Install Ultralytics

Ultralytics offers a variety of installation methods, including pip, conda, and Docker. You can install YOLO via the `ultralytics` pip package for the latest stable release, or by cloning the Ultralytics GitHub repository for the most current version. Docker is also an option to run the package in an isolated container, which avoids local installation.

Watch: Ultralytics YOLO Quick Start Guide

Install

Pip install (recommended)

Install or update the `ultralytics` package using pip by running `pip install -U ultralytics`. For more details on the `ultralytics` package, visit the Python Package Index (PyPI).


```
# Install the ultralytics package from PyPI
pip install ultralytics
```

You can also install `ultralytics` directly from the Ultralytics GitHub repository. This can be useful if you want the latest development version. Ensure you have the Git command-line tool installed, and then run:

```
# Install the ultralytics package from GitHub
pip install git+https://github.com/ultralytics/ultralytics.git@main
```

Conda install

Conda can be used as an alternative package manager to pip. For more details, visit Anaconda. The Ultralytics feedstock repository for updating the conda package is available at GitHub.


```
# Install the ultralytics package using conda
conda install -c conda-forge ultralytics
```

Note

If you are installing in a CUDA environment, it is best practice to install `ultralytics`, `pytorch`, and `pytorch-cuda` in the same command. This allows the conda package manager to resolve any conflicts. Alternatively, install `pytorch-cuda` last to override the CPU-specific `pytorch` package if necessary.

```
# Install all packages together using conda
conda install -c pytorch -c nvidia -c conda-forge pytorch torchvision pytorch-cuda=11.8 ultralytics
```

Conda Docker Image

Ultralytics Conda Docker images are also available from DockerHub. These images are based on Miniconda3 and provide a straightforward way to start using `ultralytics` in a Conda environment.

```
# Set image name as a variable
t=ultralytics/ultralytics:latest-conda

# Pull the latest ultralytics image from Docker Hub
sudo docker pull $t

# Run the ultralytics image in a container with GPU support
sudo docker run -it --ipc=host --gpus all $t            # all GPUs
sudo docker run -it --ipc=host --gpus '"device=2,3"' $t # specify GPUs
```

Git clone

Clone the Ultralytics GitHub repository if you are interested in contributing to development or wish to experiment with the latest source code. After cloning, navigate into the directory and install the package in editable mode `-e` using pip.


```
# Clone the ultralytics repository
git clone https://github.com/ultralytics/ultralytics

# Navigate to the cloned directory
cd ultralytics

# Install the package in editable mode for development
pip install -e .
```

Docker

Use Docker to execute the `ultralytics` package in an isolated container, ensuring consistent performance across various environments. By selecting one of the official `ultralytics` images from Docker Hub, you avoid the complexity of local installation and gain access to a verified working environment. Ultralytics offers five main supported Docker images, each designed for high compatibility and efficiency:

- **Dockerfile:** GPU image recommended for training.
- **Dockerfile-arm64:** Optimized for ARM64 architecture, suitable for deployment on devices like Raspberry Pi and other ARM64-based platforms.
- **Dockerfile-cpu:** Ubuntu-based CPU-only version, suitable for inference and environments without GPUs.
- **Dockerfile-jetson:** Tailored for NVIDIA Jetson devices, integrating GPU support optimized for these platforms.
- **Dockerfile-python:** Minimal image with just Python and necessary dependencies, ideal for lightweight applications and development.
- **Dockerfile-conda:** Based on Miniconda3 with a conda installation of the `ultralytics` package.

Here are the commands to get the latest image and execute it:

```
# Set image name as a variable
t=ultralytics/ultralytics:latest

# Pull the latest ultralytics image from Docker Hub
sudo docker pull $t

# Run the ultralytics image in a container with GPU support
sudo docker run -it --ipc=host --gpus all $t            # all GPUs
sudo docker run -it --ipc=host --gpus '"device=2,3"' $t # specify GPUs
```

The above command initializes a Docker container with the latest `ultralytics` image. The `-it` flags assign a pseudo-TTY and keep stdin open, allowing interaction with the container. The `--ipc=host` flag sets the IPC (Inter-Process Communication) namespace to the host, which is essential for sharing memory between processes. The `--gpus all` flag enables access to all available GPUs inside the container, crucial for tasks requiring GPU computation.

Note: To work with files on your local machine within the container, use Docker volumes to mount a local directory into the container:

```
# Mount local directory to a directory inside the container
sudo docker run -it --ipc=host --gpus all -v /path/on/host:/path/in/container $t
```

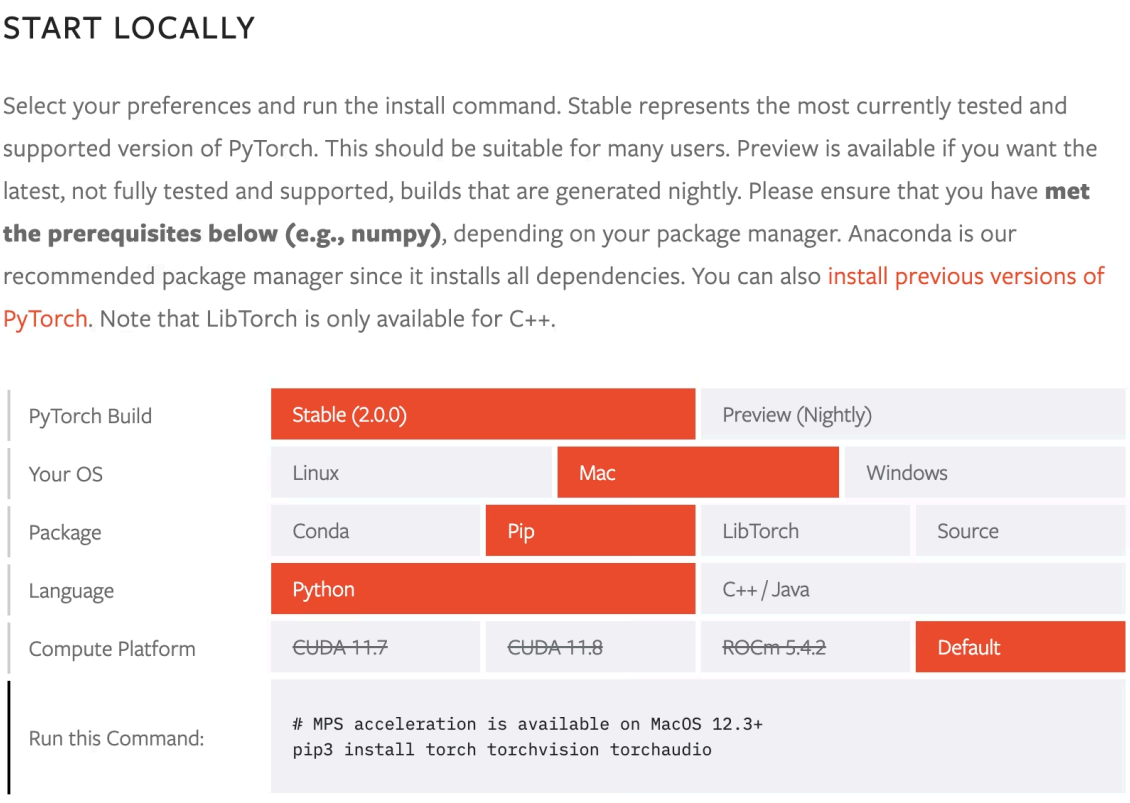
Replace `/path/on/host` with the directory path on your local machine, and `/path/in/container` with the desired path inside the Docker container.

For advanced Docker usage, explore the Ultralytics Docker Guide.

See the `ultralytics` pyproject.toml file for a list of dependencies. Note that all examples above install all required dependencies.

Tip

PyTorch requirements vary by operating system and CUDA requirements, so install PyTorch first by following the instructions at PyTorch.



Custom Installation Methods

While the standard installation methods cover most use cases, you might need a more tailored setup. This could involve installing specific package versions, omitting optional dependencies, or substituting packages like replacing `opencv-python` with the GUI-less `opencv-python-headless` for server environments.

Custom Methods

Method 1: Install without dependencies (`--no-deps`)

You can install the `ultralytics` package core without any dependencies using pip's `--no-deps` flag. This requires you to manually install all necessary dependencies afterward.

1. **Install `ultralytics` core:**

```
pip install ultralytics --no-deps
```

2. **Manually install dependencies:** You need to install all required packages listed in the `pyproject.toml` file, substituting or modifying versions as needed. For the headless OpenCV example:

```
# Install other core dependencies
pip install torch torchvision numpy matplotlib pandas pyyaml pillow psutil
requests tqdm scipy seaborn ultralytics-thop

# Install headless OpenCV instead of the default
pip install opencv-python-headless
```

Dependency Management

This method gives full control but requires careful management of dependencies. Ensure all required packages are installed with compatible versions by referencing the `ultralytics` `pyproject.toml` file.

Method 2: Install from a Custom Fork

If you need persistent custom modifications (like always using `opencv-python-headless`), you can fork the Ultralytics repository, make changes to `pyproject.toml` or other code, and install from your fork.

1. **Fork** the Ultralytics GitHub repository to your own GitHub account.
2. **Clone** your fork locally:

```
git clone https://github.com/YOUR_USERNAME/ultralytics.git
cd ultralytics
```

3. **Create a new branch** for your changes:

```
git checkout -b custom-opencv
```

4. **Modify `pyproject.toml`:** Open `pyproject.toml` in a text editor and replace the line containing `"opencv-python>=4.6.0"` with `"opencv-python-headless>=4.6.0"` (adjust version as needed).
5. **Commit and push** your changes:

```
git add pyproject.toml
git commit -m "Switch to opencv-python-headless"
git push origin custom-opencv
```

6. **Install** using pip with the `git+https` syntax, pointing to your branch:

```
pip install git+https://github.com/YOUR_USERNAME/ultralytics.git@custom-opencv
```

This method ensures that your custom dependency set is used whenever you install from this specific URL. See Method 4 for using this in a `requirements.txt` file.

Method 3: Local Clone, Modify, and Install

Similar to the standard "Git Clone" method for development, you can clone the repository locally, modify dependency files *before* installation, and then install in editable mode.

1. **Clone** the Ultralytics repository:

```
git clone https://github.com/ultralytics/ultralytics
cd ultralytics
```

2. **Modify `pyproject.toml`:** Edit the file to make your desired changes. For example, use `sed` (on Linux/macOS) or a text editor to replace `opencv-python` with `opencv-python-headless`. *Using `sed` (verify the exact line in `pyproject.toml` first):*

```
# Example: Replace the line starting with "opencv-python..."
# Adapt the pattern carefully based on the current file content
sed -i'' -e 's/^\s*"opencv-python>=.*","opencv-python-headless>=4.8.0",/'
pyproject.toml
```

   Or manually edit `pyproject.toml` to change `"opencv-python>=...` to `"opencv-python-headless>=..."`.
3. **Install** the package in editable mode (`-e`). Pip will now use your modified `pyproject.toml` to resolve and install dependencies:

```
pip install -e .
```

This approach is useful for testing local changes to dependencies or build configurations before committing them or for setting up specific development environments.

Method 4: Use `requirements.txt`

If you manage your project dependencies using a `requirements.txt` file, you can specify your custom Ultralytics fork directly within it. This ensures that anyone setting up the project gets your specific version with its modified dependencies (like `opencv-python-headless`).

1. **Create or edit `requirements.txt`:** Add a line pointing to your custom fork and branch (as prepared in Method 2).

   requirements.txt

   ```
   # Core dependencies
   numpy
   matplotlib
   pandas
   pyyaml
   Pillow
   psutil
   requests>=2.23.0
   tqdm
   torch>=1.8.0 # Or specific version/variant
   torchvision>=0.9.0 # Or specific version/variant

   # Install ultralytics from a specific git commit or branch
   # Replace YOUR_USERNAME and custom-branch with your details
   git+https://github.com/YOUR_USERNAME/ultralytics.git@custom-branch

   # Other project dependencies
   flask
   # ... etc
   ```

   Note: You don't need to list dependencies already required by your custom fork (like) here, as pip will install them based on the fork's .

2. **Install** dependencies from the file:

   ```
   pip install -r requirements.txt
   ```

This method integrates seamlessly with standard Python project dependency management workflows while allowing you to pin `ultralytics` to your customized Git source.

Use Ultralytics with CLI

The Ultralytics command-line interface (CLI) allows for simple single-line commands without needing a Python environment. CLI requires no customization or Python code; run all tasks from the terminal with the `yolo` command. For more on using YOLO from the command line, see the CLI Guide.

Example

Syntax

Ultralytics `yolo` commands use the following syntax:

```
yolo TASK MODE ARGS
```

- `TASK` (optional) is one of (detect, segment, classify, pose, obb) - `MODE` (required) is one of (train, val, predict, export, track, benchmark) - `ARGS` (optional) are `arg=value` pairs like `imgsz=640` that override defaults.

See all `ARGS` in the full Configuration Guide or with the `yolo cfg` CLI command.

Train

Train a detection model for 10 epochs with an initial learning rate of 0.01:

```
yolo train data=coco8.yaml model=yolo11n.pt epochs=10 lr0=0.01
```

Predict

Predict a YouTube video using a pretrained segmentation model at image size 320:

```
yolo predict model=yolo11n-seg.pt source='https://youtu.be/LNwODJXcvt4' imgsz=320
```

Val

Validate a pretrained detection model with a batch size of 1 and image size of 640:

```
yolo val model=yolo11n.pt data=coco8.yaml batch=1 imgsz=640
```

Export

Export a YOLOv11n classification model to ONNX format with an image size of 224x128 (no TASK required):

```
yolo export model=yolo11n-cls.pt format=onnx imgsz=224,128
```

Count

Count objects in a video or live stream using YOLO11:

```
yolo solutions count show=True

yolo solutions count source="path/to/video.mp4" # specify video file path
```

Workout

Monitor workout exercises using a YOLO11 pose model:

```
yolo solutions workout show=True

yolo solutions workout source="path/to/video.mp4" # specify video file path

# Use keypoints for ab-workouts
yolo solutions workout kpts="[5, 11, 13]" # left side
yolo solutions workout kpts="[6, 12, 14]" # right side
```

Queue

Use YOLO11 to count objects in a designated queue or region:

```
yolo solutions queue show=True

yolo solutions queue source="path/to/video.mp4" # specify video file path

yolo solutions queue region="[(20, 400), (1080, 400), (1080, 360), (20, 360)]" #
configure queue coordinates
```

Inference with Streamlit

Perform object detection, instance segmentation, or pose estimation in a web browser using Streamlit:

```
yolo solutions inference

yolo solutions inference model="path/to/model.pt" # use model fine-tuned with
Ultralytics Python package
```

Special

Run special commands to see the version, view settings, run checks, and more:

```
yolo help
yolo checks
yolo version
yolo settings
yolo copy-cfg
yolo cfg
yolo solutions help
```

Warning

Arguments must be passed as `arg=value` pairs, split by an equals `=` sign and delimited by spaces. Do not use `--` argument prefixes or commas `,` between arguments.

- `yolo predict model=yolo11n.pt imgsz=640 conf=0.25` ✅
- `yolo predict model yolo11n.pt imgsz 640 conf 0.25` ❌ (missing `=`)
- `yolo predict model=yolo11n.pt, imgsz=640, conf=0.25` ❌ (do not use `,`)
- `yolo predict --model yolo11n.pt --imgsz 640 --conf 0.25` ❌ (do not use `--`)
- `yolo solution model=yolo11n.pt imgsz=640 conf=0.25` ❌ (use `solutions`, not `solution`)

CLI Guide

Use Ultralytics with Python

The Ultralytics YOLO Python interface offers seamless integration into Python projects, making it easy to load, run, and process model outputs. Designed for simplicity, the Python interface allows users to quickly implement object detection, segmentation, and classification. This makes the

YOLO Python interface an invaluable tool for incorporating these functionalities into Python projects.

For instance, users can load a model, train it, evaluate its performance, and export it to ONNX format with just a few lines of code. Explore the Python Guide to learn more about using YOLO within your Python projects.

Example

```python
from ultralytics import YOLO

# Create a new YOLO model from scratch
model = YOLO("yolo11n.yaml")

# Load a pretrained YOLO model (recommended for training)
model = YOLO("yolo11n.pt")

# Train the model using the 'coco8.yaml' dataset for 3 epochs
results = model.train(data="coco8.yaml", epochs=3)

# Evaluate the model's performance on the validation set
results = model.val()

# Perform object detection on an image using the model
results = model("https://ultralytics.com/images/bus.jpg")

# Export the model to ONNX format
success = model.export(format="onnx")
```

Python Guide

Ultralytics Settings

The Ultralytics library includes a `SettingsManager` for fine-grained control over experiments, allowing users to access and modify settings easily. Stored in a JSON file within the environment's user configuration directory, these settings can be viewed or modified in the Python environment or via the Command-Line Interface (CLI).

Inspecting Settings

To view the current configuration of your settings:

View settings

Python

Use Python to view your settings by importing the `settings` object from the `ultralytics` module. Print and return settings with these commands:

```python
from ultralytics import settings

# View all settings
print(settings)

# Return a specific setting
value = settings["runs_dir"]
```

CLI

The command-line interface allows you to check your settings with:

```
yolo settings
```

Modifying Settings

Ultralytics makes it easy to modify settings in the following ways:

Update settings

Python

In Python, use the `update` method on the `settings` object:

```
from ultralytics import settings

# Update a setting
settings.update({"runs_dir": "/path/to/runs"})

# Update multiple settings
settings.update({"runs_dir": "/path/to/runs", "tensorboard": False})

# Reset settings to default values
settings.reset()
```

CLI

To modify settings using the command-line interface:

```
# Update a setting
yolo settings runs_dir='/path/to/runs'

# Update multiple settings
yolo settings runs_dir='/path/to/runs' tensorboard=False

# Reset settings to default values
yolo settings reset
```

Understanding Settings

The table below overviews the adjustable settings within Ultralytics, including example values, data types, and descriptions.

Name	Example Value	Data Type	Description
`settings_version`	`'0.0.4'`	`str`	Ultralytics *settings* version (distinct from the Ultralytics pip version)

Name	Example Value	Data Type	Description
`datasets_dir`	`'/path/to/datasets'`	`str`	Directory where datasets are stored
`weights_dir`	`'/path/to/weights'`	`str`	Directory where model weights are stored
`runs_dir`	`'/path/to/runs'`	`str`	Directory where experiment runs are stored
`uuid`	`'a1b2c3d4'`	`str`	Unique identifier for the current settings
`sync`	`True`	`bool`	Option to sync analytics and crashes to Ultralytics HUB
`api_key`	`''`	`str`	Ultralytics HUB API Key
`clearml`	`True`	`bool`	Option to use ClearML logging
`comet`	`True`	`bool`	Option to use Comet ML for experiment tracking and visualization
`dvc`	`True`	`bool`	Option to use DVC for experiment tracking and version control
`hub`	`True`	`bool`	Option to use Ultralytics HUB integration
`mlflow`	`True`	`bool`	Option to use MLFlow for experiment tracking
`neptune`	`True`	`bool`	Option to use Neptune for experiment tracking
`raytune`	`True`	`bool`	Option to use Ray Tune for hyperparameter tuning
`tensorboard`	`True`	`bool`	Option to use TensorBoard for visualization
`wandb`	`True`	`bool`	Option to use Weights & Biases logging
`vscode_msg`	`True`	`bool`	When a VS Code terminal is detected, enables a prompt to download the Ultralytics-Snippets extension.

Revisit these settings as you progress through projects or experiments to ensure optimal configuration.

FAQ

How do I install Ultralytics using pip?

Install Ultralytics with pip using:

```
pip install ultralytics
```

This installs the latest stable release of the `ultralytics` package from PyPI. To install the development version directly from GitHub:

```
pip install git+https://github.com/ultralytics/ultralytics.git
```

Ensure the Git command-line tool is installed on your system.

Can I install Ultralytics YOLO using conda?

Yes, install Ultralytics YOLO using conda with:

```
conda install -c conda-forge ultralytics
```

This method is a great alternative to pip, ensuring compatibility with other packages. For CUDA environments, install `ultralytics`, `pytorch`, and `pytorch-cuda` together to resolve conflicts:

```
conda install -c pytorch -c nvidia -c conda-forge pytorch torchvision pytorch-
cuda=11.8 ultralytics
```

For more instructions, see the Conda quickstart guide.

What are the advantages of using Docker to run Ultralytics YOLO?

Docker provides an isolated, consistent environment for Ultralytics YOLO, ensuring smooth performance across systems and avoiding local installation complexities. Official Docker images are available on Docker Hub, with variants for GPU, CPU, ARM64, NVIDIA Jetson, and Conda. To pull and run the latest image:

```
# Pull the latest ultralytics image from Docker Hub
sudo docker pull ultralytics/ultralytics:latest

# Run the ultralytics image in a container with GPU support
sudo docker run -it --ipc=host --gpus all ultralytics/ultralytics:latest
```

For detailed Docker instructions, see the Docker quickstart guide.

How do I clone the Ultralytics repository for development?

Clone the Ultralytics repository and set up a development environment with:

```
# Clone the ultralytics repository
git clone https://github.com/ultralytics/ultralytics

# Navigate to the cloned directory
cd ultralytics

# Install the package in editable mode for development
pip install -e .
```

This allows contributions to the project or experimentation with the latest source code. For details, visit the Ultralytics GitHub repository.

Why should I use Ultralytics YOLO CLI?

The Ultralytics YOLO CLI simplifies running object detection tasks without Python code, enabling single-line commands for training, validation, and prediction directly from your terminal. The basic syntax is:

```
yolo TASK MODE ARGS
```

For example, to train a detection model:

```
yolo train data=coco8.yaml model=yolo11n.pt epochs=10 lr0=0.01
```

Explore more commands and usage examples in the full CLI Guide.