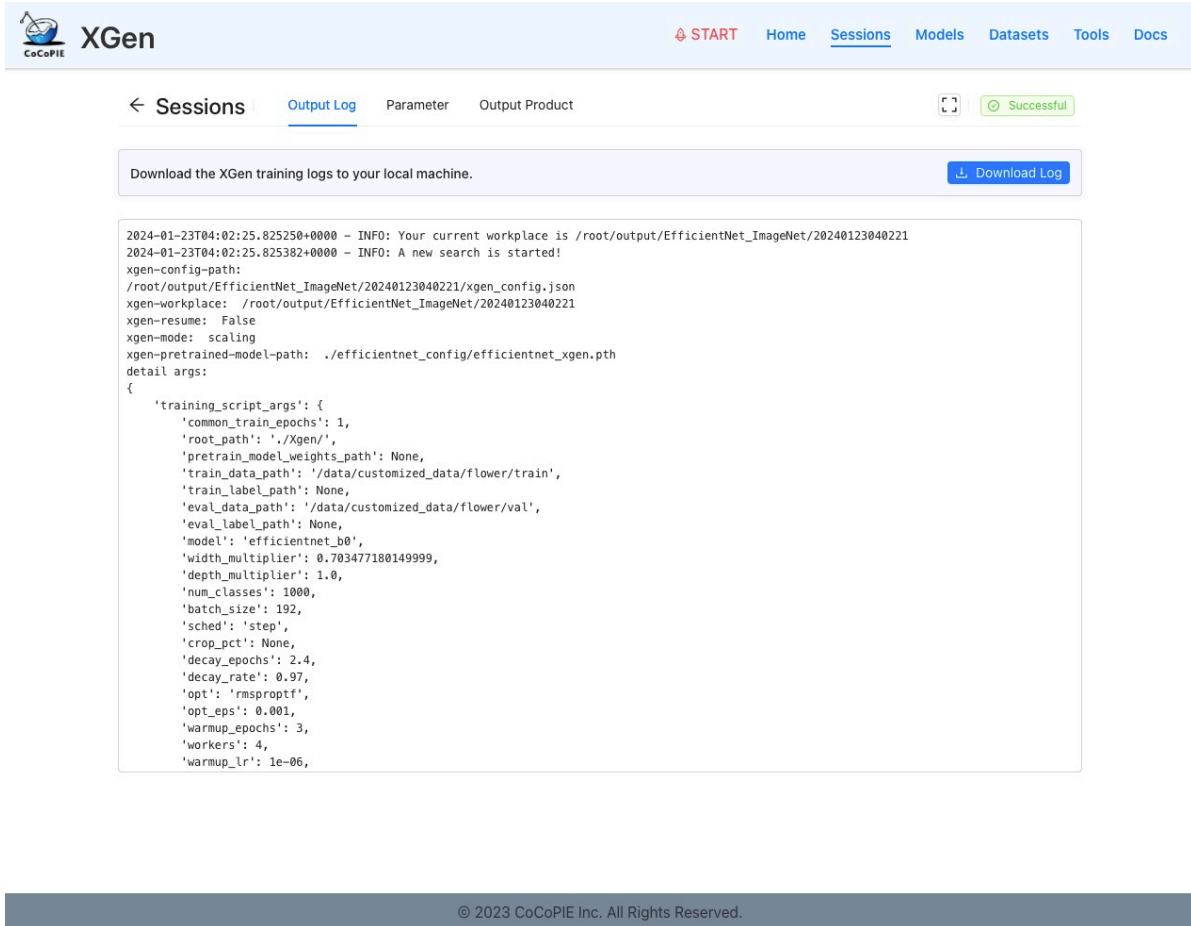
Cancel OK

© 2023 CoCoPIE Inc. All Rights Reserved.

After starting the optimization process, the optimization process will be running in background. You can close the browser or navigate to other pages. The process will not be interrupted. But please note that **please do not exit the XGen container while the optimization process is running**. If you exit the container, the process will be terminated.

## Optimization Detail Page

After clicking the **Submit** button in the optimization page, you will be redirected to the optimization detail page. This page shows the progress of the optimization process.



The screenshot displays the XGen WebUI interface. At the top, there is a navigation bar with links for **START**, **Home**, **Sessions** (selected), **Models**, **Datasets**, **Tools**, and **Docs**. Below the navigation bar, there are tabs for **Sessions**, **Output Log** (selected), **Parameter**, and **Output Product**. A green status indicator shows "Successful".

A button labeled "Download Log" is present, with the text "Download the XGen training logs to your local machine." above it. Below this, a log window displays the following information:

```

2024-01-23T04:02:25.825250+0000 - INFO: Your current workplace is /root/output/EfficientNet_ImageNet/20240123040221
2024-01-23T04:02:25.825382+0000 - INFO: A new search is started!
xgen-config-path:
/root/output/EfficientNet_ImageNet/20240123040221/xgen_config.json
xgen-workplace: /root/output/EfficientNet_ImageNet/20240123040221
xgen-resume: False
xgen-mode: scaling
xgen-pretrained-model-path: ./efficientnet_config/efficientnet_xgen.pth
detail args:
{
 'training_script_args': {
 'common_train_epochs': 1,
 'root_path': './Xgen/',
 'pretrain_model_weights_path': None,
 'train_data_path': '/data/customized_data/flower/train',
 'train_label_path': None,
 'eval_data_path': '/data/customized_data/flower/val',
 'eval_label_path': None,
 'model': 'efficientnet_b0',
 'width_multiplier': 0.703477180149999,
 'depth_multiplier': 1.0,
 'num_classes': 1000,
 'batch_size': 192,
 'sched': 'step',
 'crop_pct': None,
 'decay_epochs': 2.4,
 'decay_rate': 0.97,
 'opt': 'rmsproptf',
 'opt_eps': 0.001,
 'warmup_epochs': 3,
 'workers': 4,
 'warmup_lr': 1e-06,
 }
}

```

At the bottom of the page, a footer indicates: © 2023 CoCoPIE Inc. All Rights Reserved.

There are three tabs in the page. By default, the **Output Log** tab is selected. This tab shows the output log of the optimization process. You can see the progress of the optimization process here with automatically updated log. Here is a small square button on the right of the tabs. Click it will navigate to a full screen log page.



XGen

START

Home

Sessions

Models

Datasets

Tools

Docs

XGen Training Log

2024-01-23T04:06:39.583440+0000 - INFO: test: [ 22/22] Time: 0.010 (0.111) Loss: 0.9531 (0.9214) Acc@1: 0.0000 (0.0000) Acc@5: 0.0000 (0.0000)

2024-01-23T04:06:39.583440+0000 - INFO: eval top-1: 0.0

2024-01-23T04:06:40.850191+0000 - INFO: ===== Diagnostic Run torch.onnx.export version 2.0.1 =====

2024-01-23T04:06:40.850484+0000 - INFO: verbose: False, log level: Level.ERROR

2024-01-23T04:06:40.850583+0000 - INFO: ===== 0 NONE 0 NOTE 0 WARNING 0 ERROR =====

2024-01-23T04:06:40.850746+0000 - INFO: =====

2024-01-23T04:06:40.850864+0000 - INFO: ===== Diagnostic Run torch.onnx.export version 2.0.1 =====

2024-01-23T04:06:40.850962+0000 - INFO: verbose: False, log level: Level.ERROR

2024-01-23T04:06:40.851055+0000 - INFO: ===== 0 NONE 0 NOTE 0 WARNING 0 ERROR =====

2024-01-23T04:06:40.851142+0000 - INFO: =====

2024-01-23T04:06:41.694717+0000 - INFO: Starting compile model...

Current Phase

Quality Score	Latency(ms)	Model Size	Non-zero model parameters
0.0	0.5840706825256348	5.289M	5.289M

Closest Result:

Model Speed (latency): 0.5381345748901367

Model Quality Score: 0.02338087622852339

Model Size: 2.969M

Onnx Model Path:

/root/output/EfficientNet\_ImageNet/20240123040221/e53fc33e-7a24-4c.onnx

Pytorch Model Path:

/root/output/EfficientNet\_ImageNet/20240123040221/e53fc33e-7a24-4c.pth

Please find the info of all existing models here:

/root/output/EfficientNet\_ImageNet/20240123040221/summarized\_data.csv

Top

Bottom

Ok

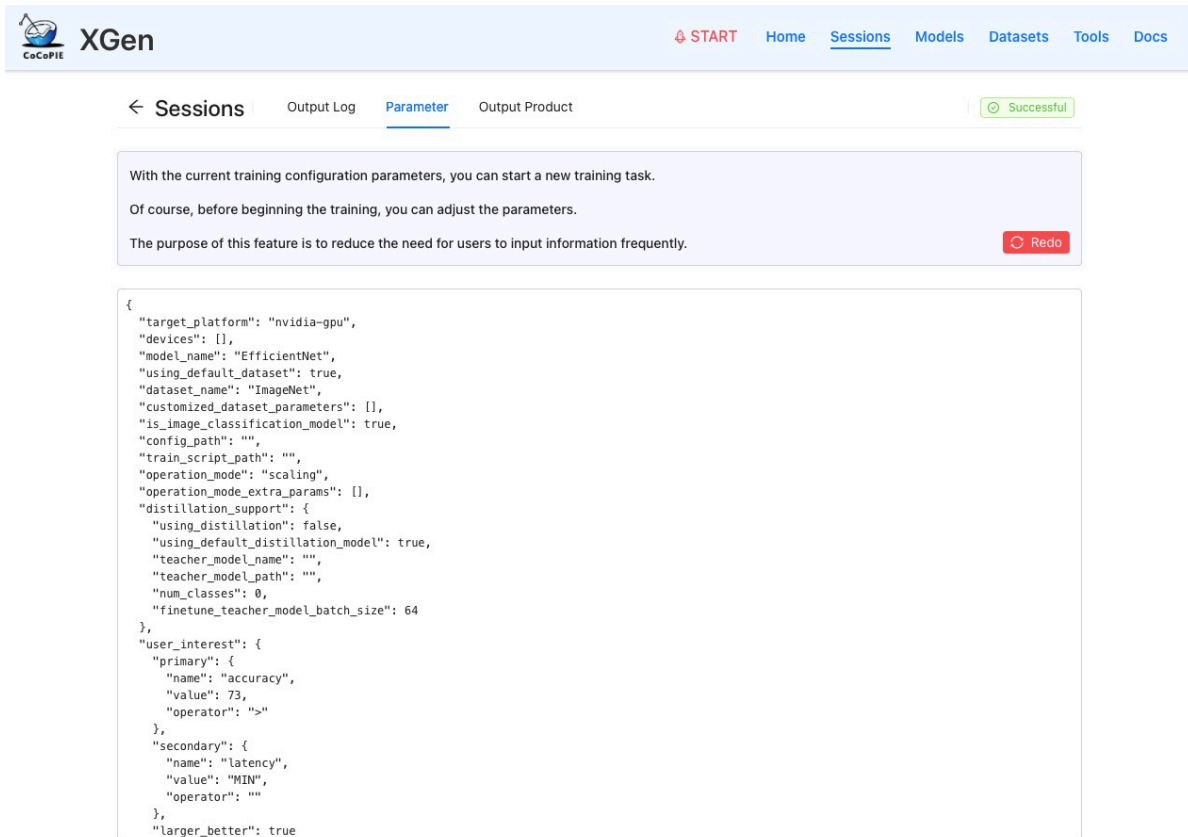
© 2023 CoCoPIE Inc. All Rights Reserved.

If interested in exact parameter details, please refer to **Parameter** tab. This tab shows the parameters used in the optimization process. You can see the parameters used in each iteration.

https://xgen.cocopie.ai/v1.4.1/12\_XGen\_WebUI\_Usage/

10/21





With the current training configuration parameters, you can start a new training task.

Of course, before beginning the training, you can adjust the parameters.

The purpose of this feature is to reduce the need for users to input information frequently.

[Redo](#)

```
{
 "target_platform": "nvidia-gpu",
 "devices": [],
 "model_name": "EfficientNet",
 "using_default_dataset": true,
 "dataset_name": "ImageNet",
 "customized_dataset_parameters": {},
 "is_image_classification_model": true,
 "config_path": "",
 "train_script_path": "",
 "operation_mode": "scaling",
 "operation_mode_extra_params": {},
 "distillation_support": {
 "using_distillation": false,
 "using_default_distillation_model": true,
 "teacher_model_name": "",
 "teacher_model_path": "",
 "num_classes": 0,
 "finetune_teacher_model_batch_size": 64
 },
 "user_interest": {
 "primary": {
 "name": "accuracy",
 "value": 73,
 "operator": ">"
 },
 "secondary": {
 "name": "latency",
 "value": "MIN",
 "operator": ""
 }
 },
 "larger_better": true
}
```

© 2023 CoCoPIE Inc. All Rights Reserved.

What's more, you can click [redo](#) button in the parameter page to prefill the optimization form with the parameters used in the iteration. Then you can start a new optimization process with the same parameters or just simply change it a little bit for another optimization try.

If the optimization process is finished, flag on the top right will turn to green with word [success](#) . Otherwise, it will be red with word [failed](#) . There are many reasons that may cause the optimization process failed. You can check the log to find the reason and fix it. Then click the [resume](#) button to resume the optimization process.

[← Sessions](#)[Output Log](#)[Parameter](#)[Failed](#)

Please carefully examine the output to determine if it meets the resume conditions. If it does, please try running it.

[Resume](#)

```

2023-11-17T07:48:59.025156+0000 - INFO: Your current workplace is /root/output/DigitNet_customized_dataset/20231117074857
2023-11-17T07:48:59.025279+0000 - INFO: A new search is started!
xgen-config-path:
/root/output/DigitNet_customized_dataset/20231117074857/xgen_config.json
xgen-workplace: /root/output/DigitNet_customized_dataset/20231117074857
xgen-resume: False
xgen-mode: pas
xgen-pretrained-model-path:
/root/output/DigitNet_MNIST/20231114025212/84cc2298-4f11-4d.pth
detail args:
{
 'origin': {
 'common_train_epochs': 3,
 'root_path': './__workplace__',
 'pretrain_model_weights_path': None,
 'train_data_path': '/data/flower/train',
 'train_label_path': None,
 'eval_data_path': None,
 'eval_label_path': None,
 'scaling_factor': 0,
 'num_classes': 10,
 'batch_size': 128,
 'learning_rate': 0.01
 },
 'general': {
 'user_id': 'test',
 'work_place': '/root/output/DigitNet_customized_dataset/20231117074857',
 'random_seed': 3407,
 'enable_ddp': False,
 'CUDA_VISIBLE_DEVICES': '0',
 'transcripts_path': None
 },
 'prune': {
 'sp_store_weights': None,
 'sp_lars': False,
 'sp_lars_trust_coef': 0.001,
 'sp_backbone': False,
 'sp_retrain': False,
 'sp_admm': False,
 'sp_admm_multi': False,
 'sp_retrain_multi': False,
 'sp_config_file': None,
 'sp_subset_progressive': False,
 'sp_admm_fixed_params': False,
 'sp_no_harden': False,

```

If the optimization is succeeded. you can see the final result in the `Output Product` tab.

← Sessions

Output Log

Parameter

Output Product

🟢 Successful

The training results are ready for you. You can download them and integrate them into your app or other applications.

The downloaded content represents the results that are closest to user requirements.

If you wish to reference other results, please refer to the table below.

Download

# Parameter	Value 1	Value 2
config_path	/root/output/EfficientNet_ImageNet/20240123040221/e53fc33e-7a24-4cargs_ai.json	/root/output/EfficientNet_ImageNet/20240123040221/464d0-40args_ai.json
work_place	/root/output/EfficientNet_ImageNet/20240123040221	/root/output/EfficientNet_ImageNet/20240123040221
onnx_path	/root/output/EfficientNet_ImageNet/20240123040221/e53fc33e-7a24-4c.onnx	/root/output/EfficientNet_ImageNet/20240123040221/464d0-40.onnx
pth_path	/root/output/EfficientNet_ImageNet/20240123040221/e53fc33e-7a24-4c.pth	/root/output/EfficientNet_ImageNet/20240123040221/464d0-40.pth
quality score	0.0233808762285233	0
model_size	2.969M	221.670K
total_nz_parameters	2.969M	221.670K
flops	239.933M	33.821M
file_name	e53fc33e-7a24-4c.pth	40a28a3f-64d0-40.pth
uuid	e53fc33e-7a24-4c	40a28a3f-64d0-40
prune_rates	0	0
width_multiplier	0.703477180149999	0.1
trained_yn	1	1
pruning		
global_sparsity	0	0
global_sparsity_prune_layer	0	0
quantized	0	0
pas_ratio	0	0
express_path	1	1
gpu_id	0	0
devices		
ios_devices		
device_platform	nvidia-gpu	nvidia-gpu
create_time	2024-01-23 04:02:37.250541	2024-01-23 04:05:10.762042
model_name	EfficientNet	EfficientNet
compiled_file_path	/root/output/EfficientNet_ImageNet/20240123040221/e53fc33e-7a24-4ccompiled_file	/root/output/EfficientNet_ImageNet/20240123040221/464d0-40compiled_file
latency	0.5381345748901367	0.4380154609680176
testing_message		
all_devices_benchmark_results		
abs_diff	73.07661912377147	73.1
closest_model	true	false
per_job_result_download	Download	Download

<

1

>

You can download the output product (best result) by clicking the top right red `Download` button. The output product is a \*.tar.gz file. If interested in result of specific iteration of optimization, Please click download button at the end of related column in the output form.

For usage of the output product, please refer to [Results](#)

## Built-in and Customized Models

Go to this page by clicking `Built-in AI Models` and `Customized AI Models` button in home page or `Models` button in the navigation bar on top of the site. For builtin models, which is provided by CoCoPIE, user can not manage it. For customized models, please upload the customized to `/root/customized_models` in the container then the uploaded model could be selected in through the `select path` button.

### Built-in Models

The `Models` page by default shows the built-in models provided by XGen. You can select a model and click the `Download` button to download the model's code maintained by XGen.


**XGen**

[START](#)
[Home](#)
[Sessions](#)
[Models](#)
[Datasets](#)
[Tools](#)
[Docs](#)

### Built-in AI Models Management

Built-in Models

Name	Category	Version	Action
<a href="#">DigitNet</a>	Image Classification 0	v1.0.7	Ready
<a href="#">EfficientNet</a>	Image classification I	v1.1.0	Ready
<a href="#">ResNet</a>	Image classification II	v1.1.0	<a href="#">Download</a>
<a href="#">MobileNet</a>	Image classification III	v1.1.0	<a href="#">Download</a>
<a href="#">ViT</a>	Image classification IV	v1.1.0	<a href="#">Download</a>
<a href="#">YOLOv6</a>	Object detection	v1.0.3	<a href="#">Download</a>
<a href="#">YOLOX</a>	Object detection	v1.0.3	Ready
<a href="#">YOLOv8</a>	Object detection	v1.0.3	<a href="#">Download</a>
<a href="#">UNet</a>	Segmentation	v1.0.6	<a href="#">Download</a>
<a href="#">R2+1d</a>	Video classification	v1.0.6	<a href="#">Download</a>
<a href="#">MobileBERT</a>	Natural language processing	v1.0.4	<a href="#">Download</a>

[<](#)
1
[>](#)

Please download the models as needed. For downloaded models, XGen will check the update of the model code every time you start a new optimization process. If there is a new version of the model code, The **Action** column will show a **Upgrade** button. Click it to upgrade the model code.

## Customized Models

Please upload the customized to `/root/customized_models` in the container then the uploaded model could be selected in through the **select path** button.

## Datasets Management

Go to this page by clicking **Built-in Datasets** and **Customized Datasets** button in home page or **Datasets** button in the navigation bar on top of the site. For customized datasets, please

upload the datasets to `/data/customized_datasets` then the uploaded dataset could be selected by the `select path` button.

## Built-in Datasets

This tab lists all datasets provided by CoCoPIE including standard datasets and tiny datasets for rapid verification and test. You can select a dataset and click the `Download` button on the top right of the dataset card to download the dataset. Meanwhile, the dataset card will show the download progress.



XGen

START

Home

Sessions

Models

Datasets

Tools

Docs

Datasets Management

Built-in Datasets

Default Datasets in Models

MNIST

Category: Image Classification

Total Size: 11.06 MB

Version: 20221004

Creator: CoCoPIE Inc.

Description: The MNIST dataset, consisting of 60,000 training images and 10,000 test images, which for handwritten digit recognition with 10 classes (digits 0-9).

ImageNet

Category: Image Classification

Total Size: 144.02 GB

Version: 20220930

Creator: CoCoPIE Inc.

Description: ImageNet is a large-scale dataset with over 1.4 million high-resolution images, organized into 1,000 object categories, used as a benchmark for training and evaluating deep learning models in computer vision.

CoCo2017

Category: Object detection

Total Size: 25.02 GB

Version: 20221205

Creator: CoCoPIE Inc.

Description: CoCo2017 is an extensive object detection dataset containing over 118,000 images and annotations for 80 different object classes, making it a widely adopted benchmark for evaluating and developing state-of-the-art object detection algorithms in computer vision.

CoCo2017 for YOLOX

Category: Object detection

Total Size: 18.91 GB

Version: 20230818

Creator: CoCoPIE Inc.

Description: CoCo2017 is an extensive object detection dataset containing over 118,000 images and annotations for 80 different object classes, making it a widely adopted benchmark for evaluating and developing state-of-the-art object detection algorithms in computer vision.

ISBI-2012

Category: Segmentation

Total Size: 15.27 MB

Version: 20220930

Creator: CoCoPIE Inc.

Description: ISBI-2012 is a dataset specifically designed for biomedical image segmentation, consisting of a collection of images with corresponding ground truth annotations, serving as a standard benchmark for evaluating and advancing algorithms in the field of biomedical image analysis and segmentation.

UCF101

Category: Video classification

Total Size: 36.53 GB

Version: 20220930

Creator: CoCoPIE Inc.

Description: UCF101 is a popular action recognition dataset that includes over 13,000 videos across 101 action categories, making it a widely used benchmark for evaluating and developing algorithms in the field of action recognition and video analysis.

Squad1.1

Category: Natural language processing

Total Size: 0 B

Version: 20221027

Creator: CoCoPIE Inc.

Description: SQuAD 1.1 (Stanford Question Answering Dataset) is a widely recognized dataset for machine reading comprehension, consisting of a large collection of passages from Wikipedia along with corresponding questions and answers, serving as a standard benchmark for evaluating and advancing question-answering models and techniques in natural language processing.

The Datasets Created By CoCoPIE for Rapid Verification

ImageNet50

Category: Image Classification

Total Size: 7.88 GB

Version: 20240110

Creator: CoCoPIE Inc.

Description: 50 subclasses selected from Imagenet, which can fast verification algorithms

Flower

Category: Image Classification

Total Size: 223.56 MB

Version: 20240110

Creator: CoCoPIE Inc.

Description: 5 categories of flower datasets, used to verify the algorithm processes

CoCo Small YOLOX

Category: Object detection

Total Size: 2.98 GB

Version: 20240126

Creator: CoCoPIE Inc.

Description: CoCo\_small is a subset of the coco2017 dataset, which contains 10 categories of data, each category has 2000 training images and 100 test images.

https://xgen.cocopie.ai/v1.4.1/12\_XGen\_WebUI\_Usage/

17/21

CoCo Small

Category:

Object detection

Total Size:

2.87 GB

Version:

20240126

Creator:

CoCoPIE Inc.

Description:

CoCo\_small is a subset of the coco2017 dataset, which contains 10 categories of data, each category has 20 00 training images and 100 test images.

Upload a dataset

Please upload the datasets to `/data/customized_datasets` then the uploaded dataset could be selected by the `select path` button.

Convenient Tools

Go to this page by clicking `Convenient Tools` button in home page or `Tools` button in the navigation bar on top of the site. This page shows some convenient tools provided by XGen, such as the benchmark tool. We will include more convenient tools in the feature.

XGen

START

Home

Sessions

Models

Datasets

Tools

Docs

Convenient Tools

Model Latency Benchmarking and Result

Benchmark Results

Measure model latency

ID	Platform	FolderName	Profiling	Status	Action
652a9438-109c-4ee1-81cd-b451b02832df	Mobile GPU/CPU (Android)	withbench2files	True	Successful	Detail
6ad8081f-1c80-4333-bdd7-147a143c7805	Mobile GPU/CPU (Android)	withbenchmark2	False	Successful	Detail
22402463-c456-4f54-808d-0249b170d13c	Mobile GPU/CPU (Android)	mobile2o333	False	Successful	Detail
00492f1b-3a1e-46c3-bc97-d9a8d4f390ae	Mobile GPU/CPU (Android)	test	False	Successful	Detail
c3ef5c00-c40d-477c-aa92-c5e9527c1e83	Mobile GPU/CPU (iOS)	ios1onnx2	False	Successful	Detail
295714e0-e15e-4c22-902b-40f54f5feae1	Mobile GPU/CPU (Android)	mobile2onnx2	False	Successful	Detail
fc2a4456-bb4b-4c0f-b2ba-a9c3a94ad8dc	Mobile GPU/CPU (Android)	mobile2	False	Successful	Detail

<

1

>



## Model Latency Benchmarking and Result

Sometimes it's essential to measure the existing latency on edged devices to determine how much optimization is needed. Thus, we need to do benchmark on the existing models. Fortunately, XGen provide a convenient tool to do this.

For benchmarking existing models latency, we offer a convenient tool in the WebUI. You can benchmark the latency of a existing **ONNX** model The benchmarking result will be shown in the WebUI. If the models to be benchmarked is not in ONNX format, please convert it to ONNX format first. For guide of converting models to ONNX format, please refer to [Checking Model Applicability](#).

There are two benchmarking modes. Normal benchmarking, which aims at measuring the latency of the model on the device. And the profiling benchmarking, which aims at do profiling on each layer of the model on edged device. The profiling benchmarking will generate a profiling log for each layer of the model.

Clicking `Measure model latency` button will navigate to the benchmarking page.

 **XGen**

[START](#) [Home](#) [Sessions](#) [Models](#) [Datasets](#) [Tools](#) [Docs](#)

[← Convenient Tools](#)

Which platform device do you want? ⓘ

☒ Mobile GPU/CPU (Android)

☐ Mobile GPU/CPU (iOS)

Please choose your device(s)

☐ android -- 3131612233002SC model: V2072A ☐ android -- DIAYLZ6955JFNF7L model: RMX2121

Folder Name

Folder to be created for placing onnx files

Including detailed time breakdown

☐

Upload ONNX Files

File name should end with .onnx. Other type of files will be rejected.



Drop files or directories to upload

Support for a single or bulk upload. Strictly prohibit from uploading company data or other band files

After submit the benchmarking form, you will be redirected to the benchmark detail page.

[START](#) [Home](#) [Sessions](#) [Models](#) [Datasets](#) [Tools](#) [Docs](#)

ONNX Benchmark

< Benchmark Results

Benchmark Detail

Successful

The detailed results of each phone's benchmark are saved in a CSV file, please download it to view.

Note that if "Including detailed time breakdown" is selected, then the first two columns in the CSV will be empty.

Download

FileName	DeviceID	Latency
lyl_origin_2.onnx	3131612233002SC	20.17 ms
lyl_origin_2.onnx	DIAYLZ6955JFNF7L	3.36 ms
lyl_origin.onnx	3131612233002SC	20.33 ms
lyl_origin.onnx	DIAYLZ6955JFNF7L	3.41 ms

2024-01-23T09:28:32.706619+0000 - INFO: Start testing...

2024-01-23T09:28:32.735082+0000 - INFO: Starting compile model...

2024-01-23T09:28:32.735346+0000 - INFO: Onnx file path: /root/benchmark\_models/withbenchmark2/input/lyl\_origin\_2.onnx

2024-01-23T09:28:34.008274+0000 - INFO: Benchmarking compiled model...

2024-01-23T09:28:34.025339+0000 - INFO: Uploading benchmark file...

2024-01-23T09:28:34.090973+0000 - INFO: Making benchmark task...

2024-01-23T09:28:34.104142+0000 - INFO: Sending benchmark task to controller...

2024-01-23T09:28:34.106996+0000 - INFO: Waiting for benchmark result...

2024-01-23T09:28:38.009569+0000 - INFO: Received benchmark result.

2024-01-23T09:28:38.446122+0000 - INFO: Starting compile model...

2024-01-23T09:28:38.446415+0000 - INFO: Onnx file path: /root/benchmark\_models/withbenchmark2/input/lyl\_origin\_2.onnx

2024-01-23T09:28:48.388881+0000 - INFO: Benchmarking compiled model...

2024-01-23T09:28:48.407968+0000 - INFO: Uploading benchmark file...

2024-01-23T09:28:48.441873+0000 - INFO: Making benchmark task...

2024-01-23T09:28:48.455730+0000 - INFO: Sending benchmark task to controller...

2024-01-23T09:28:48.459131+0000 - INFO: Waiting for benchmark result...

2024-01-23T09:31:34.113039+0000 - INFO: Received benchmark result.

2024-01-23T09:31:34.124347+0000 - INFO: Starting compile model...

2024-01-23T09:31:34.124506+0000 - INFO: Onnx file path: /root/benchmark\_models/withbenchmark2/input/lyl\_origin\_2.onnx

2024-01-23T09:31:34.891013+0000 - INFO: Benchmarking compiled model...

2024-01-23T09:31:34.909510+0000 - INFO: Uploading benchmark file...

2024-01-23T09:31:34.940842+0000 - INFO: Making benchmark task...

2024-01-23T09:31:34.953385+0000 - INFO: Sending benchmark task to controller...

2024-01-23T09:31:34.956393+0000 - INFO: Waiting for benchmark result...

2024-01-23T09:31:38.448726+0000 - INFO: Received benchmark result.

2024-01-23T09:31:38.460240+0000 - INFO: Starting compile model...

2024-01-23T09:31:38.460493+0000 - INFO: Onnx file path: /root/benchmark\_models/withbenchmark2/input/lyl\_origin\_2.onnx

2024-01-23T09:31:47.467058+0000 - INFO: Benchmarking compiled model...

2024-01-23T09:31:47.484621+0000 - INFO: Uploading benchmark file...

2024-01-23T09:31:47.513269+0000 - INFO: Making benchmark task...

2024-01-23T09:31:47.524085+0000 - INFO: Sending benchmark task to controller...

2024-01-23T09:31:47.526858+0000 - INFO: Waiting for benchmark result...

2024-01-23T09:31:52.546770+0000 - INFO: Received benchmark result.

2024-01-23T09:31:52.712717+0000 - INFO: Starting compile model...

2024-01-23T09:31:52.712961+0000 - INFO: Onnx file path: /root/benchmark\_models/withbenchmark2/input/lyl\_origin.onnx

2024-01-23T09:31:53.485920+0000 - INFO: Benchmarking compiled model...

2024-01-23T09:31:53.503782+0000 - INFO: Uploading benchmark file...

2024-01-23T09:31:53.532985+0000 - INFO: Making benchmark task...

2024-01-23T09:31:53.544750+0000 - INFO: Sending benchmark task to controller...

2024-01-23T09:31:53.547536+0000 - INFO: Waiting for benchmark result...

2024-01-23T09:31:56.585035+0000 - INFO: Received benchmark result.

2024-01-23T09:31:56.597286+0000 - INFO: Starting compile model...

2024-01-23T09:31:56.597586+0000 - INFO: Onnx file path: /root/benchmark\_models/withbenchmark2/input/lyl\_origin.onnx

2024-01-23T09:32:04.751262+0000 - INFO: Benchmarking compiled model...

2024-01-23T09:32:04.764379+0000 - INFO: Uploading benchmark file...

2024-01-23T09:32:04.798474+0000 - INFO: Making benchmark task...

2024-01-23T09:32:04.810646+0000 - INFO: Sending benchmark task to controller...

2024-01-23T09:32:04.813573+0000 - INFO: Waiting for benchmark result...

2024-01-23T09:32:10.684157+0000 - INFO: Received benchmark result.

2024-01-23T09:32:10.693399+0000 - INFO: Starting compile model...

2024-01-23T09:32:10.693596+0000 - INFO: Onnx file path: /root/benchmark\_models/withbenchmark2/input/lyl\_origin.onnx

2024-01-23T09:32:11.578500+0000 - INFO: Benchmarking compiled model...

2024-01-23T09:32:11.596013+0000 - INFO: Uploading benchmark file...

2024-01-23T09:32:11.628488+0000 - INFO: Making benchmark task...

2024-01-23T09:32:11.640842+0000 - INFO: Sending benchmark task to controller...

© 2023 CoCoPIE Inc. All Rights Reserved.

This page will show the benchmark detail after benchmark end. On the top is the benchmark result. Below is the detailed benchmark log. User can download the benchmark log by clicking the

`Download` button on the top right of the page. The result is a csv file containing the detailed benchmark result of each device of each model in different model state ( spare or not ). The result quick view just shows the best result of the benchmark.

User can close the browser or navigate to other pages. The process will not be interrupted. But please note that **please do not exit the XGen container while the benchmarking process is running**. If exit the container, the process will be terminated. User can go back to this page by clicking the `detail` button in the list of `Model Latency Benchmarking` and `Result` tab.

## XGen Docs

Go to this site. This is the link to the XGen documentation.

## Chapter 5. XGen Results

### Location of Outputs

Every time XGen runs, its outputs are saved into a folder located within the workplace. If the user does not manually designate the workplace path, either through the program's interface or configuration file, XGen automatically saves the outputs to its default workplace path. This path is typically a folder situated beneath the Output folder of XGen. By default, on the XGen container, this folder is designated as `~/output`, and on the host, the folder is usually labeled as `Output/`. Once the execution completes, XGen notifies the user of the output path on the screen.

Note:

- By default, XGen creates a new folder in the workplace and names it using the current timestamp and the model name. For example, the default workplace folder for running a model named DigitNet\_MNIST on August 15, 2023 at 3:32:25 AM would be located at `~/output/DigitNet_MNIST/20230815033225`
- In compatibility testing mode, the default workplace folder is located in `/tmp` and is automatically deleted after the test is completed.

### Outputs

The essential outputs of XGen are as follows:

- `summarized_data.csv`: The summary file contains the meta info of every run that shares the same configuration, including the performance of the models found in those runs. The essential columns of the file are as follows; some columns (e.g., power, memory size) in the file are reserved fields for future use:
  - `Onnx_path`: the produced AI model in ONNX format
  - `pth_path`: the produced AI model parameters (i.e., weights) in PyTorch format
  - `quality score`: the quality score of the produced AI model
  - `model_size`: the total number of parameters in the produced AI model
  - `flops`: the total number of floating-point operations (FLOPs) in one inference of the produced AI model

- `prune_rates`: the value of the parameter `prune_rate` in the configuration
- `global_sparsity`: average sparsity (percentage of zero weights) of the entire network
- `global_sparsity_prune_layer`: average sparsity of the pruned layers
- `device`: The IDs of the selected devices
- `compiled_file_path`: the path that holds the final AI model and code
- `latency`: the latency of the model on the slowest selected device
- `xgen-training.log`: the log file of the run. When filing a bug report, the user should include this log file in the report.
- a `*.onnx` file: the produced AI model in ONNX format
- a `*.pth` file: the produced AI model parameters (i.e., weights) in PyTorch format
- `*compiled_files` folder: A folder containing the final model and code files that are ready to be integrated into mobile apps. This folder is only produced if at least one device was selected in the run. If XGen does not know the target device, it will not compile the models. *Hint: Users may call XGen later to compile a previously generated model. See [Using the Compiler and Testing Components Only](#).*

After using XGen, the contents of the `*compiled_files` folder are all that users need to integrate the AI model into their apps. The folder's contents depend on the backend mode used in XGen.

For Android devices, XGen's backend includes two modes for code generation: `DeepOpt` and `Fallback`. `DeepOpt` conducts deeper optimizations and often produces better results than `Fallback`, but it does not yet support some operators that `Fallback` does. ([Appendix I](#) lists the supported operators of the different modes.) XGen tries both modes and automatically gives the better one to the user as the final output, without user involvement. The output of the two modes consists of different sets of files, but they are used in the same way in an app. This chapter first explains the outputs of each mode, then explains how to use the outputs in an app. For iOS devices, there is only one mode of code generation, which is the same as the `Fallback` mode for Android.

## DeepOpt Output for Android

The output of **DeepOpt** consists of 6 files in the output directory.

The directory structure is as follows:

```
output folder/android
├── include/xgen.h # XGen model deployment interface
├── include/xgen_data.h # DNN model weights file
├── include/xgen_pb.h # DNN model structure file
└── lib/libxgen.so # XGen library
```

```
├─ model/*.pb # DNN model structure file
└─ model/*.data # DNN model weights file
```

## Fallback Output for Android

The output of **Fallback** consists of 3 files in the output directory. The directory structure is as follows:

```
output folder/android
├─ include/xgen.h # XGen model deployment interface
├─ lib/libxgen.so # XGen library
└─ model/*.fallback # DNN model file (including both structure and
weights)
```

## Output for iOS

The output for **iOS** consists of five files in the output directory. The directory structure is as follows:

```
output folder/ios
.
├─ benchmark_data
│ └─ benchmark_params.json # Configuration file for benchmark settings
│ └─ *.fallback # DNN model file (including both structure and
weights)
├─ benchmark_data.zip # Compressed file for folder benchmark_data
├─ benchmark_output.json # Output from the benchmark execution
└─ xgen.framework # XGen iOS library
```

## Usage of the Results

The code generated by XGen can be easily integrated into Android or iOS apps. Integration can be static, where the AI model becomes a part of the app package, or dynamic, where the AI model can be loaded into the app during the execution. The latter approach can help minimize the static size of the app package.

Both integration methods are demonstrated in a demo app for both Android and iOS:

- Android: Click on [this link](#) to see the code for the demo app. The `localDynamicDeepOpt_XGen-Auto` branch of the git repository demonstrates the static approach using the `DeepOpt` outputs, while the `remoteDynamicDeepOpt` branch demonstrates the dynamic approach.
- iOS (static and dynamic integration): Click on [this link](#) to see the code for the demo app. The `main` branch of the git repository demonstrates the static approach, while the `download-`

`model` branch demonstrates the dynamic approach.

The following section introduces the complete API for using the output of XGen. The XGen APIs can be found in the `xgen.h` file, which contains detailed descriptions. Below is a complete list of all the APIs.

```
XGEN_EXPORT XGenHandle *XGenInit(const void *model_data, size_t model_size);
XGEN_EXPORT XGenHandle *XGenInitWithData(const void *model_data,
 size_t model_size_in_bytes,
 const void *extra_data,
 size_t data_size_in_bytes);
XGEN_EXPORT XGenHandle *XGenInitWithPower(const void *model_data,
 size_t model_size_in_bytes,
 const void *extra_data,
 size_t data_size_in_bytes,
 XGenPowerPolicy policy);
XGEN_EXPORT XGenHandle *XGenInitWithFiles(const char *model_file,
 const char *data_file,
 XGenPowerPolicy policy =
XGenPowerDefault);
XGEN_EXPORT XGenStatus XGenRun(XGenHandle *handle);
XGEN_EXPORT void XGenShutdown(XGenHandle *handle);
XGEN_EXPORT size_t XGenGetNumInputTensors(const XGenHandle *handle);
XGEN_EXPORT size_t XGenGetNumOutputTensors(const XGenHandle *handle);
XGEN_EXPORT XGenTensor *XGenGetInputTensor(XGenHandle *handle,
 size_t tensor_index);
XGEN_EXPORT XGenTensor *XGenGetOutputTensor(XGenHandle *handle,
 size_t tensor_index);
XGEN_EXPORT void XGenCopyBufferToTensor(XGenTensor *input_tensor,
 const void *input_data,
 size_t input_size_in_bytes);
XGEN_EXPORT void XGenCopyTensorToBuffer(const XGenTensor *output_tensor,
 void *output_data,
 size_t output_size_in_bytes);

XGEN_EXPORT XGenType XGenGetTensorType(const XGenTensor *tensor);
XGEN_EXPORT int32_t XGenGetTensorNumDims(const XGenTensor *tensor);
XGEN_EXPORT int32_t XGenGetTensorDim(const XGenTensor *tensor,
 int32_t dim_index);
XGEN_EXPORT size_t XGenGetTensorSizeInBytes(const XGenTensor *tensor);
XGEN_EXPORT void *XGenGetTensorData(const XGenTensor *tensor);
XGEN_EXPORT const char *XGenGetTensorName(const XGenTensor *tensor);
```

## Use in an Android application

There are five steps to follow when using an XGen-generated AI model in an Android application:

1. Initialization: Use one of the `XGenInit` functions to initialize the model.

2. Copy Input Data: Use `XGenCopyBufferToTensor` to copy the input data into the tensor.
3. Perform Prediction: Run the model by calling `XGenRun`.
4. Copy Output Data: Once the prediction is complete, use `XGenCopyTensorToBuffer` to copy the output data from the tensor.
5. Release Resources: Finally, release all resources by calling `XGenShutdown`.

Additionally, a set of APIs are available to query various tensor attributes, such as tensor name, type, and size.

### Using the output of DeepOpt

If the XGen output directory (i.e., the `*compiled_files` folder mentioned in the [XGen outputs](#) section) includes `xgen_pb.h` and `xgen_data.h`, the model is generated in the DeepOpt mode. Follow the example below or the demo app [link](#) to deploy it.

Suppose that `example_pb` and `example_pb_len` are the first and second variable names in `xgen_pb.h`, and `example_data` and `example_data_len` are the first and second variable names in `xgen_data.h`.

```
#include "xgen_data.h"
#include "xgen_pb.h"
#include "xgen.h"

int main() {
 //Initialization
 XGenHandle *h = XGenInitWithData(example_pb,
 example_pb_len,
 example_data,
 example_data_len);

 size_t num_input_tensors = XGenGetNumInputTensors(h);
 for (size_t i = 0; i < num_input_tensors; ++i) {
 XGenTensor *t = XGenGetInputTensor(h, i);
 size_t input_size = XGenGetTensorSizeInBytes(t);

 float *buffer = ... // initialize buffer with your input data
 XGenCopyBufferToTensor(t, buffer, input_size); //copy data
 }

 //Run inference
 XGenRun(h);

 size_t num_output_tensors = XGenGetNumOutputTensors(h);
 float **buffer = new float*[num_output_tensors];
 for (size_t i = 0; i < num_output_tensors; ++i) {
 XGenTensor *t = XGenGetOutputTensor(h, i);
 size_t output_size = XGenGetTensorSizeInBytes(t);
```



```

 buffer[i] = new float[(output_size + sizeof(float) - 1) / sizeof(float)];
 XGenCopyTensorToBuffer(t, buffer[i], output_size); //copy output data
 }

 // Use model output here.

 //Release resources
 XGenShutdown(h);

 return 0;
}

```

`XGenInitWithPower` can be used to fine-tune power usage for a better power-speed trade-off. The power policy can be set as one of the four predefined settings, as follows:

```

typedef enum XGenPowerPolicy {
 /* No power policy. Use whatever system default configuration is. */
 XGenPowerNone = 0,
 /* XGen default power policy is used with XGenInitWithData */
 XGenPowerDefault = 1,
 /* Performance policy uses less power than the default but still performs
 well. */
 XGenPowerPerformance = 2,
 /* Power saving policy uses even less power than performance policy. */
 XGenPowerSave = 3,
} XGenPowerPolicy;

```

In addition to using `XGenInitWithData` as in the example above, you can also initialize XGen by using `XGenInitWithFiles`, which takes in \*.pb and \*.data files, as follows:

```

XGenHandle *h = XGenInitWithFiles("wdsr.pb", "wdsr.data");

```

### Using the output of Fallback

The usage of AI models in the fallback mode is the same as in the DeepOpt mode, except that you should use `XGenInitWithFile` for initialization instead of `XGenInitWithData`. Use the following example, where `example_model_file` is the \*.fallback file located in its `model` folder:

```

XGenHandle *h = XGenInitWithFiles(example_model_file, NULL);

```

### Use in an iOS application

Using the output of XGen in an iOS application is similar to using Fallback output in an Android application, except that the model file \*.fallback is located in the `benchmark_data` folder. Users can refer to [this link](#) for the code of a demo app.

## Chapter 6. Status Monitoring

This chapter explains how users can monitor the status of XGen's executions and Device Lab.

### Monitoring XGen's Status

After entering all the configuration from the command line, the XGen console prints the total number of stages/cycles of this run. A stage refers to a main phase in the XGen execution, and one cycle in a stage refers to one iteration of the main loop in that stage. The specific meanings of a stage and a cycle depend on the selected mode of XGen; details are omitted in this manual.

During its execution, XGen prints its status to the screen and also records the info in a log file. Upon normal termination of an XGen execution, the log file is usually put inside the `output` folder in XGen (i.e., `/root/output/` inside XGen docker) as mentioned in the [Result section](#). the location will be printed on the screen right before the termination of XGen.

When there are errors, the user may check that log file for info. When the user files an error report, including that log file will be helpful.

Note: XGen sometimes terminate early before completing all the stages and cycles because it has already found a result that meets the requirement.

```
Current search has 3 stages
Stage: 1
Max Searching Cycles: 1
Stage: 2
Max Searching Cycles: 10
Stage: 3
Max Searching Cycles: 1
*****config summary*****
Current state: stage: 1/3| cycles: 1/1
Total jobs:1
processing job 1/1
Training...
==> Preparing data..
==> Building model..
```

At the end of the XGen run, there is a summary of the results of the run, including but not limited to the following:

- the statistics of the final results: a table showing the statistics of the finally produced AI model
- compiled file path: the folder that holds the finally produced AI model and code that are ready to be used in an app

- onnx model path: the produced AI model in ONNX format
- historical trained models info file: the csv file recording the info of all the runs of XGen (including this one) that share the same configuration file as this run does:

Quality Score	Latency(ms)	Model Size	Non-zero model parameters
98.92	1.67	1.196M	320.458K

Closest Result:  
Model-speed(latency): 1.67

Model Quality Score: 98.92

Model Size: 1.196M

Compiled File Path:  
/root/output/your\_own\_model/20221013024128/164379d8-8d52-4c  
4ccompiled\_file

Onnx Model Path:  
/root/output/your\_own\_model/20221013024128/164379d8-8d52-4c.onnx

Pytorch Model Path:  
/root/output/your\_own\_model/20221013024128/164379d8-8d52-4c.pth

Please find the info of all existing models here:  
/root/output/your\_own\_model/20221013024128/summarized\_data.csv

Users can go into the corresponding paths to view the XGen output files.

## Monitoring Device Lab's Status

### Where to find the daemon services log files

There are many daemon services running in the background. And logs file for each files can be found in a hidden directory under the home directory of the container. Below is where the logs file locate:

- Device Lab Controller : /root/.xgen\_controller/logs
- Device Lab Agent : /root/.xgen\_agent/logs

- Device Lab Dashboard: `/root/.xgen_dashboard/logs`
- XGen: `/root/.xgen/logs`

## How to check the status of the daemon services

The status of the daemon services can be checked by running the following command:

```
xgenctl status
```

This tool will help to determine if the daemon services are running properly. If not, it will show the error message.

## How to restart the daemon services

The daemon services can be restarted by running the following command:

```
xgenctl restart
```

Command above will restart all the services.

Or restart a specific service by running the following command:

```
xgenctl restart -s SERVICE_NAME
```

Use `xgenctl restart --help` to see what services are available.

# FAQ

## Problem Diagnosis

1. When filing an error report, please include the log file, which typically resides in a subfolder under the output folder of XGen ( `~/output` inside XGen container and `Output/` in XGen folder on the host) (see [XGen Results](#)).
2. If you have any problems pulling docker images, make sure you have logged into the Docker repository and that you have the correct permissions. See [installation instructions](#).
3. If you get an error message about the GPU, ensure that Nvidia-Driver and Nvidia-Docker are correctly installed and their versions meet the requirements. See [installation instructions](#).
4. If after XGen is started, it lists no devices, please check the connections and setup of the devices. See [installation instructions](#). Use `xgenctl status` in XGen, Agent and Controller shell to see the services status. If there is a service failure or exit, try using the command `xgenctl restart` to restart the service.
5. If the following error appears, it is possible that your devices are disconnected. Please check the connections and setup of the devices. See [installation instructions](#). Use `xgen_devices list -nm` to see what devices are connected in the agent container.

```
ValueError: benchmark testing error, latency cannot be nan
```

6. If you encounter the following error message, you probably have specified a training script path that contains no `train_script_main.py` . See [Custom AI](#).

```
ModuleNotFoundError: No module named 'train_script_main'
```

7. If you encounter the following error message, you might have run `XGen` in a directory that contains some PyTorch model files ( `*.pth` ). Try to run `XGen` in a different directory (e.g., `~/Projects/` ).

```
Fatal Python error: initsite: Failed to import the site module
```

8. If you encounter the following error message, try to run `sudo apt install gnupg2 pass` inside XGen shell to fix the issue.

```
Error saving credentials: error storing credentials - err: ...
```

9. If you running `uninstall_xgen` command, and encounter the following error message,

```
Error response from daemon: conflict: unable to delete foo (must be forced) -
image is being used by stopped container bar
```

try to run `docker rm -f bar` to remove the container first, then run `uninstall_xgen` again.

10. Errors in network or network topology shows up when run `run_xgen`. If meet error like following:

```
docker: Error response from daemon: driver failed programming external
connectivity on endpoint xgen_mq
(de6f4cc60458d22e940a9f1a15abd98ae51a447e1e101fc72b1f8a10b99f8035): (iptables
failed: iptables --wait -t nat -A DOCKER -p tcp -d 0/0 --dport 55672 -j DNAT -
-to-destination 172.18.0.2:5672 ! -i br-39207ec16e43: iptables: No
chain/target/match by that name.
(exit status 1)).
```

or error like this:

```
pymongo.errors.ServerSelectionTimeoutError: xgen_mongodb:27017: timed out,
Timeout: 30s, Topology Description: <TopologyDescription id:
654c4d2e01fe9ea16dbaa2bc, topology_type: Unknown, servers: [<ServerDescription
('xgen_mongodb', 27017) server_type: Unknown, rtt: None,
error=NetworkTimeout('xgen_mongodb:27017: timed out')>]>
```

It most likely that there is something to do with your network firewall rules which could be resulted from incorrect firewall service configuration.

**The simplest way** is to just disable the firewall services or allow required ports listed in the installation page in your firewall rules or manually insert the rule to iptables (not recommended to manipulate iptables rule directly) For closing `firewalld` service, run command below:

```
sudo systemctl stop firewalld # stop firewall service
sudo systemctl disable firewalld # disable firewall service
sudo systemctl mask --now firewalld # mask firewall service
sudo systemctl status firewalld # check firewall service status, should be
inactive and masked
```

For closing `ufw` service, run command below:

```
sudo systemctl stop ufw # stop firewall service
sudo systemctl disable ufw # disable firewall service
sudo systemctl mask --now ufw # mask firewall service
```

```
sudo systemctl status ufw # check firewall service status, should be inactive
and masked
```

Sometimes, **local network rules may override Docker's default network and bridge forwarding rules**. For example, without specifying the `bip` and `default-address-pools` for Docker, the default network segment generated by Docker is as follows:

```
$ ifconfig
```

```
br-6f017036e0bd: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
 inet 172.18.0.1 netmask 255.255.0.0 broadcast 172.18.255.255
 inet6 fe80::42:ebff:fe6d:1837 prefixlen 64 scopeid 0x20<link>
 ether 02:42:eb:6d:18:37 txqueuelen 0 (Ethernet)
 RX packets 22601980 bytes 5655627456 (5.6 GB)
 RX errors 0 dropped 0 overruns 0 frame 0
 TX packets 30346252 bytes 7482474509 (7.4 GB)
 TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

docker0: flags=4099<UP,BROADCAST,MULTICAST> mtu 1500
 inet 172.17.0.1 netmask 255.255.0.0 broadcast 172.17.255.255
 inet6 fe80::42:fbff:fe36:365a prefixlen 64 scopeid 0x20<link>
 ether 02:42:fb:36:36:5a txqueuelen 0 (Ethernet)
 RX packets 0 bytes 0 (0.0 B)
 RX errors 0 dropped 0 overruns 0 frame 0
 TX packets 23 bytes 2506 (2.5 KB)
 TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

...
```

If the firewall rules happen to cover the 172.18.0.1 or 172.17.0.1 network segment, the above connection error message will also appear.

For example, the following firewall nft rules,

```
$ sudo nft list ruleset
```

```
chain nat_PREROUTING_ZONES {
ip saddr 10.0.0.0/8 goto nat_PRE_public
ip saddr 172.16.0.0/12 goto nat_PRE_public
ip saddr 152.14.0.0/16 goto nat_PRE_public
iifname "docker0" goto nat_PRE_docker
iifname "br-08bf99834a22" goto nat_PRE_docker
goto nat_PRE_public
}

...
```

The rule `ip saddr 172.16.0.0/12 goto nat_PRE_public` covers all addresses from 172.16.0.0 to 172.31.255.255, so the data forwarding related to Docker will not be forwarded

to `nat_PRE_docker`, and XGen will display a connection error message.

There are two solutions, 1. Contact the system administrator to adjust the firewall rules to let Docker's default network pass; 2. the other is to configure the network address for Docker to avoid firewall rules. Here is a brief introduction to setting Docker network settings, edit

`/etc/docker/daemon.json` file, add the following content,

```
``bash
{
 "runtimes": {
 "nvidia": {
 "args": [],
 "path": "nvidia-container-runtime"
 }
 },
 "bip": "192.168.249.1/24",
 "default-address-pools": [{ "base": "192.168.250.0/18", "size": 24 }]
}
```

The ``bip`` specified here is the address of ``docker0``, and ``default-address-pools`` specifies the address range for Docker to create other ``br``. For more detailed configuration information and meaning, you can refer to the network section description on Docker's official website.

**For other firewall services, please refer to their official documentation.**

**Please do not manipulate iptables rule directly unless you know what you are doing.**

**Please do not disable firewall service if you are not sure about the security risk.**

**Please do not simultaneously run multiple firewall services. They may conflict with or block rules of each other.**

11. Error report associated with ADB version mismatch:

```
The host ADB version doesn't match the ADB version of XGen, please upgrade
or degrade your ADB version to \${adb_version_on_container}.
You can install matched ADB version by the following command:
1. curl -s https://dl.google.com/android/repository/platform-
tools_r33.0.3-linux.zip -o android-platform-tools.zip${NC}"
2. sudo unzip -o -q android-platform-tools.zip -d /usr/lib/android-
sdk/${NC}"
3. sudo ln -sf /usr/lib/android-sdk/platform-tools/adb /usr/bin/adb${NC}"
```

This error message is mostly shown up in the Device Lab Agent which is responsible for hosting edge devices. It is because the ADB version on the host machine is not compatible



with the ADB version on the container. The ADB version on the host and container must be the same because of the C/S architecture of ADB. To check the version of ADB, please run command `adb version`. Please ensure the output of this command is the same as the version of ADB in the container. If not, please follow the instruction in the error message to install the matched ADB version.

## Appendix I: XGen Supported DNN Operators

### Operators Supported by DeepOpt Code Generation (for Android only)

(CPU and/or GPU mean that this operator can be efficiently executed on CPU and/or GPU, respectively. In an execution of a DeepOpt model, it chooses the GPU implementation of an operator first; if it is not supported, it uses the CPU implementation.)

- `Activation` (CPU, GPU)
- `Activation3D` (CPU, GPU)
- `AdaptivePooling` (CPU, GPU)
- `BatchNorm` (CPU, GPU)
- `BiasAdd` (CPU, GPU)
- `Deconv2D` (CPU, GPU)
- `DepthToSpace` (GPU)
- `DepthwiseConv2d` (CPU, GPU)
- `LSTM-dynamic` (CPU)
- `LSTM-nonlinear` (CPU)
- `LSTM-block-cell` (CPU)
- `ChannelShuffle` (CPU, GPU)
- `Clip` (CPU, GPU)
- `Concat` (CPU, GPU)
- `Conv2D` (CPU, GPU)
- `Conv3D` (CPU, GPU)
- `Eltwise` (CPU, GPU)
- `Eltwise3D` (CPU, GPU)
- `Expand` (CPU, GPU)
- `FullyConnected` (CPU, GPU)
- `Gather` (CPU)

- `Gemm` (CPU, GPU)
- `GRU` (GPU)
- `HardSigmoid` (CPU, GPU) v1.0.28 or later
- `Identity` (CPU, GPU)
- `InstanceNorm` (CPU, GPU)
- `MatMul` (CPU, GPU)
- `Pad` (CPU, GPU)
- `PixelShuffle` (CPU, GPU)
- `Pooling` (CPU, GPU)
- `Pooling3D` (CPU, GPU)
- `Reduce` (CPU, GPU)
- `Reshape` (CPU, GPU)
- `ResizeBicubic` (CPU, GPU)
- `ResizeBilinear` (CPU, GPU)
- `ResizeNearestNeighbor` (CPU, GPU)
- `Round` (GPU) v1.0.28 or later
- `Shape` (CPU)
- `Slice` (CPU, GPU)
- `Softmax` (CPU, GPU)
- `SpaceToDepth` (GPU)
- `Squeeze` (CPU, GPU)
- `Transpose` (CPU, GPU)
- `Unsqueeze` (CPU)
- `Upsample` (CPU, GPU)
- `TopK` (CPU)
- `NMS` (CPU)
- `RPN` (CPU)

Operators Supported by Fallback Code Generation (for both Android and iOS)

## Operators with Advanced Optimizations (All on GPU only)

- Add
- Concat
- Constant
- Conv
- ConvTranspose
- Flatten
- Gemm
- GlobalAveragePool
- MaxPool
- Mul
- Relu
- Sigmoid
- Sub

## Other Supported Operators (All on GPU only)

- Abs
- Acos
- Acosh
- And
- ArgMax
- ArgMin
- Asin
- Asinh
- Atan
- Atanh
- AveragePool
- Ceil
- Compress
- ConstantOfShape

- Cos
- Cosh
- DepthToSpace
- Div
- Dropout
- Elu
- Equal
- Erf
- Exp
- EyeLike
- Floor
- GlobalLpPool
- GlobalMaxPool
- Greater
- HardSigmoid
- Hardmax
- If
- IsNaN
- LRN
- LeakyRelu
- Less
- Log
- LogSoftmax
- Loop
- LpNormalization
- LpPool
- Max
- MaxUnpool
- Mean
- MeanVarianceNormalization

- Min
- Multinomial
- Neg
- NonZero
- Not
- OneHot
- Or
- PRelu
- Pow
- RandomNormal
- RandomNormalLike
- RandomUniform
- RandomUniformLike
- Reciprocal
- Scan
- Scatter
- Selu
- Shrink
- Sign
- Sin
- Sinh
- Size
- Softplus
- Softsign
- SpaceToDepth
- Split
- Sqrt
- Sum
- Tan
- Tanh

- `TfIdfVectorizer`
- `Tile`
- `Where`
- `Xor`

## Appendix II: Dataset Preparation for Common AI

XGen provides a set of popular base AI models for some common AI tasks. For a common AI task, users may provide their own datasets and use XGen to optimize one of those base models into a model suiting the specific need.

The simplest way to use users' own dataset is to convert it to the format compatible to the expected format of the common AI model of interest. This appendix explains the expected data format of each of the common AI tasks.

### Image Classification

#### Dataset Preparation

For multi-class classification, we recommend to use [ImageNet](#) format.

The images should be arranged in the following structure:

```
train/
├── cat
│ ├── xxx.png
│ ├── xxy.png
│ ├── ...
│ └── xxz.png
├── bird
│ ├── bird1.png
│ ├── bird2.png
│ └── ...
└── dog
 ├── 123.png
 ├── nsdf3.png
 ├── ...
 └── asd932_.png
```

In this way, the users don't need to provide separate annotations: All images in the a directory (e.g., `cat` ) will be regarded as samples of that class.

Usually, the whole dataset shall be split into two sub datasets: `train` and `val` for training and validation respectively:



```
image_classification
├── my_dataset
│ ├── train
│ └── val
```

The `val` folder should be in the same structure as the `train` folder.

## Use in XGen

With dataset organized as instructed, the user may use the custom AI in XGen on any of the included image classification models that has `ImageNet` as the default dataset. Below is a sample of the inputs in a custom AI:

```
Training data absolute path should be absolute train path in the container
Training data absolute path:
/data/prepare_data/image_classification/my_dataset/train
Training labels absolute path should be null for image classification
Training labels absolute path:
Eval data absolute path should be absolute eval path in the container
Eval data absolute path: /data/prepare_data/image_classification/my_dataset/val
Eval labels absolute path should be null for image classification
Eval labels absolute path:
Number of classes is total classes in your image classification dataset
Number of classes: 5
```

## Object Detection (YOLOX)

### Dataset Preparation

For object detection, we use `COCO` format.

Organize your train and val images and labels according to the example below.

```
object_detection_datasets/
├── YourDataSetName/
│ ├── annotations/
│ │ ├── instances_train2017.json
│ │ └── instances_val2017.json
│ ├── train2017/
│ │ └── <image files for training>
│ └── val2017/
│ └── <image files for validation>
```

Note that the names of your train and val annotations must be `instances_train2017.json` and `instances_val2017.json`

## Use in XGen

Then, when using XGen, the user inputs the paths info when XGen asks.

```
Dataset absolute path should be absolute object_detection_datasets
The root directory of the dataset:
/data/prepare_data/object_detection_datasets/YourDataSetName/
Number of classes is total classes in your object detection dataset, the number
of classes must be equal to the number of names in .yaml files
Number of classes: 80
```

## Object Detection (YOLOV6)

### Dataset Preparation

For object detection, we use [YOLOv5](#) format.

Organize your train and val images and labels according to the example below.

```
object_detection_datasets
├── YourDataSetName
│ ├── images
│ │ ├── train
│ │ │ ├── train_01.png
│ │ │ ├── train_02.png
│ │ │ ├── train_03.png
│ │ │ └── ...
│ │ └── val
│ │ ├── val_01.png
│ │ ├── val_02.png
│ │ └── ...
│ └── labels
│ ├── train
│ │ ├── train_01.txt
│ │ ├── train_02.txt
│ │ ├── train_03.txt
│ │ └── ...
│ └── val
│ ├── val_01.txt
│ ├── val_02.txt
│ └── ...
```

The `labels` folder contains one text label file for each of the images unless that image contains no object. The label file specifications are as follows:

- One row per object

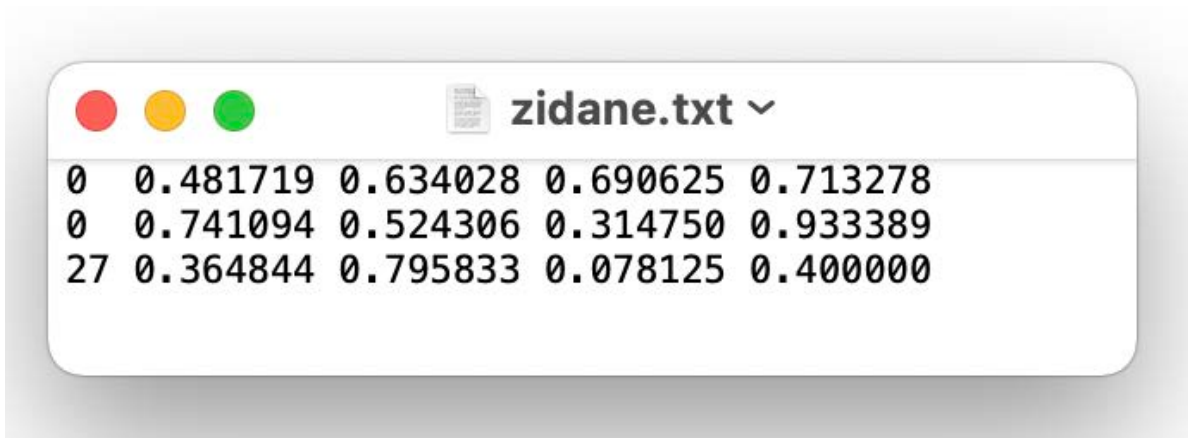
- Each row shows the following of an object in the image:

```
class x_center y_center width height,
```

where, `class` is the class label of that object (zero-indexed), `x_center` `y_center` `width` `height` are the coordinates, width and height of the bounding box of that object. The coordinates, `width` and `height` must all be normalized to 0-1 by the width and height of the image.



The label file corresponding to the above image contains 2 persons (class 0) and a tie (class 27):



## Label Images

user can use [Roboflow](#) to help prepare datasets. [Create a free Roboflow account](#), upload your dataset, and then label the images as needed. After that, user can export the labels to the needed format (YOLO v5 PyTorch format). That website also allows a user to convert a dataset with labels of other formats (e.g., COCO) to convert the labels to the required format.

Note, if you have a labeled dataset already in the COCO format, you may use `coco_json2yolo.py` to convert the label files to the format expected by YOLO. Example

```
python coco_json2yolo.py
./object_detection_datasets/coco/annotations/instances_val2017.json
```

## Use in XGen

To use XGen on the prepared dataset, the user need to create a yaml file to indicate the number and names of the classes, as illustrated as follows:

```
Classes
train: /data/object_detection_datasets/images/train
val: /data/object_detection_datasets/images/val
test: /data/object_detection_datasets/images/test
If you use coco dataset, set is_coco: True, and original json path must set there
anno_path: /data/object-detection-yolov6/coco/annotations/instances_val2017.json
number of classes
nc: 80
whether it is coco dataset, only coco dataset should be set to True.
is_coco: True
nc: 80 # number of classes
names: ['person', 'bicycle', 'car', 'motorcycle', 'airplane', 'bus', 'train',
 'truck', 'boat', 'traffic light',
 'fire hydrant', 'stop sign', 'parking meter', 'bench', 'bird', 'cat',
 'dog', 'horse', 'sheep', 'cow',
 'elephant', 'bear', 'zebra', 'giraffe', 'backpack', 'umbrella',
 'handbag', 'tie', 'suitcase', 'frisbee',
 'skis', 'snowboard', 'sports ball', 'kite', 'baseball bat', 'baseball
glove', 'skateboard', 'surfboard',
 'tennis racket', 'bottle', 'wine glass', 'cup', 'fork', 'knife', 'spoon',
 'bowl', 'banana', 'apple',
 'sandwich', 'orange', 'broccoli', 'carrot', 'hot dog', 'pizza', 'donut',
 'cake', 'chair', 'couch',
 'potted plant', 'bed', 'dining table', 'toilet', 'tv', 'laptop', 'mouse',
 'remote', 'keyboard', 'cell phone',
 'microwave', 'oven', 'toaster', 'sink', 'refrigerator', 'book', 'clock',
 'vase', 'scissors', 'teddy bear',
 'hair drier', 'toothbrush'] # class names
```

Then, when using XGen, the user inputs the paths info when XGen asks.

```

Path to the dataset description yaml file (coco.yaml in
yolov6): /data/prepare_data/object_detection_datasets/coco/coco.yaml
Number of classes is total classes in your object detection dataset, the number
of classes must be equal to the number of names in .yaml files
Number of classes: 80

```

## Object Detection (YOLOV8)

### Dataset Preparation

The data format for YOLOv8 is consistent with YOLOv6, and you can refer to the previous data description section for YOLOv6.

### Use in XGen

To use XGen on the prepared dataset, the user need to create a yaml file to indicate the number and names of the classes, as illustrated as follows:

```

path: /data/object-detection/coco # dataset root dir
train: train2017.txt # train images (relative to 'path') 118287 images
val: val2017.txt # val images (relative to 'path') 5000 images
test: test-dev2017.txt # 20288 of 40670 images, submit to
https://competitions.codalab.org/competitions/20794

Classes
names:
 0: person
 1: bicycle
 2: car
 3: motorcycle
 4: airplane
 5: bus
 6: train
 7: truck
 8: boat
 9: traffic light
 10: fire hydrant
 11: stop sign
 12: parking meter
 13: bench
 14: bird
 15: cat
 16: dog
 17: horse
 18: sheep
 19: cow
 20: elephant

```

```
21: bear
22: zebra
23: giraffe
24: backpack
25: umbrella
26: handbag
27: tie
28: suitcase
29: frisbee
30: skis
31: snowboard
32: sports ball
33: kite
34: baseball bat
35: baseball glove
36: skateboard
37: surfboard
38: tennis racket
39: bottle
40: wine glass
41: cup
42: fork
43: knife
44: spoon
45: bowl
46: banana
47: apple
48: sandwich
49: orange
50: broccoli
51: carrot
52: hot dog
53: pizza
54: donut
55: cake
56: chair
57: couch
58: potted plant
59: bed
60: dining table
61: toilet
62: tv
63: laptop
64: mouse
65: remote
66: keyboard
67: cell phone
68: microwave
69: oven
70: toaster
71: sink
72: refrigerator
73: book
```

```

74: clock
75: vase
76: scissors
77: teddy bear
78: hair drier
79: toothbrush

```

Then, when using XGen, the user inputs the paths info when XGen asks.

```

Path to the dataset description yaml file (coco.yaml in
yolov8):/data/prepare_data/object_detection_datasets/coco/coco.yaml
Number of classes is total classes in your object detection dataset, the number
of classes must be equal to the number of names in .yaml files
Number of classes: 80

```

## Object Segmentation

### Dataset Preparation

For object segmentation, we use [ISBI Challenge Dataset](#) format as it is the format expected by UNet, the included segmentation model in XGen. The original dataset struct of ISBI dataset is as follows.

```

object_segmentation_datasets
├── ISBI
│ ├── em
│ │ ├── train-volume.tif
│ │ ├── train-labels.tif
│ │ └── test-volume.tif

```

If the users' data are in that format, they can use `~/Projects/segmentation-unet/run_make_database.py` script to convert the dataset to the format required by UNet. Please customize the data path in the script before running.

```
python run_make_database.py
```

The users should make sure that the train and val images and labels are in a structure as shown as follows.

```

object_segmentation_datasets
├── ISBI
│ ├── em
│ │ ├── train
│ │ │ ├── input_001.npy
│ │ │ └── input_002.npy

```



## Use in XGen

```
Training data absolute path should be absolute train images path
Training data absolute path:
/data/prepare_data/object_segmentation_datasets/ISBI/
Training labels absolute path should be absolute train label path
Training labels absolute path:
/data/prepare_data/object_segmentation_datasets/ISBI/
Eval data absolute path should be absolute eval images path
Eval data absolute path: /data/prepare_data/object_segmentation_datasets/ISBI/
Eval labels absolute path should be absolute eval images path
Eval labels absolute path: /data/prepare_data/object_segmentation_datasets/ISBI/
```

## Video Classification

## Dataset Preparation

For video classification, the format is one derived from [UCF-101](#) dataset format.

The original structure of the UCF-101 dataset is as follows:

```
video_classification_datasets
├── ucf101
│ ├── ApplyEyeMakeup
│ │ ├── v_ApplyEyeMakeup_g008_c001.avi
│ │ └── v_ApplyEyeMakeup_g008_c002.avi
```



```

├── ...
├── ApplyLipstick
│ ├── v_ApplyLipstick_g08_c01.avi
│ ├── v_ApplyLipstick_g08_c02.avi
│ └── ...
├── Archery
│ ├── v_Archery_g08_c01.avi
│ ├── v_Archery_g08_c02.avi
│ └── ...
└── ...

```

If the users' dataset is in such a format and structure, before applying XGen on it, the user needs to convert it with several scripts included in `~/Projects/video-classification-s2-1d/` as follows:

- Extract frames from videos by running `python extract_videos.py -d ucf101`.
- Convert frames into LMDB files with `create_lmdb.py`.

```
python create_lmdb.py -d ucf101_frame -s train -vr 0 10000
python create_lmdb.py -d ucf101_frame -s val -vr 0 4000
```

The users shall make sure that after the preparation, the datasets are in the following structure:

```

video_classification_datasets
├── ucf101_frame
│ ├── v_ApplyEyeMakeup_g08_c01
│ │ ├── 00001.jpg
│ │ ├── 00002.jpg
│ │ ├── 00003.jpg
│ │ └── ...
│ ├── v_ApplyEyeMakeup_g08_c02
│ │ ├── 00001.jpg
│ │ ├── 00002.jpg
│ │ ├── 00003.jpg
│ │ └── ...
│ ├── ...
│ ├── v_ApplyLipstick_g08_c01
│ │ ├── 00001.jpg
│ │ ├── 00002.jpg
│ │ ├── 00003.jpg
│ │ └── ...
│ ├── v_ApplyLipstick_g08_c02
│ │ ├── 00001.jpg
│ │ ├── 00002.jpg
│ │ ├── 00003.jpg
│ │ └── ...
│ ├── ...
│ └── v_Archery_g08_c01
│ ├── 00001.jpg
│ └── 00002.jpg

```

```
├── 00003.jpg
├── ...
├── ...
└── ucfTrainTestlist
 ├── classInd.txt
 ├── testlist01.txt
 └── trainlist01.txt
```

The `classInd.txt` records the indices of the classes, as illustrated as follows:

```
1 ApplyEyeMakeup
2 ApplyLipstick
3 Archery
4 BabyCrawling
5 BalanceBeam
...
```

## Use in XGen

```
Training data absolute path should be absolute train images path
Training data absolute path:
/data/prepare_data/video_classification_datasets/ucf101_frame
Training labels absolute path should be null for video classification
Training labels absolute path:
Eval data absolute path should be absolute eval path in the container
Eval data absolute path:
Eval labels absolute path should be null for video classification
Eval labels absolute path:
Number of classes is total classes in your video classification dataset
Number of classes: 5
```

## Appendix III: XGen built-in API documentation

### CL (Co\_Lib)

The CL is a built-in optimization library in XGen. It is a class method that provides several built-in methods for model optimization. It includes the following functions:

```
init(self, args=None, model=None, optimizer=None, logger=None,
data_loader=None, teacher_models={}, **kwargs)
```

This function initializes the CL library.

- `args` : Arguments for initialization.
- `model` : The model to be optimized.
- `optimizer` : The optimizer to be used.
- `logger` : The logger for recording logs.
- `data_loader` : The data loader for loading data.
- `teacher_models` : Dictionary of teacher models (if applicable).
- `kwargs` : Additional keyword arguments.
- `return` : If modify model graph, a new model will be returned, otherwise it will not be returned.
- **example**

```
cl_model = CL.init(args=args_ai, model=AImodel:torch.nn.Module,
optimizer=AIoptimizer:torch.optim,
data_loader=dataLoader:torch.utils.data.dataLoader.DataLoader)
if cl_model:
 AImodel = cl_model
```

### `after_scheduler_step(epoch)`

This function is called after each scheduler step.

- `epoch` : The current epoch.

- **example**

```
CL.after_scheduler_step(epoch)
```

### `before_each_train_epoch(epoch)`

This function is called before each training epoch.

- `epoch` : The current epoch.

- **example**

```
CL.before_each_train_epoch(epoch)
```

### `update_loss(loss)`

This function updates the loss.

- `loss` : The updated loss value.
- `return` : Return new loss value.

- **example**

```
loss = CL.update_loss(loss)
```

## xgen\_tools

The `xgen_tools` is a built-in library in XGen that provides utility functions for parameter merging, weight loading, and weight saving.

```
xgen_init(user_args: Union[argparse.Namespace, dict], args_ai: dict =
None, map: dict = None) -> dict
```

This function initializes XGen. Returns the initialized `user_args` and `args_ai`, which is the XGen config loaded from a file.

- `user_args` : User-defined arguments.

- `args_ai` : AI-related arguments (ignored in the feature).
- `map` : Arguments map (maps xgen-defined args to user-defined args).
- **example**

```
user_args, args_ai = xgen_init(originalArgs, map = COCOPIE_MAP)
```

```
xgen_load(model, args_ai=None, path=None, optimizer=None)
```

- `model` : The PyTorch model.
- `args_ai` : The XGen config, including the weights path.
- `path` : If `args_ai` is provided, this parameter can be ignored.
- `optimizer` : If provided, the optimizer state dict may be loaded for resuming.

This function loads weights for the model. It can load weights from different formats of .pth files.

When using XGen, if you have prepared pretrained model weights, you can select the absolute path of your weights in the "Pretrained model absolute path" field. XGen only supports PyTorch weights and has certain requirements for the organization of your weights, as described below:

- The first method allows you to obtain the weight information for each layer of the network directly using the following approach:

```
weights = torch.load("your pretrained model weights path")
model.load_state_dict(weights)
```

- If your uploaded weights contain additional information, please ensure that the weights are saved within the "model" field:

```
weights = torch.load("your pretrained model weights path")["model"]
model.load_state_dict(weights)
```

- **example**

```
xgen_load(model, args_ai=args_ai)
```

Please make sure to follow these guidelines when organizing and loading your pretrained model weights in XGen.

```
xgen_record(args_ai, origin_model, metric, epoch=None,
optimizer=None, onnx_export=None, onnx_file_path=None)
```

- `args_ai` : The XGen config.
- `origin_model` : The user's model.
- `metric` : The evaluation metric.
- `optimizer` : The optimizer parameters for resuming (optional).
- `epoch` : The epoch of the training (-1 means the standard record).
- `onnx_export` : A callable function (may be removed in the future).
- `onnx_file_path` : The path to the generated ONNX file.
- **example**

```
xgen_record(args_ai, model, accuracy, epoch=epochCount)
```

This function saves, profiles the model, and collects data for the training.

# Appendix IV Customized Model Sample

## Prepare YOLOX for XGen Optimizations

This document explains how to revise the original [YOLOX](#) training script so that it can be optimized by XGen. It in addition gives an example configuration file for the execution of XGen, and hints on the use of multiple GPUs and data preparation.

### 1.Prepare the training script

YOLOX has a default training script, which needs to be revised so that XGen can control the training process in its optimizations of the model.

The modifications are done basically by following the [XGen Document](#). Because the training code of YOLOX is implemented in several Python files, the involved changes are in multiple files. There is a special limitation in the default YOLOX training script, which prevents the compression from flexibly scaling the Neural Network layers. So an extra minor modification is applied to remove that limitation. We detail the changes below. In the example code, we use "#XGEN: " comments to indicate the applied changes.

- Step-1. Rename the default main training script `YOLOX/tools/train.py` to `train_script_main.py`. (In the YOLOX include XGen, we put `train_script_main.py` into `YOLOX/.`)
- Steps-2 and 3. Import the XGen modules and connect the training control variables with XGen. The modified `train_script_main.py` is as follows. See `#XGEN:` for the changed places.

```
...
XGEN Introduce the xgen_tools utility.
from xgen_tools import *

def make_parser():
 ...

@logger.catch
XGEN: add an extra argument so that the XGen control variables args_ai can
be passed to the rest of the training script
def main(exp: Exp, args, args_ai):
 if exp.seed is not None:
 random.seed(exp.seed)
 torch.manual_seed(exp.seed)
```

```

 cudnn.deterministic = True
 warnings.warn(
 "You have chosen to seed training. This will turn on the CUDNN
deterministic setting, "
 "which can slow down your training considerably! You may see
unexpected behavior "
 "when restarting from checkpoints."
)

set environment variables for distributed training
configure_nccl()
configure_omp()
cudnn.benchmark = True

XGen For the "get_trainer" section, add an args_ai parameter.
trainer = exp.get_trainer(args, args_ai)
trainer.train()

XGEN: Since XGen needs to find the training_main function, create the
function with the following content:
def training_main(args_ai=None):
 configure_module()
 args = make_parser().parse_args()

 # XGEN: call xgen_init
 args, args_ai = xgen_init(args, args_ai)
 # XGEN: The 'origin' field contains the XGen control variables
 xgen_control_args = args_ai['training_script_args']
 # XGEN: For multi-GPU training, determine the number of GPUs.
 args.devices = xgen_control_args['device_num']
 # XGEN: You can use the exp_file parameter in xgen config to select the
exp_file of yolox
 exp = get_exp(XXXX)

 exp.merge(args.opts)

 # XGEN: Connect the control variables
 exp.data_dir = xgen_control_args['train_data_path']
 # XGEN: Connect the control variables
 exp.max_epoch = xgen_control_args['common_train_epochs']
 # XGEN: Connect the control variables
 exp.num_classes = xgen_control_args['num_classes']

 # XGEN: This part is for scaling (Step-11 below).
 # A simple change would be simply scaling the width of a network
 # exp.width = exp.width * xgen_control_args['scaling_factor']
 # We suggest the following change instead as such scaling empirically
gives better results
 if args.scaling_factor != 1:
 exp.width = exp.width * xgen_control_args['scaling_factor'] + 0.3375
 else:
 exp.width = exp.width * xgen_control_args['scaling_factor']

```



```

exp.depth = exp.depth * xgen_control_args['scaling_factor']

XGEN: Disable EMA as it can affect the accuracy of pruning sparsity
statistics
exp.ema = False
check_exp_value(exp)

if not args.experiment_name:
 args.experiment_name = exp.exp_name

num_gpu = get_num_devices() if args.devices is None else args.devices
assert num_gpu <= get_num_devices()

if args.cache is not None:
 exp.dataset = exp.get_dataset(cache=True, cache_type=args.cache)

dist_url = "auto" if args.dist_url is None else args.dist_url
launch(
 main,
 num_gpu,
 args.num_machines,
 args.machine_rank,
 backend=args.dist_backend,
 dist_url=dist_url,
 args=(exp, args, args_ai),
)

XGEN: Create an entry point function that calls the training_main function:
if __name__ == "__main__":
 args_ai = None
 training_main(args_ai)

```

Make corresponding modifications to `YOLOX/yolox/core/trainer.py` as it defines the trainer used in the main training script. Add the following to the beginning of the file.

```

XGEN: Import XGen's built-in optimization methods at the beginning of the
file
from co_lib import Co_Lib as CL
from xgen_tools import xgen_record, xgen_init, xgen_load, XgenArgs, helper
helper.ENABLE_HELPER = False

```

And modify the `__init__` to connect the control variables with the trainer:

```

class Trainer:
 # XGEN: add argument args_ai to get access to XGEN control variables
 def __init__(self, exp: Exp, args, args_ai):
 # init function only defines some basic attr, other attrs like model,
optimizer are built in
 # before_train methods.
 self.exp = exp
 self.args = args

```

```
#XGEN: connect the XGEN control with the training process
self.args_ai = args_ai

self.epoch = 0
...
```

Due to the addition of the `args_ai` parameter in the `trainer = exp.get_trainer(args, args_ai)` line in the `train_script_main.py` file, the corresponding modifications need to be made in the `YOLOX/yolox/exp/yolox_base.py` file.

```
XGEN: add argument args_ai
def get_trainer(self, args, args_ai):
 from yolox.core import Trainer
 # XGEN: add args_ai in the call to Trainer
 trainer = Trainer(self, args, args_ai)
 ...
```

- Steps-4 and 5. Insert the `xgen_load` for DNN model loading, and initialize XGen's optimization. The changes should be put where the model is loaded and before the training starts. For YOLOX, the location is inside file `YOLOX/yolox/core/trainer.py`. So the changes are to that file, specifically in function `before_train(self)`; after the modification, the following is inserted into that function:

```
#XGEN: after the model, train_loader, and optimizer are defined, add the
following code.
#XGEN: use xgen_load function to load pretrained model weights
xgen_load(model, self.args_ai)
#XGEN: optimization method will initialize these parameters
cl_model = CL.init(args=self.args_ai, model=model, optimizer=self.optimizer,
data_loader=self.train_loader)
if cl_model:
 model = cl_model
```

- Steps-6,7,8,9,10. Add XGEN API calls inside the training loop to keep track the training status and apply controls, and finally output the final optimized model. As the training loop is inside file `YOLOX/yolox/core/trainer.py`, the changes are in that file. The changes are as follows:

```
def train_in_epoch(self):

 #XGEN: XGen will update the parameters once before and after each epoch of
 training, so xgen-related update methods need to be added to "train_in_epoch".
 #XGEN: At the same time, in the Compatibility test function of XGEN, we
 will set max_epoch to 0, and evaluate the accuracy of the model at the same
 time.
 #XGEN: In order to support this function, we need to add some additional
 code, as follows:
```

```

#XGEN: To support XGen Compatibility test function
if self.max_epoch == 0:
 self.evaluate_and_save_model()
 #XGEN: Save the model weights and best ap, convert model to onnx in
XGen.
 xgen_record(self.args_ai, self.model, self.best_ap, epoch=-1)
for self.epoch in range(self.start_epoch, self.max_epoch):
 #XGEN: XGen-related optimization functions, before_epoch
 CL.before_each_train_epoch(epoch=self.epoch)
 self.before_epoch()
 self.train_in_iter()
 self.after_epoch()
 #XGEN: XGen-related optimization functions, after_epoch
 CL.after_scheduler_step(epoch=self.epoch)

```

```

def train_one_iter(self):
 #XGEN: During the overall model optimization process, XGen's method
 includes generating a loss function that needs to be updated alongside the
 original training loss in the train_one_iter function. This ensures that both
 the XGen loss and the training loss are updated together to optimize the model
 effectively.

 iter_start_time = time.time()
 inps, targets = self.prefetcher.next()
 inps = inps.to(self.data_type)
 targets = targets.to(self.data_type)
 targets.requires_grad = False
 inps, targets = self.exp.preprocess(inps, targets, self.input_size)
 data_end_time = time.time()
 with torch.cuda.amp.autocast(enabled=self.amp_training):
 outputs = self.model(inps, targets)

 loss = outputs["total_loss"]
 #XGEN: Update the loss function with XGen.
 loss = CL.update_loss(loss)

```

```

#XGEN: Add code in the after_epoch function to save the optimized models:
def after_epoch(self):
 self.save_ckpt(ckpt_name="latest")
 #XGEN: saves the optimized models
 xgen_record(self.args_ai, self.model, self.best_ap, epoch=self.epoch)
 if (self.epoch + 1) % self.exp.eval_interval == 0:
 all_reduce_norm(self.model)
 self.evaluate_and_save_model()

```

- Step-11. To enable network scaling, modify `train_script_main.py` based on [Configure Scaling for XGen](#) to allow XGen to control the width and depth of the network.

```
XGEN: This part is for scaling (Step-11 below).
A simple change would be simply scaling the width of a network
exp.width = exp.width * xgen_control_args['scaling_factor']
We suggest the following change instead as such scaling empirically gives
better results
if args.scaling_factor != 1:
 exp.width = exp.width * xgen_control_args['scaling_factor'] + 0.3375
else:
 exp.width = exp.width * xgen_control_args['scaling_factor']
exp.depth = exp.depth * xgen_control_args['scaling_factor']
```

- Extra step. The default YOLOX has a special limitation. When  $64 * \text{scaling\_factor}$  is not an integer, the shapes of the output tensors and input tensors expected by the next layer may mismatch. It is due to some rounding bugs in YOLOX code. Removing the bug will unlock the flexibility for the model compression.

The corresponding modified files are `YOLOX/yolox/models/yolo_pafpn.py` and `YOLOX/yolox/models/yolo_head.py`. The specific changes are as follows:

In the `_init_` function in `yolo_pafpn.py`:

```
#!/usr/bin/env python
-*- encoding: utf-8 -*-
Copyright (c) Megvii Inc. All rights reserved.

import torch
import torch.nn as nn

from .darknet import CSPDarknet
from .network_blocks import BaseConv, CSPLayer, DWConv
import math

class YOLOPAFPN(nn.Module):
 """
 YOLOv3 model. Darknet 53 is the default backbone of this model.
 """

 def __init__(
 self,
 depth=1.0,
 width=1.0,
 in_features=("dark3", "dark4", "dark5"),
 in_channels=[256, 512, 1024],
 depthwise=False,
 act="silu",
):
 super().__init__()
 self.backbone = CSPDarknet(depth, width, depthwise=depthwise, act=act)
 self.in_features = in_features
```

```

 self.in_channels = in_channels
 Conv = DWConv if depthwise else BaseConv

 self.upsample = nn.Upsample(scale_factor=2, mode="nearest")
 # XGEN: Modify the calculation method of inchanenl and out_channel to
 support any width of d size
 self.lateral_conv0 = BaseConv(
 int(math.ceil(64 * width) * 16), int(math.ceil(64 * width) * 8), 1,
1, act=act
)
 # XGEN: Modify the calculation method of inchanenl and out_channel to
 support any width of d size
 self.C3_p4 = CSPLayer(
 int(2 * math.ceil(64 * width) * 8),
 int(math.ceil(64 * width) * 8),
 round(3 * depth),
 False,
 depthwise=depthwise,
 act=act,
) # cat
 # XGEN: Modify the calculation method of inchanenl and out_channel to
 support any width of d size
 self.reduce_conv1 = BaseConv(
 int(math.ceil(64 * width) * 8), int(math.ceil(64 * width) * 4), 1,
1, act=act
)
 # XGEN: Modify the calculation method of inchanenl and out_channel to
 support any width of d size
 self.C3_p3 = CSPLayer(
 int(2 * math.ceil(64 * width) * 4),
 int(math.ceil(64 * width) * 4),
 round(3 * depth),
 False,
 depthwise=depthwise,
 act=act,
)

 # bottom-up conv
 # XGEN: Modify the calculation method of inchanenl and out_channel to
 support any width of d size
 self.bu_conv2 = Conv(
 int(math.ceil(64 * width) * 4), int(math.ceil(64 * width) * 4), 3,
2, act=act
)
 # XGEN: Modify the calculation method of inchanenl and out_channel to
 support any width of d size
 self.C3_n3 = CSPLayer(
 int(2 * math.ceil(64 * width) * 4),
 int(math.ceil(64 * width) * 8),
 round(3 * depth),
 False,
 depthwise=depthwise,
 act=act,

```

```

)

 # bottom-up conv
 # XGEN: Modify the calculation method of inchanenl and out_channel to
 support any width of d size
 self.bu_conv1 = Conv(
 int(math.ceil(64 * width) * 8), int(math.ceil(64 * width) * 8), 3,
2, act=act
)
 # XGEN: Modify the calculation method of inchanenl and out_channel to
 support any width of d size
 self.C3_n4 = CSPLayer(
 int(2 * math.ceil(64 * width) * 8),
 int(math.ceil(64 * width) * 16),
 round(3 * depth),
 False,
 depthwise=depthwise,
 act=act,
)

```

In the `_init_` function in `yolo_head.py`:

```

#!/usr/bin/env python3
-*- coding:utf-8 -*-
Copyright (c) Megvii Inc. All rights reserved.

import math
from loguru import logger

import torch
import torch.nn as nn
import torch.nn.functional as F

from yolox.utils import bboxes_iou, cxcywh2xyxy, meshgrid, visualize_assign

from .losses import IOUloss
from .network_blocks import BaseConv, DWConv

class YOLOXHead(nn.Module):
 def __init__(
 self,
 num_classes,
 width=1.0,
 strides=[8, 16, 32],
 in_channels=[256, 512, 1024],
 act="silu",
 depthwise=False,
):
 """
 Args:
 act (str): activation type of conv. Defalut value: "silu".

```

```

 depthwise (bool): whether apply depthwise conv in conv branch.
 Defalut value: False.
 """
 super().__init__()

 self.num_classes = num_classes
 self.decode_in_inference = True # for deploy, set to False

 self.cls_convs = nn.ModuleList()
 self.reg_convs = nn.ModuleList()
 self.cls_preds = nn.ModuleList()
 self.reg_preds = nn.ModuleList()
 self.obj_preds = nn.ModuleList()
 self.stems = nn.ModuleList()
 Conv = DWConv if depthwise else BaseConv
 # XGEN: Modify the calculation method of inchanenl and out_channel to
 support any width of d size
 for i in range(len(in_channels)):
 self.stems.append(
 BaseConv(
 in_channels=int(math.ceil(64 * width) * (2 ** (i + 1)) *
2),
 out_channels=int(math.ceil(64 * width) * 4),
 ksize=1,
 stride=1,
 act=act,
)
)
 # XGEN: Modify the calculation method of inchanenl and out_channel
 to support any width of d size
 self.cls_convs.append(
 nn.Sequential(
 *[
 Conv(
 in_channels=int(math.ceil(64 * width) * 4),
 out_channels=int(math.ceil(64 * width) * 4),
 ksize=3,
 stride=1,
 act=act,
),
 Conv(
 in_channels=int(math.ceil(64 * width) * 4),
 out_channels=int(math.ceil(64 * width) * 4),
 ksize=3,
 stride=1,
 act=act,
),
]
)
)
 # XGEN: Modify the calculation method of inchanenl and out_channel
 to support any width of d size
 self.reg_convs.append(

```

```

 nn.Sequential(
 *[
 Conv(
 in_channels=int(math.ceil(64 * width) * 4),
 out_channels=int(math.ceil(64 * width) * 4),
 ksize=3,
 stride=1,
 act=act,
),
 Conv(
 in_channels=int(math.ceil(64 * width) * 4),
 out_channels=int(math.ceil(64 * width) * 4),
 ksize=3,
 stride=1,
 act=act,
),
]
)
)
 # XGEN: Modify the calculation method of inchanenl and out_channel
 to support any width of d size
 self.cls_preds.append(
 nn.Conv2d(
 in_channels=int(math.ceil(64 * width) * 4),
 out_channels=self.num_classes,
 kernel_size=1,
 stride=1,
 padding=0,
)
)
 # XGEN: Modify the calculation method of inchanenl and out_channel
 to support any width of d size
 self.reg_preds.append(
 nn.Conv2d(
 in_channels=int(math.ceil(64 * width) * 4),
 out_channels=4,
 kernel_size=1,
 stride=1,
 padding=0,
)
)
 # XGEN: Modify the calculation method of inchanenl and out_channel
 to support any width of d size
 self.obj_preds.append(
 nn.Conv2d(
 in_channels=int(math.ceil(64 * width) * 4),
 out_channels=1,
 kernel_size=1,
 stride=1,
 padding=0,
)
)

```



```

self.use_l1 = False
self.l1_loss = nn.L1Loss(reduction="none")
self.bcewithlog_loss = nn.BCEWithLogitsLoss(reduction="none")
self.iou_loss = IOUloss(reduction="none")
self.strides = strides
self.grid = [torch.zeros(1)] * len(in_channels)

```

All the above file modifications can be found in the YOLOX code in XGen, `/root/Projects/object-detection-yolox/yolox_xgen`.

## 2.Prepare Config file

As XGen's optimization methods are launched through configuration files, when integrating your own model into XGen optimization, you will need to prepare the corresponding configuration file. An example is as follows:

```

{
 "general": {
 "user_id": "test",
 "work_place": null,
 "tran_scripts_path": null,
 "pretrained_model_path": "/root/Projects/object-detection-yolox/yolox_xgen/xgen.pt",
 "enable_ddp": false,
 "CUDA_VISIBLE_DEVICES": "0",
 "random_seed": 3407
 },
 "user_requirements": {
 "power": null,
 "accuracy": null,
 "accuracy_reverse_yn": 0,
 "model_size": null,
 "memory_size": null,
 "latency": 90,
 "margin": 0.1,
 "target_type": "latency",
 "max_sampling": 10,
 "searching_variable": "scaling_factor",
 "searching_range": [
 0.2,
 1
],
 "searching_granularity": null
 },
 "compiler": {
 "input_shape": "(1,3,640,640)",
 "opset_version": 11
 },
 "training_script_args": {
 "train_data_path": "/data/object-detection-yolox/coco",

```

```

"train_label_path": null,
"eval_data_path": "/data/object-detection-yolox/coco",
"eval_label_path": null,
"dataset_path": "/data/object-detection-yolox/coco",
"learning_rate": 0.01,
"common_train_epochs": 300,
"device_num": 1,
"batch_size": 16,
"root_path": "./Xgen/",
"name": "yolox-m",
"exp_file": null,
"weights": null,
"device": null,
"imgsz": 640,
"width": 0.75,
"depth": 0.67,
"num_classes": 80,
"scaling_factor": 1,
"workers": 16,
"noplots": true
},
"pruning": {
 "sp_retrain": false,
 "sp_admm": false,
 "sp_config_file": null,
 "sp_admm_update_epoch": 5,
 "sp_admm_rho": 0.001,
 "sparsity_type": "block_punched",
 "sp_admm_lr": 0.01,
 "sp_global_weight_sparsity": false,
 "sp_admm_block": "(8,4)",
 "sp_prune_ratios": 0,
 "sp_prune_threshold": -1.0
},
"pas": {
 "pas_ratio": 0,
 "pas": false,
 "limit_loss_weights": 5.0,
 "use_limit_loss": false,
 "pas_debug": false,
 "pas_rebuild": false,
 "pas_finetune_epoch": 200,
 "pas_pretrained_weight_path": null,
 "pas_ignore": null,
 "pas_searching_ratio": [0.1],
 "part_name": "backbone"
},
"train": {
 "log_save_folder": "logs",
 "eval_dataset_id": null,
 "training_dataset_id": null,
 "uuid": "yolox",
 "accuracy_reverse_yn": 0,

```

```

 "common_save_best_yn": 1
 },
 "scaling": {
 "searching_variable": "scaling_factor",
 "searching_range": [
 0.2,
 1
],
 "searching_granularity": 0.05
 }
}

```

Please see the configuration included in XGEN for YOLOX here: `/root/Projects/object-detection-yolox/yolox_xgen/yolox_config/xgen.json`.

### 3. Other considerations

In multi-GPU training, using COCO evaluation with multi-process C++ calls can lead to process conflicts.

It is recommended to remove the following code at line 293 in

`YOLOX/yolox/evaluators/coco_evaluator.py`:

```

try:
 from yolox.layers import COCOeval_opt as COCOeval
except ImportError:
 from pycocotools.cocoeval import COCOeval

 logger.warning("Use standard COCOeval.")

```

Replace it with:

```

from pycocotools.cocoeval import COCOeval

logger.warning("Use standard COCOeval.")

```

### 4. Dataset Preparation

Currently, YOLOX only supports the original COCO dataset format. To use your own dataset, you will need to convert your dataset to COCO dataset format. Please see [Dataset Preparation for Object Detection \(YOLOX\)](#) for details. The directory structure of the dataset is as follows: datasets/ - coco/ - annotations/ - instances\_train2017.json - instances\_val2017.json - train2017/ - (image files for training) - val2017/ - (image files for validation)