ONNX Profiling and Optimizations

# Objectives

Static Profiling - model architecture, memory per layer, computations

Dynamic profiling - runtime profile - memory usage, number of accesses and the CPU usage

# Current tools

Visualization tool - <https://github.com/lutzroeder/netron>

ONNX Profiler - [Tune performance | onnxruntime](https://onnxruntime.ai/docs/performance/tune-performance.html)

[onnxruntime/profiler.py at main · microsoft/onnxruntime (github.com)](https://github.com/microsoft/onnxruntime/blob/main/onnxruntime/python/tools/transformers/profiler.py)

Static Profiler - MLPERF

<https://www.nvidia.com/en-us/data-center/resources/mlperf-benchmarks/>

<https://github.com/mlperf>

# **VESPA: Static profiling for binary optimization**

VESPA is useful for obtaining profile information to feed binary optimizers like BOLT statically, i.e., with no need to execute the target application to produce profile data. To achieve this, VESPA employs machine learning techniques. First, during a training phase, VESPA is provided with a set of applications and corresponding dynamic profiles. Using these, VESPA trains a neural network model that learns the probability that branch instructions in the programs will be taken based on various program characteristics (e.g., the condition code of the branch or whether the target block is a loop header).

<https://engineering.fb.com/2022/03/15/developer-tools/vespa/>

Dynamic Profiler - MLCOMMONS

<https://mlcommons.org/en/>

OLIVE  
OLive, meaning ONNX Runtime(ORT) Go Live, is a python package that automates the process of accelerating models with [ONNX Runtime(ORT)](https://onnxruntime.ai/). It contains two parts including model conversion to [ONNX](https://onnx.ai/) with correctness checking and auto performance tuning with ORT. Users can run these two together through a single pipeline or run them independently as needed.

<https://github.com/microsoft/OLive>

## Netron (Ahmed to fill in eval)

Models to profile

LLM

https://huggingface.co/Intel/gpt-j-6B-int8-static

Netron is a viewer for neural network, deep learning, and machine learning models.

Netron supports ONNX, TensorFlow Lite, Keras, Caffe, Darknet, ncnn, MNN, PaddlePaddle, Core ML, MXNet, RKNN, MindSpore Lite, TNN, Barracuda, Tengine, TensorFlow.js, Caffe2 and UFF. Netron has experimental support for PyTorch, TensorFlow, TorchScript, OpenVINO, Torch, Vitis AI, Arm NN, BigDL, Chainer, CNTK, Deeplearning4j, MediaPipe, MegEngine, ML.NET and scikit-learn.

Netron woks by the folowing methods:

1- Install netron visualizer program

2- Netron also work in the browser and you can load your own model to visualize on the web page.

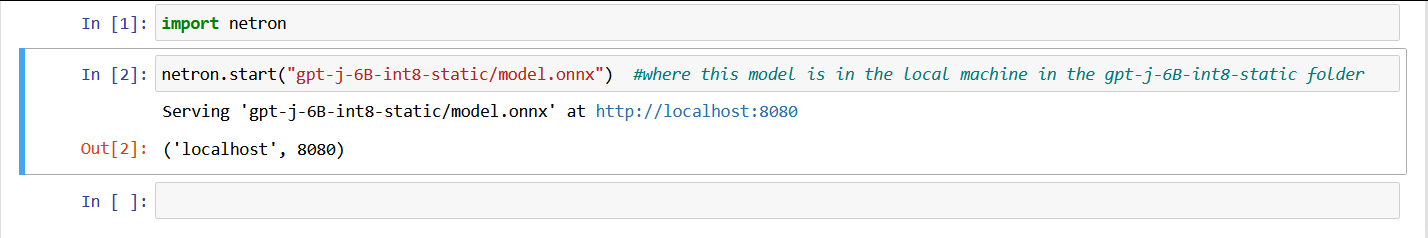
3- Write python code to install then use it in the coding program.

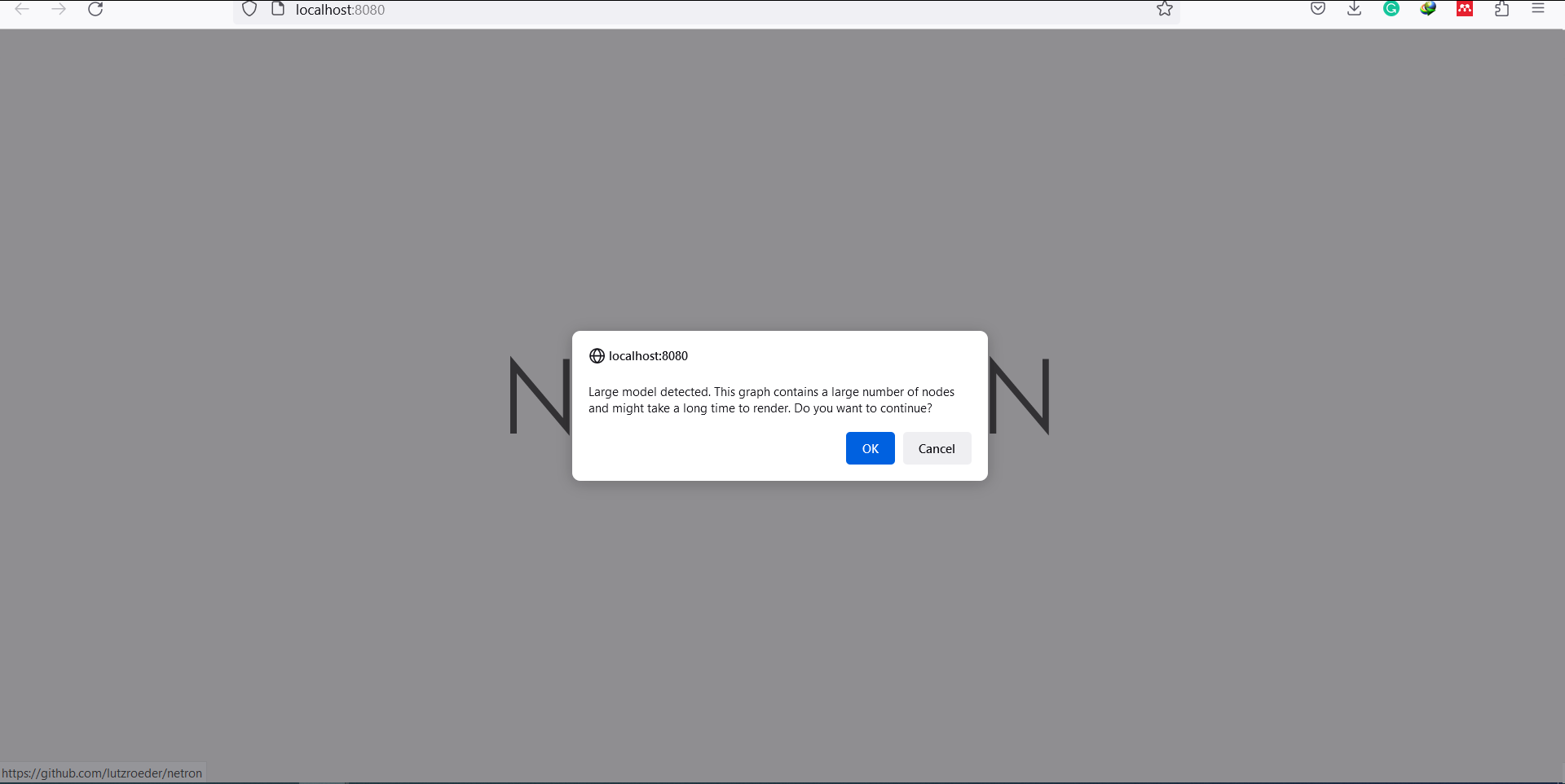
pip install netron

Import netron

netron.start(“model path on the directory”)

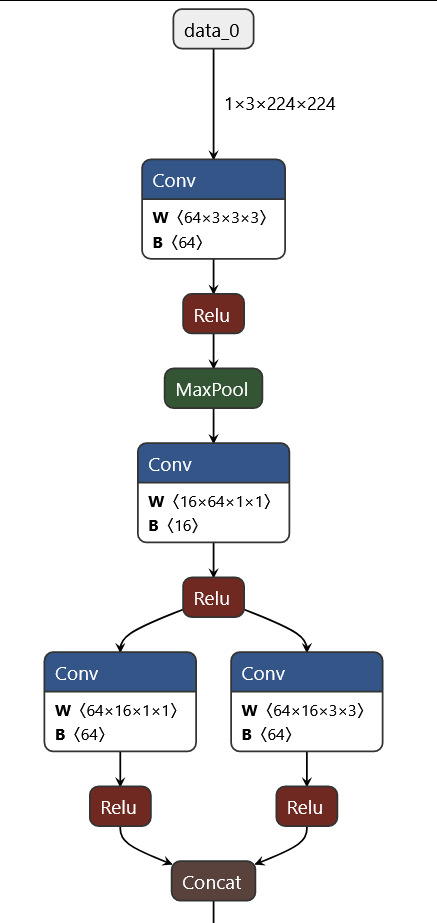
– Then open the visualized model on localhost

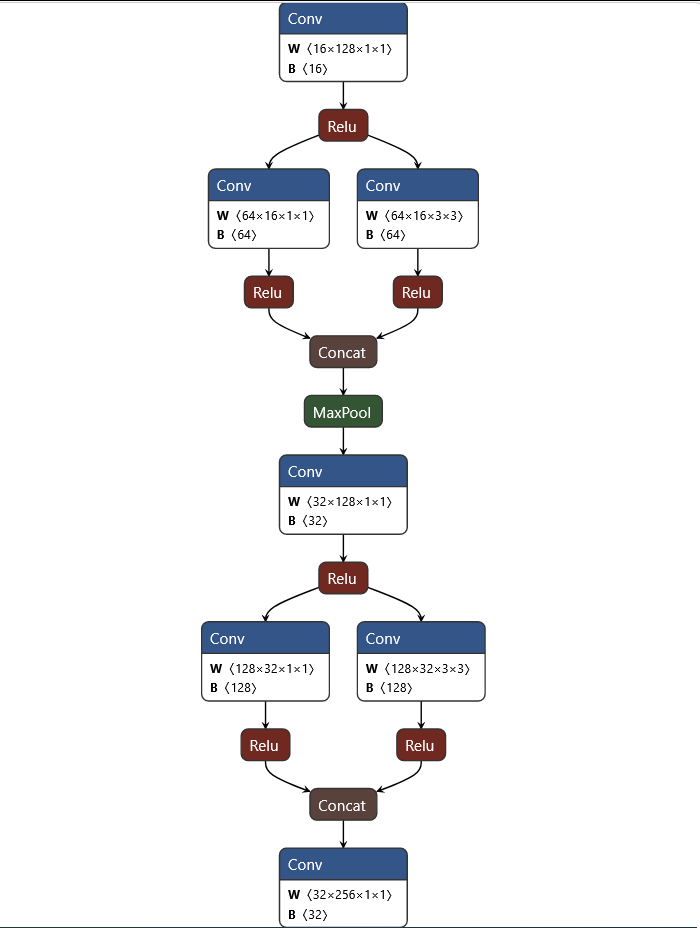


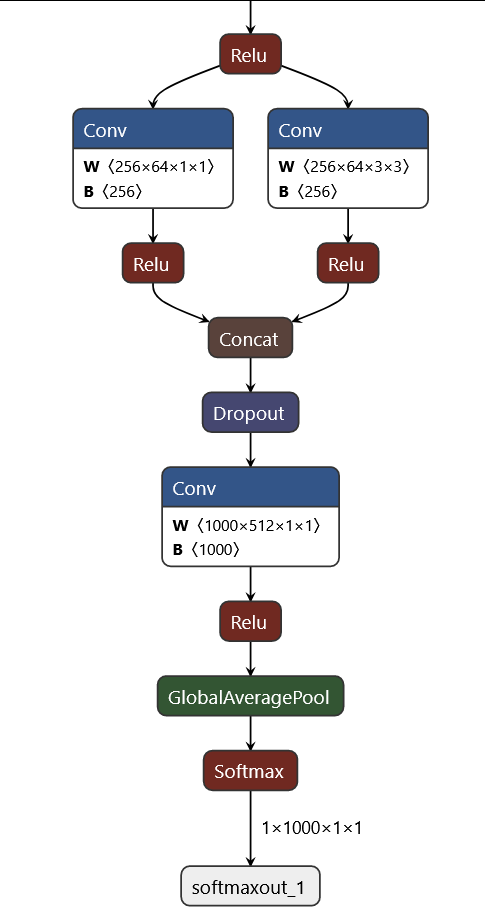












# 

# Profiling

In software engineering, profiling ("program profiling", "software profiling") is a form of dynamic program analysis that measures, for example, the space (memory) or time complexity of a program, the usage of particular instructions, or the frequency and duration of function calls. Most commonly, profiling information serves to aid program optimization, and more specifically, performance engineering.

Profiling is achieved by instrumenting either the program source code or its binary executable form using a tool called a profiler (or code profiler). Profilers may use a number of different techniques, such as event-based, statistical, instrumented, and simulation methods.

* FlameGraph

<https://github.com/brendangregg/FlameGraph>

<https://www.brendangregg.com/flamegraphs.html>

* PyTorch Profiler

<https://pytorch.org/tutorials/recipes/recipes/profiler_recipe.html#pytorch-profiler>

<https://pytorch.org/tutorials/intermediate/tensorboard_profiler_tutorial.html>

* Nividia DLProf

<https://docs.nvidia.com/deeplearning/frameworks/dlprof-user-guide/index.html>

* PyProf

<https://developer.nvidia.com/blog/profiling-and-optimizing-deep-neural-networks-with-dlprof-and-pyprof/>

* TensorBoard

<https://medium.com/analytics-vidhya/a-comprehensive-guide-for-profiling-a-deep-learning-model-37543100c0aa>

* TensorFlow Profiler

<https://www.tensorflow.org/tensorboard/tensorboard_profiling_keras>

<https://github.com/tensorflow/profiler>

<https://wiki.ncsa.illinois.edu/display/ISL20/Profile+Tensorflow+using+Tensorboard>

* Profiling JAX program

<https://jax.readthedocs.io/en/latest/profiling.html>

* Optimize TensorFlow Performance using the Profiler

<https://www.tensorflow.org/guide/profiler>

Ahmed Google Colab Notebook

<https://colab.research.google.com/drive/1AIMCEhh_cymtrnrN_n633hUqiwYYb-a2?usp=sharing>

# **CProfile**

<https://docs.python.org/3/library/profile.html>

# **Pytorch\_memlab**

A library for memory profiling. Uses torch.cuda.memory\_stats() inside.

<https://github.com/Stonesjtu/pytorch_memlab>

There are memory profiler, and memory reporter

<https://github.com/lexi-int/hse_pytorch_memlab_example>

!pip install torch transformers pytorch\_memlab

Memory Reporter

import torch

from pytorch\_memlab import LineProfiler, MemReporter, profile

from transformers import BertForTokenClassification, BertTokenizerFast

device = torch.device('cuda:0') if torch.cuda.is\_available() else torch.device('cpu')

## BERT Model

model = BertForTokenClassification.from\_pretrained(

'bert-base-cased',

num\_labels=10.cuda()

tokenizer = BertTokenizerFast.from\_pretrained('bert-base-cased')

## GPT-3

from transformers import AutoTokenizer, AutoModelForCausalLM

tokenizer = AutoTokenizer.from\_pretrained("minhtoan/gpt3-small-finetune-cnndaily-news")

model = AutoModelForCausalLM.from\_pretrained("minhtoan/gpt3-small-finetune-cnndaily-news").cuda()

reporter = MemReporter(model)

reporter.report(device=device, verbose=True)

Ahmed Notebook

<https://colab.research.google.com/drive/1AJsUjFJ5CEGrZQZ9jfBqt_juaZ0buAMI?usp=sharing>

**GPU Profiler**

TorchSpy

<https://github.com/simon-schaefer/torchspy>

PyProf

<https://github.com/NVIDIA/PyProf>

# **Exporting a Model from PyTorch to ONNX and Running it using ONNX Runtime**

<https://pytorch.org/tutorials/advanced/super_resolution_with_onnxruntime.html>

# 

# **Scalene profiler**

<https://github.com/plasma-umass/scalene>

<https://towardsdatascience.com/how-much-memory-is-your-ml-code-consuming-98df64074c8f>

# 

# **PyTorch Profiler With TensorBoard**

<https://pytorch.org/tutorials/intermediate/tensorboard_profiler_tutorial.html>

# **Torchprof**

<https://github.com/awwong1/torchprof>

# **Skyline**

<https://github.com/skylineprof/skyline>

# **Octoml-profile**

<https://github.com/octoml/octoml-profile>

<https://medium.com/pytorch/profiling-pytorch-language-models-with-octoml-profile-eda7ece6b7bd>

# **Microsoft DeepSpeed**

<https://www.deepspeed.ai/>

<https://github.com/microsoft/DeepSpeed/tree/master>

# **Flops Profiler from DeepSpeed**

<https://www.deepspeed.ai/tutorials/flops-profiler/>

<https://deepspeed.readthedocs.io/en/stable/flops-profiler.html>

# **Optimum Inference with ONNX Runtime**

<https://huggingface.co/docs/optimum/onnxruntime/usage_guides/models>

# **Resources**

<https://huggingface.co/docs/transformers/v4.19.2/en/performance>

<https://huggingface.co/docs/transformers/v4.19.2/en/benchmarks>

<https://github.com/huggingface/transformers/blob/main/examples/pytorch/benchmarking/README.md>

<https://github.com/gmlunesa/profilegen>

<https://github.com/ReidAtcheson/profiling_gpt2>

<https://github.com/microsoft/onnxruntime-training-examples>

<https://github.com/microsoft/DeepSpeedExamples/tree/master/benchmarks/communication>

<https://github.com/pytorch/kineto/tree/main/tb_plugin>