Local Install with VSCode

If you want to use VSCode, you can still follow up all the steps for “local Install” and then use VS Code to open, run and execute the notebooks. It gives a similar experience to a local install but with some additional capabilities. You may also use this if you prefer the richer GUI experience VS Code offers and if VS Code is already a part of your developer experience. High level steps to do are as follows:

1. Install VS Code on your local machine. Readers may refer to the link https://code.visualstudio.com/ for detailed setup instructions.

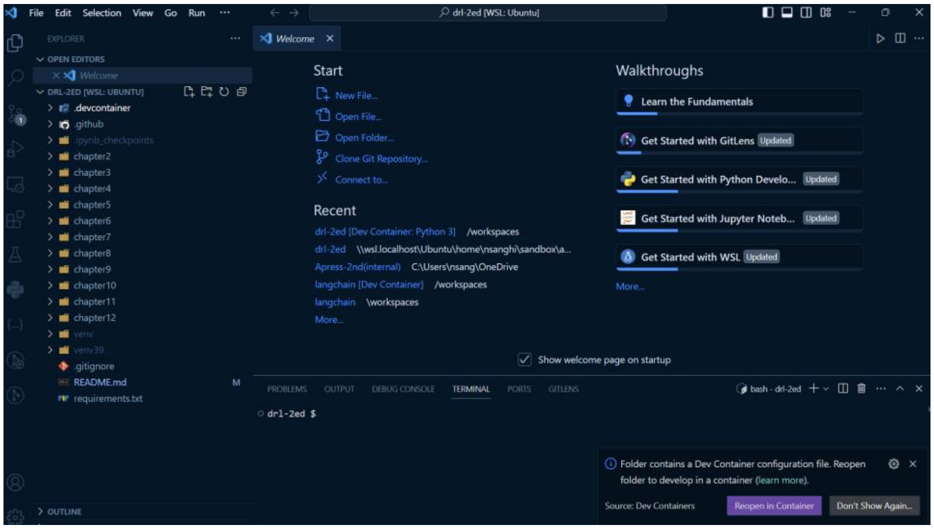
2. Next, we need to enable a few plugins for Python: Python, Python Debugger, PyLance, Jupyter, Jupyter Notebook Renderers and Jupyter Keymap, Jupyter Cell Tags and Jupyter Slide Show. Some of these are optional but installing all of them will give you a richer experience. You can read through each individual plugin and decide which ones you want to skip.

3. Navigate to the `drl-2ed` directory where you cloned/downloaded the source code. Run the following command:

   ```
   code .
   ```

   Do not forget the period (.) at the end of the command. This command instructs the system to open VS Code in the current folder.

4. You may see a pop-up at the bottom right corner with the message “*Folder contains Dev Container configuration file. Reopen folder to develop in a container*”. Please do not click on the button “*Reopen in Container*”. Instead click on the cross at the top right of the pop-up to close it.

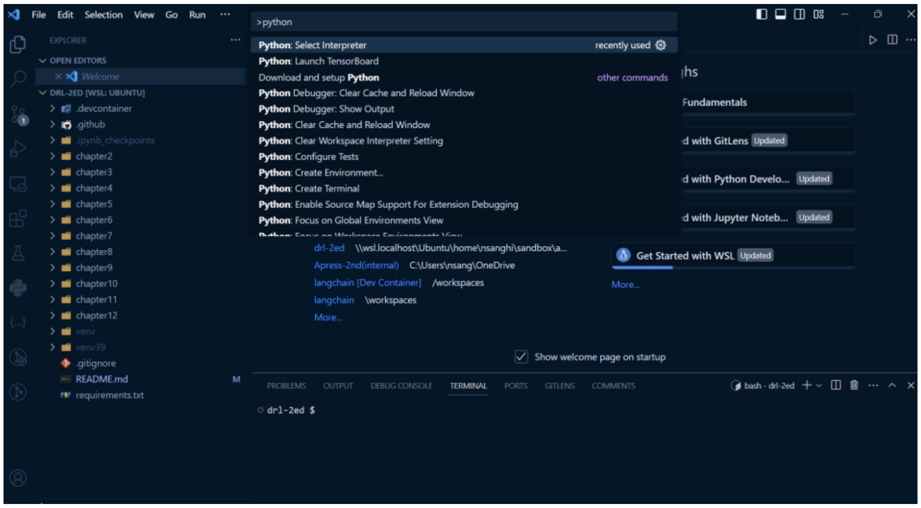


```
drl-2ed $
```

Folder contains a Dev Container configuration file. Reopen folder to develop in a container (learn more).

Figure 1-3 Opening in VS Code

5. Next, Press “Control+Shift+P” or “command+Shift+P” to open the command palette in VS Code and type “Python” to see a list of commands applicable for Python in VS Code. Choose the option “*Python: Select Interpreter*” as shown in Figure 1-4.



```
>python
```

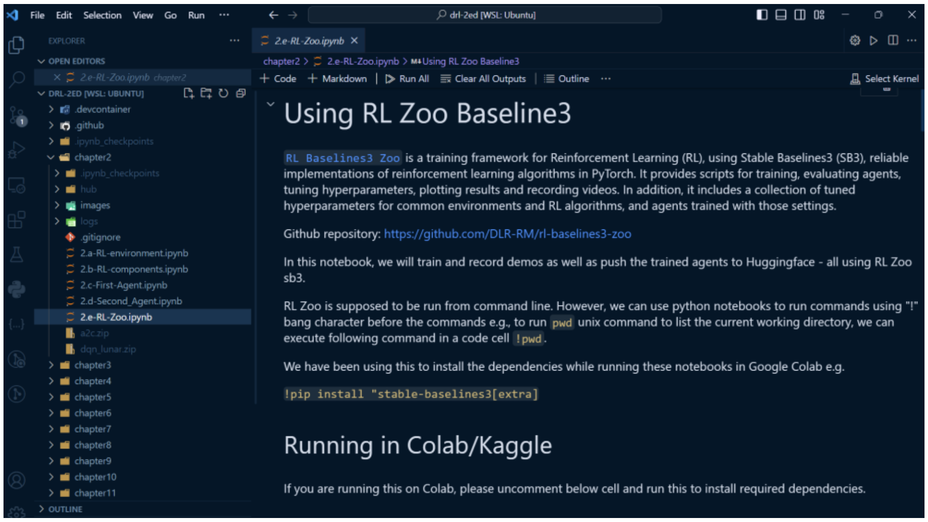
Python: Select Interpreter recently used
Python: Launch TensorBoard
Download and setup Python other commands
Python Debugger: Clear Cache and Reload Window
Python Debugger: Show Output
Python: Clear Cache and Reload Window
Python: Clear Workspace Interpreter Setting
Python: Configure Tests
Python: Create Environment...
Python: Create Terminal
Python: Enable Source Map Support For Extension Debugging
Python: Focus on Global Environments View
File Edit Selection View Go Run ···
EXPLORER
OPEN EDITORS
Welcome
DRL-2ED [WSL: UBUNTU]
.devcontainer
.github
.ipynb_checkpoints
chapter2
chapter3
chapter4
chapter5
chapter6
chapter7
chapter8
chapter9
chapter10
chapter11
chapter12
venv
venv39
.gitignore
README.md
requirements.txt
drl-2ed \\wsl.localhost\Ubuntu\home\nsanghi\sandbox\a...
Apress-2nd(internal) C:\Users\nsang\OneDrive
langchain [Dev Container] /workspaces
langchain \workspaces
More...
Fundamentals
d with GitLens Updated
d with Python Develo... Updated
d with Jupyter Noteb... Updated
Get Started with WSL Updated
More...
☑ Show welcome page on startup
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS GITLENS COMMENTS
bash - drl-2ed

```
drl-2ed $
```

Figure 1-4 Selceting Python Enviroment in VSCode

It will show all the python environments found in your system. Please choose the `venv` or `conda` environment you created for local install. Either you will find this option listed in the command palette or you can add it using the option "*Enter interpreter path…*"

6. You may now open any notebook. At times, even after you have chosen the Python environment in the previous step, you again may need to select the same environment once more as the notebook kernel before you can execute the code in the notebook. You will find that option on the top right side of the notebook and it will read as "*Select Kernel*". Please click on this and again select the same Python environment that you created for local install and the one you selected as Python Interpreter in previous Step 5.



Using RL Zoo Baseline3

`RL Baselines3 Zoo` is a training framework for Reinforcement Learning (RL), using Stable Baselines3 (SB3), reliable implementations of reinforcement learning algorithms in PyTorch. It provides scripts for training, evaluating agents, tuning hyperparameters, plotting results and recording videos. In addition, it includes a collection of tuned hyperparameters for common environments and RL algorithms, and agents trained with those settings.

Github repository: https://github.com/DLR-RM/rl-baselines3-zoo

In this notebook, we will train and record demos as well as push the trained agents to Huggingface - all using RL Zoo sb3.

RL Zoo is supposed to be run from command line. However, we can use python notebooks to run commands using "!" bang character before the commands e.g., to run `pwd` unix command to list the current working directory, we can execute following command in a code cell `!pwd`.

We have been using this to install the dependencies while running these notebooks in Google Colab e.g.

```
!pip install "stable-baselines3[extra]
```

Running in Colab/Kaggle

If you are running this on Colab, please uncomment below cell and run this to install required dependencies.

Figure 1-5 Selceting Notebook Kernel in VSCode

You are ready to explore the notebooks and execute the code cells contained in the Jupyter notebook.

Running on Google Colab (recommended for cloud option)

Google Colab is a free online platform that allows you to write and run Python code in your browser. It is based on Jupyter Notebook. Google Colab provides some advantages over Jupyter Notebook, such as:

- You can access Google Colab from any device with an internet connection, without installing anything on your local machine.
- You can use the computing power of Google's servers, including GPUs and TPUs, to speed up your code execution and save resources on your own device.
- You can easily share your notebooks with others and collaborate on them in real time.
- You can integrate your notebooks with Google Drive and other Google services, such as Google Sheets, Google Forms, and Google Maps.

To use Google Colab, you need a Google account and a web browser. You can create a new notebook by visiting https://colab.research.google.com/ or opening an existing notebook from Google Drive. You can also upload a notebook file from your local machine or import one from GitHub or other sources. Once you have a notebook open, you can write and execute code cells, add text and images, and use various tools and features provided by Google Colab. You can also customize your notebook settings, such as the runtime type, the theme, and the keyboard shortcuts.

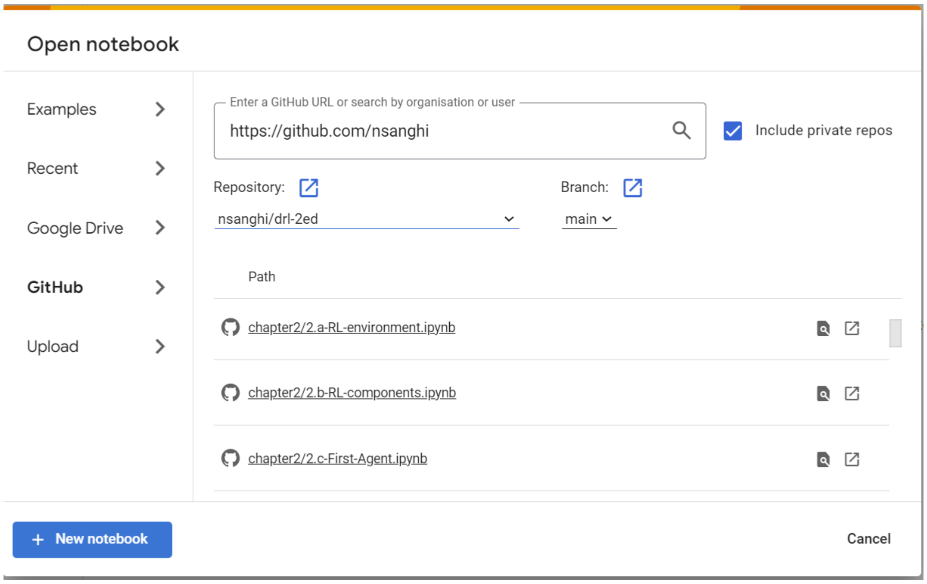
We will use GitHub option to directly open notebooks from the github repository. It is just a matter of personal preference. And if you have cloned the source code repository in github to your github account, you will also be able to make changes and issue commits to your repo giving you a seamless developer experience without any local install on your computer. Let us walk through the steps.

1. Navigate to https://colab.research.google.com/ which will open the webpage with a dialog box giving you various options to open a notebook. In our current workflow, we will choose the github option.

2. If this is the first time you are using github from inside the Colaboratory (Google Colab), clicking on github option on the dialog box will take you through an oAuth workflow to grant Google Colab access to your github account.

3. After the authentication is done and Google Colab is connected to your github account, you can use the dialog box to open a specific notebook of interest or you can directly type the url of github account housing the repository, e.g. https://github.com/nsanghi/.

4. Click on the repository dropdown to navigate to the repository. Keep the branch selection to “main” or whatever else is applicable for the repository you are trying to open. A sample of the dialog at this point is shown in Figure 1-6.



Enter a GitHub URL or search by organisation or user: https://github.com/nsanghi

☑ Include private repos

Repository: nsanghi/drl-2ed

Branch: main

Path

chapter2/2.a-RL-environment.ipynb

chapter2/2.b-RL-components.ipynb

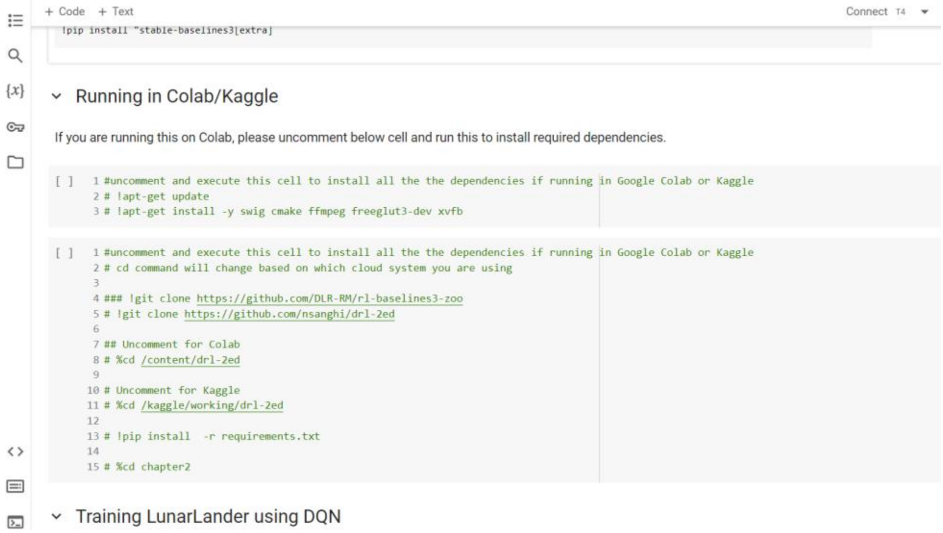
chapter2/2.c-First-Agent.ipynb

Figure 1-6 Selceting the repository and notebook in Goolge Colab

5. Select the notebook e.g. `chapter2/2.a-RL-envrionment.ipynb`. It will close the dialog and open the notebook in the browser with an interface very similar to Jupyter lab interface. There are some Google Colab specific enhancements, but most are not very relevant for our case. The only thing to note is the run time type that you can chose from the drop down labelled "Connect" on the top right of the Colab web page.

   We will be mostly using CPU environment as free tier of Google Colab gives limited access to GPU/TPU environment. Most of the notebooks can be run on a CPU environment. However, Chapter 6 onwards, once we enter the realm of Deep Reinforcement Learning and start training agents on either large text data (such as RLHF) or images (such as Atari games), we may run those notebooks with GPU enabled.

6. Google Colab comes with most of the libraries pre-installed. However, we have designed the notebooks such that you may run some additional commands from inside the notebook to install system level packages as well as required python libraries. An example of the same is the notebook "`2.e-RL-Zoo.ipynb`". Let us open this notebook in Google Colab following instructions from Step 4. Once it is opened, you will see two code execution cells which have code commented out as shown in Figure 1-7.



```
!pip install "stable-baselines3[extra]
```

Running in Colab/Kaggle

If you are running this on Colab, please uncomment below cell and run this to install required dependencies.

```
[ ]
#uncomment and execute this cell to install all the the dependencies if running in Google Colab or Kaggle
# !apt-get update
# !apt-get install -y swig cmake ffmpeg freeglut3-dev xvfb
```

```
[ ]
#uncomment and execute this cell to install all the the dependencies if running in Google Colab or Kaggle
# cd command will change based on which cloud system you are using

### !git clone https://github.com/DLR-RM/rl-baselines3-zoo
# !git clone https://github.com/nsanghi/drl-2ed

## Uncomment for Colab
# %cd /content/drl-2ed

# Uncomment for Kaggle
# %cd /kaggle/working/drl-2ed

# !pip install  -r requirements.txt

# %cd chapter2
```

Training LunarLander using DQN

Figure 1-7 Running code cells for package installation in Goolge Colab

The first cell installs the Ubuntu system level packages. The second cell clones the github repository and runs the requirements.txt file from cloned repository to install all the dependencies.

Please uncomment and run these cells before proceeding further with the notebook.

Please note that you do not need to run the commented code cells for any kind of local installation-based execution.

Running on Kaggle

Just like Google Colab, you may also use a similar functionality offered by Kaggle.
Kaggle is a platform that hosts data science and machine learning competitions, where you can find and explore various datasets, learn new skills, and share your solutions with other users. Kaggle also provides a cloud-based service called Kaggle Kernel, which allows you to run notebooks online without installing anything on your local machine. You can use Kaggle Kernel to access the datasets from the competitions, run code in Python or R, and publish your results as interactive web pages. Kaggle Kernel is a convenient way to experiment with different models and techniques, collaborate with other users, and showcase your work. You can also fork and edit existing notebooks from other users and use them as a starting point for your own projects.

Running the notebook from this book on Kaggle is pretty similar to the way you do on Google Colab. You can read more about it in the Kaggle documentation and try this out.

Using devcontainer based environments

Another way to run notebooks is to use devcontainer, which stands for development container. A ".devcontainer" is a folder in a repository that contains a Docker file and a configuration file that specify the tools and settings for a development environment. You can use ".devcontainer" to create a consistent and reproducible workspace that can run on any machine that supports Docker and Visual Studio Code (VS Code).

To use devcontainer, you need to have Docker and VS Code installed on your local machine or cloud service. You also need to install the Remote Development extension pack for VS Code, which enables you to work with remote environments. Then, you can find a repository that has a ".devcontainer" folder, which indicates that it supports devcontainer.

Once you have a devcontainer repository, you can clone it to your local machine or cloud service and open it with VS Code. VS Code will detect the .devcontainer folder and prompt you to reopen the folder in a container. This will build the Docker image and launch the container with the specified tools and settings. You can then open the notebook file and run it inside the container. You can also modify the code and push the changes back to the repository.

Using devcontainer is a convenient way to work with notebooks that require specific dependencies or configurations, without affecting your local machine or cloud service. You can also share your devcontainer setup with other users and ensure that they have the same development environment as you. Devcontainer is a useful feature for collaborating and deploying code. You can read more about it in the VS Code documentation and try it out.

The source code repository for the book has this capability enabled. We will look at two ways to use this. One will be using a local system, and another will be using github's Codespace which provides a cloud environment to run code.

Running devcontainer Locally

As discussed previously, you need to have Docker and VS Code installed locally on your machine. To install Docker, you may head over to the link https://www.docker.com/products/docker-desktop/ and install the Docker Desktop version for your system. Download and run the binary for your platform and follow all the prompts choosing the recommended/default options. VS Code can be installed from https://code.visualstudio.com/ . Once VS code is installed, please also install the Remote Development extension pack for VS Code. With these steps your local machine is now ready for local deployment of a docker container. The advantage of this approach is that your development environment is housed in a docker container, completely isolating your development environment from your local machine and irrespective of the operating system of your local machine, the code will run the same docker configuration which in our case is a ubuntu image with all the required dependencies. Let us now walk through the steps of downloading the source code and running it locally.

1. Navigate to a folder of your choice on the local machine. If you are on a windows machine, you can either choose the WSL2 subsystem or the windows subsystem.

2. Clone the source code repository locally with the command:

   ```
   git clone https://github.com/nsanghi/drl-2ed.git
   ```

   This command will create another subfolder named “`drl-2ed`” in the current folder where you ran the above command. You can also visit the publisher’s source code link to clone the repository or download and unzip the code.

3. Navigate to the `drl-2ed` folder created with the above git command and run VS Code from there with the command:

   ```
   code .
   ```

   Do not forget the period (.) at the end of the command. This command instructs the system to open VS Code in the current folder.

4. You may see a pop-up at the right corner with the message “*Folder contains Dev Container configuration file. Reopen folder to develop in a container*”. **Please click on the button** “*Reopen in Container*” to kick in the docker build. It will use the development environment image specified inside the “.devcontainer” folder of the repository to create and start the docker container based on the specifications inside this folder. This is opposite to the instructions we had for running on VS Code locally wherein you did not click the button “Reopen in Container”.

First time you do this; it has to download the Docker image as well as install the required ubuntu packages and run the "`pip install requirements.txt`" command to setup the python environment. It may take a while, say about 10 mins or so for the first time to run the whole setup process. It will save the final docker container locally and make it visible in your docker desktop.

The subsequent runs will be very fast as it will use the saved docker container. In case you need to do a rebuild of docker environment, VS Code command palette has options to do so.

When the code is opened in a .devcontainer specified docker image, you will see a status like "Dev Container: Python 3" at the bottom left of the VS Code as shown in Figure 1-8. This confirms that you are now working inside a docker based dev container.

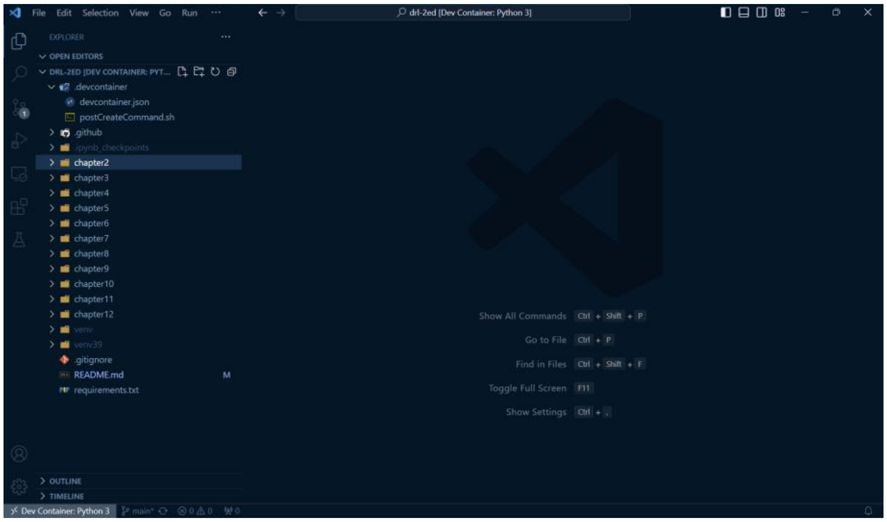
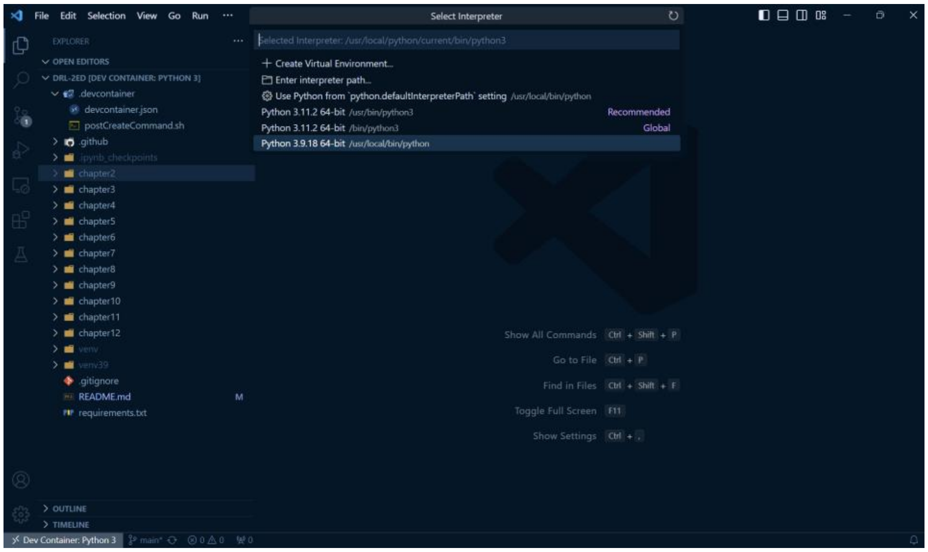


Figure 1-8 Running in Docker Container

5. Next, Press "Control+Shift+P" pr "command+Shift+P" to open the command palette in VS Code and type "Python" see a list of commands application for Python in VS Code. Choose the option "Python: Select Interpreter" as shown in Figure 1-9. This time the Python interpreter options will be different. Please choose the Python 3.9.18 version as highlighted in Figure 1-9.



Select Interpreter

Selected Interpreter: /usr/local/python/current/bin/python3

+ Create Virtual Environment...
Enter interpreter path...
Use Python from `python.defaultInterpreterPath` setting /usr/local/bin/python
Python 3.11.2 64-bit /usr/bin/python3 Recommended
Python 3.11.2 64-bit /bin/python3 Global
Python 3.9.18 64-bit /usr/local/bin/python

Figure 1-9 Selceting Python Enviroment in VSCode

6. You can open any notebook. You again may need to select the same environment as the notebook kernel before you can execute the code in the notebook. You will find that option on the top right side of the notebook and it will read as "*Select Kernel*". Please click on this and again select the same Python environment with Python 3.9.18.

   You can now execute the code just like the way you would do in a Jupyter Lab environment.

Running on Github Codespaces

GitHub Codespaces is a cloud-based development environment that allows you to create, edit, and run code from any device. You can access your codespaces from Visual Studio Code, a browser-based editor, or GitHub.com. GitHub Codespaces integrates seamlessly with GitHub features, such as pull requests, issues, actions, and packages. You can also customize your codespaces with your own dotfiles, extensions, and settings. The key advantages of using something like Codespaces are:

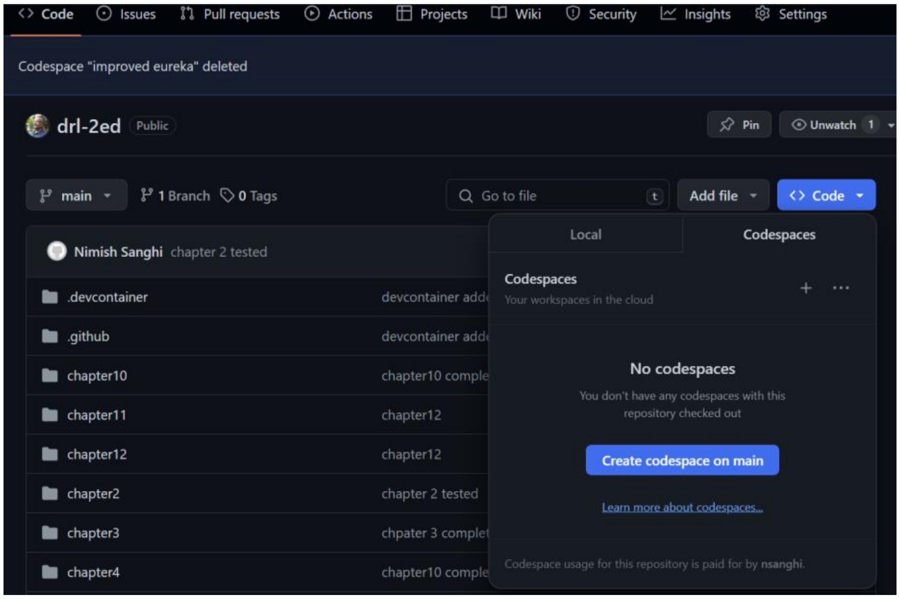
- You can start coding quickly without setting up a local environment or installing dependencies.

- You can work on multiple projects and switch between them easily without cluttering your machine.
- You can collaborate with others in real time and share your codespaces with teammates or contributors.
- You can use the same tools and workflows that you are familiar with, such as VS Code, Git, and terminal.
- You can leverage the power and scalability of the cloud and run your code on fast and secure servers.

You can start with codespaces on the free tier which will get you 60 hours per month of 2-core machines. Let us go through the steps involved in using Codespaces:

1. Navigate to the repository, e.g. https://github.com/nsanghi/drl-2ed/

2. Make sure you are logged into your github account and click on the dropdown arrow next to the blue button with text "Code". Next, you click on the button with text "Create codespace on main" as shown in Figure 1-10. It will open a new tab and start setting up the codespace using the docker container specifications provided in the ".devcontainer" folder of the repository.



Codespace "improved eureka" deleted

Nimish Sanghi chapter 2 tested

.devcontainer	devcontainer add
.github	devcontainer add
chapter10	chapter10 comple
chapter11	chapter12
chapter12	chapter12
chapter2	chapter 2 tested
chapter3	chpater 3 complet
chapter4	chapter10 comple

No codespaces

You don't have any codespaces with this repository checked out

Codespace usage for this repository is paid for by nsanghi.

Figure 1-10 Setting up Github Codespace

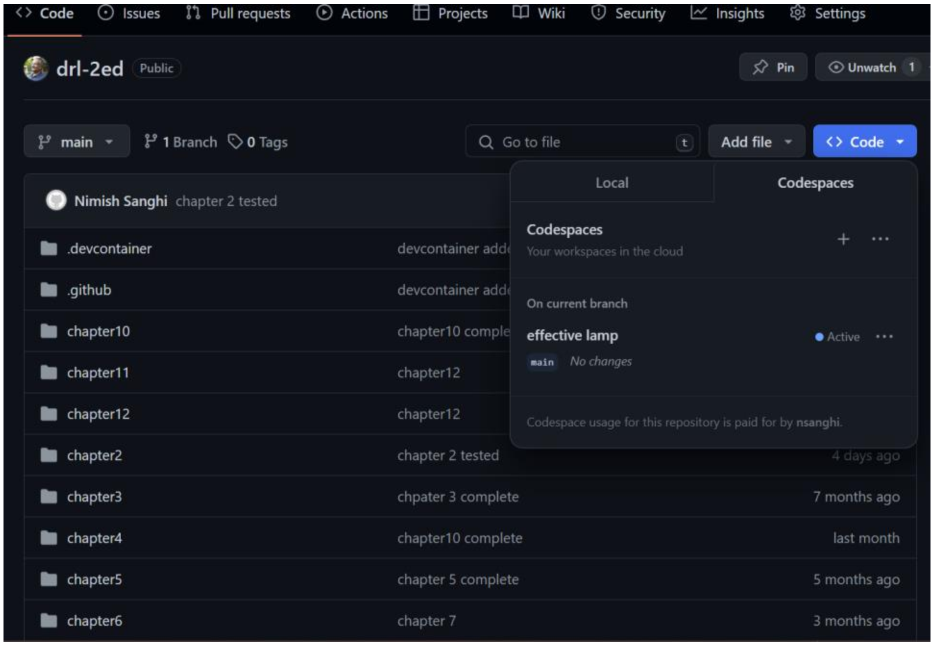
3. Once the container is set up, it will open the repository in a cloud version of VS Code and run the pip install to install all the python packages. Like the local

docker container setup, first time setup on codespaces may also take about 5-10 mins. However, subsequent runs will be faster as it will reuse the cloud saved docker container it created in the first run.

4. Similar to the option of running on local docker, you need to select the right Python interpreter with Python 3.9.18 as shown in Figure 1-9. You also need to select the same Python version as the kernel while running the notebooks.

5. You are now ready to run the code on Github codespace. This option gives you full developer experience without any local install.

6. As the docker image is stored in codespace and is tied to your github account, next time you visit the github repository and click on the dropdown next to code, you will see the previously created codespace listed as shown in Figure 1-11. You can use that to run code in subsequent visits. It will use the saved docker container and in a matter of seconds get your environment running.



Nimish Sanghi chapter 2 tested

Codespaces
Your workspaces in the cloud

On current branch
effective lamp ● Active
main No changes

Codespace usage for this repository is paid for by nsanghi.

.devcontainer	devcontainer add	
.github	devcontainer add	
chapter10	chapter10 comple	
chapter11	chapter12	
chapter12	chapter12	
chapter2	chapter 2 tested	4 days ago
chapter3	chpater 3 complete	7 months ago
chapter4	chapter10 complete	last month
chapter5	chapter 5 complete	5 months ago
chapter6	chapter 7	3 months ago

Figure 1-11 Setting up Github Codespace

7. You can click on the three dots on the right side of the container name and open the codespace container. You can also use the menu to open the cloud hosted codespace inside your local VS Code.

Running on AWS Studio Lab

AWS Studio Lab is a free tier version from AWS which can be used as an alternative to Google Colab. It provides you with certain hours of free CPU and GPU access. You need to create an account by navigating to the link https://studiolab.sagemaker.aws/ .

Once you have created the account and logged in, you can clone the github repository, create a `conda` environment and run the setup to install the ubuntu packages as well as python libraries specified in the `requirements.txt` file. The steps involved are as follows:

1. Once you are logged in, click on the button “Start runtime” to start a Jupyter instance.

2. After the runtime is running, click on the circle button “Open Project”. This will open a new tab with Jupyter Lab interface.

3. Next you need to clone the repository. Either you can open a terminal from the Jupyter Lab interface or click on the menu “Git”-> “Clone Git Repository”. Follow the instructions given in the dialog box to complete this step.

4. The next step is to create the `conda` environment with Python 3.9, install the ubuntu packages and also run the pip install. First open a terminal from the Jupyter Lab interface and run commands as follows:

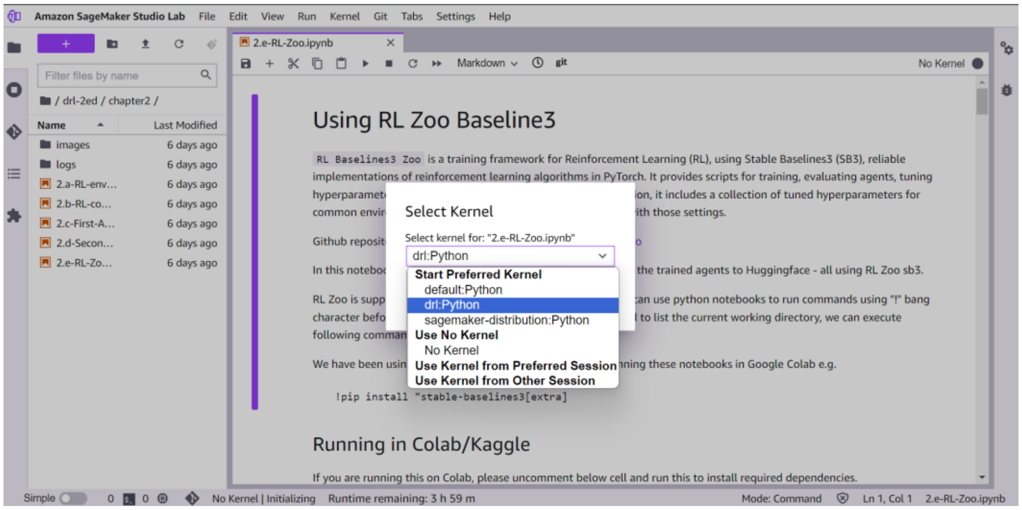
   ```
   # create conda environment and activate it
   conda create -n drl python=3.9.18
   conda activate drl

   # install required Linux packages
   conda install -c conda-forge swig cmake ffmpeg
   conda install conda-forge::xvfbwrapper
   conda install conda-forge::freeglut

   # Navigate to the drl-2ed folder and
   # install the required Python libraries
   pip install requirements.txt
   ```

5. You can now open any of the notebooks from the explorer on the left side. When presented with the option to choose the kernel, choose the option of “`drl`”, the

conda environment we created in previous step. In case you do not see the option, you may need to click on “Amazon SageMaker Studio Lab”-> “Restart JupyterLab” option.

6. Having selected the right kernel, you may start executing the notebook code cells. The conda environment you created persists between sessions making subsequent runs very fast.

Use of SageMaker Studio Lab allows you to run the code on cloud, just like a few other options that we have talked about in earlier sections. It requires no local install at all. Figure 1-12 shows a screenshot of AWS Studio Lab in action.



Name	Last Modified
images	6 days ago
logs	6 days ago
2.a-RL-env...	6 days ago
2.b-RL-co...	6 days ago
2.c-First-A...	6 days ago
2.d-Secon...	6 days ago
2.e-RL-Zo...	6 days ago

Using RL Zoo Baseline3

`RL Baselines3 Zoo` is a training framework for Reinforcement Learning (RL), using Stable Baselines3 (SB3), reliable implementations of reinforcement learning algorithms in PyTorch. It provides scripts for training, evaluating agents, tuning hyperparamet ion, it includes a collection of tuned hyperparameters for common envir ith those settings.

Github reposit o

In this noteboo the trained agents to Huggingface - all using RL Zoo sb3.

RL Zoo is supp an use python notebooks to run commands using "!" bang character befo to list the current working directory, we can execute following comman

We have been usin nning these notebooks in Google Colab e.g.

```
!pip install "stable-baselines3[extra]
```

Running in Colab/Kaggle

If you are running this on Colab, please uncomment below cell and run this to install required dependencies.

Select Kernel

Select kernel for: "2.e-RL-Zoo.ipynb"

drl:Python

Start Preferred Kernel
- default:Python
- drl:Python
- sagemaker-distribution:Python

Use No Kernel
- No Kernel

Use Kernel from Preferred Session

Use Kernel from Other Session

No Kernel | Initializing Runtime remaining: 3 h 59 m Mode: Command Ln 1, Col 1 2.e-RL-Zoo.ipynb

Figure 1-12 AWS Studio Lab

Running using Lightning.ai

Lightning AI Studio (https://lightning.ai/) is a cloud-based AI development platform (similar to Google Colab or Amazon Sagemaker Studio Lab) that aims to eliminate the hassle of setting up local environments for machine learning projects. Some key features of Lightning AI Studio are:

- It integrates popular machine learning tools into a single interface. This allows for building scalable AI apps and endpoints more easily.

- There is no environment setup required. You can code in the browser or connect your local IDE (VSCode or PyCharm). You can also easily switch between CPU and GPU with no environment changes.
- It allows hosting and sharing AI apps built with Streamlit, Gradio, React JS, etc. It also enables multi-user collaboration by coding together.
- It provides unlimited storage and the ability to upload and share files as well as connect S3 buckets.
- It enables training models at a massive scale using thousands of GPUs under paid plans.
- There is a growing list of community templates (Studios) ready for different use cases and different models which can act as your starting point.

Let us look at high level steps to run code on Lightning.ai studio.

1. First go the url https://lightning.ai/ and signup.

2. Next you click on "New Studio" button on the top right, choose defaults on the prompt and then click "Start".

3. Once a studio starts, you can click on the terminal on right side of the studio and use the terminal to a) install apt-get packages b) clone the source code repository and c) do "`pip install -r requirements.txt`" to install required Python packages and libraries.

4. At this point you are ready to start executing the Jupyter notebooks. At the time of writing, the free tier version offers a 4 CPU configuration making it a powerful experimentation platform in free tier offering. You also get access to GPU resources.

Other Options to run code

There are many more choices. In case you want to run the code on GPU machines for graphic games or 3-d environments, you can use many paid choices. A few of them are listed below. Depending on your need, you should refer to the documentation of the relevant platform to set it up for code execution:

1. Google Vertex AI notebooks
2. AWS Sagemaker
3. MS Azure ML Studio

4. Paperspace
5. Lambda labs
6. Jarvis Labs

There is a growing list of cloud GPU and CPU providers for Deep Learning workloads. For our purpose of running the source code of this book, any entry level configuration is good enough.