tract

an efficient rust neural network inference engine

&

Shipping neural networks with

torch-to-nnef

https://tinyurl.com/tract-rs

slides

FOSDEM'26

Overview

- Mathieu and Julien, ML software engineers for Sonos Voice Control
- Introducing two open-source libraries
  - tract, a Rust generic neural network inference library
  - torch-to-nnef, a companion project to export PyTorch model in tract's preferred format

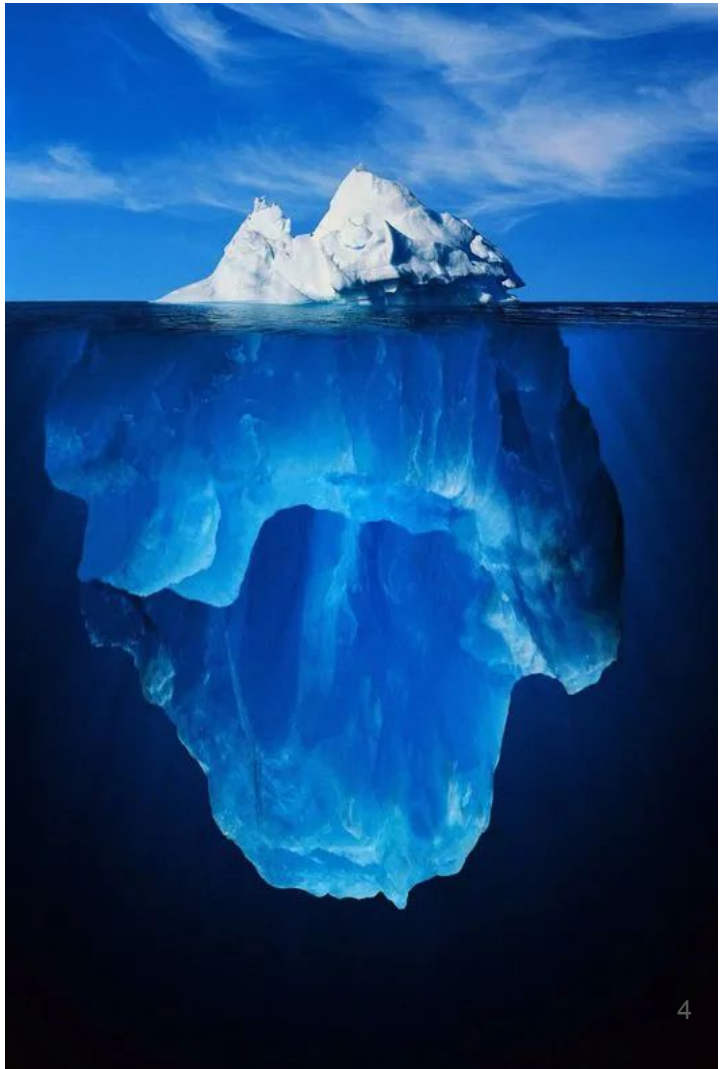
tract

an efficient rust neural network inference engine

15min

Scratching the surface

- Neural Network inference library
- “tract all the way down”: full stack, from model parsing to optimized kernels
- Have been around for 9 years or so
- In Rust, integration friendly, Apache2/MIT

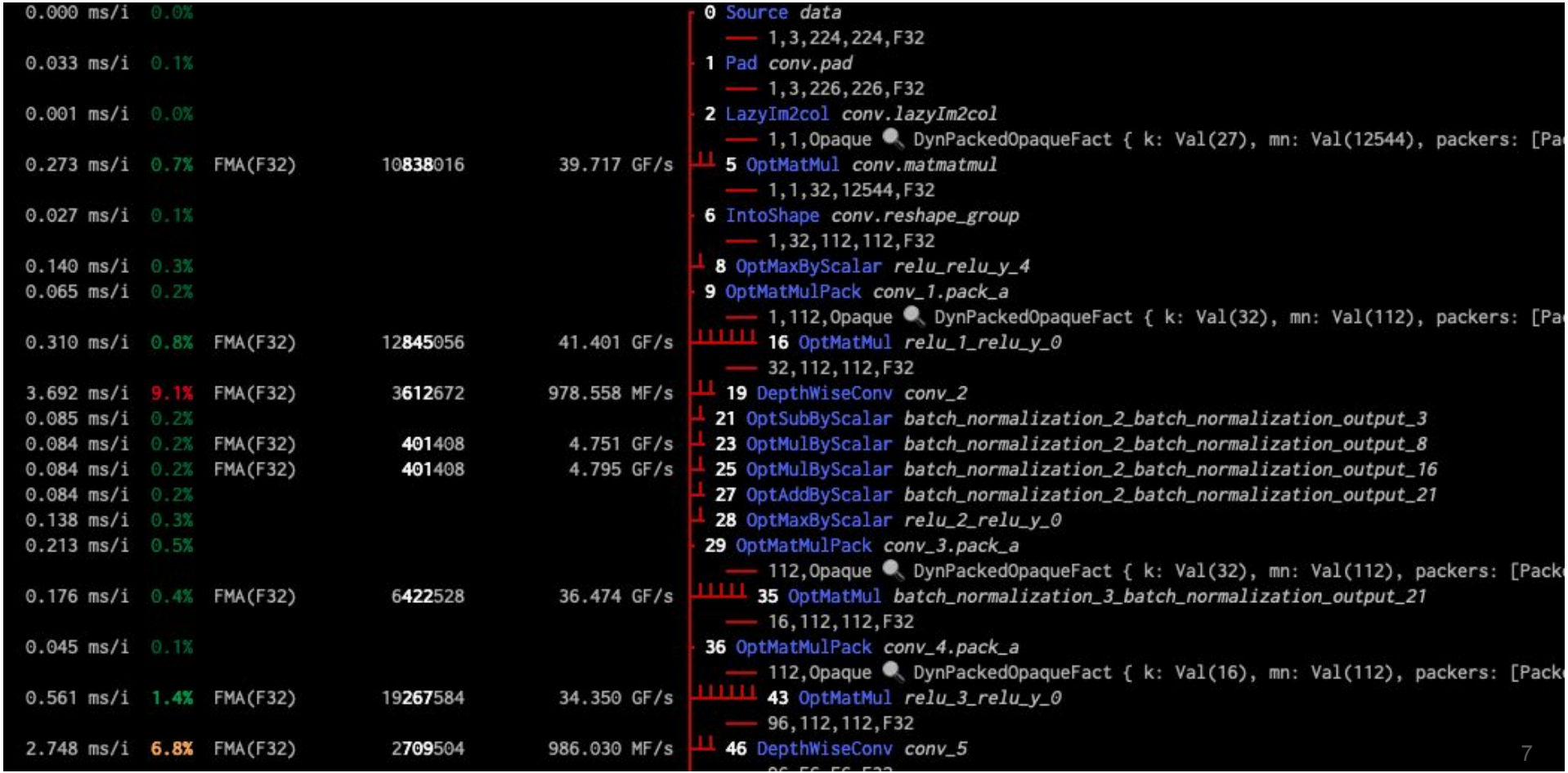


```rust
fn main() -> Result<()> {
    let nnef = tract_rs::nnef()?;
    let model = nnef.load("mobilenet_v2_1.0.onnx.nnef.tgz")?.into_runnable()?;

    // open image, resize it and make a Tensor out of it
    let image = image::open("grace_hopper.jpg").unwrap().to_rgb8();
    let resized =
        image::imageops::resize(&image, 224, 224, ::image::imageops::FilterType::Triangle);
    let input = tract_ndarray::Array4::from_shape_fn((1, 3, 224, 224), |(_, c, y, x)| {
        let mean = [0.485, 0.456, 0.406][c];
        let std = [0.229, 0.224, 0.225][c];
        (resized[(x as _, y as _)][c] as f32 / 255.0 - mean) / std
    });

    // run the model on the input
    let result: Vec<Value> = model.run([input])?;

    // find and display the max value with its index
    let best =
        &result[0].as_slice::<f32>()?.iter().zip(2..).max_by(|a, b| a.0.partial_cmp(b.0).unwrap());
    println!("result: {best:?}");
    Ok(())
}
```

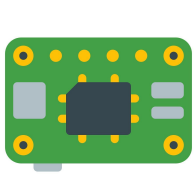
tract mobilenet_v2_1.0.nnef.tgz

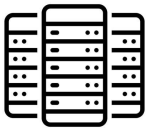
```
0 Source data
   ── 1,3,224,224,F32
3 Conv conv
   * Data format: NCHW
   * Kernel shape:[3, 3] (strides:Some([2, 2]), padding:Explicit([1, 1], [1, 1]), dilations:Some([1, 1]))
   * Kernel OIHW (groups:1)
   ── 1,32,112,112,F32
5 Max relu_relu_y_4
7 EinSum conv_1
   * NIHW,OI->OHW (F32)
   ── 32,112,112,F32
9 Sub batch_normalization_1_batch_normalization_output_3
11 Mul batch_normalization_1_batch_normalization_output_8
13 Mul batch_normalization_1_batch_normalization_output_16
```

tract mobilenet_v2_1.0.nnef.tgz -O dump --cost --profile -R

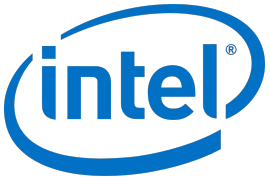


Easy to integrate



- No cumbersome third party dependency
- Easy to cross compile (Rust…)
- WASM support (run your model in a browser)
- Optimized for ARM32, ARM64, X64, WASM SIMD, metal, cuda.
- C API
- Python bindings: pip install tract







GPU support

```
let nnef = tract_rs::nnef()?;
let model = nnef.load("mobilenet_v2_1.0.onnx.nnef.tgz")?.into_runnable()?;
```

```
tract mobilenet_v2_1.0.nnef.tgz -O bench
```

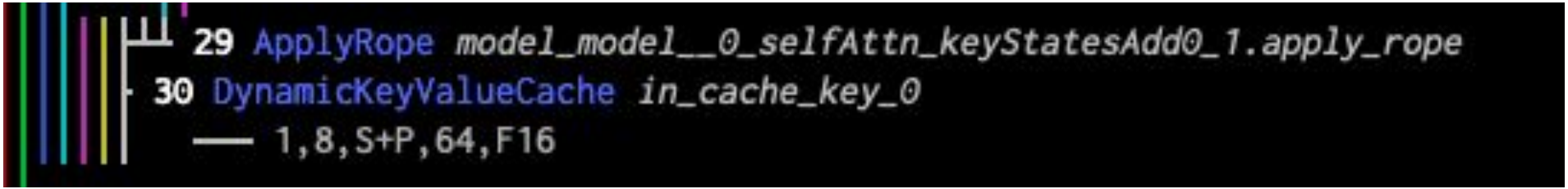
```
Bench ran 127 times, 39.493 ms/i.
```

```
let nnef = tract_rs::nnef()?;
let model = nnef.load("mobilenet_v2_1.0.onnx.nnef.tgz")?;
let model = runtime_for_name("cuda")?.prepare(model)?;
```

```
tract mobilenet_v2_1.0.nnef.tgz --cuda bench
```

```
Bench ran 2259 times, 2.204 ms/i.
```

```
tract --llm --opl Llama-3.1-8B-Instruct-f16f16.nnef.tgz
```


29 ApplyRope model_model__0_selfAttn_keyStatesAdd0_1.apply_rope
30 DynamicKeyValueCache in_cache_key_0
1,8,S+P,64,F16

ONNX models support

- ONNX is first class citizen.
- tract strong tensor typing : for each tensor we want
  - a fixed rank
  - all dimension known (symbolic expression allowed)
- ONNX protobuf format does not contain detailed tensor shapes
- Two-stage loading: load, add some shape hints, tract will infer the rest
- Still supporting symbolic shape

```rust
#[test]
fn test_inference_model() -> anyhow::Result<()> {
    ensure_models()?;
    let mut model = onnx()?.load("mobilenetv2-7.onnx")?;
    model.set_input_fact(0, "1,3,224,224,F32")?;
    let model = model.into_tract()?.into_runnable()?;
    let hopper = grace_hopper();
    let result = model.run([hopper])?;
    let view = result[0].view::<f32>()?;
    let best = view
        .as_slice()
        .unwrap()
        .iter()
        .enumerate()
        .max_by(|a, b| a.1.partial_cmp(b.1).unwrap())
        .unwrap();
    assert_eq!(best.0, 652);
    Ok(())
```

LLMs support and torch-to-nnef

- The LLM tsunami happened with specialized engines
- tract supports LLMs, but as a generic engine
  - Models structures are not hard-coded in tract
  - Instead, tract has an SDPA operator (and a couple other dedicated ones)
  - tract needs to be given the graph description in the input format: NNEF (or ONNX)
- ONNX is a bit awkward with LLM (getting better lately)

- This lead us to open-source torch-to-nnef last summer.

tinyurl.com/torch-to-nnef

docs

Shipping neural networks with

torch-to-nnef

A highly tract compatible exporter

5min

What composes a neural network model asset ?

A **neural network asset** is the packaged output of training that is designed for efficient inference. It typically contains two essential components:

A list of **tensors** with names and values (the parameters learned during the training process).

A computation graph stitching the tensors together: inputs, outputs, data types, tensor shapes, and the sequence of operators.

What is NNEF ?

NNEF stands for **N**eural **N**etwork **E**xchange **F**ormat.

It addresses the same core problem as  ONNX but focusing on inference.

Specified by the KHRONOS GROUP (consortium of ~170 companies)

Interested readers can dive into the specifications of this format here.

Why NNEF is interesting ?

Readable Graph Structure

The main `.nnef` file represents the model graph in a **simple, declarative, text-based format**

Composition

Neural-network are made of repetition of blocks, the text format promotes **reusability**, avoids repetition, and enables a **clean functional structure**.

Extensible Tensors & Quantization Logic

- Quantization metadata can live in a **separate `.quant` file**
- **Flexible Tensor Storage**
  Each tensor is stored as a binary `.dat` blob.

NNEF syntax example

```
fragment gelu_fast_approx( x: tensor<scalar> ) -> ( y: tensor<scalar> )
{
    y = 0.5 * x * (1.0 + tanh(x * 0.7978845608 * (1.0 + 0.044715 * x * x)));
}

graph network(input_0) -> (output_0)
{
    input_0 = tract_core_external(shape = [1, 2], datum_type = 'f32');
    lin_weight = variable<scalar>(label = 'lin.weight', shape = [2, 2]);
    lin_bias = variable<scalar>(label = 'lin.bias', shape = [2]);
    lin_bias_aligned_rank_expanded = unsqueeze(lin_bias, axes = [0]);
    lin__x_linear0 = linear(input_0, lin_weight, lin_bias_aligned_rank_expanded);
    gelu0 = gelu_fast_approx(lin__x_linear0);
    lin_bias_aligned_rank_expanded = unsqueeze(lin_bias, axes = [0]);
    lin_2nd_call_x = linear(gelu0, lin_weight, lin_bias_aligned_rank_expanded);
    output_0 = gelu_fast_approx(lin_2nd_call_x);
}
```

What is torch-to-nnef ?

Torch to NNEF is a Python package that:

- was born in early 2022 out of the frustration of ONNX quantization export state and our need back then for an internal project
- Creates a strong bridge between PyTorch and NNEF
- Allows a complete and flexible control to export any model.

pip install torch-to-nnef

Export Python example

```python
from pathlib import Path
from torch_to_nnef import export_model_to_nnef, TractNNEF

export_model_to_nnef(
    # any nn.Module
    model=my_model,
    # tuple of model arguments
    args=inputs_data_sample,
    # filepath to dump NNEF archive
    file_path_export=Path("vit_b_16.nnef.tgz"),
    # inference engine to target
    inference_target=TractNNEF(
        # tract version (to ensure compatible operators)
        version="0.21.13",
        # and correctness of output compared to PyTorch
        # for the provided model and input is performed
        check_io=True,
    ),
    input_names=["input"],
    output_names=["output"],
    # create a debug bundle in case model export work
    # but NNEF fail in tract
    # (either due to load error or precision mismatch)
    debug_bundle_path=Path("./debug.tgz"),
)
```

Why torch-to-nnef ?

- Export advanced quantization/dequantization functions, data-types optimized on PyTorch side seamlessly.

- Optionally control serialization of specific ***nn.Module***

- Provide additional expressions on symbolic dimensions, helping engine to reason about them (ie. Batch dimension being always >0).

- Export signal based networks easily (MFCC, MelFilterBanks, …)

Thanks for listening

Interested to use or contribute ?

github.com/sonos/tract

github.com/sonos/torch-to-nnef

Documentations and past articles:

- Technical documentation tract
- Technical documentation torch-to-nnef
- Sonos tech blog related articles: [1], [2], [3], [4]
- 2019 blog post introducing tract