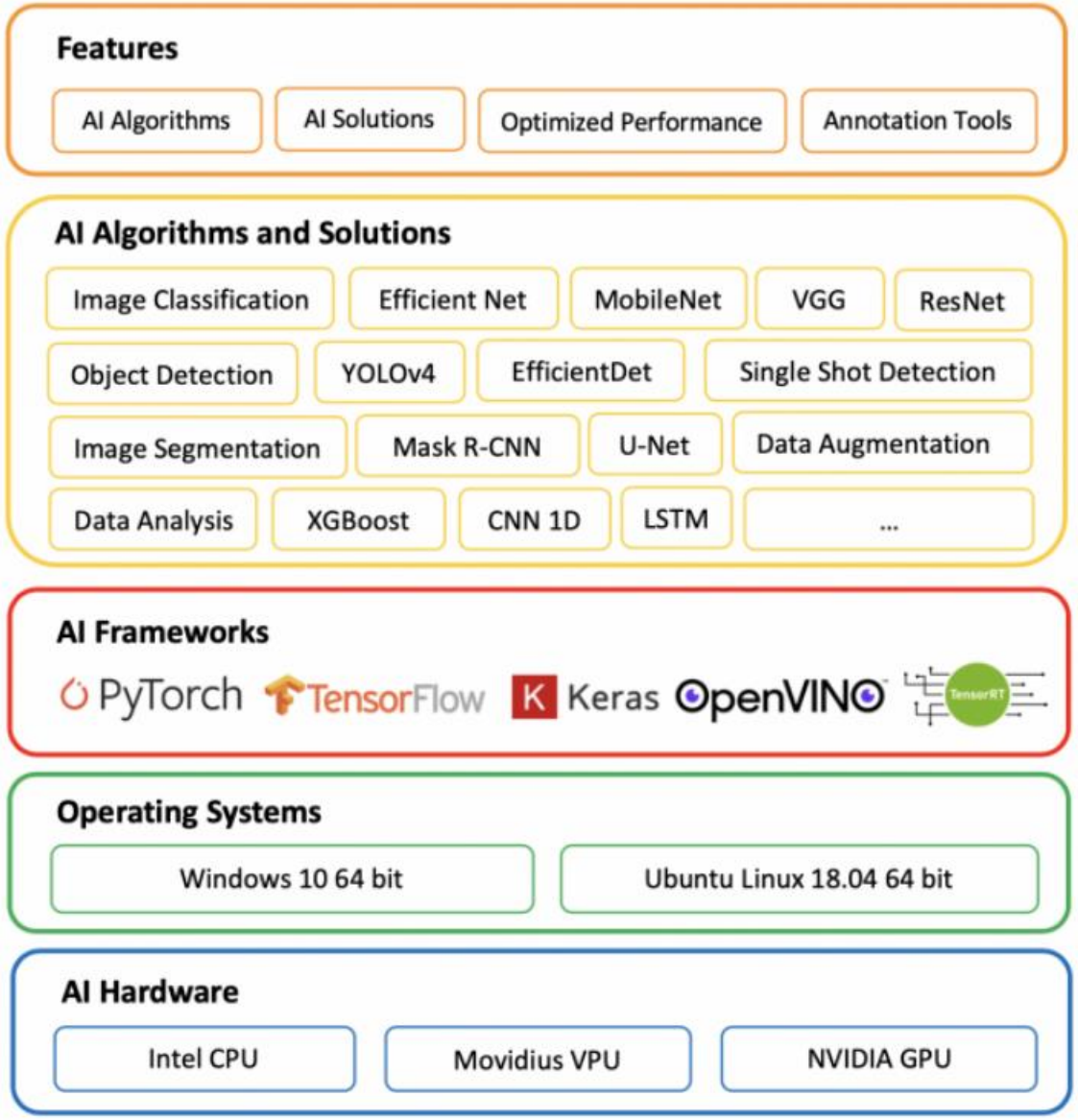
AI System architecture.

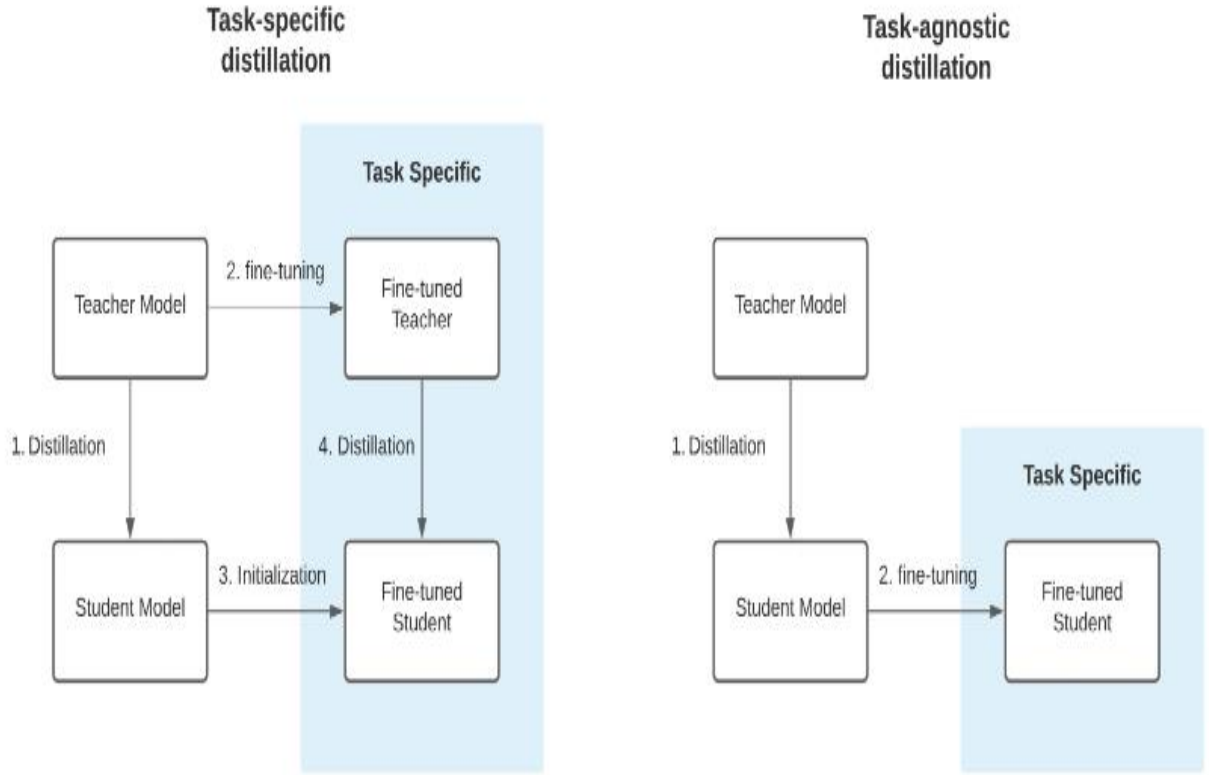


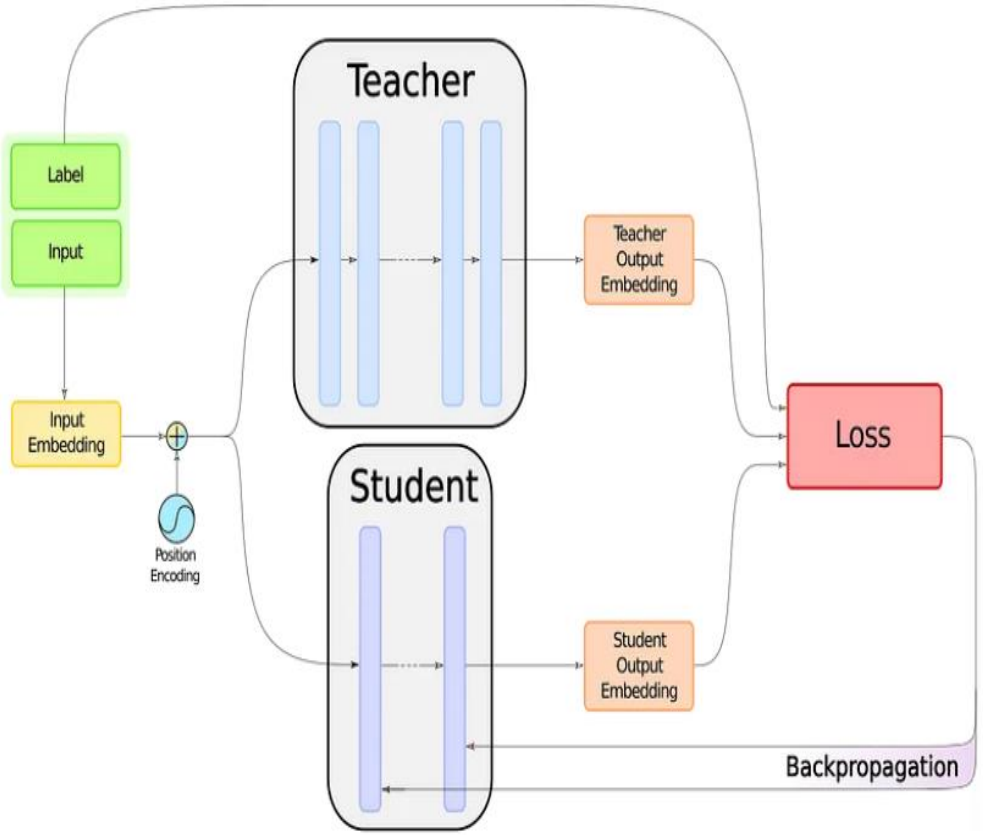
Optimizing a HuggingFace Transformer Model

- Optimization 1: Distillation
- Optimization 2: Quantization
- Optimization 3: ONNX Runtime

Optimization 1:Knowledge distillation

In machine learning, knowledge distillation is the process of transferring knowledge from a large model to a smaller one. While large models have higher knowledge capacity than small models, this capacity might not be fully utilized





Result

model	Parameter	Speed-up	Accuracy
BERT-base	109M	1x	93.2%
tiny-BERT	4M	46.5x	83.4%

Optimization 2: Quantization

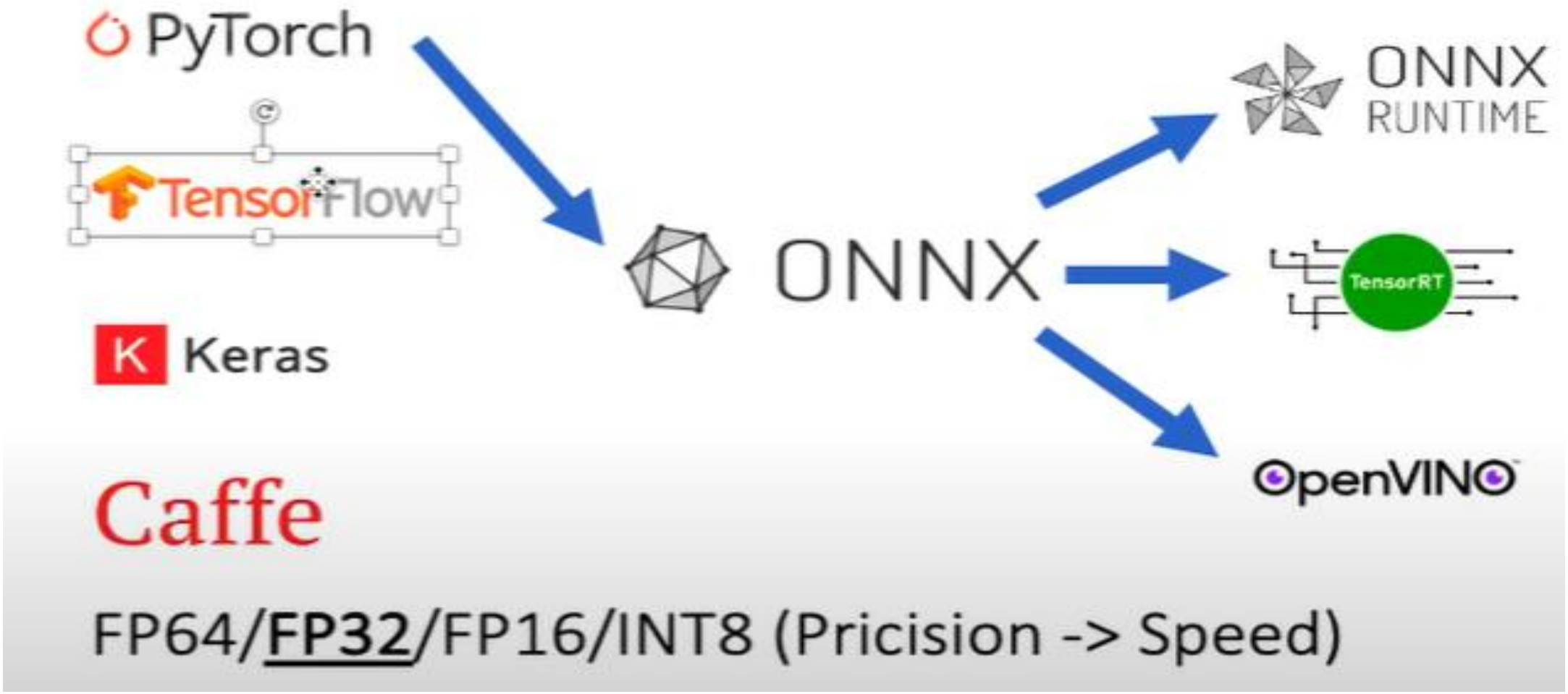
During training, most neural network weights are stored as 32-bit or even 64-bit floating point numbers. This is way more precision than we actually need. We can save considerable space and speed up execution by getting rid of some of these unnecessary decimal places. This process is known as quantization.

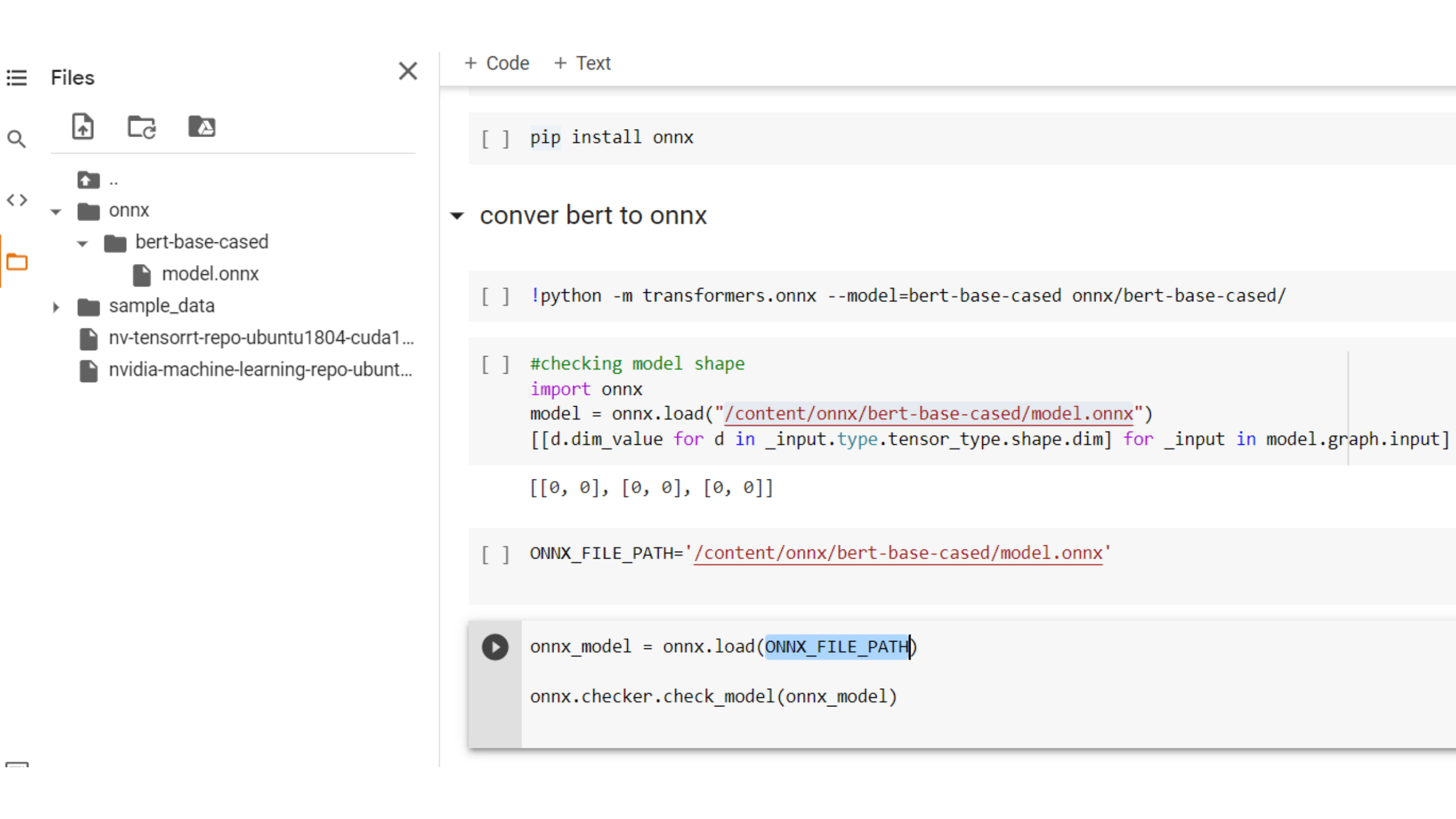
https://pytorch.org/tutorials/intermediate/dynamic_quantization_bert_tutorial.html

Prec	F1 score	Model Size	1 thread	4 threads
FP32	0.9019	438 MB	160 sec	85 sec
INT8	0.8953	181 MB	90 sec	46 sec

```
================================================================================
Model Sizes
================================================================================
FP32 Model Size: 411.00 MB
INT8 Model Size: 168.05 MB
================================================================================
BERT QA Example
================================================================================
Text:
According to PolitiFact the top 400 richest Americans "have more wealth tha
Question:
What publication printed that the wealthiest 1% have more money than those
Model Answer:
New York Times
Dynamic Quantized Model Answer:
New York Times
================================================================================
BERT QA Inference Latencies
================================================================================
CPU Inference Latency: 499.82 ms / sample
Dynamic Quantized CPU Inference Latency: 387.61 ms / sample
CUDA Inference Latency: 31.84 ms / sample
```

Optimization 3: ONNX Runtime




Files
..
onnx
bert-base-cased
model.onnx
sample_data
nv-tensorrt-repo-ubuntu1804-cuda1...
nvidia-machine-learning-repo-ubunt...
+ Code
+ Text
[ ] pip install onnx
conver bert to onnx
[ ] !python -m transformers.onnx --model=bert-base-cased onnx/bert-base-cased/
[ ] #checking model shape
import onnx
model = onnx.load("/content/onnx/bert-base-cased/model.onnx")
[[d.dim_value for d in _input.type.tensor_type.shape.dim] for _input in model.graph.input]
[[0, 0], [0, 0], [0, 0]]
[ ] ONNX_FILE_PATH='/content/onnx/bert-base-cased/model.onnx'
onnx_model = onnx.load(ONNX_FILE_PATH)
onnx.checker.check_model(onnx_model)

TensorRT Optimizations

- https://developer.nvidia.com/blog/tensorrt-3-faster-tensorflow-inference/

- https://rasa.com/blog/compressing-bert-for-faster-prediction-2/

1. Layer and tensor fusion and elimination of unused layers;
2. FP16 and INT8 reduced precision calibration;
3. Target-specific autotuning;
4. Efficient memory reuse