Install and Setup

System requirements

DGL works with the following operating systems:

- Ubuntu 16.04
- macOS X
- Windows 10

DGL requires Python version 3.6, 3.7, 3.8 or 3.9.

DGL supports multiple tensor libraries as backends, e.g., PyTorch, MXNet. For requirements on backends and how to select one, see Working with different backends.

Starting at version 0.3, DGL is separated into CPU and CUDA builds. The builds share the same Python package name. If you install DGL with a CUDA 9 build after you install the CPU build, then the CPU build is overwritten.

Install from Conda or Pip

We recommend installing DGL by `conda` or `pip`. Check out the instructions on the Get Started page.

Note

For Windows users: you will need to install Visual C++ 2015 Redistributable.

Install from source

Download the source files from GitHub.

```
git clone --recurse-submodules https://github.com/dmlc/dgl.git
```

(Optional) Clone the repository first, and then run the following:

```
git submodule update --init --recursive
```

Linux

Install the system packages for building the shared library. For Debian and Ubuntu users, run:

```
sudo apt-get update
sudo apt-get install -y build-essential python3-dev make cmake
```

For Fedora/RHEL/CentOS users, run:

```
sudo yum install -y gcc-c++ python3-devel make cmake
```

To create a Conda environment for CPU development, run:

```
bash script/create_dev_conda_env.sh -c
```

To create a Conda environment for GPU development, run:

```
bash script/create_dev_conda_env.sh -g 11.7
```

To further configure the conda environment, run the following command for more details:

```
bash script/create_dev_conda_env.sh -h
```

To build the shared library for CPU development, run:

```
bash script/build_dgl.sh -c
```

To build the shared library for GPU development, run:

```
bash script/build_dgl.sh -g
```

To further build the shared library, run the following command for more details:

```
bash script/build_dgl.sh -h
```

Finally, install the Python binding.

```
cd python
python setup.py install
# Build Cython extension
python setup.py build_ext --inplace
```

macOS

Installation on macOS is similar to Linux. But macOS users need to install build tools like clang, GNU Make, and cmake first. These installation steps were tested on macOS X with clang 10.0.0, GNU Make 3.81, and cmake 3.13.1.

Tools like clang and GNU Make are packaged in **Command Line Tools** for macOS. To install, run the following:

```
xcode-select --install
```

To install other needed packages like cmake, we recommend first installing **Homebrew**, which is a popular package manager for macOS. To learn more, see the Homebrew website.

After you install Homebrew, install cmake.

```
brew install cmake
```

Go to root directory of the DGL repository, build a shared library, and install the Python binding for DGL.

```
mkdir build
cd build
cmake -DUSE_OPENMP=off -DUSE_LIBXSMM=OFF ..
make -j4
cd ../python
python setup.py install
# Build Cython extension
python setup.py build_ext --inplace
```

Windows

You can build DGL with MSBuild. With MS Build Tools and CMake on Windows installed, run the following in VS2019 x64 Native tools command prompt.

- CPU only build:

```
MD build
CD build
cmake -DCMAKE_CXX_FLAGS="/DDGL_EXPORTS" -DCMAKE_CONFIGURATION_TYPES="Release" -
DDMLC_FORCE_SHARED_CRT=ON .. -G "Visual Studio 16 2019"
msbuild dgl.sln /m
CD ..\python
python setup.py install
```

- CUDA build:

```
MD build
CD build
cmake -DCMAKE_CXX_FLAGS="/DDGL_EXPORTS" -DCMAKE_CONFIGURATION_TYPES="Release" -
DDMLC_FORCE_SHARED_CRT=ON -DUSE_CUDA=ON .. -G "Visual Studio 16 2019"
msbuild dgl.sln /m
CD ..\python
python setup.py install
```

Working with different backends

DGL supports PyTorch, MXNet and Tensorflow backends. DGL will choose the backend on the following options (high priority to low priority)

- Use the `DGLBACKEND` environment variable:
  - You can use `DGLBACKEND=[BACKEND] python gcn.py ...` to specify the backend
  - Or `export DGLBACKEND=[BACKEND]` to set the global environment variable
- Modify the `config.json` file under "~/.dgl":
  - You can use `python -m dgl.backend.set_default_backend [BACKEND]` to set the default backend

Currently BACKEND can be chosen from mxnet, pytorch, tensorflow.

PyTorch backend

Export `DGLBACKEND` as `pytorch` to specify PyTorch backend. The required PyTorch version is 1.12.0 or later. See pytorch.org for installation instructions.

MXNet backend

Export `DGLBACKEND` as `mxnet` to specify MXNet backend. The required MXNet version is 1.6 or later. See mxnet.apache.org for installation instructions.

MXNet uses uint32 as the default data type for integer tensors, which only supports graph of size smaller than 2^32. To enable large graph training, *build* MXNet with `USE_INT64_TENSOR_SIZE=1` flag. See this FAQ for more information.

MXNet 1.5 and later has an option to enable Numpy shape mode for `NDArray` objects, some DGL models need this mode to be enabled to run correctly. However, this mode may not compatible with pretrained model parameters with this mode disabled, e.g. pretrained models from GluonCV and GluonNLP. By setting `DGL_MXNET_SET_NP_SHAPE`, users can switch this mode on or off.

Tensorflow backend

Export `DGLBACKEND` as `tensorflow` to specify Tensorflow backend. The required Tensorflow version is 2.3.0 or later. See tensorflow.org for installation instructions. In addition, DGL will set `TF_FORCE_GPU_ALLOW_GROWTH` to `true` to prevent Tensorflow take over the whole GPU memory: