Final Qualification Test

Project Mozart
iOS Personal Voice/PlayHT/FastSpeech2 Synthetic Voice Detection and the Impacts of Noise

---

Contributing Authors: Branch Hill and Jacob Pomatto

Dec 7, 2023

Acronym	Meaning
AD	Audio Deepfake
ANN	Artificial Neural Network
AR-DAD	Arabic Diversified Audio Dataset
ASV	Automatic Speaker Verification
AWGN	Additive White Gaussian Noise
BPTT	Backpropagation Through Time
CNN	Convolution Neural Network
CQCC	Constant Q Cepstral Coefficients
DBiLSTM	Deep Bidirectional Long Short-Term Memory
DL	Deep Learning
DNN	Deep Neural Network
DT	Decision Tree
DTW	Dynamic Time Warping
EER	Equivalent Error Rate
FoR	Fake or Real
FQT	Final Qualification Test
GPG	GNU Privacy Guard
HLL	Human in Loop Learning
HMM	Hidden Markov Model
HTTPS	Hypertext Transfer Protocol Secure
KNN	K-Nearest Neighbors
LFCC	Linear Frequency Cepstral Coefficient
LGBM	Light Gradient-Boosting Machine
LLR	Log-Likelihood Ratios
LR	Linear/Logistic Regression
LSTM	Long Short-Term Memory
MFCC	Mel Frequency Cepstral Coefficient
RF	Random Forest
RNN	Recurrent Neural Network
SNR	Siganl-Noise Ratio
SSAD	Semi-Supervised Anomaly Detection
SSH	Secure Shell Protocol
STFT	Short-Time Fourier Transform
STN	Spatial Transformer Network
SVM	Support Vector Machine
TCN	Temporal Convolution Network
WSL	Windows Subsystem for Linux

Table of Contents

Table of Scripts

Purpose

The final qualification test document is a comprehensive document outlining all procedures necessary to test the deep fake audio detection project. This includes initializing the development environment , training of the model, and leveraging that model to make assessment on input audio data.

Objectives and Tasks

The primary objective of the document is to replicate the process necessary to set up and test the deep fake audio detection project, so others could verify or continue work on the project without additional resources.

Assumptions

This document assumes:

1. The user is using Windows 11 or Windows 10 Build 19045 or later.
2. The user has a file compression/decompression tool.

Project File Structure

```
CS657-Audio-Deep-Fake-Detector
├── .devcontainer          (Development Container Build Information)
│
├── scripts                (Utility Bash Scripts)
│   ├── *.sh
│
├── src
│   ├── data_processor     (Orchestrates Data Pipeline)
│   ├── deepfake_detector  (Orchestrates Data Predictions)
│   ├── model_utilities    (Organizes Model Layers)
│   ├── noise_filter       (Provides Noise Filtering Methods)
│   ├── noise_generator    (Provides Noise Generation Methods)
│   ├── plotter            (Provides Graphical Plotting Utilities)
│   ├── *.py
│
├── workflows              (High-Level Demonstrable Workflow Scripts)
│   ├── *.sh
│
├── .gitignore             (Specifies Files That Git Will Not Track)
├── README.md              (Instructions To Setup Environment)
├── requirements.txt       (Python Dependencies Necessary For Project)
```

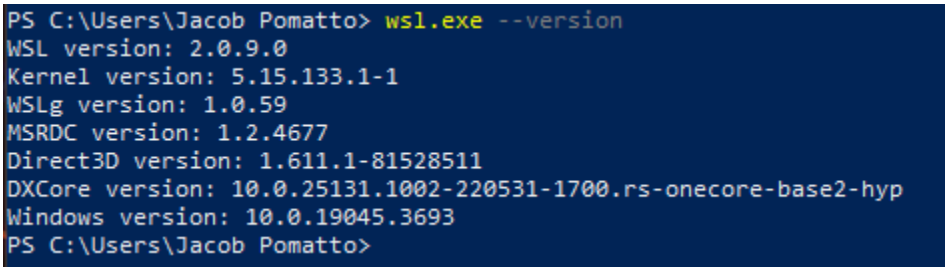
Environment Setup

Installing WSL

Windows Subsystem for Linux, or WSL, allows developers to run a Linux environment on their Windows machine without the need for a separate virtual machine. WSL is a dependency of Docker Desktop.Open the Windows Start menu.

1. Select the Windows search bar and search “Powershell”.
2. Right click the powershell application and select “Run as administrator”. Select “Yes” if prompted.
3. Enter the “wsl --install” command into the powershell terminal. Select “Yes” if prompted.
4. Restart the computer.
5. Once the computer has restarted, you may be presented with a Ubuntu installation terminal. Allow for this to finish, and then supply new credentials for a UNIX profile. Occasionally this process gets stuck; if this terminal does not progress to an installation stage, restart the computer.

6. Open the command prompt application and run the `wsl.exe –version` command to verify that it has been installed.



Installing WSL

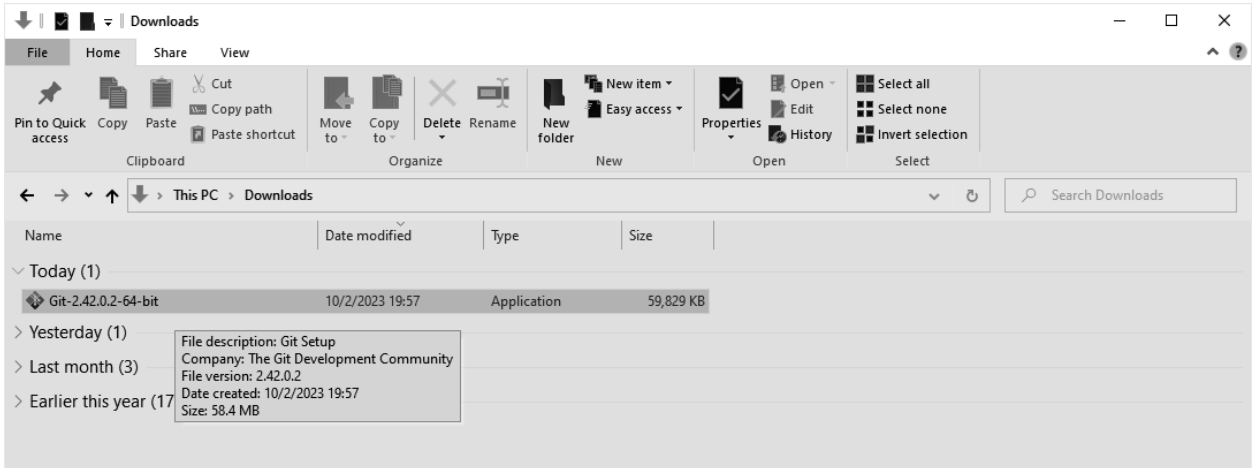
Pass/Fail	Comments
☐ Pass ☐ Fail	

Installing Git

Git is a distributed version control system that tracks changes made for a set of computer files, and enables developers to coordinate work remotely. Git is necessary to clone the software project's source code.

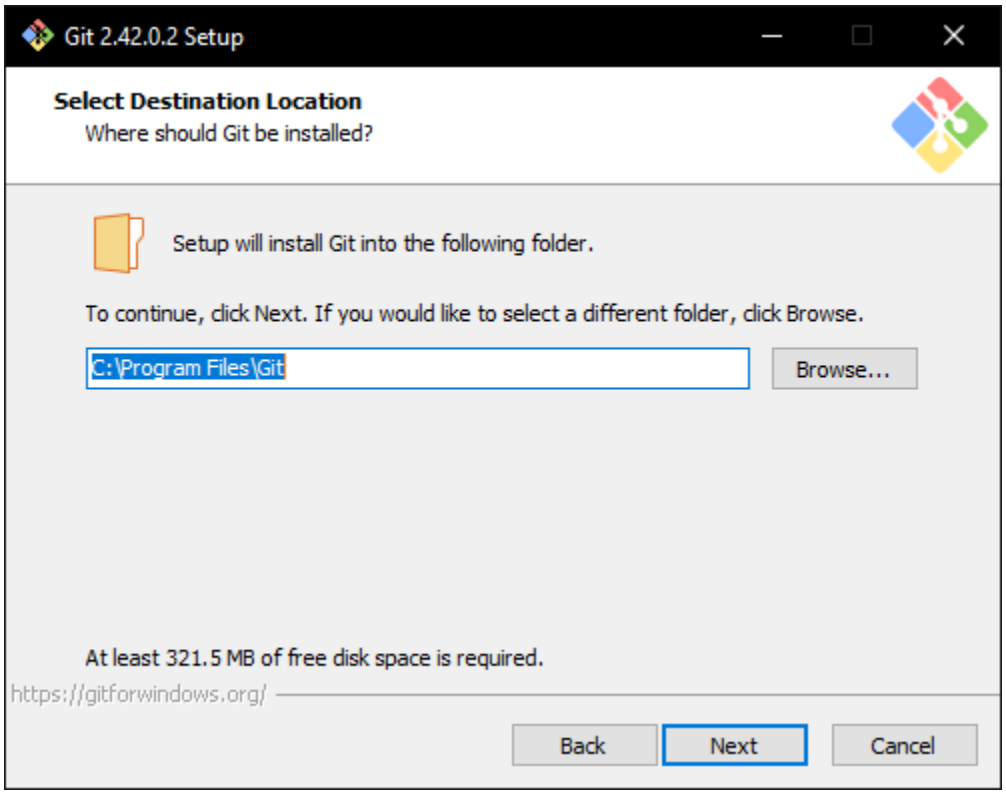
1. Download Git for Windows installer by clicking on the following link. : https://github.com/git-for-windows/git/releases/download/v2.42.0.windows.2/Git-2.42.0.2-64-bit.exe.
2. Open File Explorer and navigate to the "Downloads" folder.

3. Double click “Git-2.42.0.2-64-bit”. Select “Yes” if prompted.

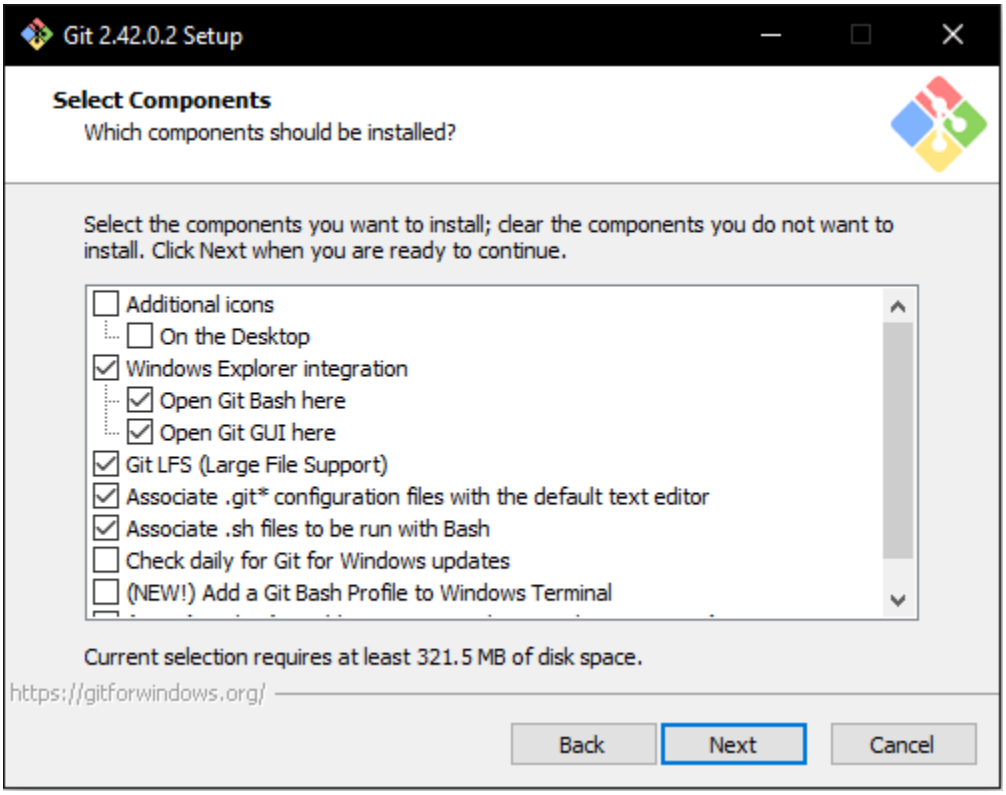


4. Read the license agreement and select “Next”.

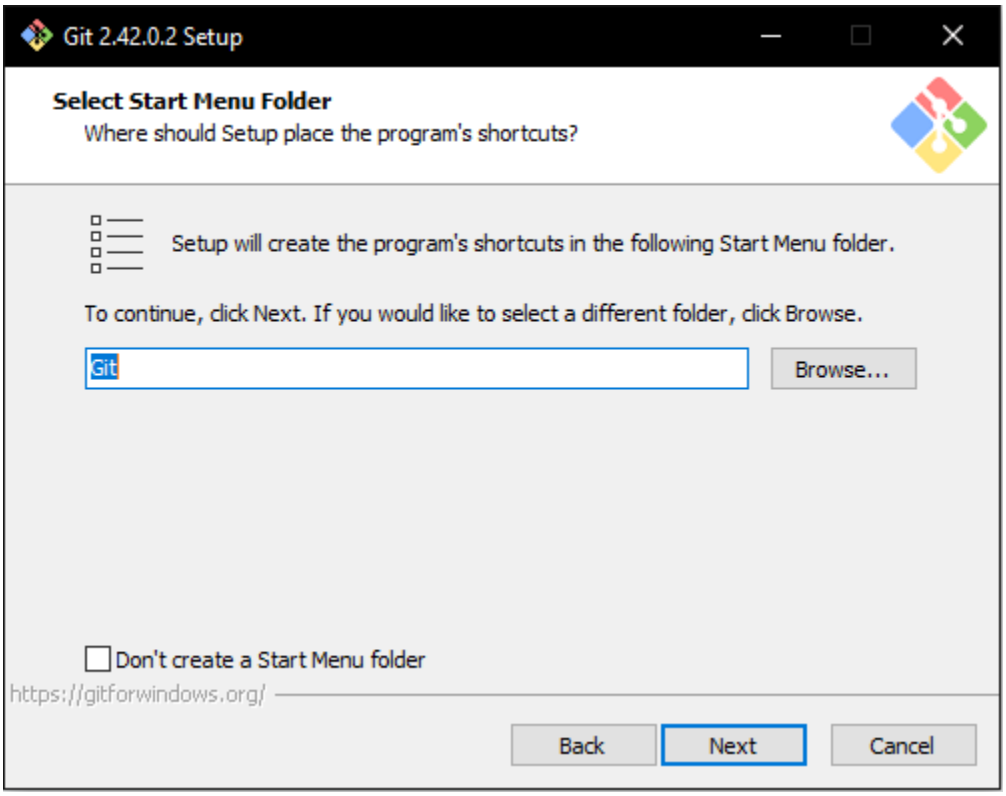
5. Use the default installation location and select “Next”.



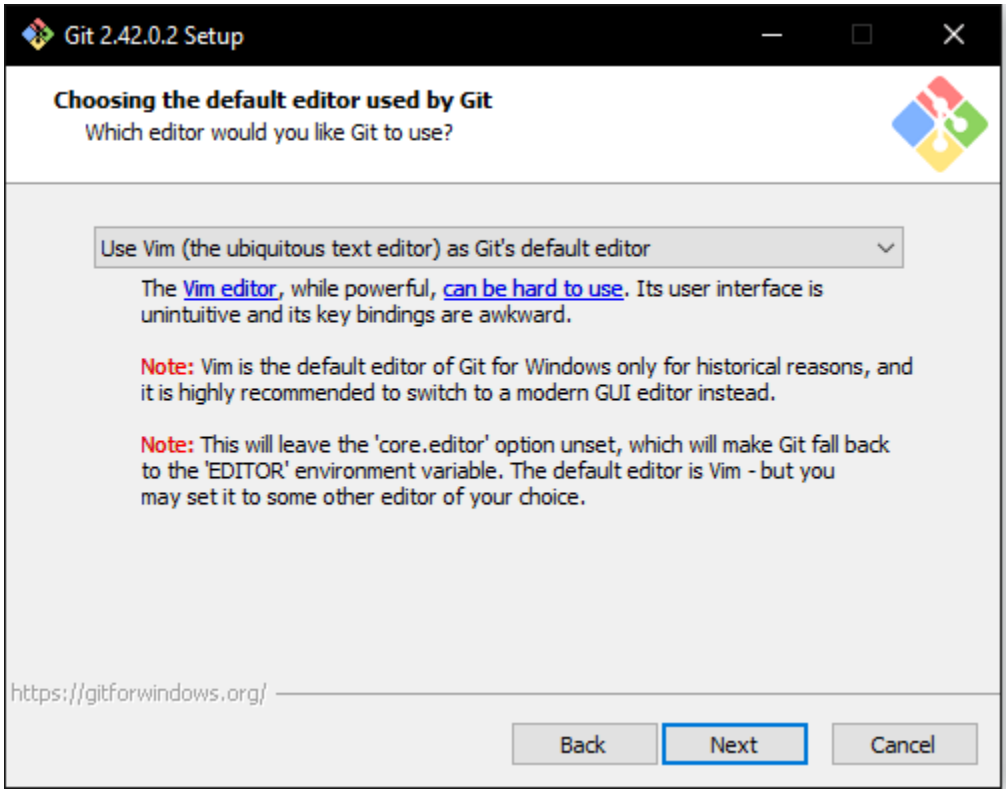
6. Use the default component options and select “Next”.



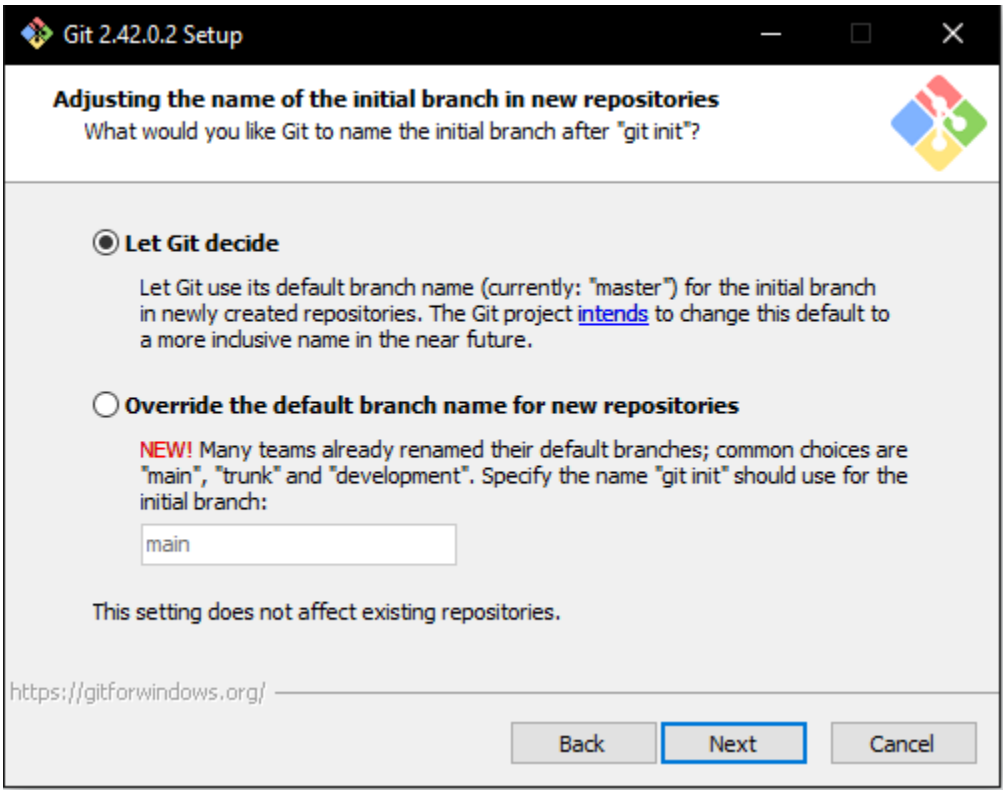
7. Use the default start menu folder and select “Next”.



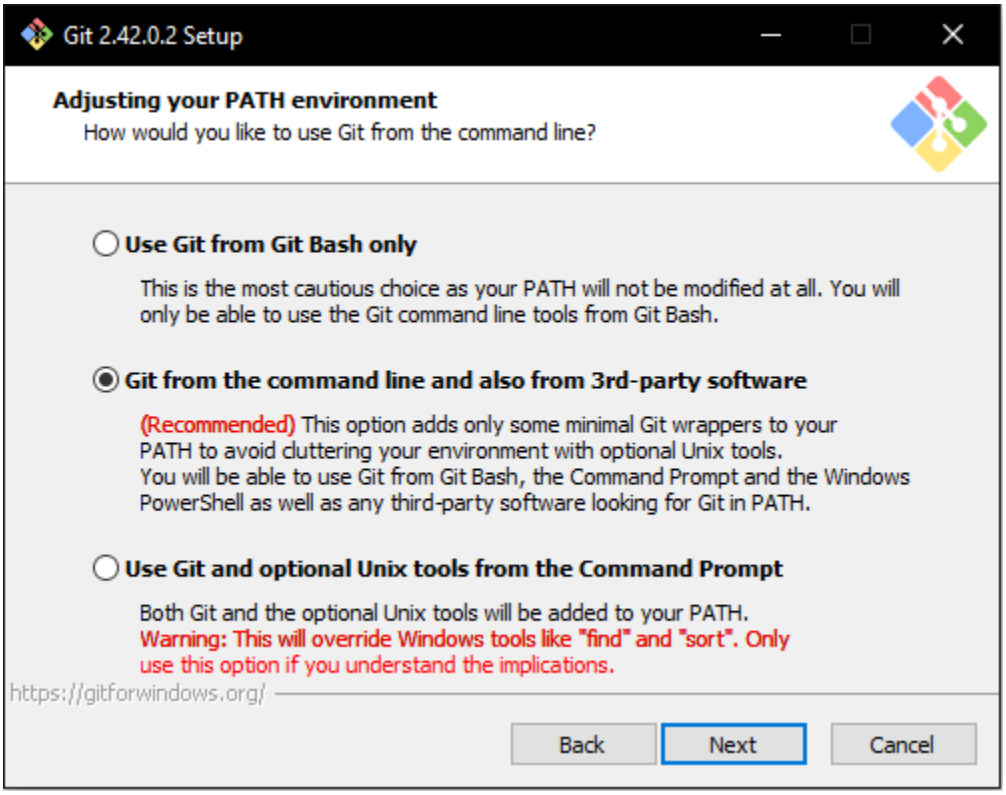
8. Use the default editor and select “Next”.



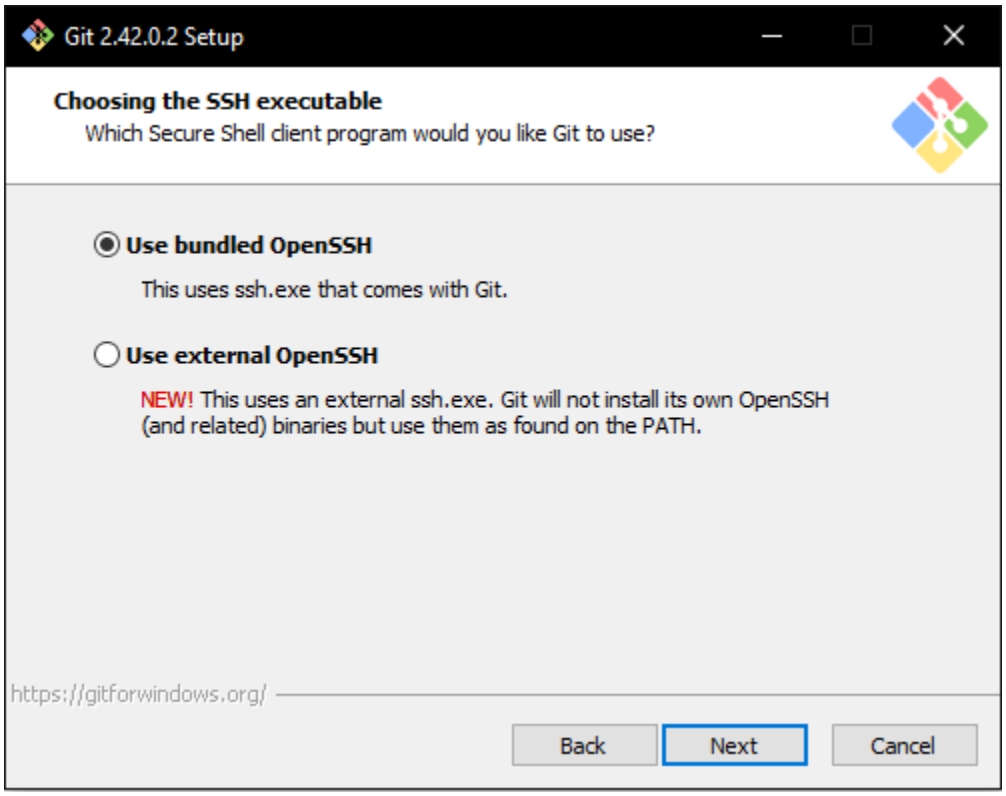
9. Use the default initial branch name and select “Next”.



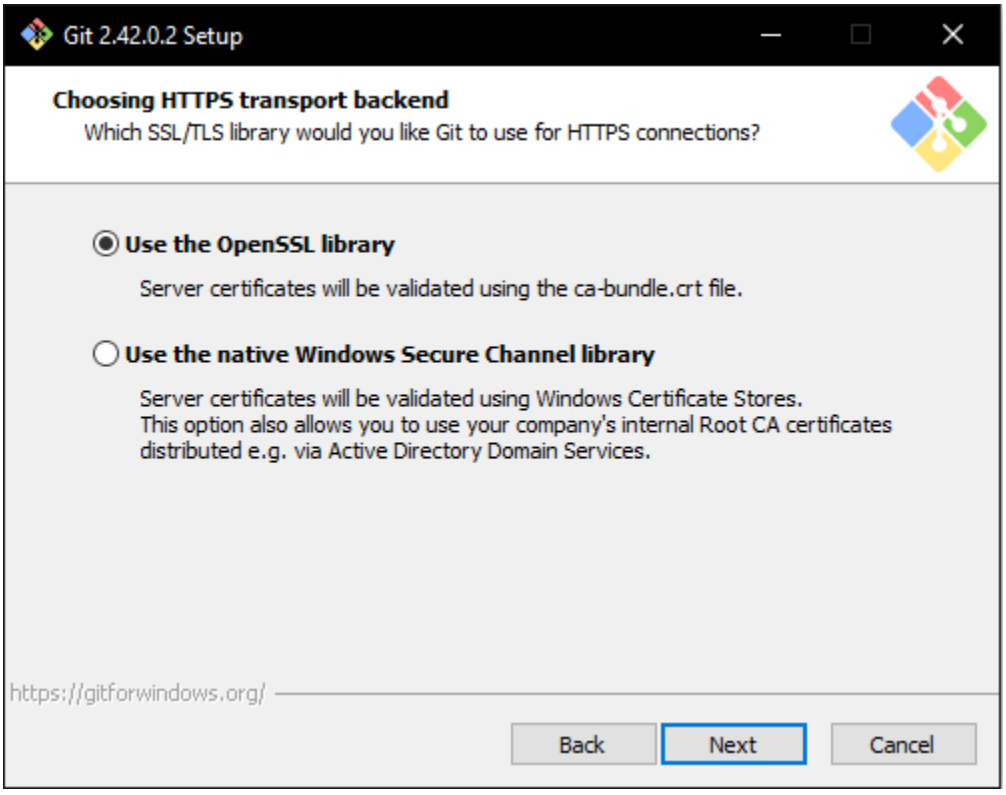
10. Use the default PATH options and select “Next”.



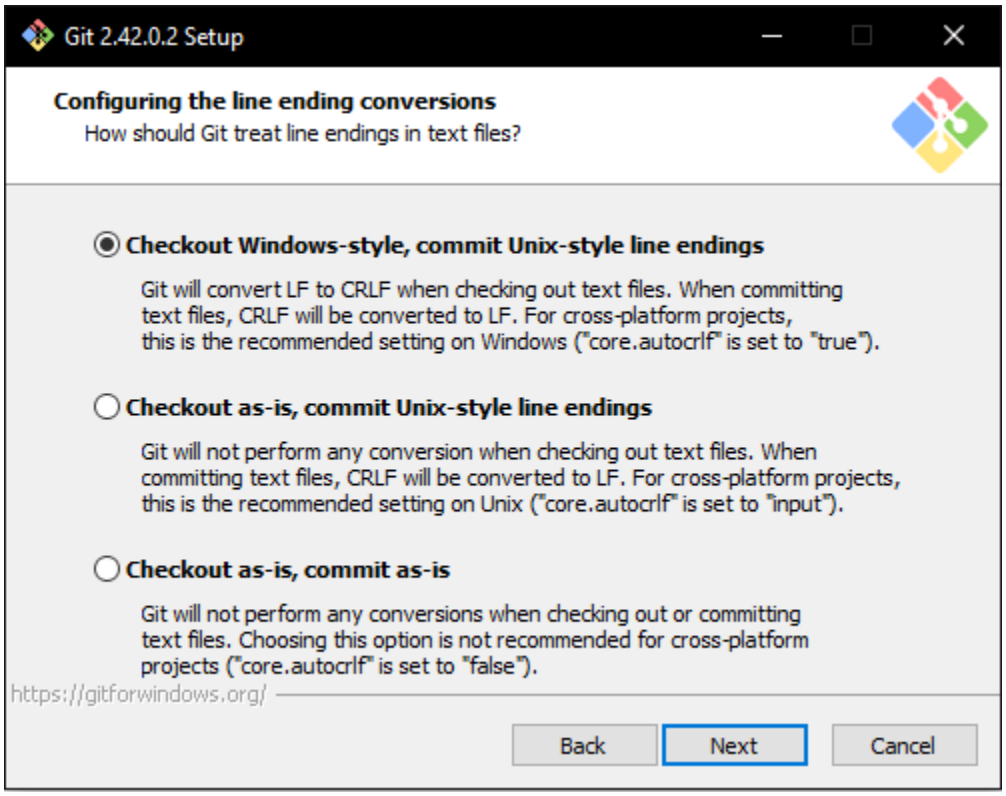
11. Use the default SSH executable option and select “Next”.



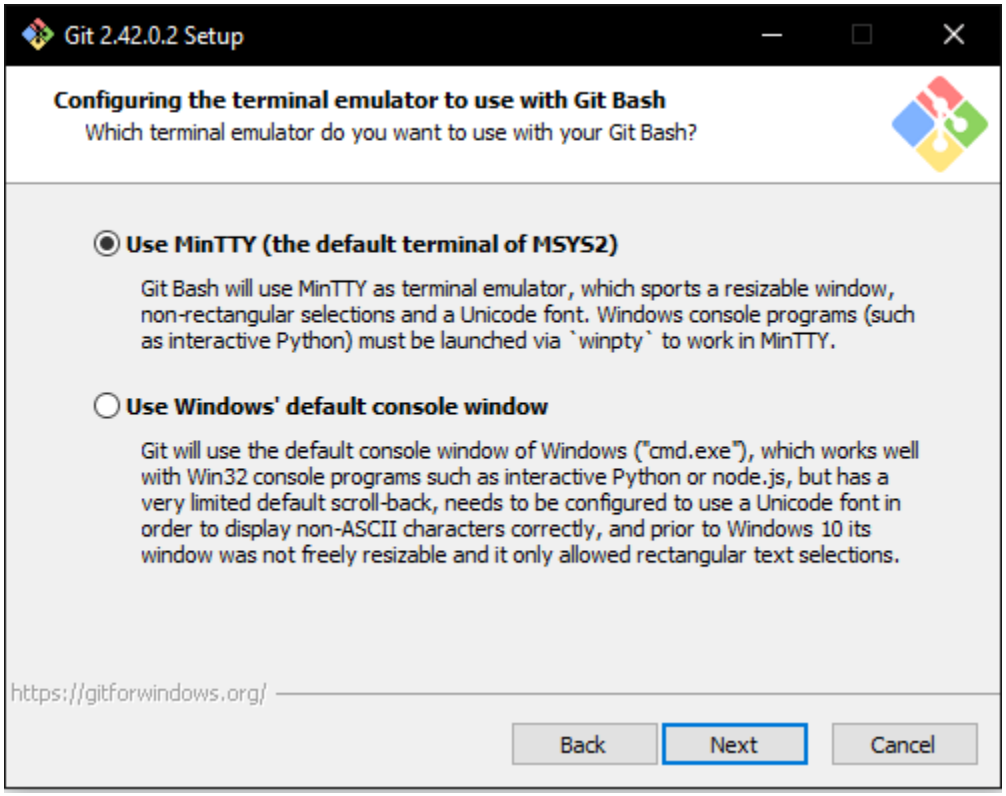
12. Use the default HTTPS transport backend and select “Next”.



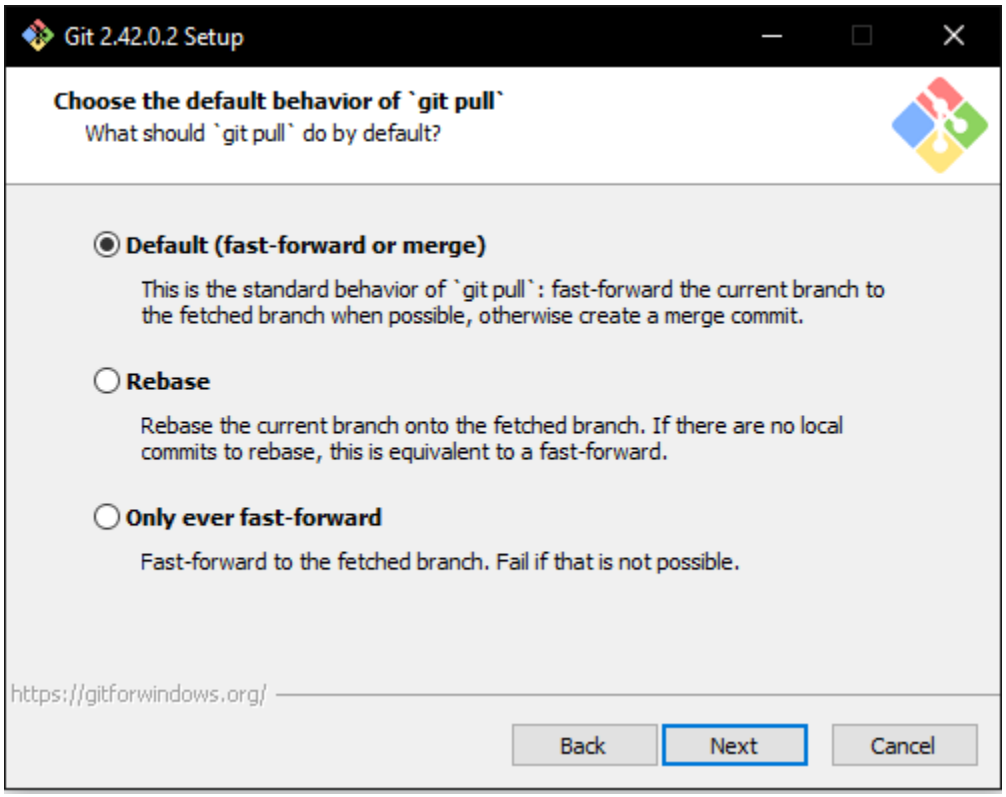
13. Use the default line ending option and select “Next”.



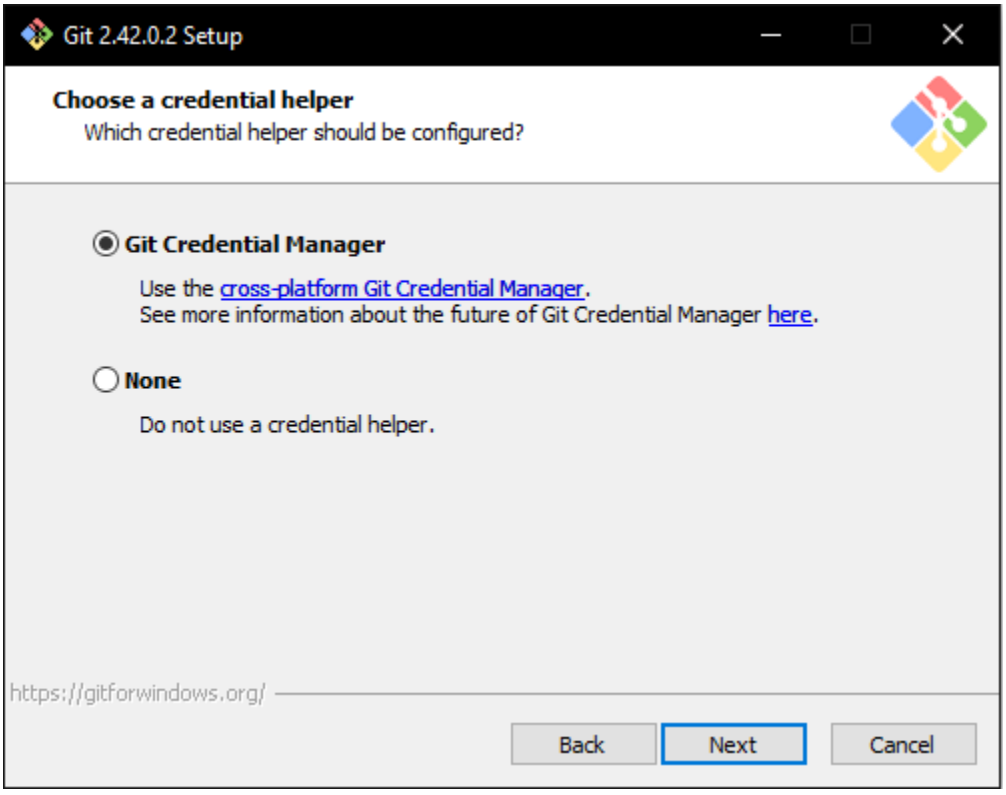
14. Use the default terminal emulator and select “Next”.



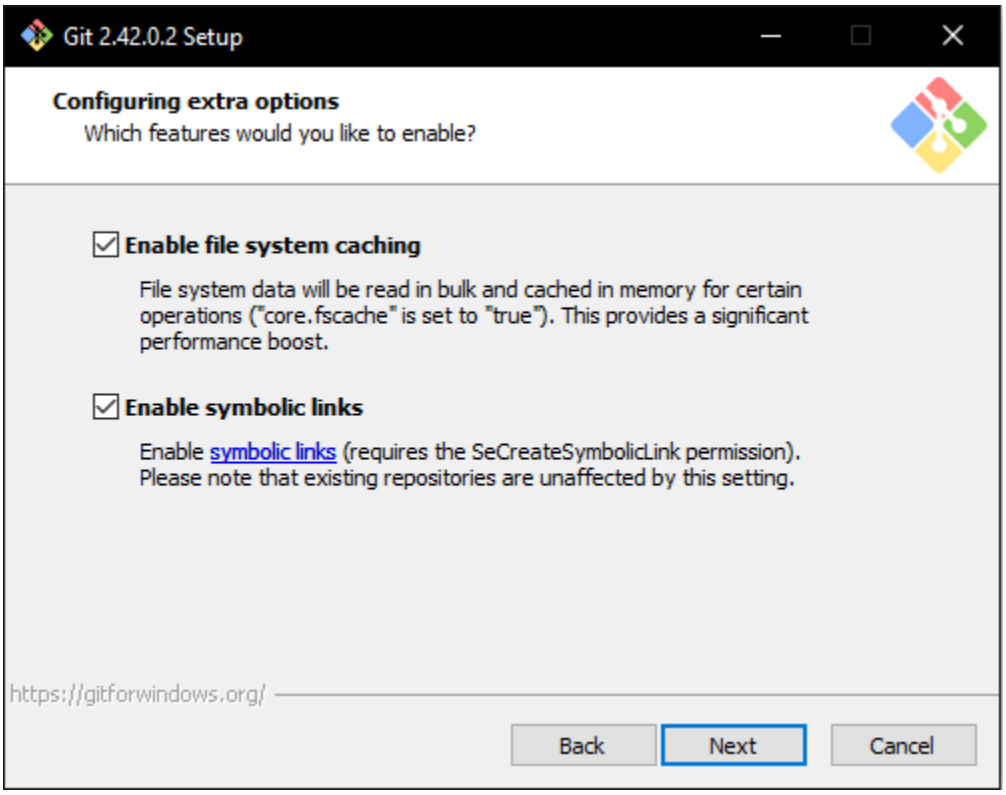
15. Use the default behavior of “git pull” and select “Next”.



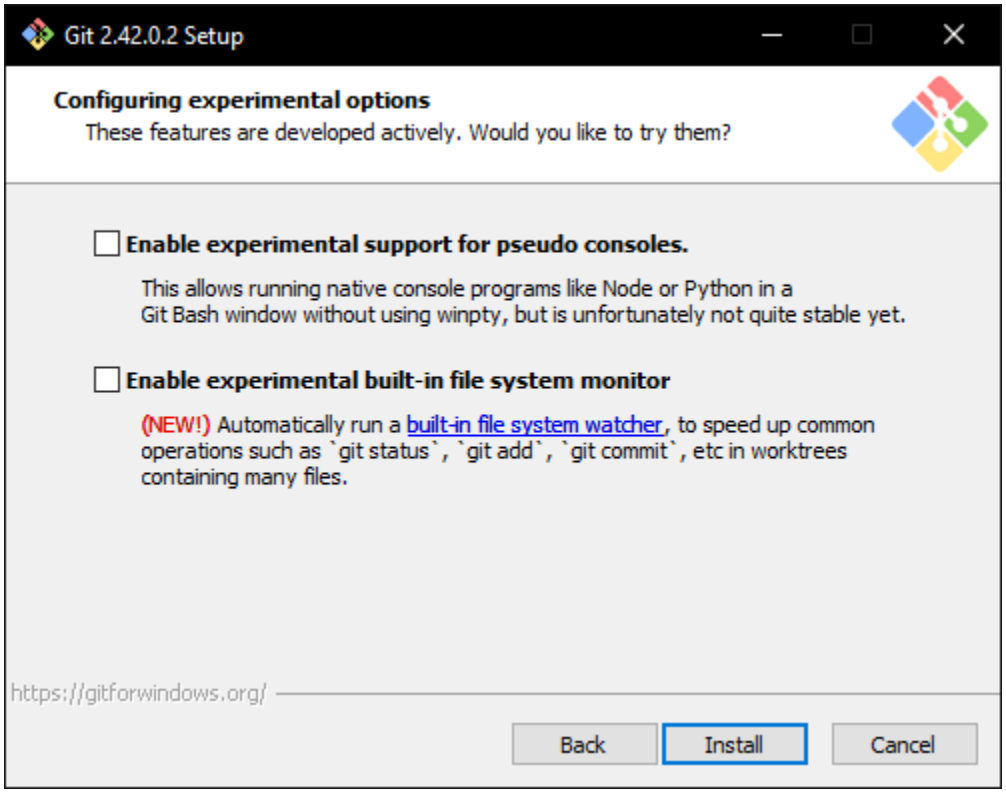
16. Use the default credential behavior option and select “Next”.



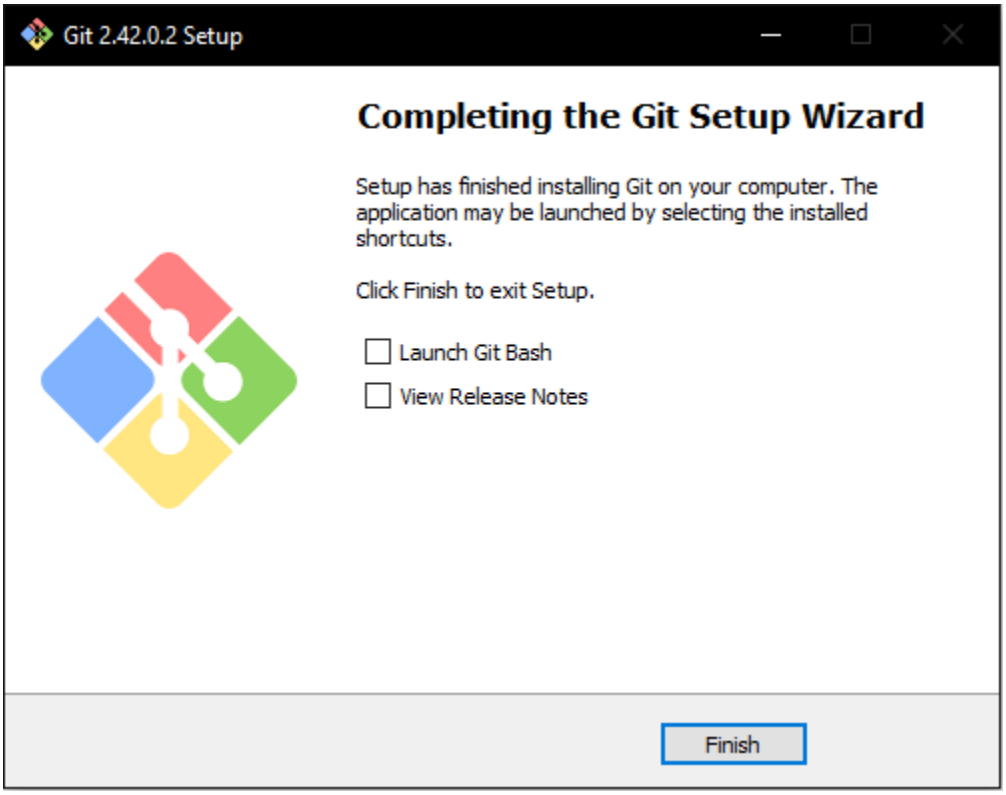
17. Use the default extra options and select “Next”.



18. Use the default experimental options and select “Install”.



19. Once the installation is complete, uncheck “View Release Notes” and select “Finish” to close the installer window.



Installing Git

Pass/Fail	Comments
☐ Pass ☐ Fail	

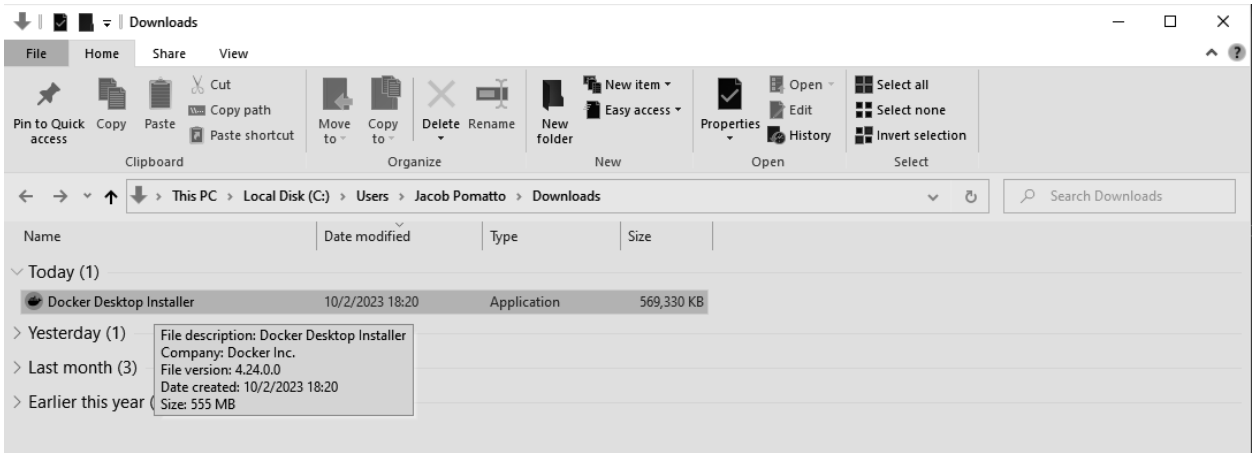
Installing Docker Desktop

Docker Desktop is a secure, out of the box containerization software offering developers a toolkit for building and running applications. This project leverages containers to provide an environment-in-a-box for python development.

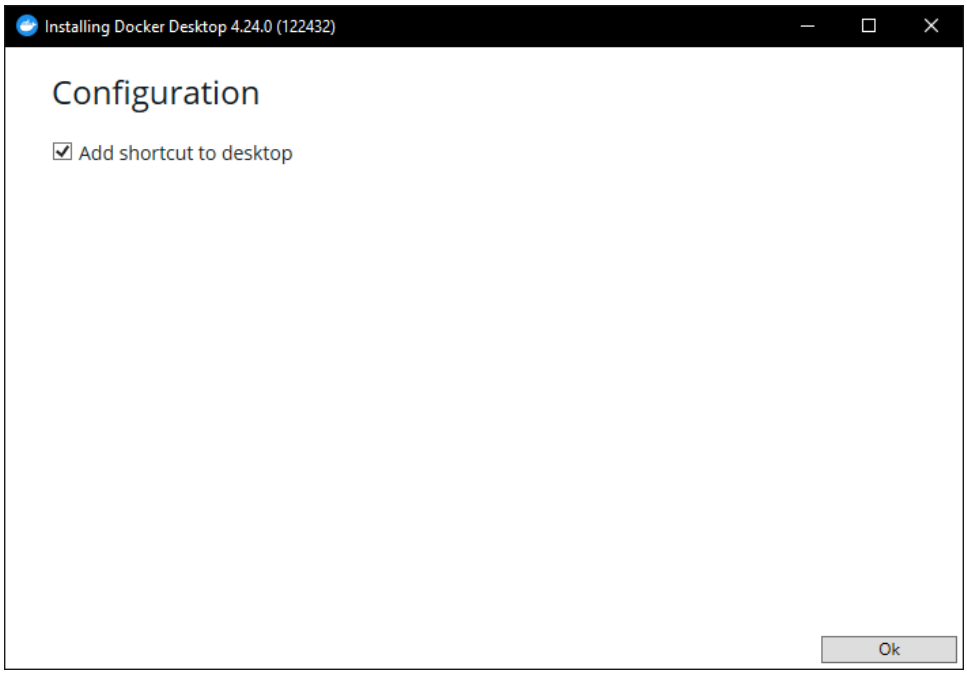
1. Download Docker Desktop for Windows installer by clicking on the following link: https://desktop.docker.com/win/main/amd64/Docker%20Desktop%20Installer.exe

?_gl=1*s8yam4*_ga*MTMyMjA0NjM4OS4xNjkzMDc1MDc5*_ga_XJWPQMJYHQ*MTY5MzA3NTA3OC4xLjEuMTY5MzA3NTUxMS42MC4wLjA.

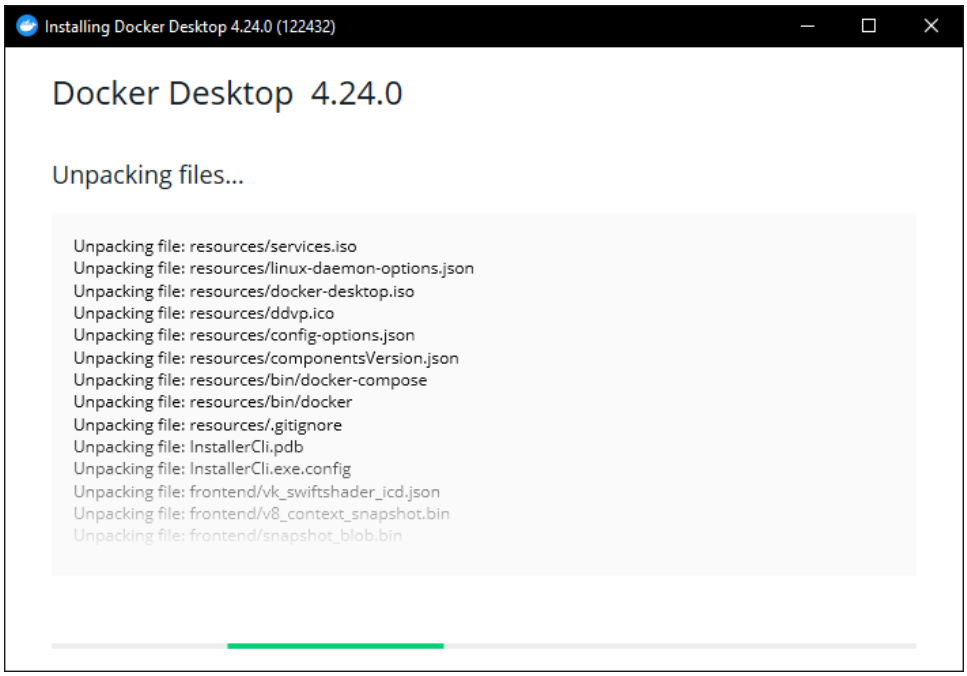
2. Open File Explorer and navigate to the “Downloads” folder.
3. Double click “Docker Desktop Installer”. Select “Yes” if prompted.



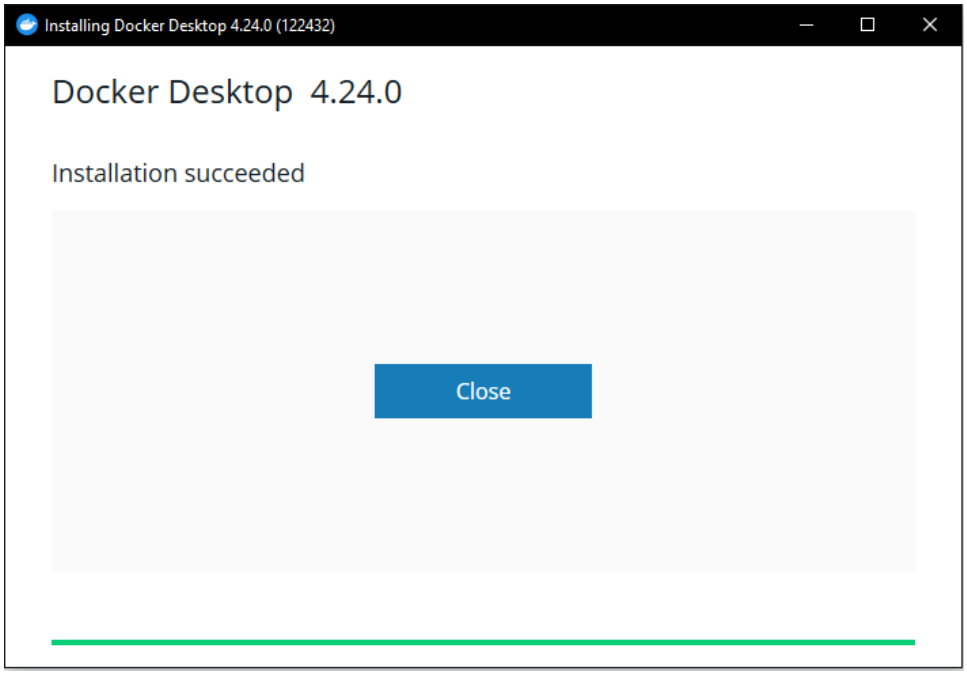
4. When prompted if the installer may make changes to the system, select “yes”.
5. On the Configuration screen, select “Ok”.



6. Wait for the installer to finish unpacking the Docker Desktop application files.

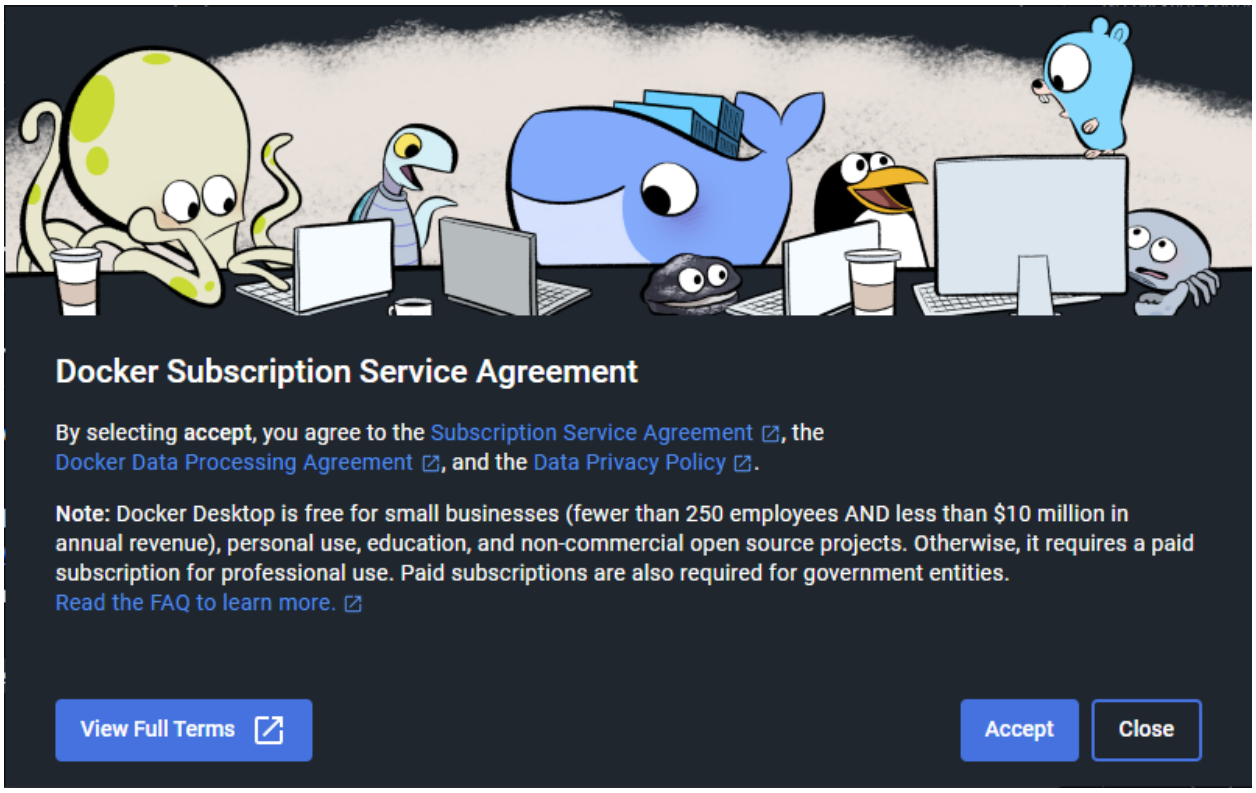


7. Once the installation is complete, select “Close” to close the installer.

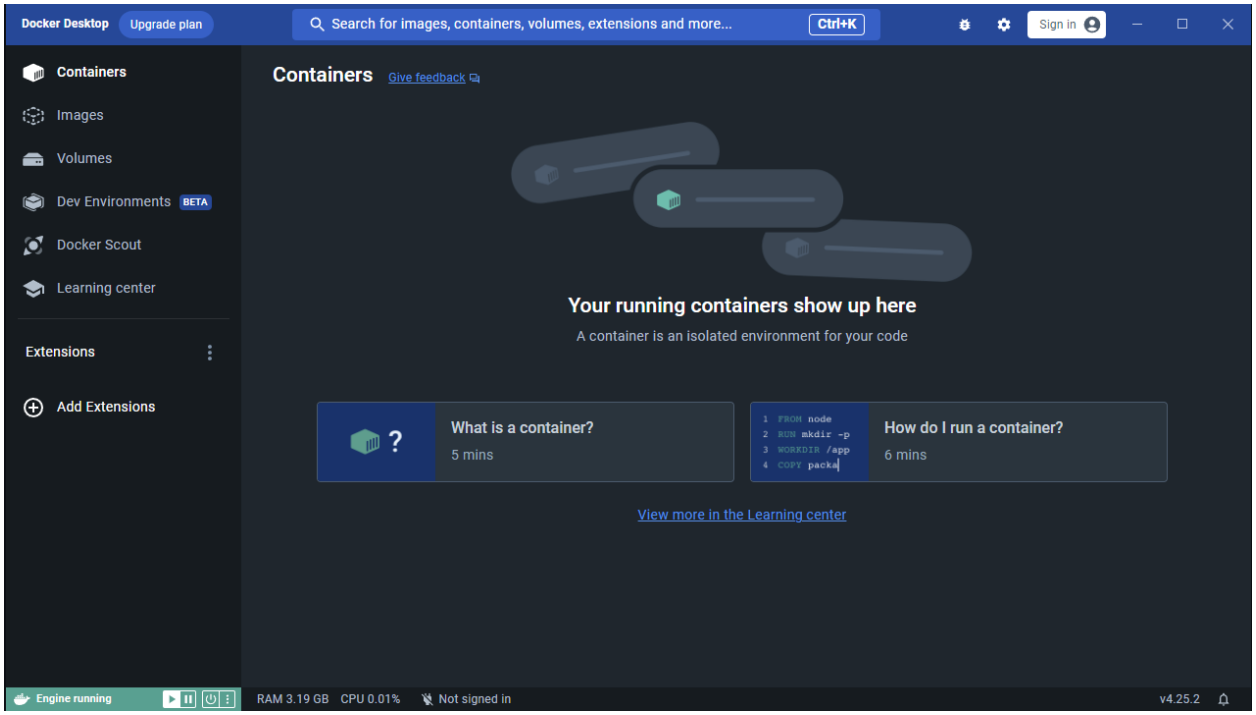


8. Select the Windows search bar and type “Docker Desktop” and open the application.

9. When presented with the service agreement, select “Accept”.



10. Continue without signing in and skip optional questionnaires.
11. Verify that Docker Desktop is at the home screen.



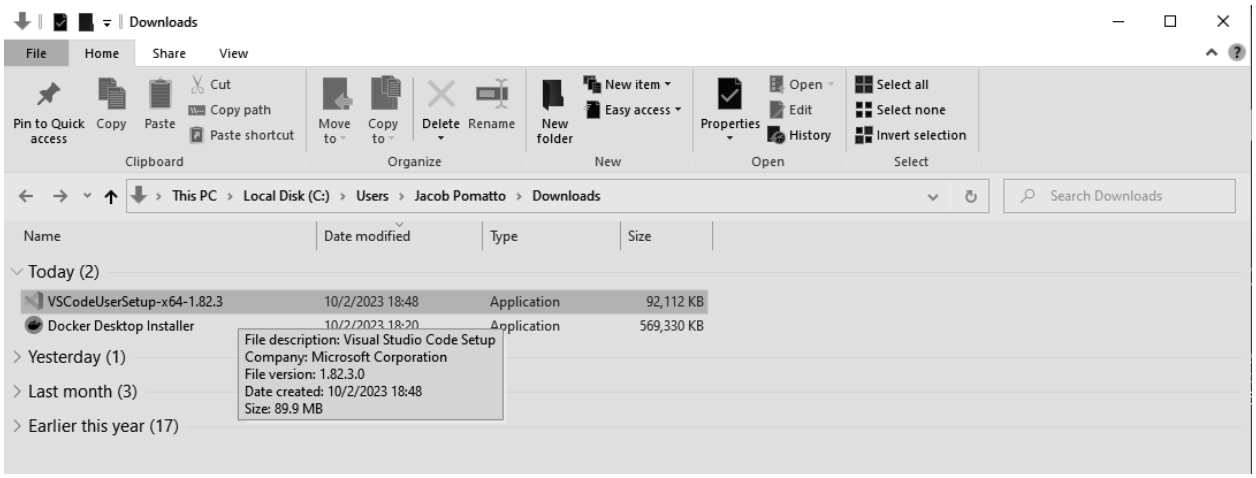
Installing Docker Desktop

Pass/Fail	Comments
☐ Pass ☐ Fail	

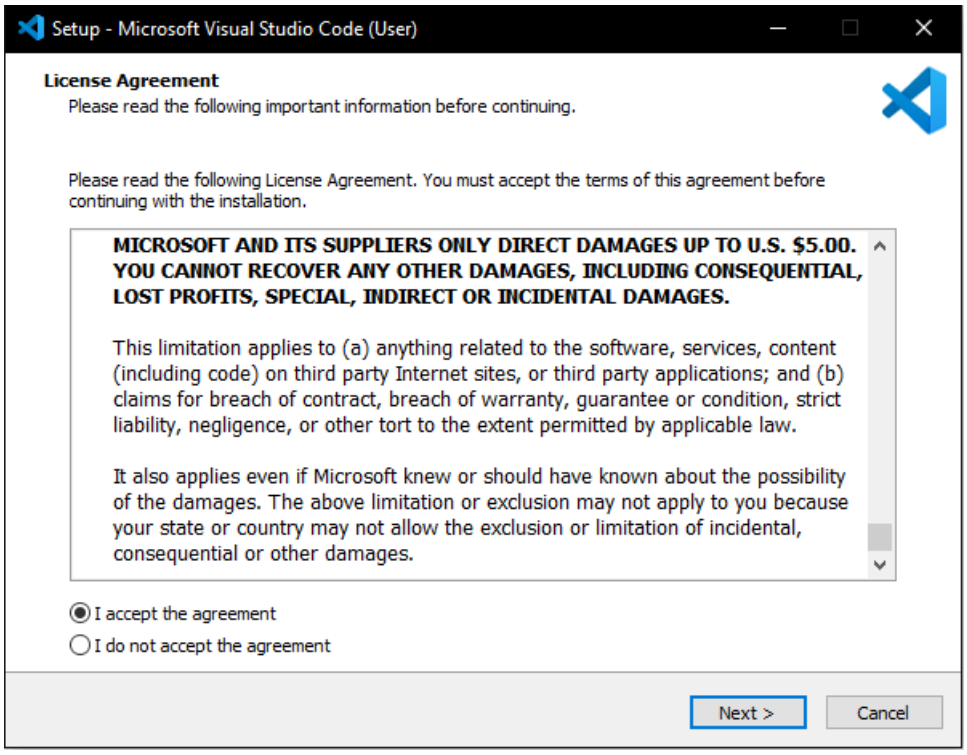
Installing Visual Studio Code

Visual Studio Code is a modern source-code editor that offers numerous plugins to enhance developer productivity.

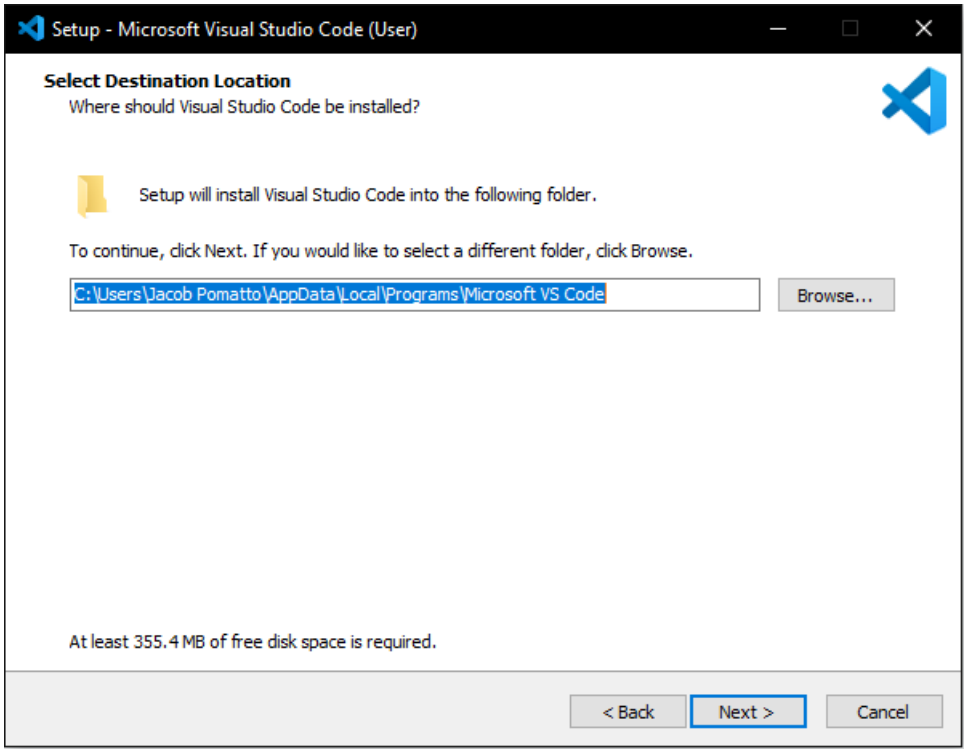
1. Download the VSCode for Windows installer by clicking on the following link: https://code.visualstudio.com/sha/download?build=stable&os=win32-x64-user
2. Open file explorer and navigate to the “Downloads” folder.
3. Double click “VSCodeUserSetup-x64-1.84.2”. Select “Yes” if prompted.



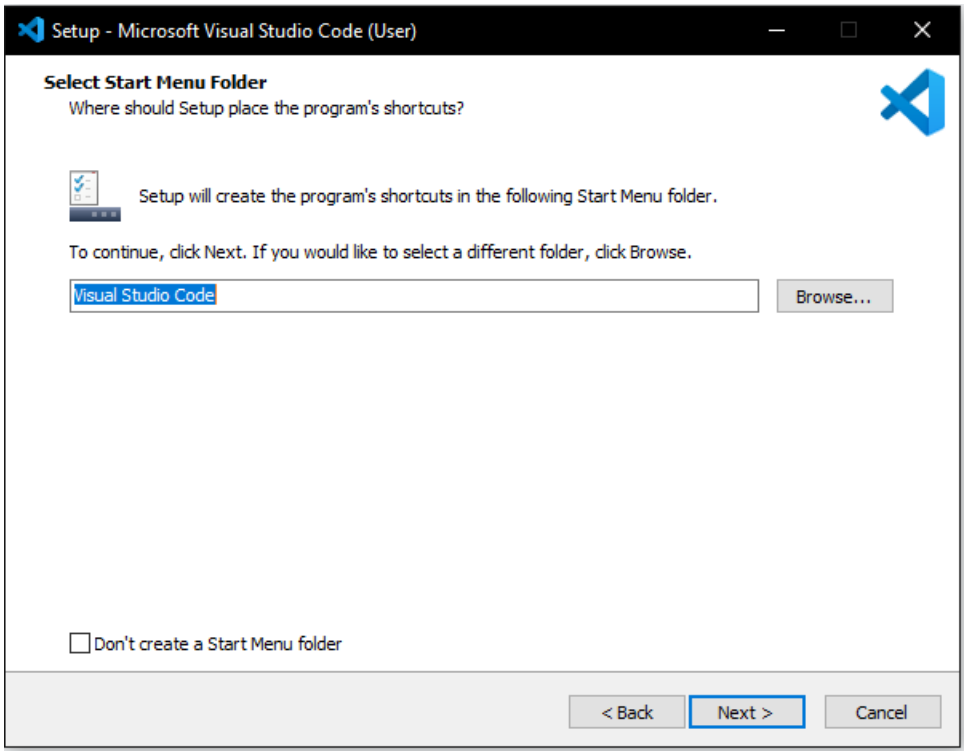
4. Read the License Agreement, select “I accept the agreement”, and select “Next”.



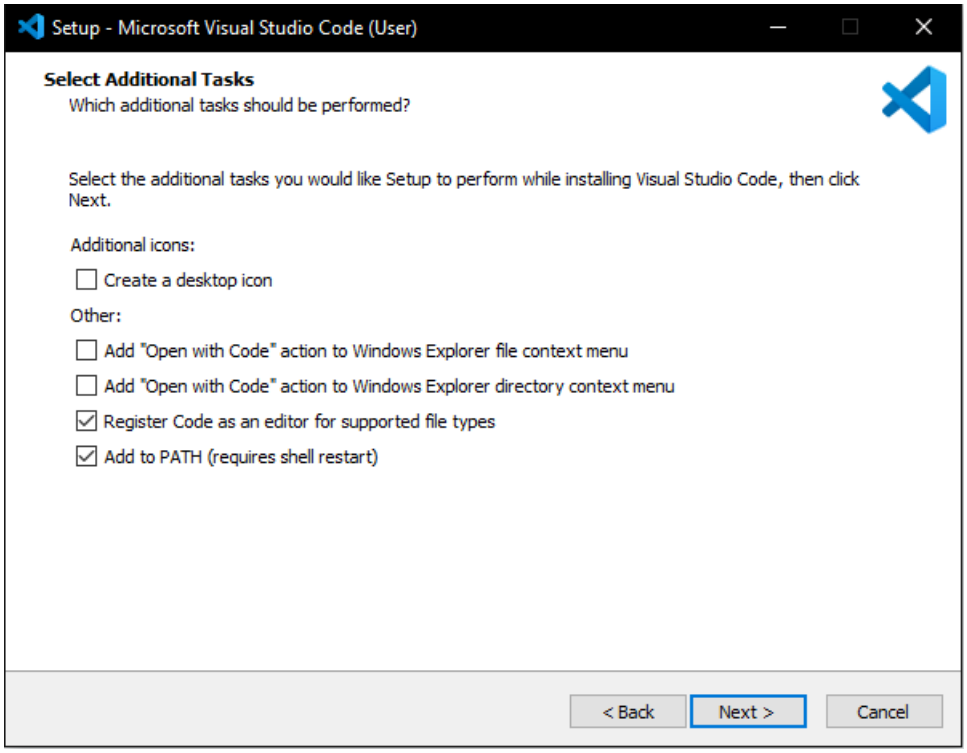
5. Use the default destination location and select “Next”.



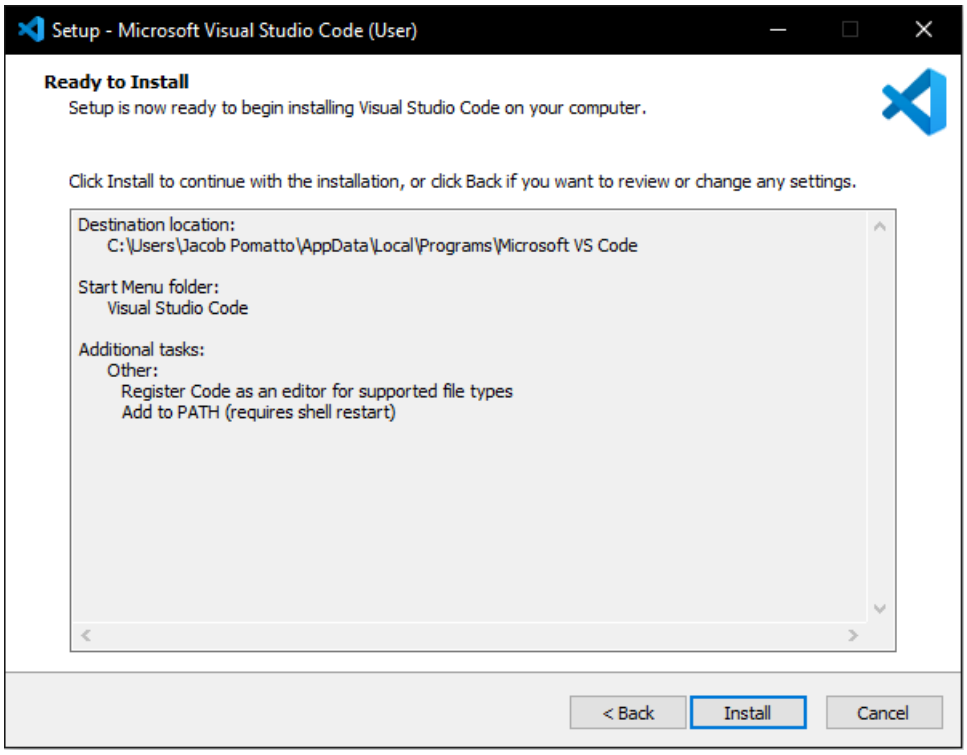
6. Use the default start menu folder and select “Next”.



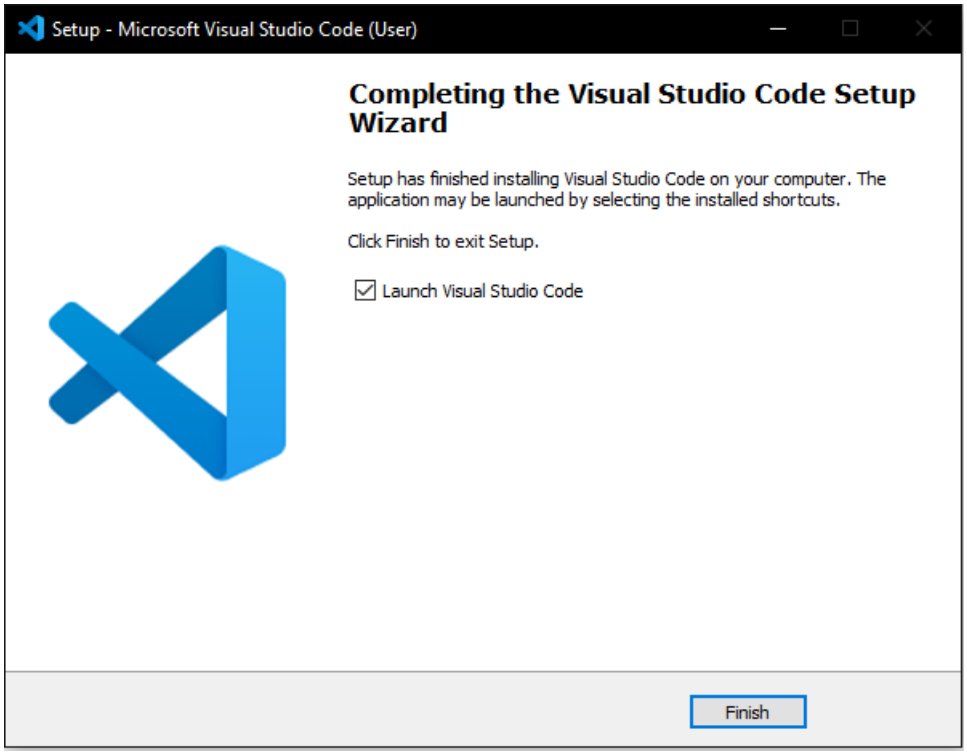
7. Use the default additional tasks and select “Next”.



8. Review install options and select “Install”.



9. Once the installation is complete, leave “Launch Visual Studio Code” checked and select “Finish” to close the installer.



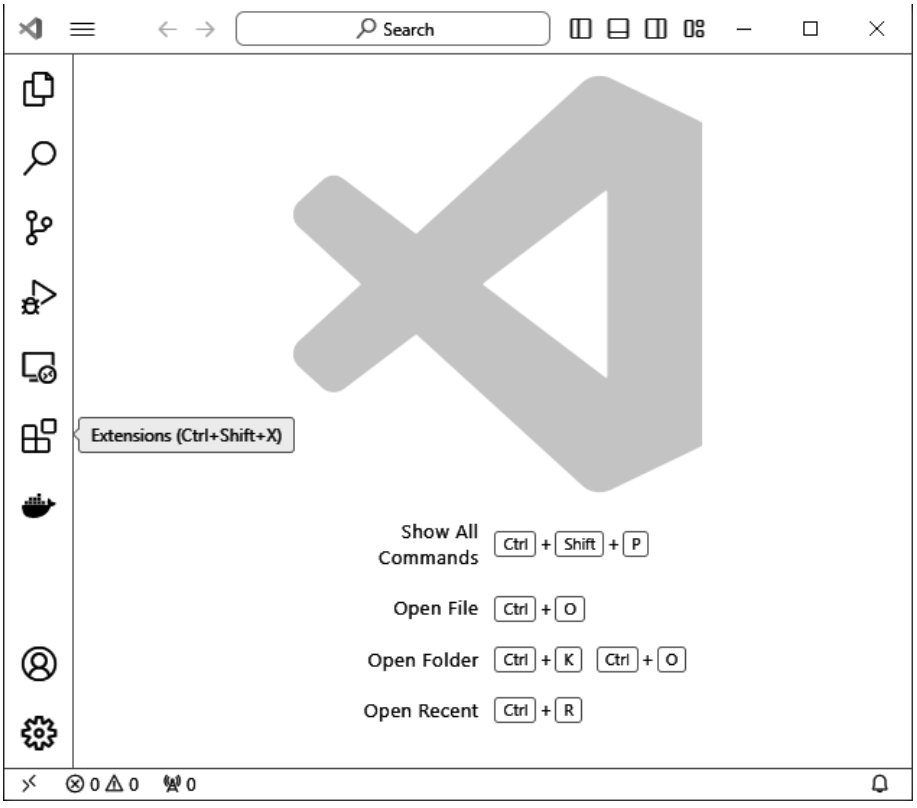
Installing Visual Studio Code

Pass/Fail	Comments
☐ Pass ☐ Fail	

Installing Dev Containers Extension

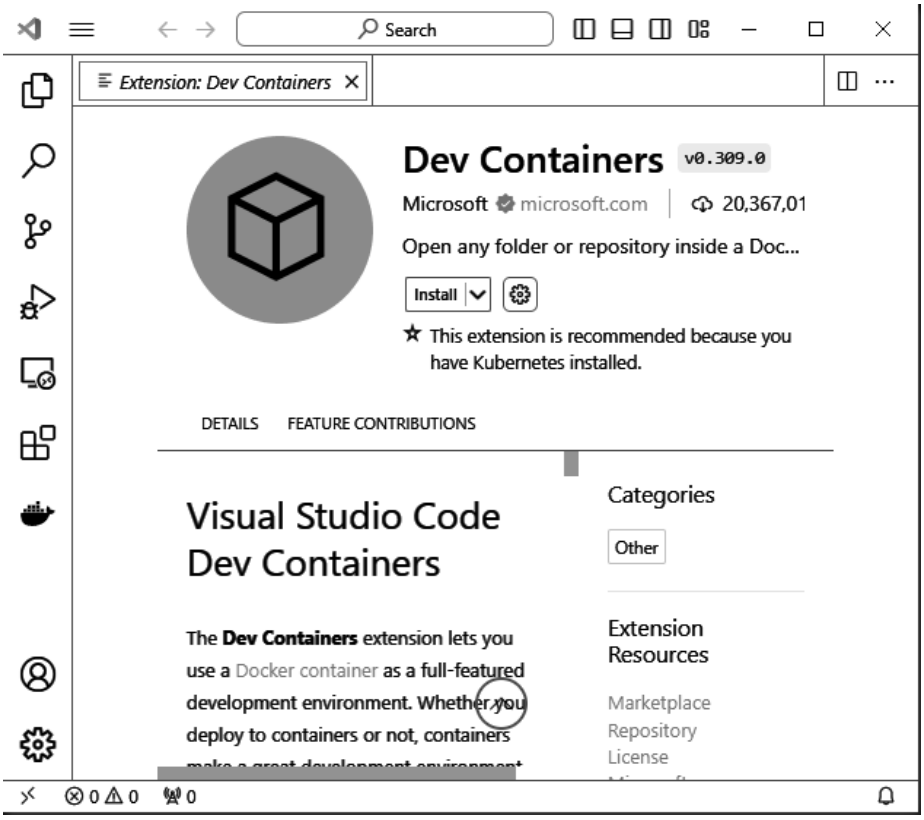
The Dev Containers extensions allows developers to use a container as a full-featured development environment. In the scope of this project, it provides all the necessary dependencies, tooling, and configuration to develop with Python.

1. Open the Visual Studio Code extensions menu.



2. Select the search bar and search “Dev Container”.

3. Select the “Dev Container” extension by Microsoft and install.



Installing Dev Containers

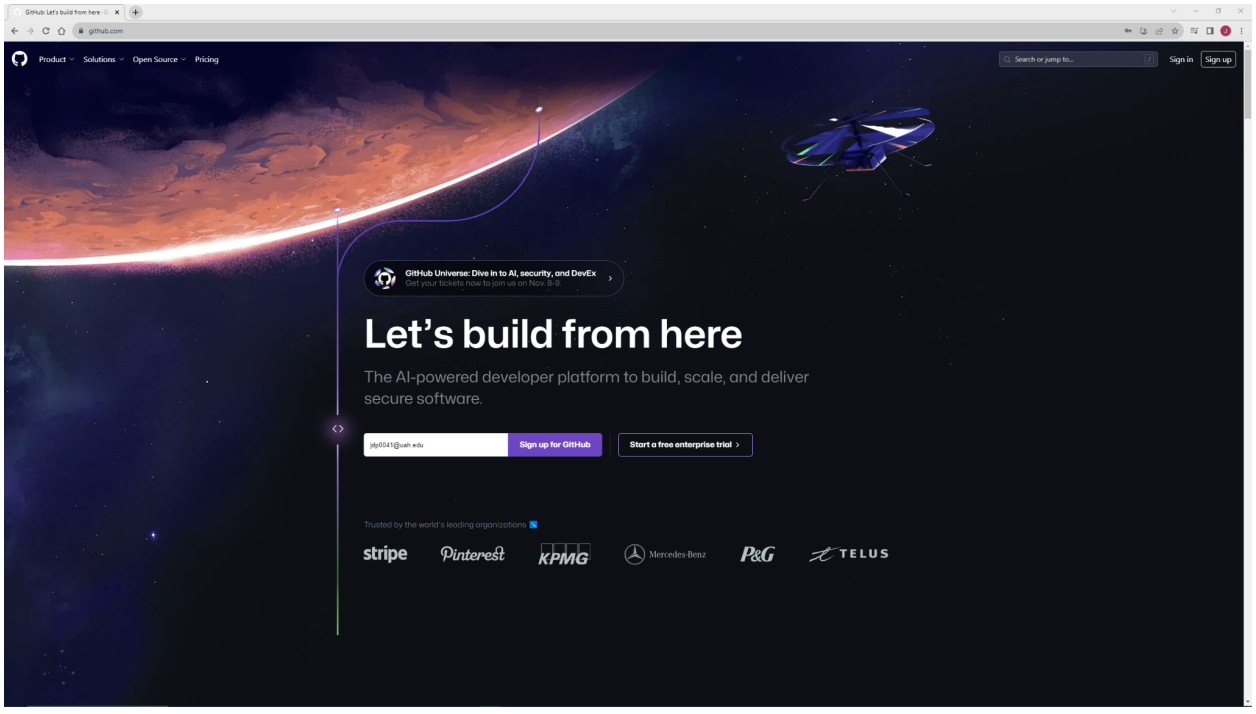
Pass/Fail	Comments
☐ Pass ☐ Fail	

Project Setup

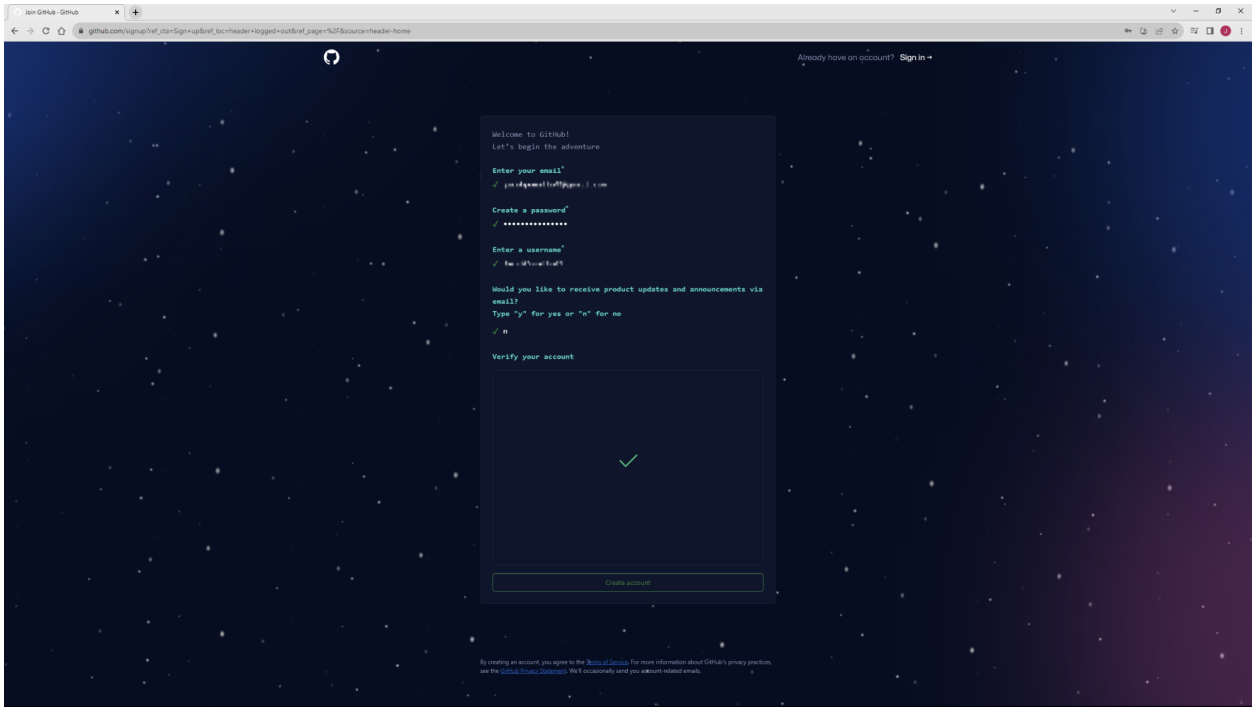
Downloading Source Code

Using Git, the project must be cloned from the remote repository to the developer's local machine to execute the application.

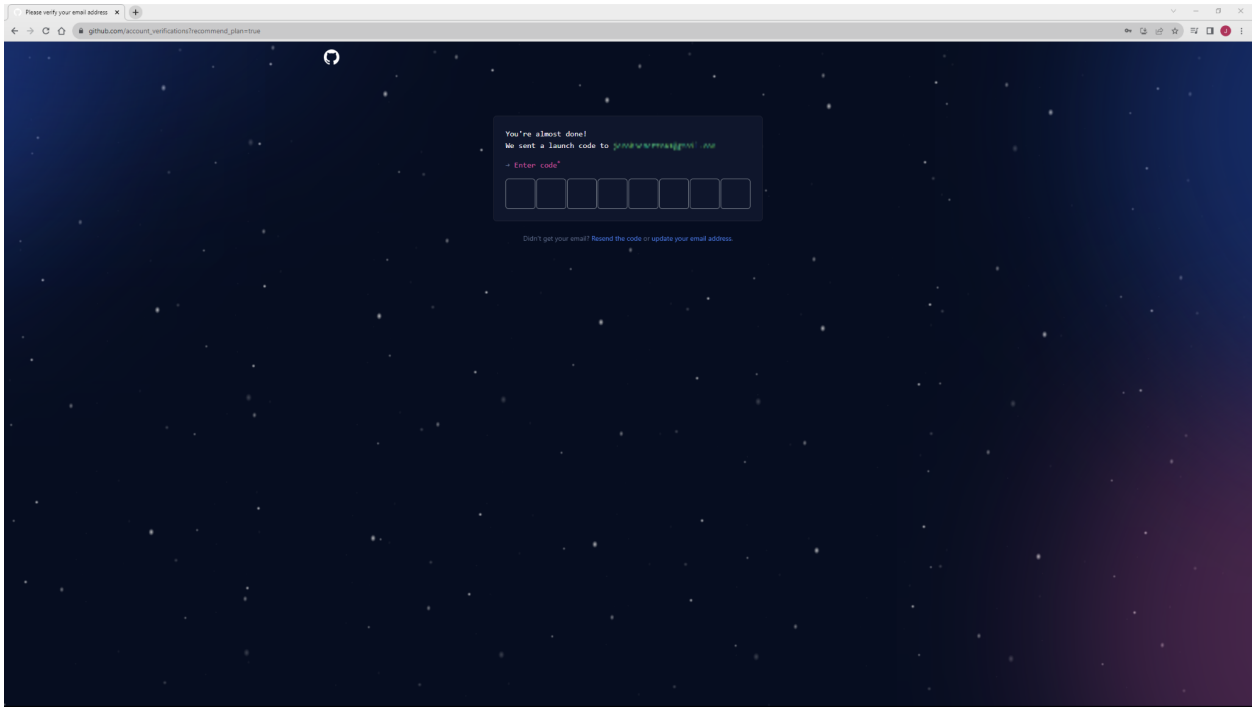
1. Navigate to https://github.com/
2. If you already have a GitHub account, login and skip to step 6.
3. On the GitHub landing page select the "Sign Up" button found in the top right corner.



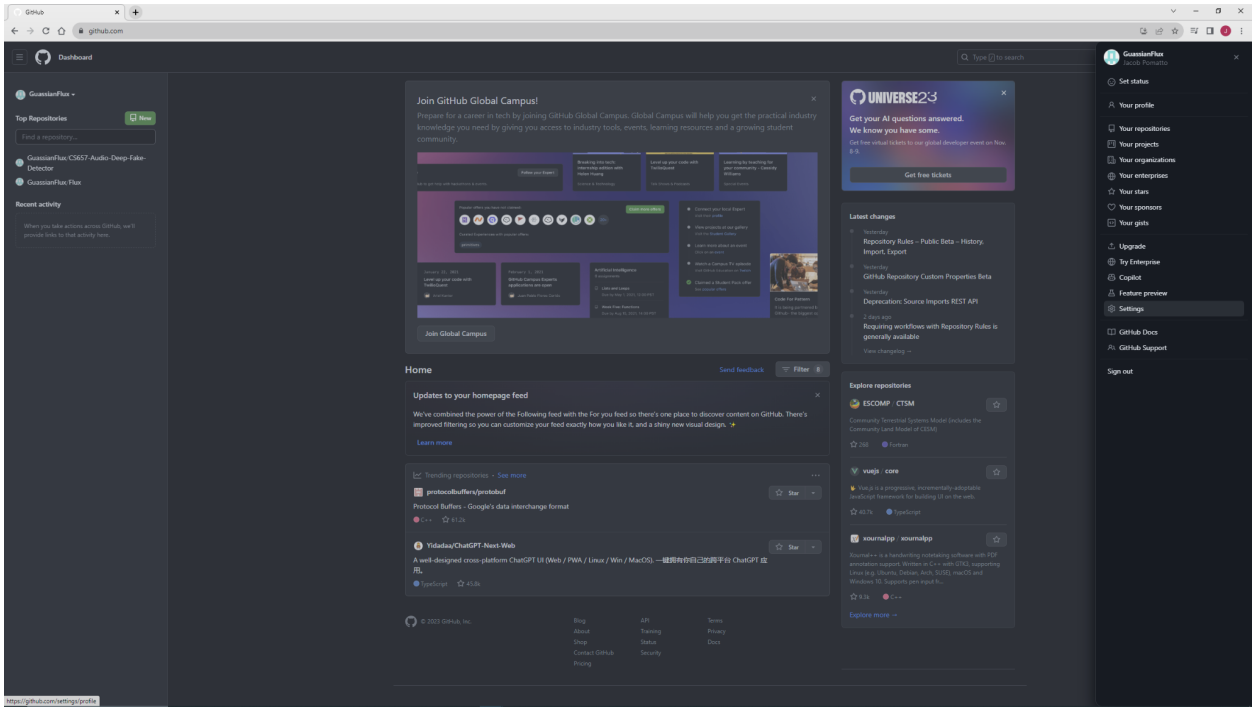
4. Enter the prompted account information and verify that you are not a robot. Once completed, select “Create Account”.



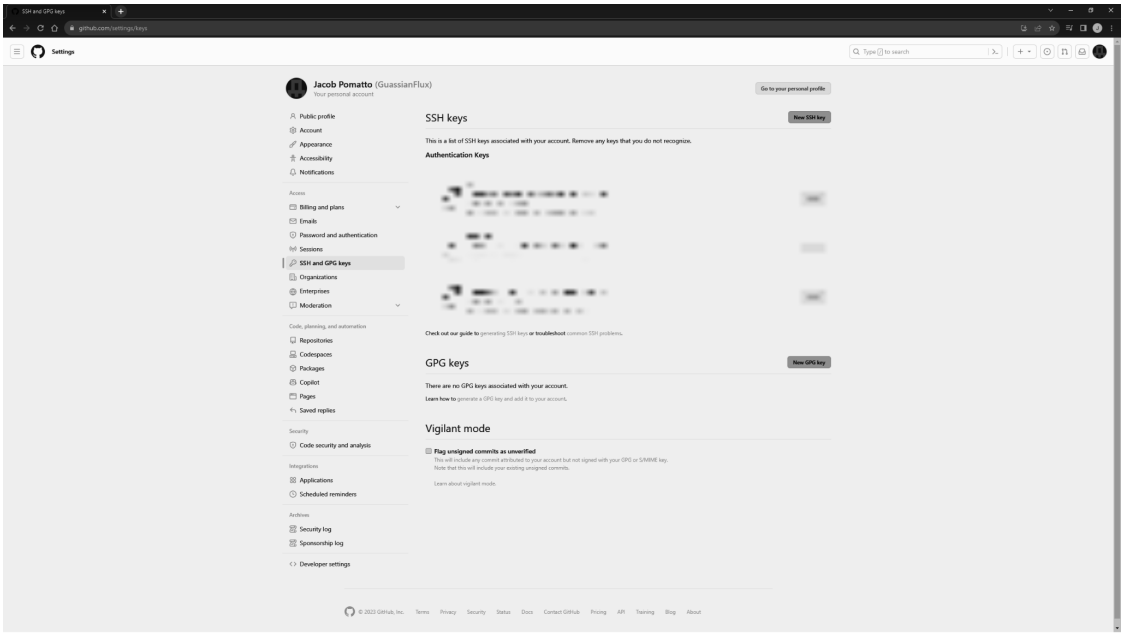
5. You’ll receive an email with a verification code at the email address previously provided. Obtain that code and enter it into the prompt.



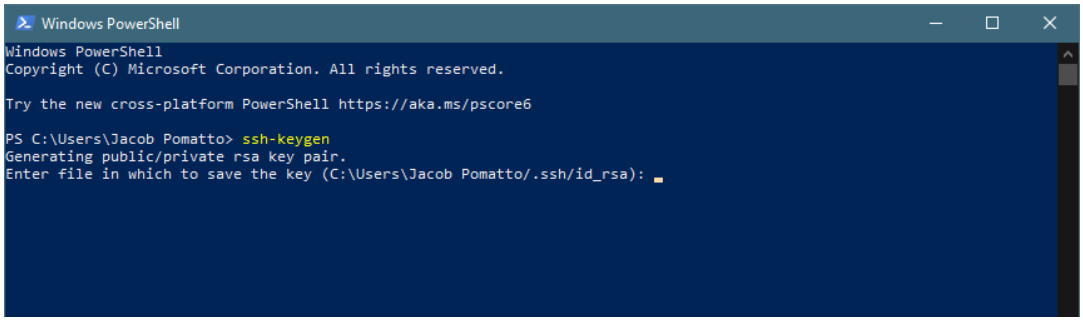
6. If you've already generated an SSH key for your Github account, skip to step 13.
7. Once signed in, click on the circular icon in the top-right corner of the screen to open an account menu and then select "Settings".



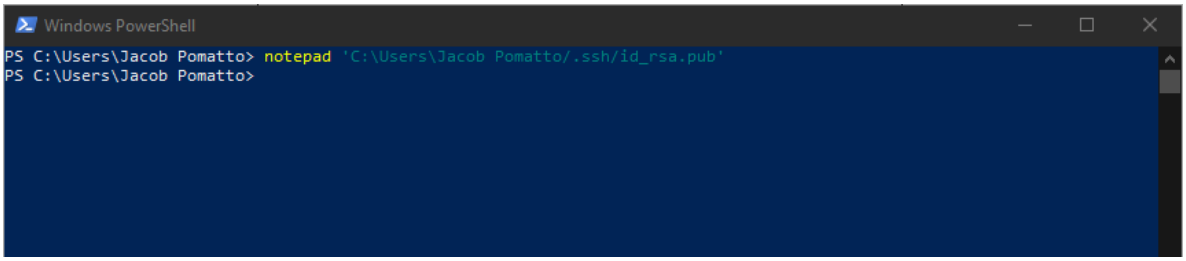
8. In the Access settings group, select the "SSH and GPG keys" option and then select "New SSH Key".



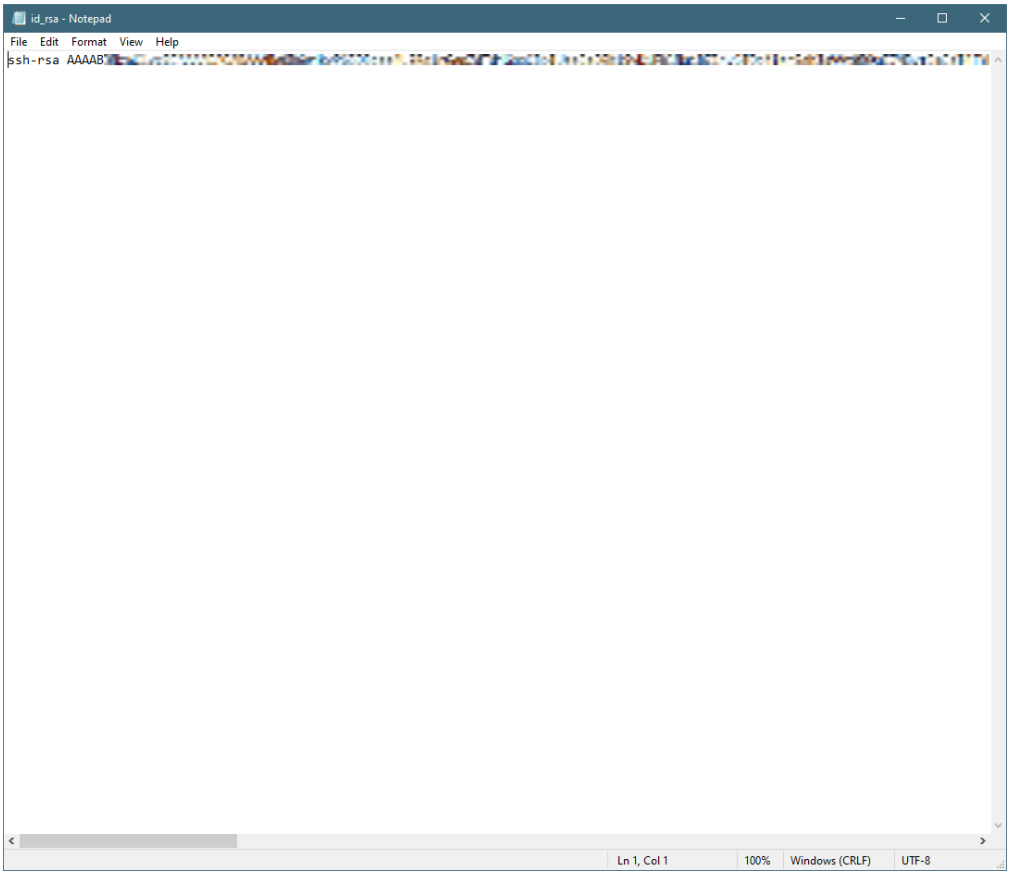
9. Leave this page open and open a PowerShell terminal. Enter the command “ssh-keygen” and use the default file location.



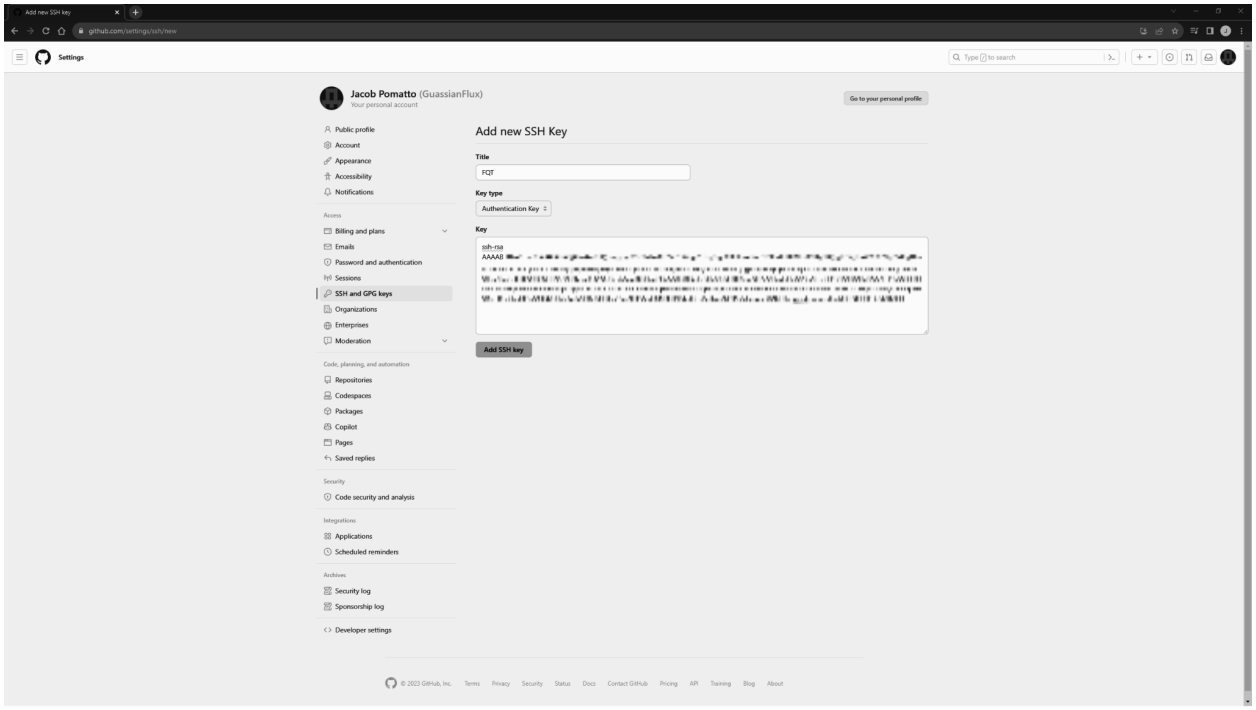
10. Open the generated public key with Notepad.



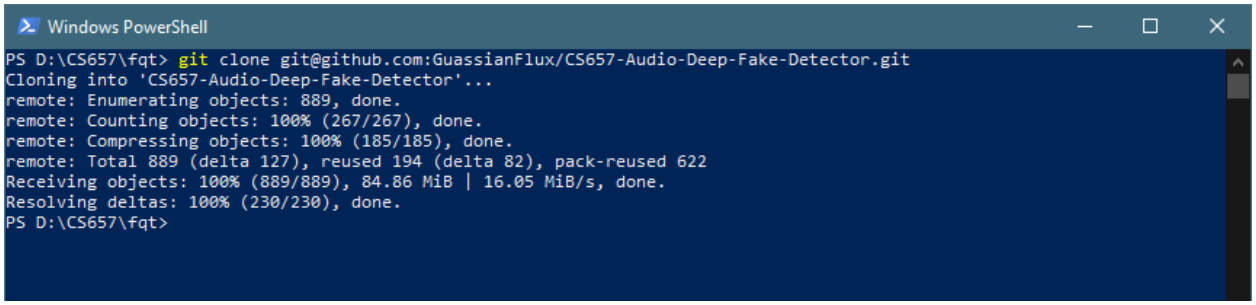
11. Copy the SSH key into your clipboard.



12. Navigate back to the GitHub New SSH Key page. Give your key a title and paste the key from your clipboard into the “Key” text box. Select “Add SSH Key”.



13. In a new PowerShell terminal, navigate to a directory of your choice to store the project and then run the command “git clone git@github.com:GuassianFlux/CS657-Audio-Deep-Fake-Detector.git”.



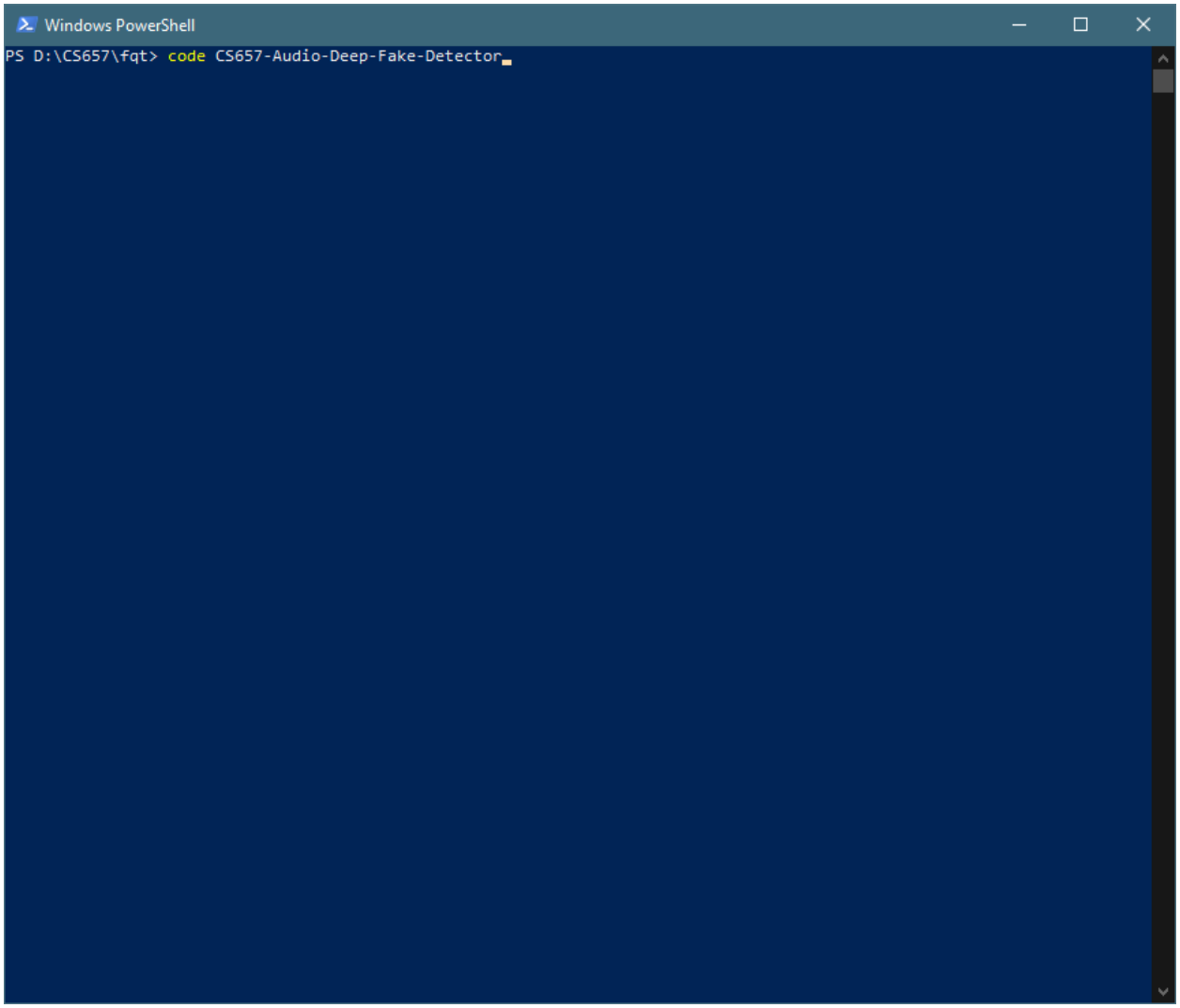
Downloading Source Code

Pass/Fail	Comments
☐ Pass ☐ Fail	

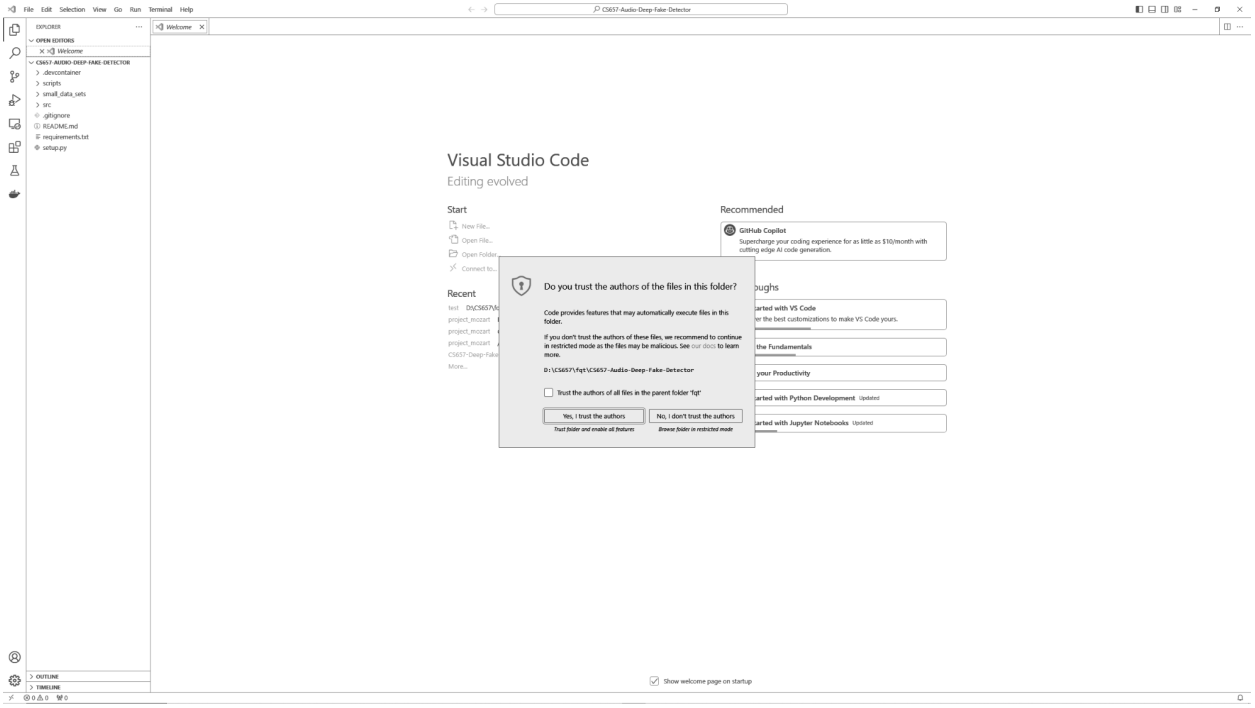
Installing Software Dependencies

Although the Dev Containers extension offers the Python environment, additional software dependencies are necessary to support the project. Modern machine learning libraries are included into the project to accelerate development productivity.

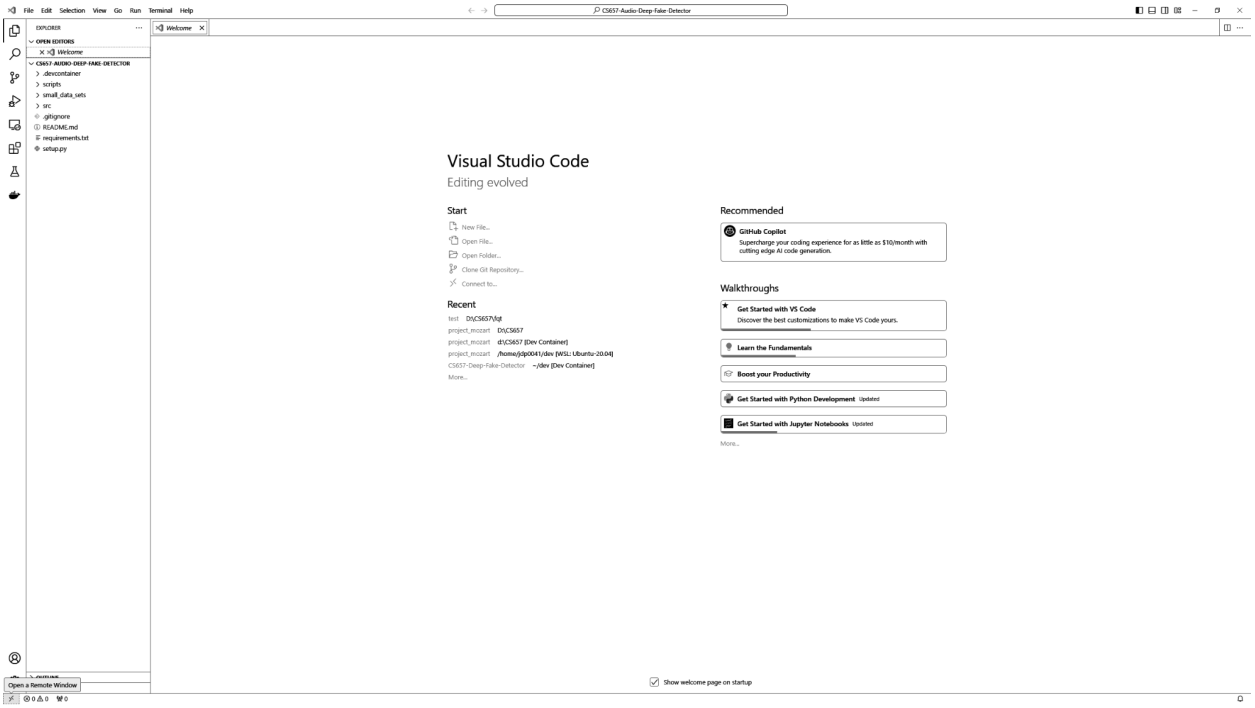
1. Open the project with Visual Studio Code using the command “code CS657-Audio-Deep-Fake-Detector”.



2. Select “Yes, I trust the authors” to enable extensions.

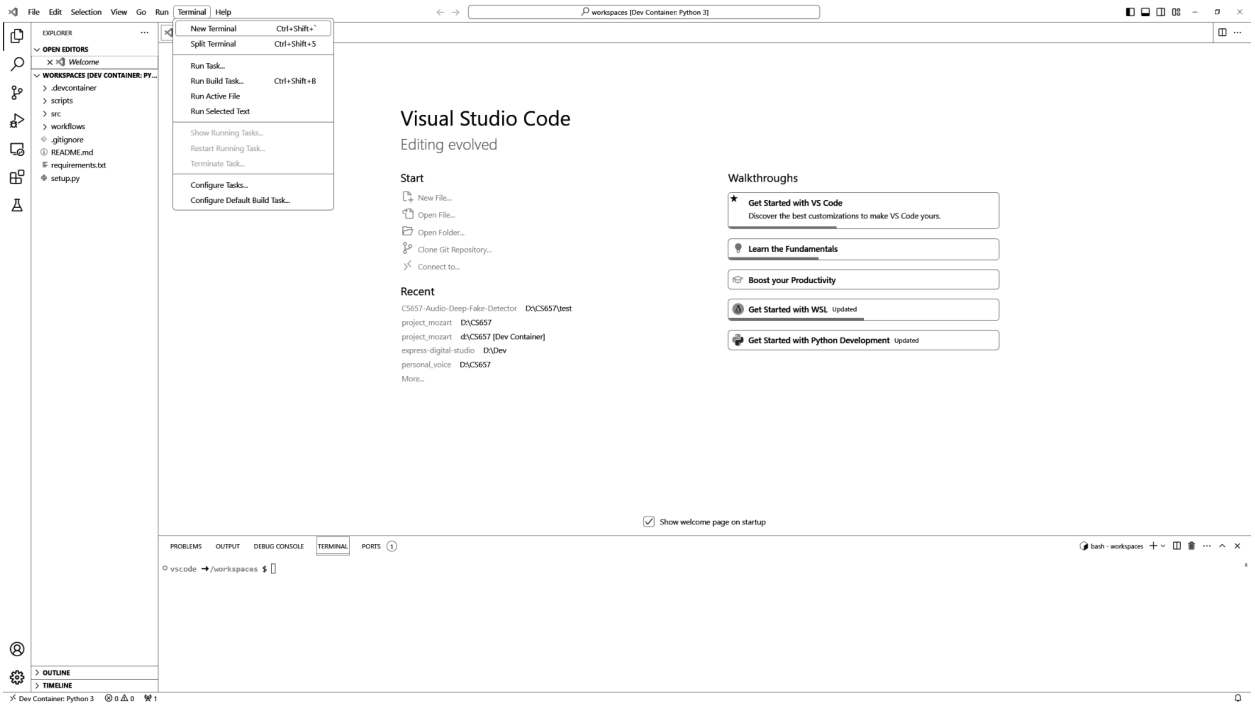


3. In the bottom left corner of the screen, select “Open a Remote Window”.

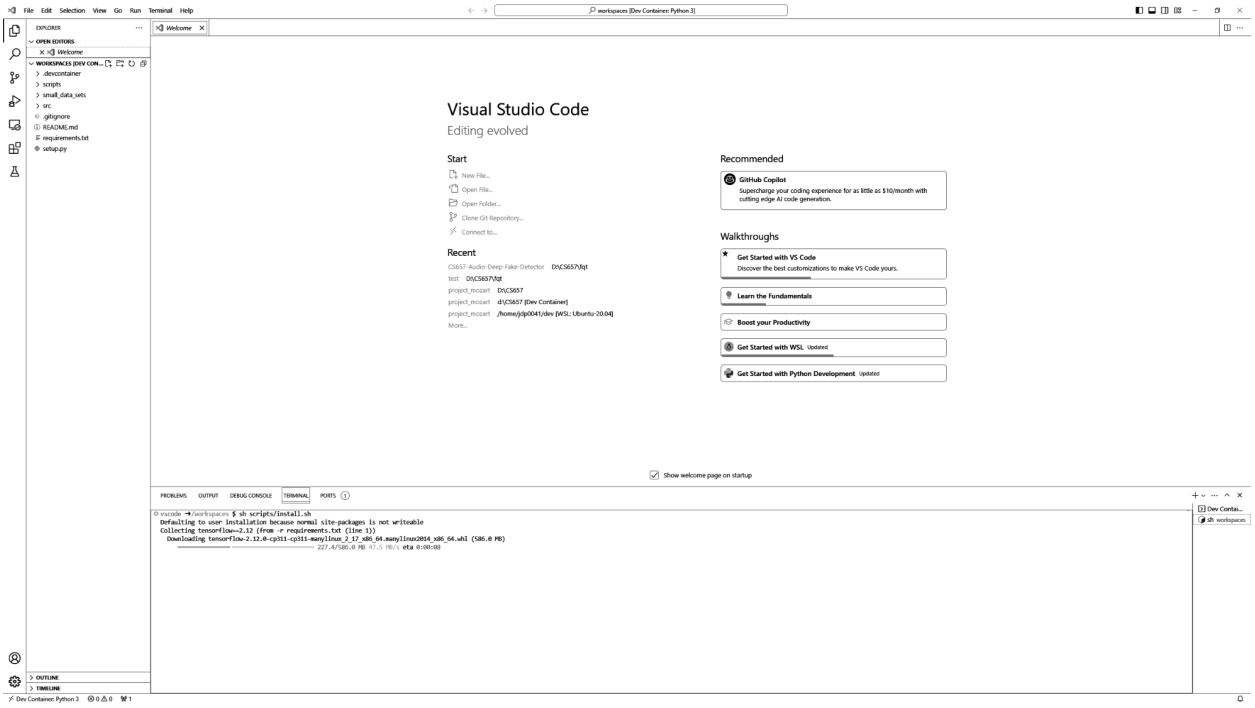


4. In the menu that displayed from the previous action, select “Reopen in Container” to enter the development container. This process may take a few minutes.

5. Once in the container, create a new terminal by selecting “Terminal” > “New Terminal” found at the top of the screen. [pass/fail]



6. Run the command “sh scripts/install.sh” to install all of the necessary software dependencies.



Installing Software Dependencies

Pass/Fail	Comments
☐ Pass ☐ Fail	

Training Model

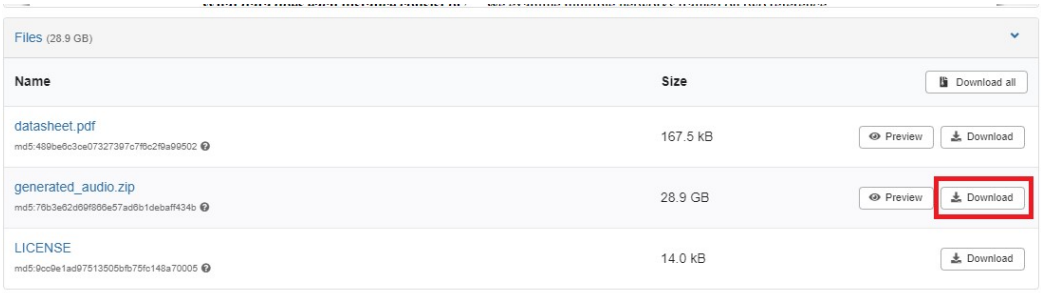
Downloading and Configuring Datasets

To train the audio deep fake detection model, datasets of real and deep fake audio files must be sourced. The audio datasets used for this project were also used for the WaveFake Research Project.

The GitHub repo can be found at the following link: https://github.com/RUB-SysSec/WaveFake.

The Research paper associated with this research can be found at the following link: https://arxiv.org/pdf/2111.02813.pdf.

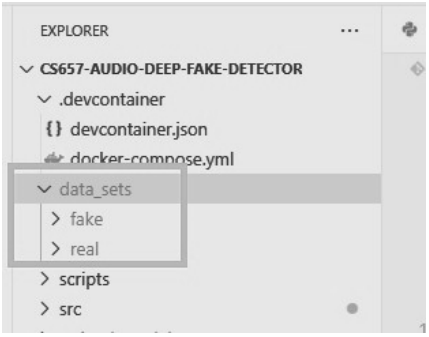
1. Download the generated (fake) audio from the following link: https://zenodo.org/records/5642694. This dataset contains over 100,000 audio files that have been created using different synthesis algorithms. Once downloaded, extract the package.



2. Download the real audio data from the following link: https://keithito.com/LJ-Speech-Dataset/. This dataset was used to create some of the generated audio.

3. During the FQT, the two above downloads will be initiated, but pre-downloaded instances of the data will be used for the remainder of the procedure.

4. Create a folder called “data_sets” that is adjacent to the “src” and “scripts” folder. Create one child folder for fake data and one child folder for real data. Place all of the real and fake data from the links above in the appropriate folders. You can use the following subset for testing:
    - Fake - generated_audio/common_voices_prompts_from_conformer_fastspeech2_pwg_ljspeech/gen_(0-200).wav
    - Real - LJSpeech-1.1/wavs/LJ001-*.wav



Downloading and Configuring Datasets

Pass/Fail	Comments
☐ Pass ☐ Fail	

Loading Data

The machine learning libraries used in the project enable developers to load datasets and supply labels based on the file organization of those data. The audio files must be organized in two directories: real and fake. Use the following steps to test this and view the output when a dataset is loaded.

1. Navigate into "/workspaces/workflows" and run the shell script entitled "1_Loading_Data.sh" to load the real and fake data in the data_sets folder. If you receive parsing errors, run the following command "sed -i -e 's/\r$//' *.sh".

```
workflows > $ 1_Loading_Data.sh
1   #!/bin/bash
2
3   # Workflow 1 - Loading Data: Test for loading a dataset from a directory structure
4   echo "=== Workflow 1 - Loading Data: Test for loading a dataset from a directory structure ==="
5
6   dataset_path='/workspaces/data_sets'
7
8   # Fit a new model
9   python /workspaces/src/load_data.py $dataset_path
```

2. Ensure the output of the script shows the total number of files in the dataset, the number of classes, and training/validation split.

```
vscode ➜/workspaces/workflows $ ./1_Loading_Data.sh
=== Workflow 1 - Loading Data: Test for loading a dataset from a directory structure ===
Loading data sets from /workspaces/data_sets
Found 407 files belonging to 2 classes.
Using 326 files for training.
Using 81 files for validation.
```

Loading Data

Pass/Fail	Comments
☐ Pass	

☐ Fail	

Fitting Model

Though the model's architecture is already designed, the internal weights of the model must be trained to fit the detection use case of the project. Use the following steps to test this and view the output when a dataset is loaded and the model is trained.

1. Run the shell script entitled "2_Fitting_Model.sh" to load the data and train the model.

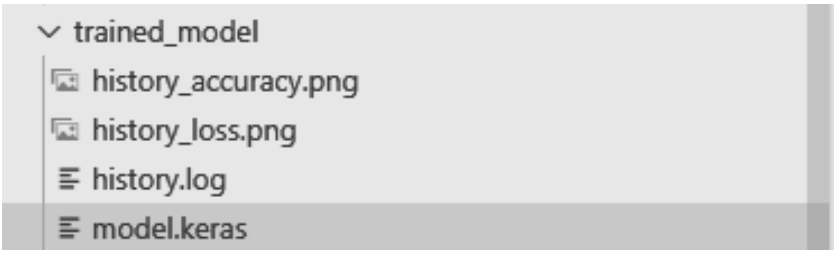
workflows > $ 2_Fitting_Model.sh

```
#!/bin/bash

# Workflow 2 - Fitting Model: Normal training of the model with the default dataset
echo "=== Workflow 2 - Fitting Model: Normal training of the model with the default dataset ==="

dataset_path='/workspaces/data_sets'
trained_model_path='/workspaces/trained_model'

# Fit a new model
python /workspaces/src/fit_model.py $dataset_path -tmp $trained_model_path
```

2. Ensure the output of the script will show the information related to the loaded data, the metrics after each epoch, and the path where the model is saved to file.

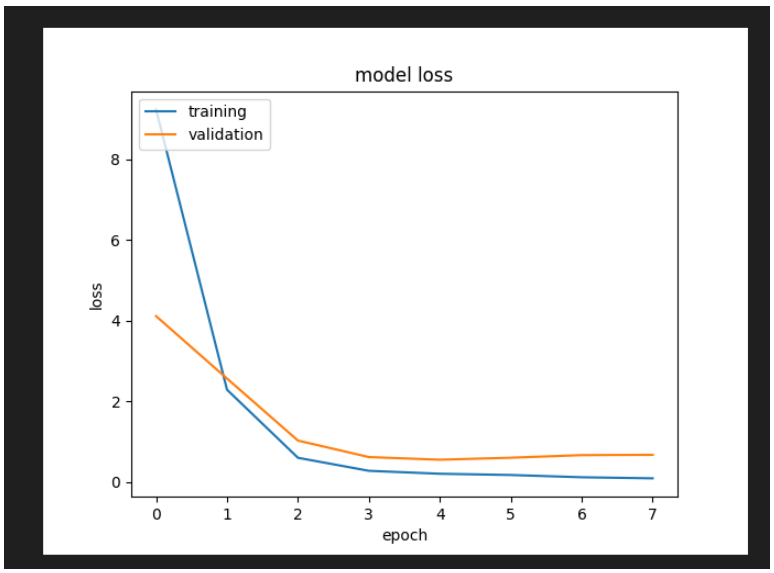
```
vscode ➜/workspaces/workflows $ ./2_Fitting_Model.sh
=== Workflow 2 - Fitting Model: Normal training of the model with the default dataset ===
Loading data sets from /workspaces/data_sets
Found 407 files belonging to 2 classes.
Using 326 files for training.
Using 81 files for validation.
Epoch 1/25
6/6 [==============================] - 13s 2s/step - loss: 9.2273 - accuracy: 0.5859 - val_loss: 4.1102 - val_accuracy: 0.7188
Epoch 2/25
6/6 [==============================] - 8s 1s/step - loss: 2.2877 - accuracy: 0.7975 - val_loss: 2.5641 - val_accuracy: 0.6562
Epoch 3/25
6/6 [==============================] - 8s 1s/step - loss: 0.6012 - accuracy: 0.8620 - val_loss: 1.0276 - val_accuracy: 0.7656
Epoch 4/25
6/6 [==============================] - 8s 1s/step - loss: 0.2785 - accuracy: 0.8558 - val_loss: 0.6191 - val_accuracy: 0.7344
Epoch 5/25
6/6 [==============================] - 8s 1s/step - loss: 0.2059 - accuracy: 0.9509 - val_loss: 0.5535 - val_accuracy: 0.7031
Epoch 6/25
6/6 [==============================] - 8s 1s/step - loss: 0.1751 - accuracy: 0.9632 - val_loss: 0.6029 - val_accuracy: 0.7656
Epoch 7/25
6/6 [==============================] - 9s 1s/step - loss: 0.1189 - accuracy: 0.9632 - val_loss: 0.6670 - val_accuracy: 0.7812
Epoch 8/25
6/6 [==============================] - 8s 1s/step - loss: 0.0924 - accuracy: 0.9693 - val_loss: 0.6737 - val_accuracy: 0.7812
Epoch 8: early stopping
Saving model to path /workspaces/trained_model/model.keras
```

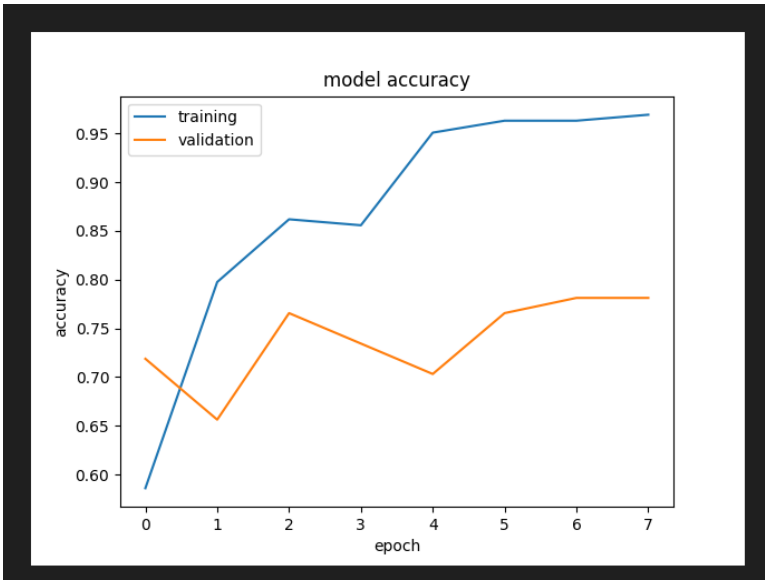
3. Ensure that the "trained_model" folder was created adjacent to the "src" folder. This folder should contain the saved keras model. And should contain several files that represent the metrics on the training.



trained_model > history.log

```
epoch,accuracy,loss,val_accuracy,val_loss
0,0.5858895778656006,9.227331161499023,0.71875,4.110232830047607
1,0.7975460290908813,2.287679433822632,0.65625,2.5641212463378906
2,0.8619632124900818,0.6012099385261536,0.765625,1.0275564193725586
3,0.8558282256126404,0.278491735458374,0.734375,0.6190707683563232
4,0.9509202241897583,0.20594993233680725,0.703125,0.553461492061615
5,0.9631901979446411,0.1751130372285843,0.765625,0.6028811931610107
6,0.9631901979446411,0.11894137412309647,0.78125,0.667044997215271
7,0.9693251252174377,0.09238245338201523,0.78125,0.673671543598175

```





Fitting Model

Pass/Fail	Comments
☐ Pass ☐ Fail	

Making Predictions

Loading Saved Models

Previously fit models may be loaded from storage to quickly adapt the deep fake detector to different scenarios.

1. Run the shell script entitled “3_Load_Saved_Model.sh” to test and view the output when a model is loaded from a file. Before running the script, make sure the keras file is in the “trained_model” folder.

```
workflows > $ 3_Load_Saved_Model.sh
1  #!/bin/bash
2
3  # Workflow 3 - Load Saved Model: Loading a model that has been saved to file
4  echo "=== Workflow 3 - Load Saved Model: Loading a model that has been saved to file ==="
5
6  trained_model_path='/workspaces/trained_model'
7
8  # Load saved model
9  python /workspaces/src/load_saved_model.py $trained_model_path
```

2. Ensure that the output of the script shows the metrics and layer architecture of the loaded model.

```
vscode ➜/workspaces/workflows $ ./3_Load_Saved_Model.sh
=== Workflow 3 - Load Saved Model: Loading a model that has been saved to file ===
   epoch  accuracy      loss  val_accuracy  val_loss
0      0  0.585890  9.227331      0.718750  4.110233
1      1  0.797546  2.287679      0.656250  2.564121
2      2  0.861963  0.601210      0.765625  1.027556
3      3  0.855828  0.278492      0.734375  0.619071
4      4  0.950920  0.205950      0.703125  0.553461
5      5  0.963190  0.175113      0.765625  0.602881
6      6  0.963190  0.118941      0.781250  0.667045
7      7  0.969325  0.092382      0.781250  0.673672
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 1122, 127, 32)     320

 max_pooling2d (MaxPooling2  (None, 561, 63, 32)       0
 D)

 dropout (Dropout)           (None, 561, 63, 32)       0

 flatten (Flatten)           (None, 1130976)           0

 dense (Dense)               (None, 128)               144765056

 dense_1 (Dense)             (None, 1)                 129

=================================================================
Total params: 144765505 (552.24 MB)
Trainable params: 144765505 (552.24 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Loading Saved Models

Pass/Fail	Comments
☐ Pass ☐ Fail	

Making Predictions

When supplied with a new audio file or test dataset, the trained model may predict whether it is real or fake in origin.

1. Run the shell script entitled “4_Making_Predictions.sh” to test and view the output of making a prediction on a dataset with a loaded model. Before running the script, make sure the keras file is in the “trained_model” folder. This script will load the trained model. Load the dataset and make a prediction on a subset of the dataset. The prediction dataset can be replaced by another test dataset folder that has real and fake children.

workflows > $ 4_Making_Predictions.sh

```
#!/bin/bash

# Workflow 2: Prediction of the dataset with existing model
echo "=== Workflow 4: Prediction of the dataset with existing model ==="

dataset_path='/workspaces/data_sets'
trained_model_path='/workspaces/trained_model'
prediction_data_path='/workspaces/prediction_data'

# Make predictions
python /workspaces/src/make_predictions.py $dataset_path -tmp $trained_model_path -pdp $prediction_data_path
```

2. Ensure that the output of the script shows that the dataset was loaded.

```
vscode ➜/workspaces/workflows $ ./4_Making_Predictions.sh
=== Workflow 4: Prediction of the dataset with existing model ===
Loading data sets from /workspaces/data_sets
```

3. Ensure that the out of the script shows that the model was loaded and correctly displays the metrics and architecture of the model.

	epoch	accuracy	loss	val_accuracy	val_loss
0	0	0.622699	4.612930	0.703125	1.858111
1	1	0.769939	0.992546	0.750000	0.739502
2	2	0.837423	0.327513	0.781250	0.569403
3	3	0.926380	0.199120	0.734375	0.604590
4	4	0.929448	0.175378	0.750000	0.600693
5	5	0.966258	0.107851	0.781250	0.563935
6	6	0.981595	0.074296	0.781250	0.584582
7	7	0.993865	0.049397	0.765625	0.657892
8	8	0.996933	0.031415	0.781250	0.692003

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 1122, 127, 32)     320

 max_pooling2d (MaxPooling2  (None, 561, 63, 32)       0
 D)

 dropout (Dropout)           (None, 561, 63, 32)       0

 flatten (Flatten)           (None, 1130976)           0

 dense (Dense)               (None, 128)               144765056

 dense_1 (Dense)             (None, 1)                 129

=================================================================
Total params: 144765505 (552.24 MB)
Trainable params: 144765505 (552.24 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

4. Ensure that the output of the scripts shows the predictions that were made on the test dataset files and the accuracy of the prediction.

```
Making dataset predictions...
1/1 [==============================] - 0s 228ms/step
WAV 1 Prediction: real, Actual: real, Confidence: 91.6523%
conv2d_input (1, 1124, 129, 1)
conv2d (1, 1122, 127, 32)
max_pooling2d (1, 561, 63, 32)
dropout (1, 561, 63, 32)
flatten (1, 1130976)
dense (1, 128)
dense_1 (1, 1)
WAV 2 Prediction: fake, Actual: fake, Confidence: 97.4554%
WAV 3 Prediction: fake, Actual: fake, Confidence: 99.9397%
WAV 4 Prediction: fake, Actual: fake, Confidence: 99.0572%
WAV 5 Prediction: real, Actual: real, Confidence: 99.9997%
WAV 6 Prediction: real, Actual: real, Confidence: 99.9498%
WAV 7 Prediction: real, Actual: real, Confidence: 99.9924%
WAV 8 Prediction: fake, Actual: real, Confidence: 89.9884%
WAV 9 Prediction: fake, Actual: fake, Confidence: 73.3782%
WAV 10 Prediction: real, Actual: real, Confidence: 62.1102%
WAV 11 Prediction: fake, Actual: fake, Confidence: 97.7663%
WAV 12 Prediction: real, Actual: real, Confidence: 50.0224%
WAV 13 Prediction: fake, Actual: real, Confidence: 96.9325%
WAV 14 Prediction: fake, Actual: fake, Confidence: 83.6395%
WAV 15 Prediction: fake, Actual: fake, Confidence: 96.1454%
WAV 16 Prediction: fake, Actual: fake, Confidence: 99.1222%
WAV 17 Prediction: fake, Actual: fake, Confidence: 99.3980%
Number correct: 15 out of 17
Accuracy: 88.2353%
```

Making Predictions

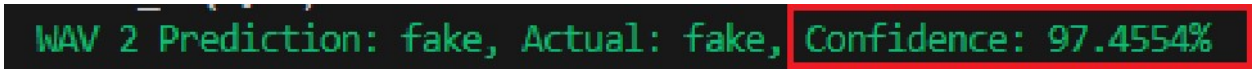
Pass/Fail	Comments
☐ Pass ☐ Fail	

Verifying Prediction Output

After making a prediction on a dataset, the prediction scripts will print output in the terminal that will show the prediction made for each file in the test dataset. A prediction, actual classification, and confidence level will be shown for each of the files in the dataset. There will also be an accuracy displayed at the bottom of the output.

1. Verify that the output for each wav file shows the prediction and actual classification.

2. Verify that the output for each wav file shows the confidence level of the predictions



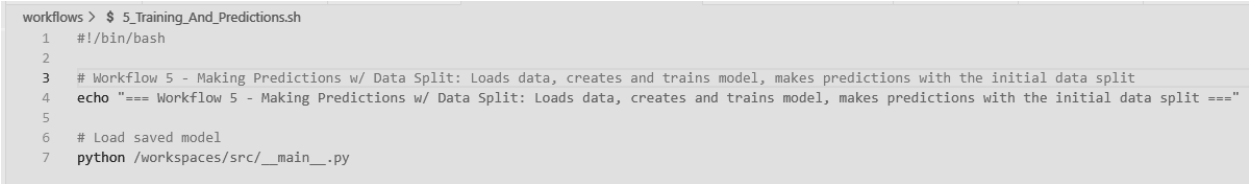
Interpreting Output

Pass/Fail	Comments
☐ Pass ☐ Fail	

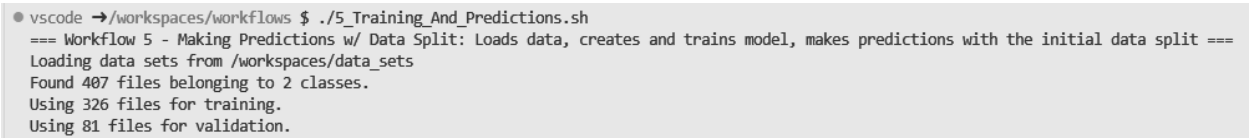
Training and Making Predictions with a Single Dataset

Training and prediction can be done with one script when the dataset is split properly. The data can be split into training, validation, and test datasets.

1. Run the shell script called “5_Training_And_Predictions.sh” to test the entire process of creating a dataset, splitting the data, fitting the model, saving the model, loading the model, and making predictions with the validation data.



2. Verify that the output shows the data split



3. Verify that the output shows all epoch metrics from fitting the model

```
Epoch 1/25
6/6 [==============================] - 7s 986ms/step - loss: 4.9031 - accuracy: 0.6319 - val_loss: 2.8830 - val_accuracy: 0.6094
Epoch 2/25
6/6 [==============================] - 5s 816ms/step - loss: 1.6283 - accuracy: 0.7301 - val_loss: 1.0996 - val_accuracy: 0.7500
Epoch 3/25
6/6 [==============================] - 5s 818ms/step - loss: 0.5658 - accuracy: 0.8160 - val_loss: 0.8126 - val_accuracy: 0.7500
Epoch 4/25
6/6 [==============================] - 5s 829ms/step - loss: 0.1647 - accuracy: 0.9325 - val_loss: 0.7880 - val_accuracy: 0.6875
Epoch 5/25
6/6 [==============================] - 5s 886ms/step - loss: 0.1127 - accuracy: 0.9601 - val_loss: 0.6426 - val_accuracy: 0.7812
Epoch 6/25
6/6 [==============================] - 5s 885ms/step - loss: 0.0626 - accuracy: 0.9724 - val_loss: 0.6559 - val_accuracy: 0.7656
Epoch 7/25
6/6 [==============================] - 5s 880ms/step - loss: 0.0405 - accuracy: 0.9939 - val_loss: 0.6251 - val_accuracy: 0.7969
Epoch 8/25
6/6 [==============================] - 5s 901ms/step - loss: 0.0246 - accuracy: 0.9969 - val_loss: 0.6474 - val_accuracy: 0.7812
Epoch 9/25
6/6 [==============================] - 6s 910ms/step - loss: 0.0166 - accuracy: 1.0000 - val_loss: 0.7245 - val_accuracy: 0.7969
Epoch 10/25
6/6 [==============================] - 7s 1s/step - loss: 0.0119 - accuracy: 1.0000 - val_loss: 0.7608 - val_accuracy: 0.7812
Epoch 10: early stopping
```

4. Verify that the output shows the path of the keras model, the metrics, and the layer architecture of the model.

```
Saving model to path /workspaces/trained_model/model.keras
   epoch  accuracy      loss  val_accuracy  val_loss
0      0  0.631902  4.903122      0.609375  2.883035
1      1  0.730061  1.628287      0.750000  1.099602
2      2  0.815951  0.565821      0.750000  0.812592
3      3  0.932515  0.164665      0.687500  0.788040
4      4  0.960123  0.112716      0.781250  0.642584
5      5  0.972393  0.062553      0.765625  0.655865
6      6  0.993865  0.040477      0.796875  0.625073
7      7  0.996933  0.024617      0.781250  0.647375
8      8  1.000000  0.016605      0.796875  0.724477
9      9  1.000000  0.011922      0.781250  0.760805
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 1122, 127, 32)     320

 max_pooling2d (MaxPooling2  (None, 561, 63, 32)       0
 D)

 dropout (Dropout)           (None, 561, 63, 32)       0

 flatten (Flatten)           (None, 1130976)           0

 dense (Dense)               (None, 128)               144765056

 dense_1 (Dense)             (None, 1)                 129

=================================================================
Total params: 144765505 (552.24 MB)
```

5. Verify that the predictions made of the validation data is shown in the script output

```
Making dataset predictions...
1/1 [==============================] - 0s 119ms/step
WAV 1 Prediction: real, Actual: real, Confidence: 95.5988%
conv2d_input (1, 1124, 129, 1)
conv2d (1, 1122, 127, 32)
max_pooling2d (1, 561, 63, 32)
dropout (1, 561, 63, 32)
flatten (1, 1130976)
dense (1, 128)
dense_1 (1, 1)
WAV 2 Prediction: fake, Actual: fake, Confidence: 98.3600%
WAV 3 Prediction: fake, Actual: fake, Confidence: 99.9833%
WAV 4 Prediction: fake, Actual: fake, Confidence: 96.4075%
WAV 5 Prediction: real, Actual: real, Confidence: 100.0000%
WAV 6 Prediction: real, Actual: real, Confidence: 99.9964%
WAV 7 Prediction: real, Actual: real, Confidence: 99.9998%
WAV 8 Prediction: fake, Actual: real, Confidence: 83.7892%
WAV 9 Prediction: fake, Actual: fake, Confidence: 50.1674%
WAV 10 Prediction: real, Actual: real, Confidence: 85.7522%
WAV 11 Prediction: fake, Actual: fake, Confidence: 99.5641%
WAV 12 Prediction: real, Actual: real, Confidence: 88.6998%
WAV 13 Prediction: fake, Actual: real, Confidence: 93.5647%
WAV 14 Prediction: fake, Actual: fake, Confidence: 80.4446%
WAV 15 Prediction: fake, Actual: fake, Confidence: 92.0841%
WAV 16 Prediction: fake, Actual: fake, Confidence: 99.8195%
WAV 17 Prediction: fake, Actual: fake, Confidence: 99.6310%
Number correct: 15 out of 17
Accuracy: 88.2353%
```

Training and Making Predictions with a Single Dataset

Pass/Fail	Comments
☐ Pass ☐ Fail	

Experimentation

Adding White Noise

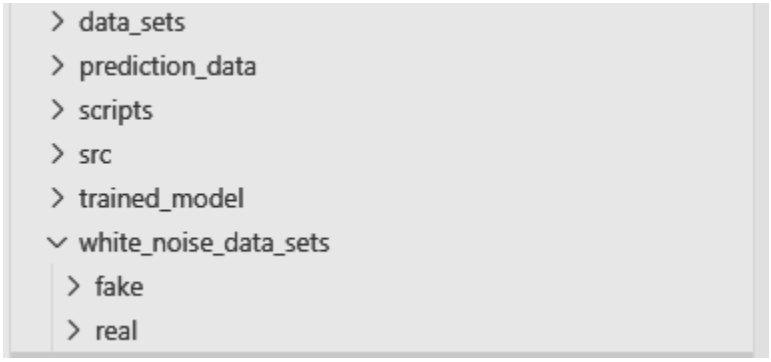
To determine how noise may impact the detection accuracy of the model, a provided script will add white noise to the requested audio files.

1. Run the shell script entitled “6_Adding_White_Noise.sh”

workflows > $ 6_Adding_White_Noise.sh

```
#!/bin/bash

# Workflow 6 - Adding White Noise: Create white noise datasets
echo "=== Workflow 6 - Adding White Noise: Create white noise datasets ==="

root_noise_dir='/workspaces/white_noise_data_sets'
if [ -d $root_noise_dir ]; then
    rm -r $root_noise_dir
fi
mkdir $root_noise_dir

training_dataset_path='/workspaces/data_sets'
real_dataset_path='/workspaces/data_sets/real'
real_white_noise_dataset_path='/workspaces/white_noise_data_sets/real'
fake_dataset_path='/workspaces/data_sets/fake'
fake_white_noise_dataset_path='/workspaces/white_noise_data_sets/fake'

# Add white noise to real data
python /workspaces/src/make_noise_dataset.py $real_dataset_path $real_white_noise_dataset_path --snr 15 --type white

# Add white noise to fake data
python /workspaces/src/make_noise_dataset.py $fake_dataset_path $fake_white_noise_dataset_path --snr 15 --type white
```

2. Verify that a folder named “white_noise_data_sets” was created adjacent to the “data_sets” folder.



3. Open the real and fake subfolders to make sure that the wav files have been generated.

4. Click on one of the wav files and click the play button to listen to the file and to make sure that white noise has been added.

Adding White Noise

Pass/Fail	Comments
☐ Pass ☐ Fail	

Adding Burst Noise

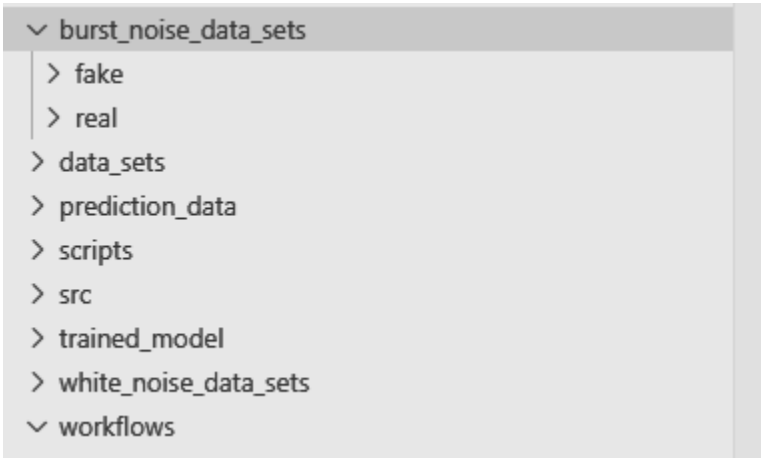
To determine how noise may impact the detection accuracy of the model, a provided script will add burst noise to the requested audio files.

1. Run the script entitled “7_Adding_Burst_Noise.sh”.

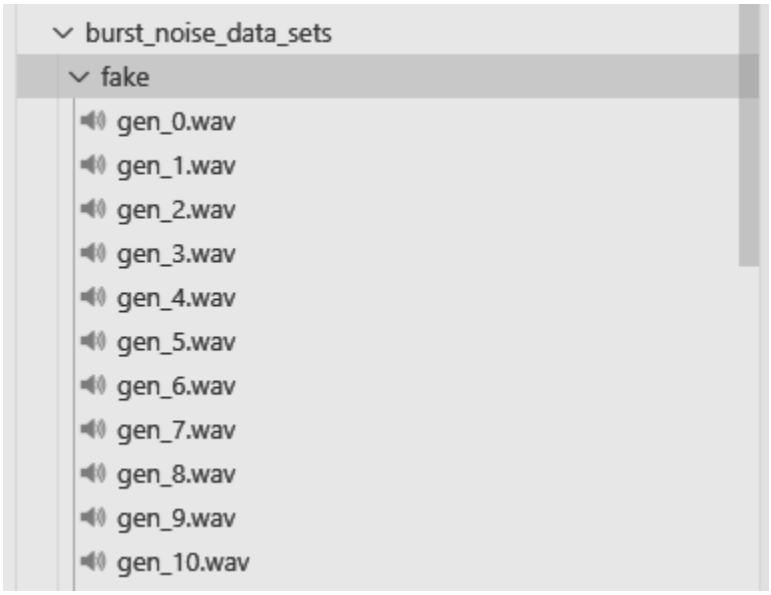
workflows > $ 7_Adding_Burst_Noise.sh

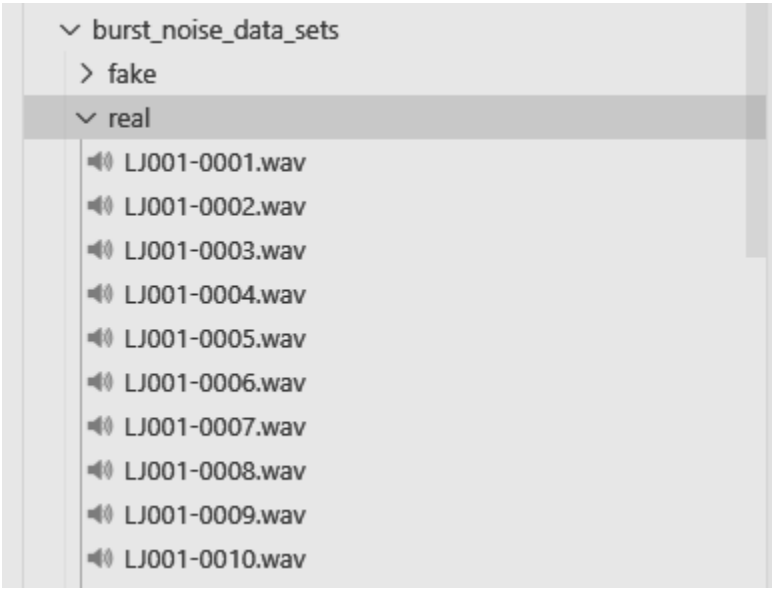
```
#!/bin/bash

# Workflow 7 - Adding Burst Noise: Create burst noise datasets
echo "=== Workflow 7 - Adding Burst Noise: Create burst noise datasets ==="

root_noise_dir='/workspaces/burst_noise_data_sets'
if [ -d $root_noise_dir ]; then
    rm -r $root_noise_dir
fi
mkdir $root_noise_dir

training_dataset_path='/workspaces/data_sets'
real_dataset_path='/workspaces/data_sets/real'
real_burst_noise_dataset_path='/workspaces/burst_noise_data_sets/real'
fake_dataset_path='/workspaces/data_sets/fake'
fake_burst_noise_dataset_path='/workspaces/burst_noise_data_sets/fake'

# Add burst noise to real data
python /workspaces/src/make_noise_dataset.py $real_dataset_path $real_burst_noise_dataset_path --snr 15 --type burst

# Add burst noise to fake data
python /workspaces/src/make_noise_dataset.py $fake_dataset_path $fake_burst_noise_dataset_path --snr 15 --type burst
```

2. Verify that a folder named “burst_noise_data_sets” was created adjacent to the “data_sets” folder.



3. Open the real and fake subfolders and make sure there are generated files.





4. Click on one of the wav files and click the play button to listen to the file and to make sure that burst noise has been added.

Adding Burst Noise

Pass/Fail	Comments
☐ Pass ☐ Fail	

Noise Filtering

To determine how noise filtering may impact the detection accuracy of the model, a provided script will remove noise from the requested audio files.

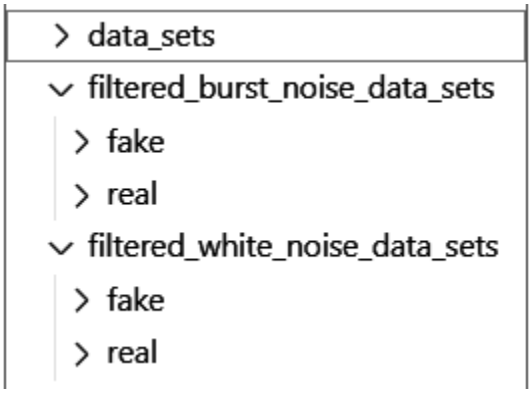
1. Run the script entitled “8_Noise_Filtering.sh” to filter noise from the files in the white noise dataset. Make sure this dataset has been created before running the script.

workflows > $ 8_Noise_Filtering.sh

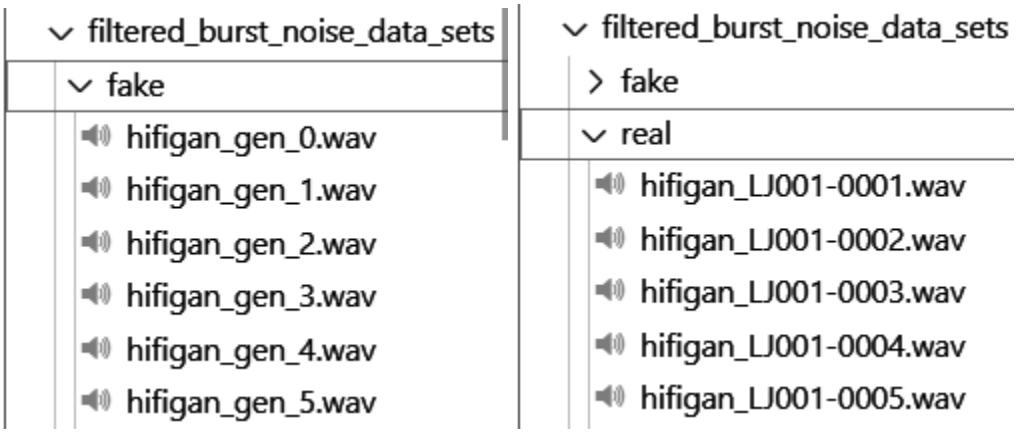
```
#!/bin/bash

# Workflow 8 - Noise Filtering: Filtering the white noise dataset
echo "=== Workflow 8 - Noise Filtering: Filtering the white noise dataset ==="


real_white_noise_dataset_path='/workspaces/white_noise_data_sets/real'
fake_white_noise_dataset_path='/workspaces/white_noise_data_sets/fake'


root_filtered_noise_dir='/workspaces/filtered_noise_data_sets'
if [ -d $root_filtered_noise_dir ]; then
    rm -r $root_filtered_noise_dir
fi
mkdir $root_filtered_noise_dir

real_filtered_noise_dir='/workspaces/filtered_noise_data_sets/real'
if [ -d $real_filtered_noise_dir ]; then
    rm -r $real_filtered_noise_dir
fi
mkdir $real_filtered_noise_dir

fake_filtered_noise_dir='/workspaces/filtered_noise_data_sets/fake'
if [ -d $fake_filtered_noise_dir ]; then
    rm -r $fake_filtered_noise_dir
fi
mkdir $fake_filtered_noise_dir

# Filter noise for real data
python /workspaces/src/filter_noise_dataset.py $real_white_noise_dataset_path $real_filtered_noise_dir

# Filter noise for fake data
python /workspaces/src/filter_noise_dataset.py $fake_white_noise_dataset_path $fake_filtered_noise_dir
```

```
vscode ➜/workspaces/workflows $ ./8_Noise_Filtering.sh
=== Workflow 8 - Noise Filtering: Filtering the white noise dataset ===
vscode ➜/workspaces/workflows $
```

2. Verify that a folders named “filtered_white_noise_data_sets” and “filtered_burst_noise_data_sets” have been created adjacent to the “data_sets” folder.



3. Open the real and fake subfolders for each and make sure files have been created.



4. Listen to the files and make sure the white noise has been removed. It should sound similar to the original audio before the static or popping noises were introduced.

Noise Filtering

Pass/Fail	Comments
☐ Pass ☐ Fail	

Collecting Results

With all noise variants of the dataset produced, various metrics may be collected to evaluate the performance of the model. Given the random variability in training the model, the results produced during the FQT may not exactly match the provided screenshots.

1. Execute the script entitled “9_Collect_Metrics.sh”. You may optionally provide the number of test runs to be conducted. If a value is not specified, it will only run once.

```
vscode ➜ /workspaces $ sh workflows/9_Collect_Metrics.sh
```

2. Once the script completes, verify that the performance metrics have been displayed in the terminal.

```
Baseline Training and Prediction:
Accuracy: 84.6154
EER: 7.0312

Baseline Training and White Noise Prediction:
Accuracy: 86.1538
EER: 6.25

White Noise Training and Prediction:
Accuracy: 87.6923
EER: 5.4688

Baseline Training and Burst Noise Prediction:
Accuracy: 78.4615
EER: 10.1562

Baseline Training and Filtered Burst Noise Prediction:
Accuracy: 83.0769
EER: 7.8125

Burst Noise Training and Prediction:
Accuracy: 86.1538
EER: 6.25

Baseline Training and Filtered White Noise Prediction:
Accuracy: 78.4615
EER: 10.1562
```

3. Arrange the metrics into a presentable table

FQT	Avg Prediction Accuracy	Equivalent Error Rate
Baseline	84.6154	7.0312
White Noise - No Training	86.1538	6.2500
White Noise - Training	87.6923	5.4688
White Noise - Filtered	78.4615	10.1562
Burst Noise - No Training	83.0769	7.8125
Burst Noise - Training	86.1538	6.2500
Burst Noise - Filtered	78.4615	10.1562

Collecting Results

Pass/Fail	Comments
☐ Pass ☐ Fail	

Appendix

Project Repository

The source code for this project is public on GitHub and can be found at the following location:
https://github.com/GuassianFlux/CS657-Audio-Deep-Fake-Detector

Repository Contents

The project repository contains the following contents:

- Instructions for setting up the development environment
- DevContainer configuration files
- Scripts for installing all dependencies
- Source code for loading datasets, fitting the model, visualizing data, and making predictions
- High level workflow scripts
- Documentation:
  - Design Manual: Outlines the software design and project architecture
  - User Manual: Explains and demonstrates how the software is used
  - Final Qualification Test: Explains how to setup the development environment and reproduce results
  - Research Paper: Summaries the project from beginning to end and presents the final results

How to Contribute

Use the following command to clone the code repository.

```
$ git clone https://github.com/GuassianFlux/CS657-Audio-Deep-Fake-Detector
```

Scripts

install.sh

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

1_Loading_Data.sh

```
#!/bin/bash

# Workflow 1 - Loading Data: Test for loading a dataset from a directory
structure
echo "=== Workflow 1 - Loading Data: Test for loading a dataset from a
directory structure ==="
```

```
dataset_path='/workspaces/data_sets'

# Fit a new model
python /workspaces/src/load_data.py $dataset_path
```

2_Fitting_Model.sh

```
#!/bin/bash

# Workflow 2 - Fitting Model: Normal training of the model with the default
dataset
echo "=== Workflow 2 - Fitting Model: Normal training of the model with the
default dataset ==="

dataset_path='/workspaces/data_sets'
trained_model_path='/workspaces/trained_model'

# Fit a new model
python /workspaces/src/fit_model.py $dataset_path -tmp $trained_model_path
```

3_Load_Saved_Model.sh

```
#!/bin/bash

# Workflow 3 - Load Saved Model: Loading a model that has been saved to file
echo "=== Workflow 3 - Load Saved Model: Loading a model that has been saved
to file ==="

trained_model_path='/workspaces/trained_model'

# Load saved model
python /workspaces/src/load_saved_model.py $trained_model_path
```

4_Making_Predictions.sh

```
#!/bin/bash

# Workflow 4: Prediction of the dataset with existing model
echo "=== Workflow 4: Prediction of the dataset with existing model ==="

dataset_path='/workspaces/data_sets'
trained_model_path='/workspaces/trained_model'
prediction_data_path='/workspaces/prediction_data'
```

```
# Make predictions
python /workspaces/src/make_predictions.py $dataset_path -tmp
$trained_model_path -pdp $prediction_data_path
```

5_Training_And_Predictions.sh

```
#!/bin/bash

# Workflow 5 - Making Predictions w/ Data Split: Loads data, creates and
trains model, makes predictions with the initial data split
echo "=== Workflow 5 - Making Predictions w/ Data Split: Loads data, creates
and trains model, makes predictions with the initial data split ==="

# Load saved model
python /workspaces/src/__main__.py
```

6_Adding_White_Noise.sh

```
#!/bin/bash

# Workflow 6 - Adding White Noise: Create white noise datasets
echo "=== Workflow 6 - Adding White Noise: Create white noise datasets ==="

root_noise_dir='/workspaces/white_noise_data_sets'
if [ -d $root_noise_dir ]; then
    rm -r $root_noise_dir
fi
mkdir $root_noise_dir

training_dataset_path='/workspaces/data_sets'
real_dataset_path='/workspaces/data_sets/real'
real_white_noise_dataset_path='/workspaces/white_noise_data_sets/real'
fake_dataset_path='/workspaces/data_sets/fake'
fake_white_noise_dataset_path='/workspaces/white_noise_data_sets/fake'

# Add white noise to real data
python /workspaces/src/make_noise_dataset.py $real_dataset_path
$real_white_noise_dataset_path --snr 15 --type white

# Add white noise to fake data
python /workspaces/src/make_noise_dataset.py $fake_dataset_path
$fake_white_noise_dataset_path --snr 15 --type white
```

7_Adding_Burst_Noise.sh

```
#!/bin/bash

# Workflow 7 - Adding Burst Noise: Create burst noise datasets
```

```
echo "=== Workflow 7 - Adding Burst Noise: Create burst noise datasets ==="

root_noise_dir='/workspaces/burst_noise_data_sets'
if [ -d $root_noise_dir ]; then
    rm -r $root_noise_dir
fi
mkdir $root_noise_dir

training_dataset_path='/workspaces/data_sets'
real_dataset_path='/workspaces/data_sets/real'
real_burst_noise_dataset_path='/workspaces/burst_noise_data_sets/real'
fake_dataset_path='/workspaces/data_sets/fake'
fake_burst_noise_dataset_path='/workspaces/burst_noise_data_sets/fake'

# Add burst noise to real data
python /workspaces/src/make_noise_dataset.py $real_dataset_path
$real_burst_noise_dataset_path --snr 15 --type burst

# Add burst noise to fake data
python /workspaces/src/make_noise_dataset.py $fake_dataset_path
$fake_burst_noise_dataset_path --snr 15 --type burst
```

8_Noise_Filtering.sh

```
#!/bin/bash

# Workflow 8 - Noise Filtering: Filtering the white noise dataset
echo "=== Workflow 8 - Noise Filtering: Filtering the white noise dataset ==="


real_white_noise_dataset_path='/workspaces/white_noise_data_sets/real'
fake_white_noise_dataset_path='/workspaces/white_noise_data_sets/fake'


root_filtered_noise_dir='/workspaces/filtered_noise_data_sets'
if [ -d $root_filtered_noise_dir ]; then
    rm -r $root_filtered_noise_dir
fi
mkdir $root_filtered_noise_dir

real_filtered_noise_dir='/workspaces/filtered_noise_data_sets/real'
if [ -d $real_filtered_noise_dir ]; then
    rm -r $real_filtered_noise_dir
fi
mkdir $real_filtered_noise_dir

fake_filtered_noise_dir='/workspaces/filtered_noise_data_sets/fake'
if [ -d $fake_filtered_noise_dir ]; then
    rm -r $fake_filtered_noise_dir
fi
mkdir $fake_filtered_noise_dir

# Filter noise for real data
```

```
python /workspaces/src/filter_noise_dataset.py $real_white_noise_dataset_path
$real_filtered_noise_dir

# Filter noise for fake data
python /workspaces/src/filter_noise_dataset.py $fake_white_noise_dataset_path
$fake_filtered_noise_dir
```

9_Collect_Metrics.sh

```
#!/bin/bash

# Workflow 9 - Collect Metrics: Evaluate the model against datasets
echo "=== Workflow 9 - Collect Metrics: Evaluate the model against datasets
==="
default=1
count=${1:-$default}

dataset_path='/workspaces/data_sets'
white_noise_dataset_path='/workspaces/white_noise_data_sets'
burst_noise_dataset_path='/workspaces/burst_noise_data_sets'
filtered_noise_dataset_path='/workspaces/filtered_noise_data_sets'

trained_model_path='/workspaces/trained_model'
prediction_data_path='/workspaces/prediction_data'

metrics_root_path='/workspaces/metrics'

if [ -d $metrics_root_path ]; then
    rm -r $metrics_root_path
fi
mkdir $metrics_root_path
mkdir $metrics_root_path/baseline
mkdir $metrics_root_path/white_noise_train
mkdir $metrics_root_path/burst_noise_train
mkdir $metrics_root_path/white_noise_no_train
mkdir $metrics_root_path/burst_noise_no_train
mkdir $metrics_root_path/filtered_noise

for i in $(seq 1 $count)
do
    python /workspaces/src/fit_model.py $dataset_path -tmp $trained_model_path
    python /workspaces/src/make_predictions.py $dataset_path -tmp
$trained_model_path -pdp $prediction_data_path >
$metrics_root_path/baseline/metric_$i.txt
    python /workspaces/src/make_predictions.py $white_noise_dataset_path -tmp
$trained_model_path -pdp $prediction_data_path >
$metrics_root_path/white_noise_no_train/metric_$i.txt
    python /workspaces/src/make_predictions.py $burst_noise_dataset_path -tmp
$trained_model_path -pdp $prediction_data_path >
$metrics_root_path/burst_noise_no_train/metric_$i.txt
    python /workspaces/src/make_predictions.py $filtered_noise_dataset_path
-tmp $trained_model_path -pdp $prediction_data_path >
$metrics_root_path/filtered_noise/metric_$i.txt
```

```
    python /workspaces/src/fit_model.py $white_noise_dataset_path -tmp
$trained_model_path
    python /workspaces/src/make_predictions.py $white_noise_dataset_path -tmp
$trained_model_path -pdp $prediction_data_path >
$metrics_root_path/white_noise_train/metric_$i.txt
    python /workspaces/src/fit_model.py $burst_noise_dataset_path -tmp
$trained_model_path
    python /workspaces/src/make_predictions.py $burst_noise_dataset_path -tmp
$trained_model_path -pdp $prediction_data_path >
$metrics_root_path/burst_noise_train/metric_$i.txt
done

echo 'Baseline Training and Prediction:'
python /workspaces/src/parse_metrics.py $metrics_root_path/baseline
echo '\nBaseline Training and White Noise Prediction:'
python /workspaces/src/parse_metrics.py
$metrics_root_path/white_noise_no_train
echo '\nBaseline Training and Burst Noise Prediction:'
python /workspaces/src/parse_metrics.py
$metrics_root_path/burst_noise_no_train
echo '\nBaseline Training and Filtered Noise Prediction:'
python /workspaces/src/parse_metrics.py $metrics_root_path/filtered_noise
echo '\nWhite Noise Training and Prediction:'
python /workspaces/src/parse_metrics.py $metrics_root_path/white_noise_train
echo '\nBurst Noise Training and Prediction:'
python /workspaces/src/parse_metrics.py $metrics_root_path/burst_noise_train
```

Citations

1. Sakoe, H., and S. Chiba. “Dynamic programming algorithm optimization for spoken word recognition.” IEEE Transactions on Acoustics, Speech, and Signal Processing, vol. 26, no. 1, 1978, pp. 43–49, https://doi.org/10.1109/tassp.1978.1163055.
2. Rabiner, L.R. “A tutorial on Hidden Markov models and selected applications in speech recognition.” *Proceedings of the IEEE*, vol. 77, no. 2, 1989, pp. 257–286, https://doi.org/10.1109/5.18626.
3. Bhan, Sarthak. “What Is Dynamic Time Warping?” *Medium*, MLearning.ai, 18 July 2022, medium.com/mlearning-ai/what-is-dynamic-time-warping-253a6880ad12.
4. Almutairi, Zaynab, and Hebah Elgibreen. “A review of modern audio deepfake detection methods: Challenges and future directions.” Algorithms, vol. 15, no. 5, 2022, p. 155, https://doi.org/10.3390/a15050155.
5. Frank, Joel, and Lea Schönherr. “WaveFake: A Data Set to Facilitate Audio Deepfake Detection.” *CoRR*, abs/2111.02813, 2021, https://doi.org/https://doi.org/10.48550/arXiv.2111.02813.
6. Wang, Run, et al. “DeepSonar: Towards Effective and Robust Detection of AI-Synthesized Fake Voices.” *Proceedings of the 28th ACM International Conference on Multimedia*, 2020, https://doi.org/https://doi.org/10.48550/arXiv.2005.13770.
7. Dimitrakakis, Christos & Bengio, Samy. (2011). Phoneme and Sentence-Level Ensembles for Speech Recognition. EURASIP J. Audio, Speech and Music Processing. 2011. 10.1155/2011/426792.
8. “Using Deep Learning Models / Convolutional Neural Networks.” *Using Deep Learning Models / Convolutional Neural Networks*, 2023, docs.ecognition.com/eCognition_documentation/User%20Guide%20Developer/8%20Classification%20-%20Deep%20Learning.htm.
9. Mishra, Vidushi & AGARWAL, SMT & PURI, NEHA. (2018). COMPREHENSIVE AND COMPARATIVE ANALYSIS OF NEURAL NETWORK. INTERNATIONAL JOURNAL OF COMPUTER APPLICATION. 2. 10.26808/rs.ca.i8v2.15.
10. Saeed, Mehreen. “An Introduction to Recurrent Neural Networks and the Math That Powers Them.” *MachineLearningMastery.Com*, Guiding Tech Media, 5 Jan. 2023, machinelearningmastery.com/an-introduction-to-recurrent-neural-networks-and-the-math-that-powers-them/.
11. K, Bharath. “Introduction to Deep Neural Networks.” *DataCamp*, DataCamp, 4 July 2023, www.datacamp.com/tutorial/introduction-to-deep-neural-networks.
12. Yu, H.; Tan, Z.-H.; Ma, Z.; Martin, R.; Guo, J. Guo spoofing detection in automatic speaker verification systems using DNN classifiers and dynamic acoustic features. *IEEE Trans. Neural Netw. Learn. Syst.* **2018**, *29*, 4633–4644. [**Google Scholar**] [**CrossRef**][**Green Version**]
13. Alzantot, M.; Wang, Z.; Srivastava, M.B. Deep residual neural networks for audio spoofing detection. *arXiv CoRR* **2019**, arXiv:1907.00501. [**Google Scholar**]
14. Lai, C.-I.; Chen, N.; Villalba, J.; Dehak, N. ASSERT: Anti-spoofing with squeeze-excitation and residual networks. *arXiv* **2019**, arXiv:1904.01120. [**Google Scholar**]
15. Aravind, P.R.; Nechiyil, U.; Paramparambath, N. Audio spoofing verification using deep convolutional neural networks by transfer learning. *arXiv* **2020**, arXiv:2008.03464. [**Google Scholar**]
16. Lataifeh, M.; Elnagar, A.; Shahin, I.; Nassif, A.B. Arabic audio clips: Identification and discrimination of authentic cantillations from imitations. *Neurocomputing* **2020**, *418*, 162–177. [**Google Scholar**] [**CrossRef**]
17. Rodríguez-Ortega, Y.; Ballesteros, D.M.; Renza, D. A machine learning model to detect fake voice. In *Applied Informatics*; Florez, H., Misra, S., Eds.; Springer International Publishing: Cham, Switzerland, 2020; pp. 3–13. [**Google Scholar**]

18. Wang, R.; Juefei-Xu, F.; Huang, Y.; Guo, Q.; Xie, X.; Ma, L.; Liu, Y. Deepsonar: Towards effective and robust detection of ai-synthesized fake voices. In Proceedings of the the 28th ACM International Conference on Multimedia, Seattle, WA, USA, 12–16 October 2020; Association for Computing Machinery: New York, NY, USA, 2020; pp. 1207–1216. [**Google Scholar**]
19. Subramani, N.; Rao, D. Learning efficient representations for fake speech detection. In Proceedings of the The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, 7–12 February 2020; pp. 5859–5866. [**Google Scholar**]
20. Shan, M.; Tsai, T. A cross-verification approach for protecting world leaders from fake and tampered audio. *arXiv* **2020**, arXiv:2010.12173. [**Google Scholar**]
21. Wijethunga, R.L.M.A.P.C.; Matheesha, D.M.K.; Al Noman, A.; De Silva, K.H.V.T.A.; Tissera, M.; Rupasinghe, L. Rupasinghe deepfake audio detection: A deep learning based solution for group conversations. In Proceedings of the 2020 2nd International Conference on Advancements in Computing (ICAC), Malabe, Sri Lanka, 10–11 December 2020; Volume 1, pp. 192–197. [**Google Scholar**]
22. Jiang, Z.; Zhu, H.; Peng, L.; Ding, W.; Ren, Y. Self-supervised spoofing audio detection scheme. In Proceedings of the INTERSPEECH 2020, Shanghai, China, 25–29 October 2020; pp. 4223–4227. [**Google Scholar**]
23. Chintha, A.; Thai, B.; Sohrawardi, S.J.; Bhatt, K.M.; Hickerson, A.; Wright, M.; Ptucha, R. Ptucha recurrent convolutional structures for audio spoof and video deepfake detection. *IEEE J. Sel. Top. Signal. Process.* **2020**, *14*, 1024–1037. [**Google Scholar**] [**CrossRef**]
24. Singh, A.K.; Singh, P. Detection of ai-synthesized speech using cepstral & bispectral statistics. In Proceedings of the 2021 IEEE 4th International Conference on Multimedia Information Processing and Retrieval (MIPR), Tokyo, Japan, 8–10 September 2021; pp. 412–417. [**Google Scholar**]
25. Lei, Z.; Yang, Y.; Liu, C.; Ye, J. Siamese convolutional neural network using gaussian probability feature for spoofing speech detection. In Proceedings of the INTERSPEECH, Shanghai, China, 25–29 October 2020; pp. 1116–1120. [**Google Scholar**]
26. Ballesteros, D.M.; Rodriguez-Ortega, Y.; Renza, D.; Arce, G. Deep4SNet: Deep learning for fake speech classification. *Expert Syst. Appl.* **2021**, *184*, 115465. [**Google Scholar**] [**CrossRef**]
27. Bartusiak, E.R.; Delp, E.J. Frequency domain-based detection of generated audio. In Proceedings of the Electronic Imaging; Society for Imaging Science and Technology, New York, NY, USA, 11–15 January 2021; Volume 2021, pp. 273–281. [**Google Scholar**]
28. Borrelli, C.; Bestagini, P.; Antonacci, F.; Sarti, A.; Tubaro, S. Synthetic speech detection through short-term and long-term prediction traces. *EURASIP J. Inf. Secur.* **2021**, *2021*, 2. [**Google Scholar**] [**CrossRef**]
29. Khalid, H.; Kim, M.; Tariq, S.; Woo, S.S. Evaluation of an audio-video multimodal deepfake dataset using unimodal and multimodal detectors. In Proceedings of the 1st Workshop on Synthetic Multimedia, ACM Association for Computing Machinery, New York, NY, USA, 20 October 2021; pp. 7–15. [**Google Scholar**]
30. Khochare, J.; Joshi, C.; Yenarkar, B.; Suratkar, S.; Kazi, F. A deep learning framework for audio deepfake detection. *Arab. J. Sci. Eng.* **2021**, *47*, 3447–3458. [**Google Scholar**] [**CrossRef**]
31. Liu, T.; Yan, D.; Wang, R.; Yan, N.; Chen, G. Identification of fake stereo audio using SVM and CNN. *Information* **2021**, *12*, 263. [**Google Scholar**] [**CrossRef**]
32. Camacho, S.; Ballesteros, D.M.; Renza, D. Fake speech recognition using deep learning. In *Applied Computer Sciences in Engineering*; Figueroa-García, J.C., Díaz-Gutierrez, Y., Gaona-García, E.E.,

Orjuela-Cañón, A.D., Eds.; Springer International Publishing: Cham, Switzerland, 2021; pp. 38–48. [**Google Scholar**]

33. Arif, T.; Javed, A.; Alhameed, M.; Jeribi, F.; Tahir, A. Voice spoofing countermeasure for logical access attacks detection. *IEEE Access* **2021**, *9*, 162857–162868. [**Google Scholar**] [**CrossRef**]
34. Dennis, Jonathan & Dat, Tran & Li, Haizhou. (2011). Spectrogram Image Feature for Sound Event Classification in Mismatched Conditions. Signal Processing Letters, IEEE. 18. 130 - 133. 10.1109/LSP.2010.2100380.