Artifact for OOPSLA Paper 230
Shiftry: RNN Inference in 2 KB of RAM

GETTING STARTED GUIDE

Hardware Requirements

1. Arduino Uno Board + B USB Cable (https://www.arduino.cc/en/Guide/ArduinoUno) (If not available it will be difficult to reproduce runtime numbers. We provide instructions for the case an Uno board is not available, so that all numbers apart from runtime can be reproduced. If a different Arduino board is available, it can be tried out however the sketch size and runtime numbers will be different. We strongly recommend an Uno board)
2. Windows 10 Machine, preferably with 16GB+ RAM (minimum 8GB), 8+ Logical Processors (more is better).

We have tested the artifact on a Lenovo Thinkpad T495S running Windows 10 with AMD Ryzen 7 Pro @2.30Ghz.
We also provide instructions for building on Ubuntu (should work for any Linux distribution, we tested on Windows Subsystem for Linux) if a Windows machine is not available.
Please note that we have extensively tested the artifact for Windows (individually checked all commands).
In the documentation, we follow windows conventions for filenames. The commands given have been executed on Windows Powershell.

Installing Dependencies: Windows

Please follow the steps in order. If one software is already installed, you can skip to the next step.
Recommended versions are those used by us to run the code. However, any subsequent version should be okay to use.
1. Install Python 3 (Recommended 3.7.8: https://www.python.org/downloads/release/python-378/)
    -While installing python, there would be an option to "Disable path length limit". Please ensure to check this option. As an alternative, follow this link (https://www.howtogeek.com/266621/how-to-make-windows-10-accept-file-paths-over-260-characters/) to enable long paths in Windows.
2. Run the following on Windows powershell:
    a. pip install antlr4-python3-runtime==4.7
    b. pip install numpy==1.16.4
3. Install Microsoft Visual Studio Community/Professional/Enterprise Edition
    -While installing Visual Studio, install Python, C++ and .NET development environments
4. Install Java Runtime Environment (Recommended Version 8 Update 261: https://java.com/en/download/)
5. Install Arduino IDE (Recommended 1.8.10: https://www.arduino.cc/download_handler.php?f=/arduino-1.8.10-windows.exe)
6. Connect the Arduino Uno board to the Windows 10 machine (skip if an Uno board is not available)

Installing Dependencies: Ubuntu

Please install the following. If one software is already installed, you can skip to the next step.
1. Python 3 (Should be pre-installed on most distributions)
2. Install pip for Python 3 (sudo apt install -y python3-pip)
3. Install the following through pip: (you may need to use pip3 instead of pip)
    a. pip install antlr4-python3-runtime==4.7

b. pip install numpy==1.16.4

4. Install g++ (should be already installed on many distributions) (sudo apt-get install g++)
5. Ensure make is installed (The linux version of the compiler utilises makefile to build and execute output code)
6. Install Java (sudo apt install openjdk-8-jre-headless)
7. Install Arduino IDE (https://www.arduino.cc/download_handler.php?f=/arduino-1.8.10-linux64.tar.xz)
8. Connect the Arduino Uno board to the Ubuntu machine (skip if an Uno board is not available)

Placing the datasets and trained models

1. The datasets and trained models are included with the artifact ('datasets' and 'model' directories)
2. Check that both the 'datasets' and 'model' directory should be within the 'SeeDot' directory. (As a check, 'ls SeeDot' command should give the following files: .vscode, datasets, model, seedot, Results.csv, SeeDot-dev.py, SeeDot.py) (On Ubuntu, ls -a SeeDot should give the same result)
3. The 'datasets' directory contains the input/output traces for multiple datasets, and the 'model' directory contains the model parameters and the input code.

Both 'datasets' and 'model' have subdirectories in the following way: <algorithm>\<dataset_name>\<...relevant files...>

In 'datasets\<algorithm>\<dataset_name>\', there would be two files: 'train.npy', 'test.npy'

In 'model\<algorithm>\<dataset_name>\', there would be one file called 'input.sd', and multiple other .npy files which represent model parameters. If there are multiple files with extension .sd, they can be safely ignored

Running an Example

1. Navigate to .\SeeDot\
2. Run the following command: 'python SeeDot-dev.py -a rnn -d HAR-6 -m red_disagree -v fixed -t arduino'
3. It is possible you may encounter outputs which look like the following: (observed on Windows)

```
>> Traceback (most recent call last):
>>   File "SeeDot-dev.py", line 392, in <module>
>>     obj.parseArgs()
>>   File "SeeDot-dev.py", line 101, in parseArgs
>>     shutil.rmtree(config.tempdir)
>>   File "C:\Users\t-aak\AppData\Local\Programs\Python\Python37\lib\shutil.py", line 516, in rmtree
>>     return _rmtree_unsafe(path, onerror)
>>   File "C:\Users\t-aak\AppData\Local\Programs\Python\Python37\lib\shutil.py", line 404, in _rmtree_unsafe
>>     onerror(os.rmdir, path, sys.exc_info())
>>   File "C:\Users\t-aak\AppData\Local\Programs\Python\Python37\lib\shutil.py", line 402, in _rmtree_unsafe
>>     os.rmdir(path)
>> OSError: [WinError 145] The directory is not empty: 'temp'
```

These errors are caused due to the OS not being able to clear the generated files used in the previous run of the compiler. In case such a message appears, simply run the compiler again (sometimes might be required multiple times till the error clears)

4. The output should be as similar to the follows (the numbers may vary but the structure should be the same):

-Approximate runtime on our machine: 200 seconds

-Each output line is preceded by '>> ', lines not preceded by '>> ' is an explanatory note

-The output is approximately 560 lines long
-If the output matches the following then the compiler is working as expected
-The arduino code would be dumped in SeeDot\arduinodump\arduino\16\rnn\HAR-6
-Refer to the STEP_BY_STEP_GUIDE to use the arduino code and reproduce the results of the paper

```
>> ======================================
>> Executing on rnn fixed HAR-6 arduino
>> ======================================
>>
>> ----------------------
>> Collecting profile data
>> ----------------------
>> Generating input files for float training dataset...done
>>
>> Generating code...completed
>> Build...success
>> Execution...success
>> Accuracy is 99.802%
>>
```

The above is to run a floating point code to compute data driven scales for variables in the code (Refer to Section 5.1 in the paper)

```
>> --------------------------------------------------
>> Performing search to find the best scaling factor
>> --------------------------------------------------
>>
>> Generating input files for fixed training dataset...done
>>
>> Generating code...completed
>> Generating code...completed
>> Generating code...completed
>> Testing with DDS and scale of X as 0
>> Generating code...completed
>> Testing with DDS and scale of X as -1
>> Generating code...completed
>> Testing with DDS and scale of X as -2
>> Generating code...completed
>> Testing with DDS and scale of X as -3
>> Generating code...completed
>> Testing with DDS and scale of X as -4
>> Generating code...completed
```

```
>> Testing with DDS and scale of X as -5
>> Generating code...completed
>> Testing with DDS and scale of X as -6
>> Generating code...completed
>> Testing with DDS and scale of X as -7
>> Generating code...completed
>> Testing with DDS and scale of X as -8
>> Generating code...completed
>> Testing with DDS and scale of X as -9
>> Generating code...completed
>> Testing with DDS and scale of X as -10
>> Generating code...completed
>> Testing with DDS and scale of X as -11
>> Generating code...completed
>> Testing with DDS and scale of X as -12
>> Generating code...completed
>> Testing with DDS and scale of X as -13
>> Generating code...completed
>> Testing with DDS and scale of X as -14
>> Generating code...completed
>> Testing with DDS and scale of X as -15
>> Generating code...completed
>> Build...success
>> Execution...success
>> Accuracy at scale factor 0 is 29.366%, Disagreement Count is 5356, Reduced Disagreement Count is 5344
>>
>> Accuracy at scale factor -1 is 71.412%, Disagreement Count is 2167, Reduced Disagreement Count is 2158
>>
>> Accuracy at scale factor -2 is 91.406%, Disagreement Count is 649, Reduced Disagreement Count is 642
>>
>> Accuracy at scale factor -3 is 98.207%, Disagreement Count is 130, Reduced Disagreement Count is 125
>>
>> Accuracy at scale factor -4 is 99.578%, Disagreement Count is 24, Reduced Disagreement Count is 20
>>
>> Accuracy at scale factor -5 is 99.736%, Disagreement Count is 9, Reduced Disagreement Count is 7
>>
>> Accuracy at scale factor -6 is 99.802%, Disagreement Count is 2, Reduced Disagreement Count is 1
>>
>> Accuracy at scale factor -7 is 99.789%, Disagreement Count is 3, Reduced Disagreement Count is 2
>>
>> Accuracy at scale factor -8 is 99.763%, Disagreement Count is 3, Reduced Disagreement Count is 3
```

```
>>
>> Accuracy at scale factor -9 is 99.776%, Disagreement Count is 2, Reduced Disagreement Count is 2
>>
>> Accuracy at scale factor -10 is 99.776%, Disagreement Count is 2, Reduced Disagreement Count is 2
>>
>> Accuracy at scale factor -11 is 99.736%, Disagreement Count is 5, Reduced Disagreement Count is 5
>>
>> Accuracy at scale factor -12 is 99.341%, Disagreement Count is 35, Reduced Disagreement Count is 35
>>
>> Accuracy at scale factor -13 is 95.703%, Disagreement Count is 311, Reduced Disagreement Count is 311
>>
>> Accuracy at scale factor -14 is 55.055%, Disagreement Count is 3403, Reduced Disagreement Count is 3398
>>
>> Accuracy at scale factor -15 is 5.088%, Disagreement Count is 7201, Reduced Disagreement Count is 7186
>>
>>
>> Search completed
>>
>> ------------------------------------------------
>> Best performing scaling factors with accuracy, disagreement, reduced disagreement:
>> [(-11, (99.736, 5, 5)), (-10, (99.776, 2, 2)), (-9, (99.776, 2, 2)), (-8, (99.763, 3, 3)), (-7, (99.789, 3, 2))]
```

As mentioned in the paper, for the input variable 'X', data driven scaling often does not give the best scale (Section 5.1). So, in the above segment we explore across multiple scales to see which one fits best.

```
>> Scales computed in native bitwidth. Starting exploration over other bitwidths.
>> Generating code...completed
>> Generating code...completed
>> Generating code...completed
>> Generating code...completed
```

Multiple lines hidden for brevity

```
>> Generating code...completed
>> Generating code...completed
>> Generating code...completed
>> Build...success
>> Execution...success
>> Demoted vars: ('tmp20',)
>>
>> Offset 0 (Code ID 1): Accuracy 93.357%, Disagreement Count 500, Reduced Disagreement Count 494
```

>>
>> Offset -1 (Code ID 2): Accuracy 98.603%, Disagreement Count 97, Reduced Disagreement Count 94
>>
>> Offset -2 (Code ID 3): Accuracy 99.249%, Disagreement Count 48, Reduced Disagreement Count 45
>>
>> Demoted vars: ('tmp22',)
>>
>> Offset 0 (Code ID 4): Accuracy 83.999%, Disagreement Count 1220, Reduced Disagreement Count 1209
>>
>> Offset -1 (Code ID 5): Accuracy 95.624%, Disagreement Count 331, Reduced Disagreement Count 324
>>
>> Offset -2 (Code ID 6): Accuracy 98.985%, Disagreement Count 74, Reduced Disagreement Count 68
>>
>> Demoted vars: ('tmp24',)
>>
>> Offset 0 (Code ID 7): Accuracy 99.697%, Disagreement Count 8, Reduced Disagreement Count 8
>>
>> Offset -1 (Code ID 8): Accuracy 75.405%, Disagreement Count 1854, Reduced Disagreement Count 1852
>>
>> Offset -2 (Code ID 9): Accuracy 25.939%, Disagreement Count 5619, Reduced Disagreement Count 5607
>>
>> Demoted vars: ('tmp26',)
>>
>> Offset 0 (Code ID 10): Accuracy 99.051%, Disagreement Count 69, Reduced Disagreement Count 63
>>
>> Offset -1 (Code ID 11): Accuracy 99.183%, Disagreement Count 49, Reduced Disagreement Count 48
>>
>> Offset -2 (Code ID 12): Accuracy 80.256%, Disagreement Count 1498, Reduced Disagreement Count 1490
>>
>> Demoted vars: ('tmp27',)
>>
>> Offset 0 (Code ID 13): Accuracy 85.185%, Disagreement Count 1130, Reduced Disagreement Count 1119
>>
>> Offset -1 (Code ID 14): Accuracy 96.639%, Disagreement Count 254, Reduced Disagreement Count 247
>>
>> Offset -2 (Code ID 15): Accuracy 99.156%, Disagreement Count 59, Reduced Disagreement Count 54
>>
>> Demoted vars: ('tmp28',)
>>
>> Offset 0 (Code ID 16): Accuracy 94.016%, Disagreement Count 451, Reduced Disagreement Count 445
>>

>> Offset -1 (Code ID 17): Accuracy 98.682%, Disagreement Count 93, Reduced Disagreement Count 89
>>
>> Offset -2 (Code ID 18): Accuracy 97.891%, Disagreement Count 151, Reduced Disagreement Count 148
>>
>> Demoted vars: ('tmp30',)
>>
>> Offset 0 (Code ID 19): Accuracy 95.624%, Disagreement Count 331, Reduced Disagreement Count 324
>>
>> Offset -1 (Code ID 20): Accuracy 99.104%, Disagreement Count 59, Reduced Disagreement Count 56
>>
>> Offset -2 (Code ID 21): Accuracy 99.670%, Disagreement Count 18, Reduced Disagreement Count 14
>>
>> Demoted vars: ('tmp32',)
>>
>> Offset 0 (Code ID 22): Accuracy 99.473%, Disagreement Count 33, Reduced Disagreement Count 29
>>
>> Offset -1 (Code ID 23): Accuracy 12.350%, Disagreement Count 6647, Reduced Disagreement Count 6636
>>
>> Offset -2 (Code ID 24): Accuracy 10.175%, Disagreement Count 6815, Reduced Disagreement Count 6800
>>
>> Demoted vars: ('tmp34',)
>>
>> Offset 0 (Code ID 25): Accuracy 99.499%, Disagreement Count 29, Reduced Disagreement Count 26
>>
>> Offset -1 (Code ID 26): Accuracy 35.469%, Disagreement Count 4890, Reduced Disagreement Count 4883
>>
>> Offset -2 (Code ID 27): Accuracy 7.869%, Disagreement Count 6991, Reduced Disagreement Count 6976
>>
>> Demoted vars: ('tmp35',)
>>
>> Offset 0 (Code ID 28): Accuracy 99.631%, Disagreement Count 21, Reduced Disagreement Count 17
>>
>> Offset -1 (Code ID 29): Accuracy 1.872%, Disagreement Count 7445, Reduced Disagreement Count 7430
>>
>> Offset -2 (Code ID 30): Accuracy 17.174%, Disagreement Count 6285, Reduced Disagreement Count 6271
>>
>> Demoted vars: ('tmp36',)
>>
>> Offset 0 (Code ID 31): Accuracy 99.631%, Disagreement Count 21, Reduced Disagreement Count 17
>>
>> Offset -1 (Code ID 32): Accuracy 1.568%, Disagreement Count 7468, Reduced Disagreement Count 7453

>>
>> Offset -2 (Code ID 33): Accuracy 15.711%, Disagreement Count 6395, Reduced Disagreement Count 6383
>>
>> Demoted vars: ('tmp37',)
>>
>> Offset 0 (Code ID 34): Accuracy 99.473%, Disagreement Count 33, Reduced Disagreement Count 29
>>
>> Offset -1 (Code ID 35): Accuracy 1.529%, Disagreement Count 7471, Reduced Disagreement Count 7456
>>
>> Offset -2 (Code ID 36): Accuracy 13.576%, Disagreement Count 6558, Reduced Disagreement Count 6544
>>
>> Demoted vars: ('tmp12',)
>>
>> Offset 0 (Code ID 37): Accuracy 99.460%, Disagreement Count 34, Reduced Disagreement Count 30
>>
>> Offset -1 (Code ID 38): Accuracy 1.410%, Disagreement Count 7480, Reduced Disagreement Count 7465
>>
>> Offset -2 (Code ID 39): Accuracy 3.018%, Disagreement Count 7359, Reduced Disagreement Count 7344
>>
>> Demoted vars: ('tmp40',)
>>
>> Offset 0 (Code ID 40): Accuracy 99.723%, Disagreement Count 6, Reduced Disagreement Count 6
>>
>> Offset -1 (Code ID 41): Accuracy 95.624%, Disagreement Count 318, Reduced Disagreement Count 317
>>
>> Offset -2 (Code ID 42): Accuracy 14.209%, Disagreement Count 6509, Reduced Disagreement Count 6496
>>
>> Demoted vars: ('tmp42',)
>>
>> Offset 0 (Code ID 43): Accuracy 99.750%, Disagreement Count 8, Reduced Disagreement Count 6
>>
>> Offset -1 (Code ID 44): Accuracy 43.482%, Disagreement Count 4273, Reduced Disagreement Count 4273
>>
>> Offset -2 (Code ID 45): Accuracy 0.119%, Disagreement Count 7578, Reduced Disagreement Count 7563
>>
>> Demoted vars: ('tmp43',)
>>
>> Offset 0 (Code ID 46): Accuracy 99.750%, Disagreement Count 8, Reduced Disagreement Count 6
>>
>> Offset -1 (Code ID 47): Accuracy 40.438%, Disagreement Count 4508, Reduced Disagreement Count 4504
>>

>> Offset -2 (Code ID 48): Accuracy 0.171%, Disagreement Count 7574, Reduced Disagreement Count 7559
>>
>> Demoted vars: ('W1',)
>>
>> Offset 0 (Code ID 49): Accuracy 99.710%, Disagreement Count 15, Reduced Disagreement Count 11
>>
>> Offset -1 (Code ID 50): Accuracy 6.564%, Disagreement Count 7089, Reduced Disagreement Count 7074
>>
>> Offset -2 (Code ID 51): Accuracy 1.450%, Disagreement Count 7477, Reduced Disagreement Count 7462
>>
>> Demoted vars: ('W2',)
>>
>> Offset 0 (Code ID 52): Accuracy 99.710%, Disagreement Count 11, Reduced Disagreement Count 9
>>
>> Offset -1 (Code ID 53): Accuracy 17.609%, Disagreement Count 6249, Reduced Disagreement Count 6237
>>
>> Offset -2 (Code ID 54): Accuracy 2.148%, Disagreement Count 7423, Reduced Disagreement Count 7409
>>
>> Demoted vars: ('U1',)
>>
>> Offset 0 (Code ID 55): Accuracy 99.723%, Disagreement Count 6, Reduced Disagreement Count 6
>>
>> Offset -1 (Code ID 56): Accuracy 37.156%, Disagreement Count 4775, Reduced Disagreement Count 4762
>>
>> Offset -2 (Code ID 57): Accuracy 26.638%, Disagreement Count 5567, Reduced Disagreement Count 5553
>>
>> Demoted vars: ('U2',)
>>
>> Offset 0 (Code ID 58): Accuracy 99.736%, Disagreement Count 7, Reduced Disagreement Count 6
>>
>> Offset -1 (Code ID 59): Accuracy 99.341%, Disagreement Count 37, Reduced Disagreement Count 36
>>
>> Offset -2 (Code ID 60): Accuracy 48.319%, Disagreement Count 3920, Reduced Disagreement Count 3911
>>
>> Demoted vars: ('Bg',)
>>
>> Offset 0 (Code ID 61): Accuracy 99.710%, Disagreement Count 9, Reduced Disagreement Count 8
>>
>> Offset -1 (Code ID 62): Accuracy 27.732%, Disagreement Count 5481, Reduced Disagreement Count 5469
>>
>> Offset -2 (Code ID 63): Accuracy 35.495%, Disagreement Count 4894, Reduced Disagreement Count 4881

>>
>> Demoted vars: ('Bh',)
>>
>> Offset 0 (Code ID 64): Accuracy 99.723%, Disagreement Count 6, Reduced Disagreement Count 6
>>
>> Offset -1 (Code ID 65): Accuracy 70.397%, Disagreement Count 2240, Reduced Disagreement Count 2233
>>
>> Offset -2 (Code ID 66): Accuracy 19.692%, Disagreement Count 6093, Reduced Disagreement Count 6078
>>
>> Demoted vars: ('FC1',)
>>
>> Offset 0 (Code ID 67): Accuracy 99.763%, Disagreement Count 5, Reduced Disagreement Count 4
>>
>> Offset -1 (Code ID 68): Accuracy 38.856%, Disagreement Count 4639, Reduced Disagreement Count 4626
>>
>> Offset -2 (Code ID 69): Accuracy 2.649%, Disagreement Count 7386, Reduced Disagreement Count 7371
>>
>> Demoted vars: ('FC2',)
>>
>> Offset 0 (Code ID 70): Accuracy 99.723%, Disagreement Count 10, Reduced Disagreement Count 8
>>
>> Offset -1 (Code ID 71): Accuracy 70.766%, Disagreement Count 2207, Reduced Disagreement Count 2204
>>
>> Offset -2 (Code ID 72): Accuracy 3.980%, Disagreement Count 7284, Reduced Disagreement Count 7271
>>
>> Demoted vars: ('FCBias',)
>>
>> Offset 0 (Code ID 73): Accuracy 99.750%, Disagreement Count 4, Reduced Disagreement Count 4
>>
>> Offset -1 (Code ID 74): Accuracy 99.025%, Disagreement Count 60, Reduced Disagreement Count 59
>>
>> Offset -2 (Code ID 75): Accuracy 96.138%, Disagreement Count 281, Reduced Disagreement Count 279
>>
>> Demoted vars: ('X',)
>>
>> Offset 0 (Code ID 76): Accuracy 98.168%, Disagreement Count 133, Reduced Disagreement Count 128
>>
>> Offset -1 (Code ID 77): Accuracy 99.104%, Disagreement Count 60, Reduced Disagreement Count 56
>>
>> Offset -2 (Code ID 78): Accuracy 95.743%, Disagreement Count 312, Reduced Disagreement Count 310
>>

>> Offset -3 (Code ID 79): Accuracy 55.226%, Disagreement Count 3392, Reduced Disagreement Count 3386
>>
>> Offset -4 (Code ID 80): Accuracy 5.048%, Disagreement Count 7204, Reduced Disagreement Count 7189
>>
>> Offset -5 (Code ID 81): Accuracy 1.898%, Disagreement Count 7443, Reduced Disagreement Count 7428
>>
>> Offset -6 (Code ID 82): Accuracy 1.582%, Disagreement Count 7467, Reduced Disagreement Count 7452
>>
>> Offset -7 (Code ID 83): Accuracy 1.239%, Disagreement Count 7493, Reduced Disagreement Count 7478
>>
>> Offset -8 (Code ID 84): Accuracy 1.028%, Disagreement Count 7509, Reduced Disagreement Count 7494
>>

In the above section, we demote one variable from 16 to 8 bits at a time, modifying the scale appropriately. The results obtained from each demotion are shown above. (Refer to Section 5.2, Stage III)

>> Generating code...completed
>> Generating code...completed
>> Generating code...completed

Multiple lines hidden for brevity

>> Generating code...completed
>> Generating code...completed
>> Generating code...completed
>> Build...success
>> Execution...success
>> Demoted vars: ('FCBias',)
>>
>> Offset 0 (Code ID 1): Accuracy 99.750%, Disagreement Count 4, Reduced Disagreement Count 4
>>
>> Demoted vars: ('FCBias', 'FC1')
>>
>> Offset 0 (Code ID 2): Accuracy 99.750%, Disagreement Count 6, Reduced Disagreement Count 5
>>
>> Demoted vars: ('FCBias', 'FC1', 'tmp40')
>>
>> Offset 0 (Code ID 3): Accuracy 99.750%, Disagreement Count 8, Reduced Disagreement Count 6
>>
>> Demoted vars: ('FCBias', 'FC1', 'tmp40', 'U1')
>>

>> Offset 0 (Code ID 4): Accuracy 99.723%, Disagreement Count 6, Reduced Disagreement Count 6
>>
>> Demoted vars: ('FCBias', 'FC1', 'tmp40', 'U1', 'Bh')
>>
>> Offset 0 (Code ID 5): Accuracy 99.723%, Disagreement Count 6, Reduced Disagreement Count 6
>>
>> Demoted vars: ('FCBias', 'FC1', 'tmp40', 'U1', 'Bh', 'U2')
>>
>> Offset 0 (Code ID 6): Accuracy 99.710%, Disagreement Count 9, Reduced Disagreement Count 8
>>
>> Demoted vars: ('FCBias', 'FC1', 'tmp40', 'U1', 'Bh', 'U2', 'tmp42')
>>
>> Offset 0 (Code ID 7): Accuracy 99.697%, Disagreement Count 12, Reduced Disagreement Count 10
>>
>> Demoted vars: ('FCBias', 'FC1', 'tmp40', 'U1', 'Bh', 'U2', 'tmp42', 'tmp43')
>>
>> Offset 0 (Code ID 8): Accuracy 99.697%, Disagreement Count 12, Reduced Disagreement Count 10
>>
>> Demoted vars: ('FCBias', 'FC1', 'tmp40', 'U1', 'Bh', 'U2', 'tmp42', 'tmp43', 'tmp24')
>>
>> Offset 0 (Code ID 9): Accuracy 99.644%, Disagreement Count 16, Reduced Disagreement Count 14
>>
>> Demoted vars: ('FCBias', 'FC1', 'tmp40', 'U1', 'Bh', 'U2', 'tmp42', 'tmp43', 'tmp24', 'Bg')
>>
>> Offset 0 (Code ID 10): Accuracy 99.631%, Disagreement Count 19, Reduced Disagreement Count 16
>>
>> Demoted vars: ('FCBias', 'FC1', 'tmp40', 'U1', 'Bh', 'U2', 'tmp42', 'tmp43', 'tmp24', 'Bg', 'FC2')
>>
>> Offset 0 (Code ID 11): Accuracy 99.605%, Disagreement Count 21, Reduced Disagreement Count 18
>>
>> Demoted vars: ('FCBias', 'FC1', 'tmp40', 'U1', 'Bh', 'U2', 'tmp42', 'tmp43', 'tmp24', 'Bg', 'FC2', 'W2')
>>
>> Offset 0 (Code ID 12): Accuracy 99.565%, Disagreement Count 24, Reduced Disagreement Count 21
>>
>> Demoted vars: ('FCBias', 'FC1', 'tmp40', 'U1', 'Bh', 'U2', 'tmp42', 'tmp43', 'tmp24', 'Bg', 'FC2', 'W2', 'W1')
>>
>> Offset 0 (Code ID 13): Accuracy 99.381%, Disagreement Count 42, Reduced Disagreement Count 37
>>
>> Demoted vars: ('FCBias', 'FC1', 'tmp40', 'U1', 'Bh', 'U2', 'tmp42', 'tmp43', 'tmp24', 'Bg', 'FC2', 'W2', 'W1', 'tmp30')
>>
>> Offset -2 (Code ID 14): Accuracy 98.972%, Disagreement Count 71, Reduced Disagreement Count 67

>>
>> Demoted vars: ('FCBias', 'FC1', 'tmp40', 'U1', 'Bh', 'U2', 'tmp42', 'tmp43', 'tmp24', 'Bg', 'FC2', 'W2', 'W1', 'tmp30', 'tmp35')
>>
>> Offset 0 (Code ID 15): Accuracy 98.814%, Disagreement Count 83, Reduced Disagreement Count 79
>>
>> Demoted vars: ('FCBias', 'FC1', 'tmp40', 'U1', 'Bh', 'U2', 'tmp42', 'tmp43', 'tmp24', 'Bg', 'FC2', 'W2', 'W1', 'tmp30', 'tmp35', 'tmp36')
>>
>> Offset 0 (Code ID 16): Accuracy 98.814%, Disagreement Count 83, Reduced Disagreement Count 79
>>
>> Demoted vars: ('FCBias', 'FC1', 'tmp40', 'U1', 'Bh', 'U2', 'tmp42', 'tmp43', 'tmp24', 'Bg', 'FC2', 'W2', 'W1', 'tmp30', 'tmp35', 'tmp36', 'tmp34')
>>
>> Offset 0 (Code ID 17): Accuracy 98.985%, Disagreement Count 72, Reduced Disagreement Count 67
>>
>> Demoted vars: ('FCBias', 'FC1', 'tmp40', 'U1', 'Bh', 'U2', 'tmp42', 'tmp43', 'tmp24', 'Bg', 'FC2', 'W2', 'W1', 'tmp30', 'tmp35', 'tmp36', 'tmp34', 'tmp32')
>>
>> Offset 0 (Code ID 18): Accuracy 98.511%, Disagreement Count 106, Reduced Disagreement Count 102
>>
>> Demoted vars: ('FCBias', 'FC1', 'tmp40', 'U1', 'Bh', 'U2', 'tmp42', 'tmp43', 'tmp24', 'Bg', 'FC2', 'W2', 'W1', 'tmp30', 'tmp35', 'tmp36', 'tmp34', 'tmp32', 'tmp37')
>>
>> Offset 0 (Code ID 19): Accuracy 98.511%, Disagreement Count 106, Reduced Disagreement Count 102
>>
>> Demoted vars: ('FCBias', 'FC1', 'tmp40', 'U1', 'Bh', 'U2', 'tmp42', 'tmp43', 'tmp24', 'Bg', 'FC2', 'W2', 'W1', 'tmp30', 'tmp35', 'tmp36', 'tmp34', 'tmp32', 'tmp37', 'tmp12')
>>
>> Offset 0 (Code ID 20): Accuracy 98.445%, Disagreement Count 109, Reduced Disagreement Count 106
>>
>> Demoted vars: ('FCBias', 'FC1', 'tmp40', 'U1', 'Bh', 'U2', 'tmp42', 'tmp43', 'tmp24', 'Bg', 'FC2', 'W2', 'W1', 'tmp30', 'tmp35', 'tmp36', 'tmp34', 'tmp32', 'tmp37', 'tmp12', 'tmp20')
>>
>> Offset -2 (Code ID 21): Accuracy 97.720%, Disagreement Count 165, Reduced Disagreement Count 161
>>
>> Demoted vars: ('FCBias', 'FC1', 'tmp40', 'U1', 'Bh', 'U2', 'tmp42', 'tmp43', 'tmp24', 'Bg', 'FC2', 'W2', 'W1', 'tmp30', 'tmp35', 'tmp36', 'tmp34', 'tmp32', 'tmp37', 'tmp12', 'tmp20', 'tmp26')
>>
>> Offset -1 (Code ID 22): Accuracy 97.417%, Disagreement Count 190, Reduced Disagreement Count 185
>>
>> Demoted vars: ('FCBias', 'FC1', 'tmp40', 'U1', 'Bh', 'U2', 'tmp42', 'tmp43', 'tmp24', 'Bg', 'FC2', 'W2', 'W1', 'tmp30', 'tmp35', 'tmp36', 'tmp34', 'tmp32', 'tmp37', 'tmp12', 'tmp20', 'tmp26', 'tmp27')
>>
>> Offset -2 (Code ID 23): Accuracy 96.402%, Disagreement Count 270, Reduced Disagreement Count 264
>>

```
>>      Demoted vars: ('FCBias', 'FC1', 'tmp40', 'U1', 'Bh', 'U2', 'tmp42', 'tmp43', 'tmp24', 'Bg', 'FC2', 'W2', 'W1', 'tmp30', 'tmp35', 'tmp36', 'tmp34', 'tmp32', 'tmp37',
'tmp12', 'tmp20', 'tmp26', 'tmp27', 'X')
>>
>>      Offset -1 (Code ID 24): Accuracy 94.688%, Disagreement Count 398, Reduced Disagreement Count 393
>>
>>      Demoted vars: ('FCBias', 'FC1', 'tmp40', 'U1', 'Bh', 'U2', 'tmp42', 'tmp43', 'tmp24', 'Bg', 'FC2', 'W2', 'W1', 'tmp30', 'tmp35', 'tmp36', 'tmp34', 'tmp32', 'tmp37',
'tmp12', 'tmp20', 'tmp26', 'tmp27', 'X', 'tmp22')
>>
>>      Offset -2 (Code ID 25): Accuracy 94.688%, Disagreement Count 398, Reduced Disagreement Count 393
>>
>>      Demoted vars: ('FCBias', 'FC1', 'tmp40', 'U1', 'Bh', 'U2', 'tmp42', 'tmp43', 'tmp24', 'Bg', 'FC2', 'W2', 'W1', 'tmp30', 'tmp35', 'tmp36', 'tmp34', 'tmp32', 'tmp37',
'tmp12', 'tmp20', 'tmp26', 'tmp27', 'X', 'tmp22', 'tmp28')
>>
>>      Offset -1 (Code ID 26): Accuracy 93.173%, Disagreement Count 516, Reduced Disagreement Count 509
>>
>>      Best scaling factor = -11
>>
```

In the above section, using the order and scale obtained from the previous section, we cumulatively keep demoting variables to 8 bit until the accuracy threshold is breached (Section 5.2 Stage IV). The variables thus demoted are mentioned below (Demoted Vars with Offsets). In the following section, the final code is generated, run on the testing dataset to show the final accuracy numbers, and the arduino code is generated and dumped.

```
>>      -------------------------------
>>      Prediction on testing dataset
>>      -------------------------------
>>
>>      Setting max scaling factor to -11
>>
>>      Demoted Vars with Offsets: {'FCBias': 0, 'FC1': 0, 'tmp40': 0, 'U1': 0, 'Bh': 0, 'U2': 0, 'tmp42': 0, 'tmp43': 0, 'tmp24': 0, 'Bg': 0, 'FC2': 0, 'W2': 0, 'W1': 0, 'tmp30': -2,
'tmp35': 0, 'tmp36': 0, 'tmp34': 0, 'tmp32': 0, 'tmp37': 0, 'tmp12': 0}
>>
>>      Generating input files for fixed testing dataset...done
>>
>>      Generating code...completed
>>      Build...success
>>      Execution...success
>>      Accuracy at scale factor -11 is 88.893%, Disagreement Count is 113, Reduced Disagreement Count is 57
>>
>>      -------------------------------
>>      Generating code for arduino...
>>      -------------------------------
```

>>
>> Generating input files for fixed testing dataset...done
>>
>> Generating code...completed
>>
>> Arduino sketch dumped in the folder arduinodump\arduino

5. The Arduino sketch can be found in SeeDot\arduinodump\arduino\16\rnn\HAR-6\. Copy all the files to SeeDot\arduinodump\arduino\.
6. Double click on Seedot\arduinodump\arduino\arduino.ino. Arduino IDE will open up with all files loaded.
7. IF YOU HAVE THE ARDUINO BOARD:
   -Go to Tools\Board<board name>. If <board name> is not Arduino/Genuino Uno, click on the item and change the board to Arduino/Genuino Uno
   -Click to Tools\Port<port name and board name>. Select the port on which the Uno board is connected.
   -Click on the arrow (upload) on the top left of Arduino IDE. This will compile the Arduino code and upload it onto the Uno board. It will also print out the sketch size in the message box to the bottom of Arduino IDE. Open Tools\Serial Monitor and observe the output. It should look as follows (numbers may be different. For some datasets, the prediction would be incorrect and the 'Correct prediction' message would be replaced by an incorrect prediction warning. The output may take a minute or two to show up. The time is shown in microseconds. This output will appear one after the other in a loop. For models with short inference time, consider switching off autoscroll):

   >> 1: Predicted label: 1; True label: 1; Correct prediction
   >>
   >> ----------
   >> Average prediction time:
   >> 51234567.89
   >> ----------

   IF YOU DO NOT HAVE THE ARDUINO BOARD:
   - Go to Tools\Board<board name>. If <board name> is not Arduino/Genuino Uno, click on the item and change the board to Arduino/Genuino Uno
   -Click on the tick mark (verify) on the top left of Arduino IDE. This will compile the Arduino code and print the sketch size in the message box towards the bottom of Arduino IDE. Message would be similar to the following (numbers may differ):

   >> Sketch uses 11722 bytes (36%) of program storage space. Maximum is 32256 bytes.
   >> Global variables use 362 bytes (17%) of dynamic memory, leaving 1686 bytes for local variables. Maximum is 2048 bytes.

STEP BY STEP INSTRUCTIONS GUIDE

Results this artifact supports

All results of Shiftry mentioned in Section 7 can be reproduced through this artifact. They include:

1. Algorithms: ProtoNN, Bonsai, FastGRNN are supported
2. Datasets:
   -ProtoNN and Bonsai: Cifar, CR*2, Curet, Letter, MNIST*2, USPS*2, Ward
   -FastGRNN: DSA, Google*2, HAR*2, MNIST

(The *2 signifies two different datasets bearing the same name are used)

(Check out the exact names of the datasets from SeeDot\datasets\<algorithm>\ which will contain folders bearing the names of the datasets)

The accuracy and model size of sketch on Uno can all be evaluated through the artifact.

The performance on Arduino Uno can be evaluated if the hardware is available, either in person or remotely.

Important Note:

(1) Performance numbers will vary from what is recorded in the paper FOR PROTONN AND BONSAI datasets. The performance numbers were recorded for an earlier version of Arduino Uno. In the version we are currently using (1.8.10), in all of the output codes we observe a 5-20% speedup (up to 50% in a few cases) for all codes running on Arduino.

(2) Model size (sketch size) numbers can vary slightly from what is recorded in the paper (+- few bytes) due to small updates in the boilerplate code.

(3) For FastGRNN, we also present estimated RAM usage in the paper. For FastGRNN, we note that in addition to sketch size, the RAM used by temporary variables was also a barrier to being able to run on Arduino Uno, hence we also give an estimate for RAM usage on Uno. This calculation is rough and only relevant to FastGRNN, not to ProtoNN and Bonsai. The RAM usage presented in the artifact is a closer estimate than the one presented in the paper, so it can be expected to be approx 10-30% higher than the values given in the paper.

(4) The approximate compile times on our machine are given in the last section in this file for reference.

Results this artifact does not support

1. None of our benchmarks required the usage of defragmentation (Section 6.1). Hence this artifact does support it.
2. The wakeword and industrial datasets are proprietary datasets which we are unable to distribute.

Primer to invoking the compiler

1. Please go through GETTING_STARTED guide's Running an Example section to ensure the compiler is set up correctly if not already done.
2. Note the command used in the above: 'python SeeDot-dev.py -a rnn -d HAR-6 -m red_disagree -v fixed -t arduino'. The arguments are as follows:
   - -a: algorithm to run, options are 'rnn' (for FastGRNN), 'protonn' (for ProtoNN), 'bonsai' (for Bonsai)
   - -d: dataset to run on, options are in the file 'SeeDot\SeeDot-dev.py' class 'Dataset', parameter 'all'
   - -m: metric which is optimized upon in the intermediate stages, options are 'acc' (for accuracy), 'disagree' (for disagreement), 'red_disagree' (for reduced disagreement)
   - -v: whether to generate floating point of fixed point code, options are 'fixed' or 'float'
   - -t: target device, options are 'arduino' or 'x86'
3. Note the default values for some of the parameters:
   - -d: if left blank, the compiler is run for all datasets which are in the file 'SeeDot\SeeDot-dev.py' class 'Dataset', parameter 'default'. So to run the compiler for all the datasets for FastGRNN, edit file 'SeeDot\SeeDot-dev.py' class 'Dataset', parameter 'default' to point to the FastGRNN datasets, and run the compiler ignoring the -d parameter. To run all the datasets for ProtoNN or Bonsai, edit file 'SeeDot\SeeDot-dev.py' class 'Dataset', parameter 'default' to point to the Bonsai\ProtoNN datasets, and run the compiler ignoring the -d parameter.
   - -a: if left blank, the compiler will be run both for ProtoNN and Bonsai (not FastGRNN). To generate code both for ProtoNN and Bonsai, simply ignore this flag (examples below)

NOTE: SeeDot-dev.py should only be invoked from the terminal while inside the \SeeDot\ directory (within the same directory as the SeeDot-dev.py file) (For example, running python SeeDot\SeeDot-dev.py i.e. outside the \SeeDot\ directory will crash the compiler)

Reproducing results

For FastGRNN

1. Edit file 'SeeDot\SeeDot-dev.py' class 'Dataset', parameter 'default' to point to the FastGRNN datasets (uncomment the line corresponding to FastGRNN and comment the other)
2. Run the command: 'python SeeDot-dev.py -a rnn -m red_disagree -v fixed -t arduino' <If you want to run for a a few datasets, add '-d <dataset-name>' to the command (without quotes) (only one dataset-name at a time)>
3. In the folder SeeDot\arduinodump\arduino\16\rnn\<dataset>\ there will be 6 files: 'compileConfig.h', 'model.h', 'predict.cpp', 'res', 'library.h', 'ram.usage'. The file 'res' will have the accuracy for the particular dataset, which can be tallied with the paper (Section 7 Table 4) and the file 'ram.usage' gives the RAM usage estimate, which can be tallied with the paper (Section 7 Figure 12)
4. Copy the 6 files in point 3 for each dataset one by one to SeeDot\arduinodump\arduino\ and open arduino.ino in Arduino IDE. Click on the arrow button on top left (upload) which will generate the arduino sketch and upload it to the board.
    -When the sketch is uploaded, the message box will show the size of the sketch which is the model size, and can be tallied with the paper (Section 7 Table 4)
    -Open Tools/Serial Monitor which will generate the time taken per prediction, which can be tallied with the paper (Section 7 Table 4)
5. For results on RAM consumption, we estimate the RAM usage in the Arduino code by summing up the sizes of all local variables declared within the FastGRNN Loop. Other sources of RAM usage like local variables, loops, function call overheads, are ignored as in this artifact the memory optimisation is only applied within one loop, and we illustrate RAM usage reduction within the loop where 10-15 variables are condensed to the space of 2-4 variables. As noted above, the current estimate may be higher than the values presented in the paper.

For ProtoNN, Bonsai

For these two datasets, the results are presented comparing to the floating point results. We discuss how to generate both fixed point code and floating point code
1. Edit file 'SeeDot\SeeDot-dev.py' class 'Dataset', parameter 'default' to point to the ProtoNN/Bonsai datasets (uncomment the line corresponding to ProtoNN/Bonsai and comment the other)

2. Run the command: 'python SeeDot-dev.py -m red_disagree -v fixed -t arduino' (ignoring -a flag will execute the compiler for both protonn and bonsai) <If you want to run for a a few datasets, add '-d <dataset-name>' to the command (without quotes) (only one dataset-name at a time)>
3. In the folder SeeDot\arduinodump\arduino\16\<algorithm>\<dataset>\ there will be 6 files: 'compileConfig.h', 'model.h', 'predict.cpp', 'library.h', 'res', 'ram.usage'. The file 'res' will have the accuracy for the particular dataset. 'ram.usage' can be ignored.

4. Run the command: 'python SeeDot-dev.py -m red_disagree -v float -t arduino' (ignoring -a flag will execute the compiler for both protonn and bonsai) <If you want to run for a a few datasets, add '-d <dataset-name>' to the command (without quotes) (only one dataset-name at a time)>
5. In the folder SeeDot\arduinodump\arduino\float\<algorithm>\<dataset>\ there will be 5 files: 'compileConfig.h', 'model.h', 'predict.cpp', 'library.h', 'ram.usage'. The accuracy can be checked on STDOUT (for floating point code the accuracy is directly printed on the terminal). These results can be compared with fixed point results in point 3 and matched with the paper (Section 7 Figures 6, 7)

6. For each algorithm (ProtoNN, Bonsai):
    For each dataset (Cifar, CR*2, Curet, Letter, MNIST*2, USPS*2, Ward):

        6a. Copy the 6 files inside SeeDot\arduinodump\arduino\16\<algorithm>\<dataset>\ to SeeDot\arduinodump\arduino\ and open arduino.ino in Arduino IDE. Click on the arrow button (upload) which will generate the arduino sketch and upload it to the board.
            -When the sketch is uploaded, the message box will show the size of the sketch which is the model size

-Open Tools\Serial Monitor which will generate the time taken per prediction

6b. Copy the 5 files inside SeeDot\arduinodump\arduino\float\<algorithm>\<dataset>\ to SeeDot\arduinodump\arduino\ and open arduino.ino in Arduino IDE. Click on the arrow button (upload) which will generate the arduino sketch and upload it to the board.

-When the sketch is uploaded, the message box will show the size of the sketch which is the model size, and can be compared to fixed point code in point 6 and tallied with the paper (Section 7 Figures 10, 11)

-Open Tools/Serial Monitor which will generate the time taken per prediction, and can be compared to fixed point code in point 6 and tallied with the paper (Section 7 Figures 8, 9)

IMPORTANT NOTE: For floating point code of Bonsai, cr-multiclass and curet-multiclass do not run on Arduino Uno as the models are too big. For them, no speedup numbers are reported in the paper. The floating point codes will compile on Arduino IDE, sketch size will be displayed but the sketch size will be reported as too high to fit on the board

Numerical values of various benchmarks

We provide the raw numbers used to construct graphs in Section 7. Note performance and model size numbers can differ due to differences in Arduino IDE versions. (Runtimes will be faster and model sizes will be better, all across the table for protonn and bonsai)

Bonsai:

(paper shows difference floating point - fixed point)

Accuracy	Fixed point code	Floating point code
cifar-binary	75.56	75.95
cr-binary	74.496	74.602
cr-multiclass	9.922	10.757
curet-multiclass	41.981	45.545
letter-multiclass	64.38	65.04
mnist-binary	95.9	95.96
mnist-multiclass	92.87	93.56
usps-binary	94.469	94.818
usps-multiclass	91.579	91.629
ward-binary	94.821	95.132

(paper shows the ratio floating point / fixed point)

Runtime (microseconds)	Fixed point code	Floating point code
cifar-binary	11201	44810
cr-binary	11480	44480
cr-multiclass	29745	does not run
curet-multiclass	31485	does not run
letter-multiclass	9281	30505
mnist-binary	14205	42653
mnist-multiclass	18412	55244
usps-binary	9841	40572
usps-multiclass	9513	37277
ward-binary	18858	52560

(paper shows ratio fixed point / floating point)

Model size (bytes)	Fixed point code	Floating point code
cifar-binary	13284	23978
cr-binary	13262	24004
cr-multiclass	13764	29214
curet-multiclass	15342	33096
letter-multiclass	8082	13664
mnist-binary	14688	24546
mnist-multiclass	15674	28320
usps-binary	12914	24288
usps-multiclass	9764	16704
ward-binary	15782	26220

ProtoNN:

(paper shows difference floating point - fixed point)

Accuracy	Fixed point code	Floating point code
cifar-binary	76.35	76.45
cr-binary	72.747	72.906
cr-multiclass	32.115	32.742
curet-multiclass	52.815	54.526
letter-multiclass	81.78	84.02
mnist-binary	94.95	95.21
mnist-multiclass	91.71	92.06
usps-binary	93.174	94.27
usps-multiclass	92.078	92.526
ward-binary	94.045	94.873

(paper shows the ratio floating point / fixed point)

Runtime (microseconds)	Fixed point code	Floating point code
cifar-binary	17057	49519
cr-binary	27770	95394
cr-multiclass	36123	159273
curet-multiclass	33038	160415
letter-multiclass	30647	136711
mnist-binary	17244	53897
mnist-multiclass	18523	63578
usps-binary	8823	26422
usps-multiclass	14128	51073
ward-binary	22524	63774

(paper shows ratio fixed point / floating point)

Model size (bytes)	Fixed point code	Floating point code
cifar-binary	10538	16926
cr-binary	13396	25702
cr-multiclass	13318	27988
curet-multiclass	13670	29976
letter-multiclass	9672	20596
mnist-binary	12118	21274
mnist-multiclass	12402	22112
usps-binary	7694	11470
usps-multiclass	8628	14722
ward-binary	14320	24776

Time needed for benchmarking

The given figures are runtimes on our machine, a Lenovo Thinkpad T495S running Windows 10 with AMD Ryzen 7 Pro @2.30Ghz with 8 logical processors

1. Command 'python SeeDot-dev.py -a rnn -m red_disagree -v fixed -t arduino' (For RNN Benchmarking)
    - Total time taken: **168 minutes**
    - For individual datasets:
    - dsa: 26 minutes
    - Google-12: 21 minutes
    - Google-30: 25 minutes
    - HAR-2: 9 minutes
    - HAR-6: 9 minutes
    - MNIST-10: 78 minutes
2. Command 'python SeeDot-dev.py -m red_disagree -v fixed -t arduino' (For ProtoNN/Bonsai Benchmarking)
    - Total time taken: **186 minutes**
    - For individual algorithms:
    - protonn: 108 minutes
        - For individual datasets:
        - cifar-binary: 18 minutes
        - cr-binary: 3 minutes
        - cr-multiclass: 2 minutes
        - curet-multiclass: 4 minutes
        - letter-multiclass: 2 minutes
        - mnist-binary: 37 minutes
        - mnist-multiclass: 32 minutes
        - usps-binary: 3 minutes
        - usps-multiclass: 2 minutes
        - ward-binary: 5 minutes
    - bonsai: 78 minutes

For individual datasets:

cifar-binary:	12 minutes
cr-binary:	2 minutes
cr-multiclass:	2 minutes
curet-multiclass:	3 minutes
letter-multiclass:	2 minutes
mnist-binary:	25 minutes
mnist-multiclass:	24 minutes
usps-binary:	2 minutes
usps-multiclass:	2 minutes
ward-binary:	4 minutes

3. Command 'python SeeDot-dev.py -m red_disagree -v float -t arduino' (For ProtoNN/Bonsai Floating point code)

total runtime: 10 minutes (For ProtoNN, Bonsai, 10 datasets each)

4. If the compiler is too slow on your machine, please randomly subsample the dataset in train.npy (and optionally in test.npy). For example, the MNIST dataset contains 60000 datapoints. If it is desired to make the compiler run faster, randomly subsample, say 10%, or 6000 datapoints out of the full dataset (store the subsampled dataset in the same file train.npy) and the compiler should speed up. However, results in that case may be slightly different (in terms of accuracy and runtimes) than the ones reported in the paper.

The remainder part of the documentation is meant for those who wish to study the compiler in greater detail and extend it for further research

Design of the Compiler

This file presents a high level overview of how the compiler is written.
Not all files are described. Only a few files, which describe the backbone of the compiler are described.
For an example of how to extend the compiler, check out the ADDING_FUNCTIONS guide.
The Shiftry compiler is built on top of the Seedot framework.
The organisation of this description is as follows:

<Description>: <Class Name>
    <Member function A> (All methods are in the order which they are called by the parent)
    Description of A
        ->Method B invoked by Member function A
        Description of B
            ->Method C invoked by function B
            Description of C
        ->Second method invoked by Member function A
    . ..

Driver code: main.py
    1. main.py::convert()
    This method converts the model parameters and inputs from floating point to fixed point code by invoking the Converter object (converter.py)
    2. main.py::performSearch()
    This method is responsible for generating different scales and bitwidths according which candidate target codes would be generated.
        The following two methods are invoked repeatedly for multiple stage of the exploration. These stages are the ones described as annotations to the program output in the GETTING_STARTED guide
        -> main.py::partialCompile()
        This is invoked by performSearch() with one particular assignment of scales and bitwidths. This method calls the Compiler object described below and generates the appropriate code
        -> main.py::runAll()
        This is invoked by performSearch() and it calls the Predictor object (predictor.py) which takes the code generated by partialCompile(), builds, executes and evaluates the accuracy of the target code on the given dataset
    3. main.py::compileFixedForTarget() / main.py::compileFloatForTarget()
    These methods are used to dump the final output code which can be run on an Arduino Uno.

Compiler code: compiler.py
    1. compiler.py::run()
    This method takes the input code, information about variable scales and bitwidths and generates the fixed point code through the following methods.
    -> compiler.py::genAST()

This method takes in the input code and converts it into an AST
-> compiler.py::InferType()
This object is used to type check the input AST and annotate the AST with the types of all variables
-> compiler.py::compile()
-> compiler.py::readDataDrivenScales()
This method is called during fixed point code generation, and it computes the ranges of values taken by intermediate variables for data driven scaling (Section 5.1 of the paper)
-> compiler.py::IRBuilder
This object takes the type annotated AST and translates it into the output code, which is a sequence of function calls
-> compiler.py::codegen.printAll()
This method takes the output of IRBuilder and generates the output code, either for X86 or Arduino

Writing a Program

This is a tutorial on how to write a program using Shiftry

Check out 'SeeDot\seedot\compiler\antlr\seedot.g4' for the syntax supported.
The following is a non exhaustive list of operations currently supported. For some of them, example usages are pointed to inside the provided models:

-> initialising a tensor to zero (Used in ProtoNN/Bonsai all examples)

```
let var = init([10, 20, 64], 0.0) in
```

-> transpose (Used in ProtoNN all examples)

```
let trans_var = var^T in
```

-> reshape (Used in all provided examples for variable X)

```
let A = init([10, 4, 2, 10], 0.0) in
    ...
let Ashape = reshape(A, (20, 40), (1, 3, 2, 4)) in
```

(20, 40) refers to the shape of the reshaped tensor, (1, 4, 3, 2) represents the order in which the original array will be traversed
The statement is equivalent to the torch statement Ashape = A.permute(0, 2, 1, 3).reshape(20, 40)

-> splice (Used in FastGRNN most examples)

```
let A = B[0:+2][1:+3] in
```

within each square bracket [a:+b], a represents the starting index (can be any expression) and b is a fixed integer represent the size of the splice along that axis

-> maxpool

```
let A = maxpool(B, 3) in
```

this is the standard maxpool operator in ML applications. The second argument represents the kernel size

-> indexing (Used in Bonsai all examples)

```
let A = init([10, 20, 30], 0.0) in
let B = A[7]                    //B now has a size of [20, 30]
let C = A[7][18][29]    //C is a scalar
```

-> conditional (Used in Bonsai all examples)

```
let A = B >= 0 ? C : D in
```

-> summation loops (Used in ProtoNN all examples)

```
let res = $(i = [0:40])(
        A[i]
) in
```

this translates to res = A[0] + A[1] + A[2] + A[3] + ... + A[39]

-> for loops (Used in FastGRNN all examples)

```
let H = init([1, 20], 0.0) in
let res = loop(i = [0:40], H)(
        let G = A * H + B in
        G
) in
```

the variable H is an accumulator here. the accumulator must be specified along with the counter in the loop header. after this, within the loop body, the accumulator will be treated like a global variable whose value can be used and updated. within the loop body, the last expression (in this case G) is what is assigned to the accumulator H in each iteration

-> relu, exp (used in ProtoNN), argmax (used in all models which are not binary classifiers), sgn (used in binary classifiers), tanh (used in Bonsai, FastGRNN), sigmoid (used in Bonsai, FastGRNN)

```
let A = <func name>(B) in
```

these operators are applied pointwise (except sgn, which expects a scalar). sgn is the signum operator (1 if the variable is positive, 0 if negative)

-> matrix addition, multiplication, subtraction

Supported for 2D matrices

-> sparse matrix multiplication

```
let X = [1, 100] in (-4, 4) in
let val = W |*| X in
```

the second argument MUST be X

the first argument MUST be a model parameter

-> hadamard multiplication

```
let C = A <*> B in
```

supports 2D matrices, element wise matrix multiplication

We take the example of a ProtoNN model for 'letter-multiclass' dataset to demonstrate how to code in Shiftry. This code already exists in the dataset provided, but we give the tutorial in a way that can be extended to any other algorithm on any dataset.
As a system which generates ML inference code, Shiftry assumes that the user has access to a trained model and atleast some validation data for tuning the scales of variables and judging the performance of the code.

Example Program

1. In Shiftry, ProtoNN is expressed as follows:

```
>> let X   = (16, 1)   in [-2.864300, 2.935500] in
>> let W  = (10, 16)    in [-5.634220, 4.518250] in
>> let B  = (104, 10, 1) in [-14.832600, 16.682200] in
```

```
>> let Z  = (104, 26, 1) in [-4.798770, 9.011270] in
>> let g2 = 0.032470 in
>>
>> let WX = W |*| X in
>> let res = $(i = [0:104])
>> (
>>       let del = WX - B[i] in
>>       Z[i] * exp(-g2 * (del^T * del))
>> ) in
>> argmax(res)
```

2. This code should be placed in a file called 'input.sd' and placed in the folder SeeDot\model\<algorithm>\<dataset>\. Here, algorithm is protonn and dataset is letter-multiclass.
3. In SeeDot-dev.py, class Dataset, add 'letter-multiclass' to either the 'common' or 'extra' field, if not present in either. This is a sanity check to guard against errors like, for example mixing up the algorithm and dataset names while running the compiler.
4. In SeeDot/config.py class Algorithm, add a parameter append "protonn" to the list parameter 'all'. This is a similar sanity check as in point 3.
5. In the code above, the first 4 lines signify that X, W, B, Z are tensors which must be read from the file. Shiftry recognises that X is the input datapoint, and rest are the model parameters.
6. 'let param = (num1, num2, ...) in [min, max] in' means that param is a tensor to be read from file. it's dimensions are (num1, num2, ...) and its minima is 'min' and maxima is 'max' (used for scale computation of model parameters)
7. All parameters are read from <param_name>.npy file present within SeeDot\model\<algorithm>\<dataset>\. So in this case, there should be files called W.npy, B.npy, Z.npy within Seedot\model\protonn\letter-multiclass.
IMPORTANT NOTE: While generating the model files, it is recommended to compress the weights into a 2D tensor, as Shiftry will not be able to read .npy's with more than 2 dimensions.
So in the above example, B has dimensions (104, 10, 1), so while creating B.npy, reshape B to (104, 10) or (1, 1040) or (1040, 1) before saving the tensor.
8. The variable X will be read from train.npy or test.npy file present within SeeDot\dataset\<algorithm>\<dataset>\. Shiftry expects thes files to have the dimensions (number of datapoints, number of features per datapoint + 1). For each datapoint, we append the class label (which in this case can be a number between 1 and 26 inclusive) at the start of the datapoint. So, in this case, we have a 16-dimensionsal input X and one class Y, so one particular datapoint in train.npy or test.npy can look like
[Y, X0, X1, X2, X3, X4, X5, X6, X7, X8, X9, X10, X11, X12, X13, X14, X15]
9. train.npy and test.npy should be disjoint subsamples of the entire dataset for the best results. For the tuning of scales and bitwidths in the fixed point code, Shiftry uses train.npy and for the final evaluation of accuracy, Shiftry evaluates on test.npy
10. To run the code on the newly added dataset, run 'python Seedot-dev.py -a <algorithm> -d <dataset> -v fixed -t arduino -m red_disagree'

Known Issues

1. The variable X, must either be reshaped or sparsematmulled.
    -> let Xprocessed = reshape(X, (10, 20), (1, 2)) in ...
    -> let Xprocessed = W |*| X in ...
    These are the only two operations supported for variable X, however they do not affect expressibility (reshape can be an identity operation)
2. The variable X must be declared like:
    let X = (number of features, 1) in

where number of features is a single integer. for multidimensional input, X must be flattened first into a 1D vector and through the reshape command can be brought back to the appropriate shape

3. Operations are not permitted over arbitrary tensors. Check out SeeDot\seedot\Predictor\library_fixed.h to see what operators support operands of what dimensions. Keep in mind that the reshape command works for arbitrary dimensions, and can be used to bring tensors to the appropriate dimensionality.

4. exponentiation is only supported for negative numbers. Feeding positive numbers into a exp() will be undefined for fixed point code.

Extending the Compiler

Shiftry translates the given input code into a sequence of function calls. The functions are already implemented in a library. What if the model being implemented needs a function which does not already implemented in the library? We take the example of convolution, which is currently not implemented in the artifact.

Note: We only show how to add a new node for X86 architecture. Adding the node for Arduino is omitted for brevity, but follows very closely with the X86 guidelines.

STEP 1: Operator description

We present an implementation of convolution which supports padding, strides, dilations, as well as groups (for depthwise separable convolutions) which is similar to what is provided by pytorch (https://pytorch.org/docs/stable/generated/torch.nn.Conv2d.html?highlight=conv2d#torch.nn.Conv2d)

Suppose our syntax looks like the following:

let A = <some N*H*W*C tensor>

let B = <some G*FH*FW*C*Cout tensor>

let C = conv2d(A, B, {s 1 1}, {p 0 0 1 1}, {d 1 1}, {g 2})

In this syntax, s represents the 2 stride parameters (vertical, horizontal), p represents the 4 padding parameters (up, down, left, right), d represents the 4 dilation parameters (vertical, horiontal), g represents the number of groups. For details please refer to the pytorch documentation of conv2d given above

STEP 2: Adding the grammar in the file

1. The grammar is in the file 'SeeDot\seedot\compiler\antlr\seedot.g4'. Add the following: (remove the '>>') (feel free to change symbols if preferred)

```
>> |      Conv2d '(' expr ',' expr ','
>>                 '{s' IntConst IntConst '}' ','
>>                 '{p' IntConst IntConst IntConst IntConst'}' ','
>>                 '{d' IntConst IntConst '}' ','
>>                 '{g'IntConst'}' ')'                                    # convolution
```

to the rules for expr. The name after the # (convolution) is the name which would be generated for this operator when we generate the parser. (Watch out for methods named visitConvolution below)

2. After updating the grammar file, we need to generate a new parser. Navigate to SeeDot\seedot\lib\ in a terminal and execute the following command (remove '>>'):

```
>> java -jar .\antlr-4.7-complete.jar ..\compiler\antlr\seedot.g4 -visitor -Dlanguage=Python3 -o kl
```

This will generate the new parser and associated files in the folder SeeDot\seedot\lib\compiler\antlr\

3. Within the folder SeeDot\seedot\lib\compiler\antlr\ there will be 6 files. Delete seedotListener.py (it is not needed) and copy the rest of the files to the directory SeeDot\seedot\compiler\antlr\ (there already will be 5 files of the same name which you must overwrite). After this Shiftry's parser will be updated.

STEP 3: Adding new nodes in the AST

Since we are adding a new rule for the expr, we will need to add a new node for our convolution operation to the abstract syntax tree (AST).

1. Go to the file SeeDot\seedot\compiler\ast\ast.py

Here, we will add a node which captures all the relevant information for the convolution node. We have discusse above that the node has an input image, an input filter, properties called stride, padding, dilation, groups. So we add the following class to the end of the file: (remove '>>')

```
>> class Convolution(ASTNode):
>>
>>    def __init__(self, expr1, expr2, stride, padding, dilation, groups):
>>        super().__init__()
>>        self.expr1 = expr1
>>        self.expr2 = expr2
>>        self.stride = stride
>>        self.padding = padding
>>        self.dilation = dilation
>>        self.groups = groups
```

2. We also need a mechanism to construct an object of the class Convolution. Navigate to SeeDot\seedot\compiler\ast\astBuilder.py, and add the following method as a member of the class ASTBuilder: (remove '>>')

```
>> def visitConvolution(self, ctx: SeeDotParser.ConvolutionContext):
>>        expr1 = self.visit(ctx.expr(0))
>>        expr2 = self.visit(ctx.expr(1))
>>        stride = [int(ctx.IntConst(i).getText()) for i in range(0, 2)]
>>        padding = [int(ctx.IntConst(i).getText()) for i in range(2, 6)]
>>        dilation = [int(ctx.IntConst(i).getText()) for i in range(6, 8)]
>>        groups = int(ctx.IntConst(8).getText())
>>        return AST.Convolution(expr1, expr2, stride, padding, dilation, groups)
```

We explain the above builder method. Take a look at the addition to the grammar file in STEP 1. In a convolution node, the first two sub expressions signify the two inputs. Since they are subexpressions within our convolution expression, we must recurse down both of them one after the other. ctx.expr(0) refers to first subexpression, ctx.expr(1) refers to the second sub expressions.

After the first two subexpressions, we have 2 integers (0, 1) for the stride parameter, 4 integers (2, 3, 4, 5) for padding, 2 integers (6, 7) for dilation, and 1 integer (8) for groups. all of these numbers are read by the parser in the same order, and we extract the numbers thus. The ctx.expr(), ctx.IntConst(i).getText() etc. are pre generated from the parser in STEP 2. After extracting all parameters, we simply construct the convolution class and return.

3. Navigate to SeeDot\seedot\compiler\ast\astVisitor.py. The ASTVisitor class is inherited by the type checker, IR generator etc. In the visit() method, before the else branch, add the following: (remove '>>')

```
>> elif isinstance(node, AST.Convolution):
>>         return self.visitConvolution(node)
```

4. Edit some boilerplate code. In SeeDot\seedot\compiler\ast\mtdAST.py, within the MtdAST class, add the following method: (remove '>>')

```
>> def visitConvolution(self, node: AST.Convolution, mtd: dict):
>>     node.metadata.update(mtd)
>>     self.visit(node.expr1, mtd)
>>     self.visit(node.expr2, mtd)
```

Include all the subexpressions of the node which is being added, but no need to add int constants (which are not recursively explored)

In SeeDot\seedot\compiler\ast\printAST.py, within the PrintAST class, add the following method: (remove '>>')

```
>> def visitConvolution(self, node: AST.Convolution):
>>     node.expr1.printLevel = node.expr2.printLevel = node.printLevel + 1
>>     print(indent * node.printLevel, "conv(", )
>>     self.visit(node.expr1)
>>     self.visit(node.expr2)
>>     print(",", node.stride, ',', node.padding, ',', node.dilation, ',',node.groups, ')')
```

This is for pretty printing. Feel free to edit if not pretty enough.

STEP 4: Type Checking

Navigate to SeeDot\seedot\compiler\type.py and add the following code as a member method of Type class (remove '>>'):

```
>> def visitConvolution(self, node: ast.Convolution):
>>     node.expr1.gamma = dict(node.gamma)
>>     eType = self.visit(node.expr1)
>>
>>     assert eType.dim == 4
>>     [n, h, w, cin] = eType.shape
>>
>>     node.expr2.gamma = dict(node.gamma)
>>     fType = self.visit(node.expr2)
>>
>>     assert fType.dim == 5
>>     [g, hf, wf, cin_, cout] = fType.shape
>>
>>     assert cin_ * g == cin
```

```
>>    assert g == node.groups
>>
>>    assert hf % 2 == wf % 2 == 1, "Odd filter sizes supported"
>>
>>    for i in range(0,4):
>>       assert node.padding[i] >= 0, "Padding cannot be negative"
>>    assert node.stride[0] > 0 and node.stride[1] > 0, "Stride must be positive"
>>    assert node.dilation[0] > 0 and node.dilation[1] > 0, "Dilation must be positive"
>>
>>    hout = (h + node.padding[0] + node.padding[1] - node.dilation[0] * (hf - 1) - 1) // node.stride[0] + 1
>>    wout = (w + node.padding[2] + node.padding[3] - node.dilation[1] * (wf - 1) - 1) // node.stride[1] + 1
>>    shape = [n, hout, wout, g * cout]
>>
>>    node.type = Tensor(shape)
>>    return node.type
```

This function checks whether the types of the inputs are compatible with the outputs, whether there are illegal values and so on.
For example, the following are the checks made for convolution:
1. The input image should be of dimension 4 corresponding to (batch, number of rows, number of columns, channels), and the filter should match the number of channels
2. Our implementation does not support even filter sizes, so we can make a check here to only allow odd filter sizes or throw an exception
3. The values for padding should not be negative, neither should dilations be negative
The method also computes the type of the output node, given the input (look at the assignment node.type = Tensor(shape), which is returned)
For other operators, different checks may be there. For example for addition, the dimensions of both inputs should match and the output should also have the same dimension as inputs.

STEP 5: Implementing the operator
This part can be tricky as it involves a good understanding of fixed point arithmetic and how to best tune the hyperparameters which are introduced because of fixed point. It is recommended to go through MatAdd, MatMul, Exponentiation function codes to get an idea

1. We first add an implementation of the operator in floating point. We navigate to SeeDot\seedot\Predictor\library_float.h and add the following (remove '>>'):

```
>> void Convolution(float *A, const float *B, float *C, float *tmp, MYINT N, MYINT H, MYINT W, MYINT CIN, MYINT HF, MYINT WF, MYINT CINF, MYINT COUTF, MYINT HOUT, MYINT WOUT, MYINT HPADL, MYINT HPADR, MYINT WPADL, MYINT WPADR, MYINT HSTR, MYINT WSTR, MYINT HDL, MYINT WDL, MYINT G, MYINT shrA, MYINT shrB, MYINT H1, MYINT H2);
```

2. We navigate to SeeDot\seedot\Predictor\library_float.cpp and add the following (remove '>>'):

```
>> void Convolution(float *A, const float *B, float *C, float *tmp, MYINT N, MYINT H, MYINT W, MYINT CIN, MYINT HF, MYINT WF, MYINT CINF, MYINT COUTF, MYINT HOUT, MYINT WOUT, MYINT HPADL, MYINT HPADR, MYINT WPADL, MYINT WPADR, MYINT HSTR, MYINT WSTR, MYINT HDL, MYINT WDL, MYINT G, MYINT shrA, MYINT shrB, MYINT H1, MYINT H2) {
>>      MYITE HOffsetL = HDL*(HF/2) - HPADL;
>>      MYITE WOffsetL = WDL*(WF/2) - WPADL;
```

```
>>      MYITE HOffsetR = HDL*(HF/2) - HPADR;
>>      MYITE WOffsetR = WDL*(WF/2) - WPADR;
>>
>>      for(MYITE n = 0; n < N; n++) {
>>              for(MYITE h = HOffsetL, hout = 0; h < H - HOffsetR; h += HSTR, hout++) {
>>                      for(MYITE w = WOffsetL, wout = 0; w < W - WOffsetR; w += WSTR, wout++) {
>>                              for(MYITE g = 0; g < G; g++) {
>>                                      for(MYITE co = 0; co < COUTF; co ++) {
>>
>>                                              MYITE counter = 0;
>>                                              for(MYITE hf = -(HF/2); hf <= HF/2; hf++) {
>>                                                      for(MYITE wf = -(WF/2); wf <= WF/2; wf++) {
>>                                                              for(MYITE ci = 0; ci < CINF; ci++) {
>>
>>                                                                      float a = (((h + HDL * hf) < 0) || ((h + HDL * hf) >= H) || ((w + WDL * wf) < 0) || ((w + WDL * wf)
      >= W)) ? 0 : A[n * H * W * CIN + (h + HDL * hf) * W * CIN + (w + WDL * wf) * CIN + (ci + g * CINF)];
>>
>>                                                                      float b = B[g * HF * WF * CINF * COUTF + (hf + HF/2) * WF * CINF * COUTF + (wf + WF/2) * CINF
      * COUTF + ci * COUTF + co];
>>
>>                                                                      tmp[counter] = a * b;
>>                                                                      counter++;
>>                                                              }
>>                                                      }
>>                                              }
>>
>>                                              MYITE totalEle = HF * WF * CINF;
>>                                              MYITE count = HF * WF * CINF, depth = 0;
>>                                              bool shr = true;
>>
>>                                              while (depth < (H1 + H2)) {
>>                                                      if (depth >= H1)
>>                                                              shr = false;
>>
>>                                                      for (MYITE p = 0; p < (totalEle / 2 + 1); p++) {
>>                                                              float sum;
>>                                                              if (p < (count >> 1))
>>                                                                      sum = tmp[2 * p] + tmp[(2 * p) + 1];
>>                                                              else if ((p == (count >> 1)) && ((count & 1) == 1))
>>                                                                      sum = tmp[2 * p];
>>                                                              else
```

```
>>                                                      sum = 0;
>>
>>                                              if (shr)
>>                                                      tmp[p] = sum;
>>                                              else
>>                                                      tmp[p] = sum;
>>                                      }
>>                                      count = (count + 1) >> 1;
>>
>>                                      depth++;
>>                              }
>>
>>                              C[n * HOUT * WOUT * (COUTF * G) + hout * WOUT * (COUTF * G) + wout * (COUTF * G) + (co + g * COUTF)] = tmp[0];
>>                      }
>>              }
>>      }
>>  }
>> }
```

In most cases, this implementation itself can act as a reference for adding other operators. The code from MYITE counter = 0; to MYITE totalEle = HF * WF * CINF is the standard convolution operation, and the while loop in the code is tree sum addition (From SeeDot, Gopinath et al., PLDI 2019)

3. Navigate to SeeDot\seedot\Predictor\library_fixed.h and append the following (remove the '>>') (lines not beginning with '>>' are explanatory lines and can be ignored):

```
>> template<class TypeA, class TypeB, class TypeTemp, class TypeC>
>> void Convolution(TypeA *A, const TypeB *B, TypeC *C, TypeTemp *tmp, MYINT N, MYINT H, MYINT W, MYINT CIN, MYINT HF, MYINT WF, MYINT CINF, MYINT COUTF,
        MYINT HOUT, MYINT WOUT, MYINT HPADL, MYINT HPADR, MYINT WPADL, MYINT WPADR, MYINT HSTR, MYINT WSTR, MYINT HDL, MYINT WDL, MYINT G,
        MYINT shrA, MYINT shrB, MYINT H1, MYINT H2, MYINT demote) {
// Most parameters are self explanatory ones (parameters like inputs, outputs, sizes of inputs and outputs and filters). TypeA, TypeB, TypeTemp, TypeC can be int8_t,
int16_t or int32_t, and represent the bitwidths for input image, input filter, temp buffer, output image.
// shrA, shrB, H1, H2, demote are parameters pertaining to fixed point code:
// shrA and shrB and demote are used for controlling scale during fixed point multiplication (Refer to Section 2.2 and Section 3 of the paper)
// H1 and H2 are parameters of tree sum (the while loop in the function), which is used to sum up long vectors without losing significant bits
>>      MYITE HOffsetL = HDL*(HF/2) - HPADL;
>>      MYITE WOffsetL = WDL*(WF/2) - WPADL;
>>      MYITE HOffsetR = HDL*(HF/2) - HPADR;
>>      MYITE WOffsetR = WDL*(WF/2) - WPADR;
>>
>>      for(MYITE n = 0; n < N; n++) {
```

```
>>                 for(MYITE h = HOffsetL, hout = 0; h < H - HOffsetR; h += HSTR, hout++) {
>>                         for(MYITE w = WOffsetL, wout = 0; w < W - WOffsetR; w += WSTR, wout++) {
>>                                 for(MYITE g = 0; g < G; g++) {
>>                                         for(MYITE co = 0; co < COUTF; co ++) {
>>
>>                                                 MYITE counter = 0;
>>                                                 for(MYITE hf = -(HF/2); hf <= HF/2; hf++) {
>>                                                         for(MYITE wf = -(WF/2); wf <= WF/2; wf++) {
>>                                                                 for(MYITE ci = 0; ci < CINF; ci++) {
>>
>>                                                                         TypeTemp a = (TypeTemp) (((h + HDL * hf) < 0) || ((h + HDL * hf) >= H) || ((w + WDL * wf) < 0) ||
        ((w + WDL * wf) >= W)) ? 0 : A[n * H * W * CIN + (h + HDL * hf) * W * CIN + (w + WDL * wf) * CIN + (ci + g * CINF)];
>>
>>                                                                         TypeTemp b = (TypeTemp) B[g * HF * WF * CINF * COUTF + (hf + HF/2) * WF * CINF * COUTF +
        (wf + WF/2) * CINF * COUTF + ci * COUTF + co];
>>
>>                                                                         tmp[counter] = a * b;
>>                                                                         counter++;
>>                                                                 }
>>                                                         }
>>                                                 }
>>
>>                                                 MYITE totalEle = HF * WF * CINF;
>>                                                 MYITE count = HF * WF * CINF, depth = 0;
>>                                                 bool shr = true;
>>
>>                                                 while (depth < (H1 + H2)) {
>>                                                         if (depth >= H1)
>>                                                                 shr = false;
>>
>>                                                         for (MYITE p = 0; p < (totalEle / 2 + 1); p++) {
>>                                                                 TypeTemp sum;
>>                                                                 if (p < (count >> 1)) {
>>                                                                         if (shr)
>>                                                                                 sum = tmp[2 * p] / 2 + tmp[(2 * p) + 1] / 2;
>>                                                                         else
>>                                                                                 sum = tmp[2 * p] + tmp[(2 * p) + 1];
>>                                                                 }
>>                                                                 else if ((p == (count >> 1)) && ((count & 1) == 1)) {
>>                                                                         if (shr)
>>                                                                                 sum = tmp[2 * p] / 2;
>>
```

```
>>                                                                      else
>>                                                                              sum = tmp[2 * p];
>>                                                              }
>>                                                              else
>>                                                                      sum = 0;
>>
>>                                                              tmp[p] = sum;
>>                                                      }
>>                                                      count = (count + 1) >> 1;
>>
>>                                                      depth++;
>>                                              }
>>
>>                                              C[n * HOUT * WOUT * (COUTF * G) + hout * WOUT * (COUTF * G) + wout * (COUTF * G) + (co + g * COUTF)] =
        Saturate<TypeC>(((tmp[0] / shrA) / shrB) / demote);
>>                                      }
>>                              }
>>                      }
>>              }
>>      }
>> }
```

STEP 6: Handling the operator in IR

This part can also be tricky as the user would need to know how to compute bitwidths, scales for the outputs given the inputs' parameters. It is recommended to study MatAdd, MatMul, Exponentiation, TanH functions for a good understanding

1. Navigate to SeeDot\seedot\compiler\ir\irBuilder.py, and add the following as a member method of the IRBuilder class: (remove '>>') (lines not beginning with '>>' are comments and can be ignored)

```
>> def visitConvolution(self, node: AST.Convolution):
>>
>>    (prog_in_A, expr_in_A) = self.visit(node.expr1)
>>    (prog_in_B, expr_in_B) = self.visit(node.expr2)
>>
#     The above two are required to explore the AST of the input subexpression. If a node has n subexpressions, there would be n statements instead of the two above for each subexpression
>>
>>    [expr_treeSum, expr_out] = self.getTempVars(2)
>>
#     In the above a new temporary variable will be assigned to both the output, and the temporary buffer used by the function
>>
>>    [N, H, W, Cin] = node.expr1.type.shape
```

```
>>     [G, Hf, Wf, CinF, CoutF] = node.expr2.type.shape
>>
#      The type checked inputs' dimensions are extracted from node, as out implementation expects these values. It may or may not be necessary to extract all such information
>>
>>     type_treeSum = Type.Tensor([Hf * Wf * CinF])
>>     type_out = node.type
>>
#      In the convolution operator, a temporary buffer is created whose type can be inferred from the inputs' types, but will not be encountered in the source code. Hence they are declared here
>>
>>     bitwidth_in_A, scale_in_A = self.getBitwidthAndScale(expr_in_A.idf)
>>     bitwidth_in_B, scale_in_B = self.getBitwidthAndScale(expr_in_B.idf)
>>     if self.ddsEnabled:
>>         bitwidth_out, scale_out = self.getBitwidthAndScale(expr_out.idf)
>>         bitwidth_temp, scale_temp = self.getBitwidthAndScale(expr_out.idf, native=True)
>>     else:
>>         bitwidth_out = config.wordLength // 2 if expr_out.idf in self.demotedVarsList else config.wordLength
>>         scale_out, scale_temp = None, None
>>         bitwidth_temp = bitwidth_out
>>
#      ddsEnabled flag being true means that the output's scale is computed through profiling. if the flag is false, the scale would have to be computed manually. The getBitwidtAndScale method always returns the scale and the bitwidth assignment (It is known which all variables are in 8 bits before the IRBuilder class is invoked)
>>
>>     intv_in_A, intv_in_B = (0, 0), (0, 0)
>>     intv_out = (0, 0)
>>
>>     shr_A, shr_B, H1, H2, demote, scale_out = self.getShrTreeSumAndDemoteParamsForMul(bitwidth_in_A, scale_in_A, bitwidth_in_B, scale_in_B, bitwidth_temp, scale_temp, bitwidth_out, scale_out, Hf * Wf * CinF)
>>
#      The helper method getShrTreeSumAndDemoteParamsForMul can be directly used to infer scales for multiplication outputs, and also compute the parameters which the fixed point implementation would expect for multiplication. There are many such helper methods, check out the node for MatAdd for addition.
>>
>>     shr_A = self.formatShr(shr_A)
>>     shr_B = self.formatShr(shr_B)
>>
>>     expr_in_A.inputVar = False
>>     expr_in_B.inputVar = False
>>     expr_out.inputVar = False
>>     expr_treeSum.inputVar = False
>>
```

```
>>    if forFixed():
>>       self.varsForBitwidth[expr_treeSum.idf] = bitwidth_temp
>>
>>    comment = IR.Comment('conv(%s, %s)' %(expr_in_A.idf,expr_in_B.idf), self.counter_inst+1)
>>    self.allDepths[self.counter_inst+1] = self.curDepth
>>
>>    bitwidth_mul = self.getTempBitwidth(bitwidth_in_A, bitwidth_in_B, "mul")
>>    if self.vbwEnabled:
>>       self.varsForBitwidth[expr_treeSum.idf] = bitwidth_mul
>>
#     The temporary buffer variables are introduced by the compiler, and are not profiled by the input code. Their bitwidths are set using the values of bitwidths of the input
>>
>>    argMap = {
>>       expr_in_A: "A",
>>       expr_in_B: "B",
>>       expr_out: "C",
>>       expr_treeSum: "tmp",
>>       IR.Int(N): "N",
>>       IR.Int(H): "H",
>>       IR.Int(W): "W",
>>       IR.Int(Cin): "CIN",
>>       IR.Int(Hf): "HF",
>>       IR.Int(Wf): "WF",
>>       IR.Int(CinF): "CINF",
>>       IR.Int(CoutF): "COUTF",
>>       IR.Int(type_out.shape[1]): "HOUT",
>>       IR.Int(type_out.shape[2]): "WOUT",
>>       IR.Int(node.padding[0]): "HPADL",
>>       IR.Int(node.padding[1]): "HPADR",
>>       IR.Int(node.padding[2]): "WPADL",
>>       IR.Int(node.padding[3]): "WPADR",
>>       IR.Int(node.stride[0]): "HSTR",
>>       IR.Int(node.stride[1]): "WSTR",
>>       IR.Int(node.dilation[0]): "HDL",
>>       IR.Int(node.dilation[1]): "WDL",
>>       IR.Int(G): "G",
>>       shr_A: "shrA",
>>       shr_B: "shrB",
>>       IR.Int(H1): "H1",
>>       IR.Int(H2): "H2"
>>    }
```

```
  >>    if self.vbwEnabled:
  >>        argMap[IR.Int(demote)] = "demote"
  >>
  >>    if not self.vbwEnabled:
  >>        funcCall = IR.FuncCall("Convolution", argMap) #, {expr_treeSum.idf: type_treeSum})
  >>    else:
  >>        funcCall = IR.FuncCall("Convolution" + ("<int%d_t, int%d_t, int%d_t, int%d_t>"%(bitwidth_in_A, bitwidth_in_B, bitwidth_mul, bitwidth_out)), argMap) #,
{expr_treeSum.idf: type_treeSum})
  >>
  #     The argMap variable holds the arguments of the function call. Each key corresponds to one argument, the value is not required but is added for reference
  >>
  >>    self.counter_inst += 1
  >>    self.updateLiveRange([expr_in_A, expr_in_B, expr_out, expr_treeSum])
  >>
  #     The above updateLiveRange method updates the live ranges of the variable being alive. Refer to Section 6 in the paper about Memory Management
  >>
  >>
  >>    profile = IR.FuncCall("Profile4", {
  >>        expr_out: "Var",
  >>        IR.Int(N): "I",
  >>        IR.Int(type_out.shape[1]): "J",
  >>        IR.Int(type_out.shape[2]): "K",
  >>        IR.Int(CoutF * G): "L",
  >>        IR.String(expr_out): "VarName"
  >>    })
  >>    if forFloat():
  >>        self.independentVars.append(expr_out.idf)
  >>
  #     For convolution, the output variable is profiled from the data to infer it's scale. For any variable which needs to be profiled, the above function call is added (it gets
added to the floating point code), If a variable's output scale is known (for example sigmoid's output is always bounded by +-1, so it need not be profiled and the output scale
can be set directly)
  >>
  >>    prog_conv = IR.Prog([comment, funcCall, profile] if forFloat() and self.ddsEnabled else [comment, funcCall])
  >>
  >>    prog_out = IRUtil.concatPrograms(prog_in_A, prog_in_B, prog_conv)
  >>
  >>    # Update context for output variable
  >>    self.varDeclarations[expr_out.idf] = type_out
  >>    self.varScales[expr_out.idf] = scale_out
  >>    self.varIntervals[expr_out.idf] = intv_out
  >>
```

```
>>    self.varDeclarations[expr_treeSum.idf] = type_treeSum
>>    self.varScales[expr_treeSum.idf] = scale_temp
>>    self.varIntervals[expr_treeSum.idf] = (0, 0)
>>
#     varDeclarations variable is to make a list of variables which need to be instantiated in the output code
>>
>>    self.log.print(comment.msg)
>>    self.log.print("\tInput1: scale = %d, interval = [%d, %d]" % (
>>       (self.varScales[expr_in_A.idf],) + self.varIntervals[expr_in_A.idf]))
>>    self.log.print("\tInput2: scale = %d, interval = [%d, %d]" % (
>>       (self.varScales[expr_in_B.idf],) + self.varIntervals[expr_in_B.idf]))
>>    self.log.print("\tOutput: scale = %d, interval = [%d, %d]" % (
>>       (self.varScales[expr_out.idf],) + self.varIntervals[expr_out.idf]))
>>
>>    return (prog_out, expr_out)
```

STEP 7: Use the operator

After all these steps, one can simply use the operator in the input Shiftry code, and the compiler will handle the rest