**Project: Fingerprinting ML model using timing analysis and side channel**

**Description:**

In this project, we want to use a power side channel attack and convert the ML models running information into timing information. Then, through the timing analysis, we want to detect which ML model was running and what data was being processed.

In order to convert the ML model info to timing information and create a power side-channel attack, we are going to use the frequency throttling technique. Please read the following papers first, to understand the general idea behind these concepts:

1-Frequency Throttling Side-Channel Attack:

<https://dl.acm.org/doi/pdf/10.1145/3548606.3560682>

2-Hertzbleed: Turning Power Side-Channel Attacks Into Remote Timing Attacks on x86:

<https://www.usenix.org/system/files/sec22-wang-yingchen.pdf>

In order to get familiar with fingerprinting of ML models, please read the following paper:

3-SECURITY ANALYSIS OF DEEP NEURAL NETWORKS OPERATING IN THE PRESENCE OF CACHE SIDE-CHANNEL ATTACKS:

<https://arxiv.org/pdf/1810.03487.pdf>

—------------------------------------------------------------------------------------------------------

**Infrastructure:**

**Linux-based server access:**

Please install Team Viewer on your system.

Connect to the server through the following info:

**Software:**

We need to develope both C programs and Python scripts.

How to compile:

Use visual studio code to edit the software. You can compile the C program by opening the C program and then from the” terminal” tab, please select “Run Build Task”. It will compile the code and generate an executable file in the same path.

To compile C, we used clang++-10 and we passed -mavx and -mfma flags to compiler.

Here is the json file for vs code compiler:

VS code task for compiling C code:

{

"version": "2.0.0",

"tasks": [

{

"type": "cppbuild",

"label": "C/C++: clang++-10 build active file",

"command": "/usr/bin/clang++-10",

"args": [

"-fcolor-diagnostics",

"-fansi-escape-codes",

"-g",

"-mavx",

"-mfma",

"${file}",

"-o",

"${fileDirname}/${fileBasenameNoExtension}"

],

"options": {

"cwd": "${fileDirname}"

},

"problemMatcher": [

"$gcc"

],

"group": "build",

"detail": "compiler: /usr/bin/clang++-10"

}

]

}

orchestrator.cpp:

The C program manages the whole operation. It uses a pipe file to synchronize the execution of AVX instructions with the Python program. Then it forks two separate processes. Each process will be assigned to a specific core.

The parent process executes 10000 AVX instructions 20000 times in a For loop and saves the execution time of each loop in a text file that will be later used for timing analysis. In order to synchronize the parent process with the child process, the C program uses shared memory.

The child process executes the Python program and then finishes.

ml\_pipeline.py:

This Python script uses a pipe file to sync with a C program.

it will download pretrain models and save them for later use.

regression.py:

This Python script runs a regression to run all models with all kernels for the number times set by user.

Ts\_processor.py:

This script will be run when all the data is collected. Then it can be used to parse the dataset, prepare and clean features, then train a classifier. After the classification, it can ne used to evaluate the model.

$ python3 ts\_processor.py train mm reload

$ python3 ts\_processor.py train hif mm reload

$ python3 ts\_processor.py mm reload

$ python3 ts\_processor.py mm reload unknown

**—------------------------------------------------------------------------------------**

**Appendix 1:**

**Changing package power limit for frequency throttling:**

Learning link:

<https://gist.github.com/Mnkai/5a8edd34bd949199224b33bd90b8c3d4>

Commands:

If “msr” module is not loaded into the Linux kernel (if the system gets reset, this must be reloaded!)Please first load it into the kernel with the below command. It may require sudo access.

$ modprobe msr

In order to check what the current value of the package power limit is, you can read the register with the address “0x610” by “rdmsr” command. It requires sudo access to run the command.

$ sudo rdmsr 0x610

The output of reading would be similar to this:

38493005a8**3d0 =122W**

avxThe first three hex values bolded above show the current limit. You can convert that value to Watt as follow:

0x3d0 = 976 -> If we divide it by 8 (power unit), we get the Watt value = 976/8 = 122 Watt

In order to set a new power limit, you should write the new value to address “0x610” by “wrmsr” command. Please make sure only to change the first 3 hex values and keep the others the same, similar below:

For example, if we want to set the package power limit to 23 watts, then we have to multiply it by 8 which gives us 184 and the hex value for that is 0xb8. Since we need to modify three hex value, we should write 0x0b8 to that register as follow:

$ sudo wrmsr 0x610 0x38493005a8**0b8**

**—----------------------------------------------------------------**

**Appendix B:**

**Monitoring CPU frequency and system status:**

To see the online CPU frequency of each core, you can run the following command:

$ watch grep \"cpu MHz\" /proc/cpuinfo

To see the system status, run the following command, it shows the load on each CPU and memory usage, and active process:

$ htop

If you want to exit, you can push: Ctrl+c

**# if frequency does not change, follow this:**

https://www.cnx-software.com/2022/09/08/how-to-check-tdp-pl1-and-pl2-power-limits-in-windows-and-linux/

To see system current status:

$powercap-info -p intel-rapl

To see runtime power consumption status:

$sudo powerstat -R -D

To set new power limit:

$sudo powercap-set -p intel-rapl -z 0 -c 1 -l 15000000

***Set ubuntu on max frequency:***

sudo apt-get update

sudo apt-get install linux-tools-generic linux-tools-$(uname -r)

sudo cpupower frequency-set -g performance

Then use this to see results:

cat /sys/devices/system/cpu/cpu\*/cpufreq/scaling\_governor

—------------------------------------------------------------