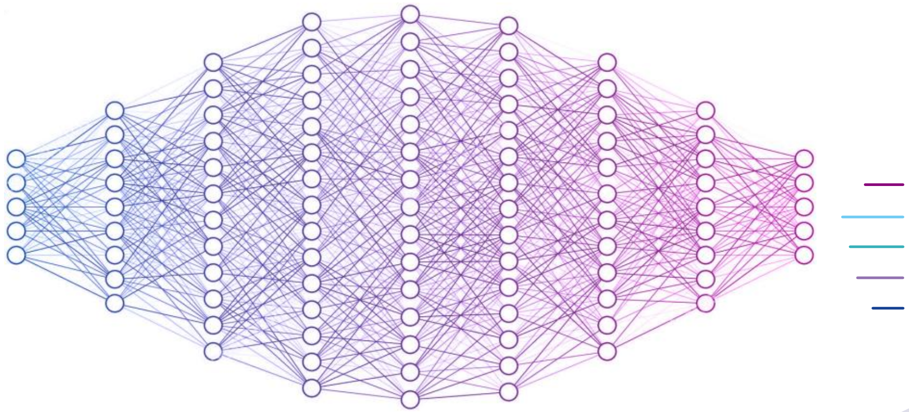
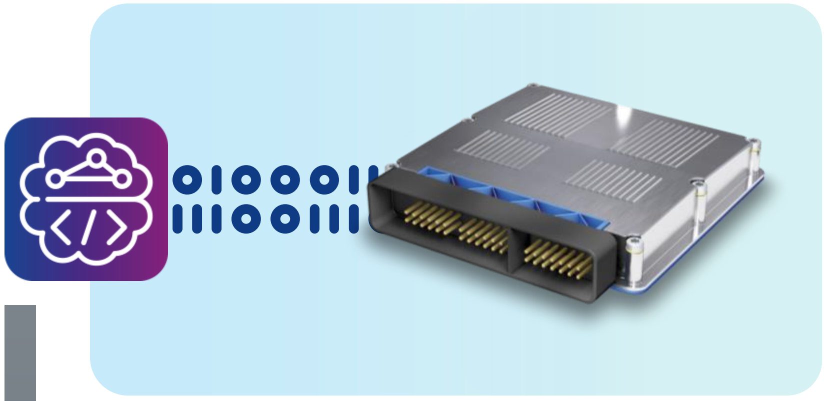
Bridging the gap AI world - embedded world





Virtual sensors
saving cost

BMS with **higher energy efficiency**

AI based ultrasound parking with **higher performance and reduced cost**

Embedded
AI Coder

Embedded AI Coder

Benefits

Generating C-code from trained neural networks for embedded systems

AI without new costly hardware

Enabling **AI on today's microcontrollers & microprocessors**. No need for specialized AI hardware.

Functional safety support

Suitable for **safety-relevant projects** (subject to ISO-26262)

Proven in use

Developed at Bosch and **proven in use** across the Bosch corporation

State-of-the-art resource efficiency

Minimizing resource consumption on **highly constrained** embedded systems

Artificial Intelligence in products

Why AI on Embedded Systems?

Improve efficiency of products

Example: Object height classification in ultrasonic parking systems

Classic solution: Rule-based software
- High complexity
- High maintenance effort

AI solution:
- Significantly better object classification
- Simple maintenance and thus cost savings

Decrease cost of products

Example: Virtual sensors in various actuator systems

Classic solution: Physical sensor
- High product cost
- High mechanical complexity and risk of defects

AI solution:
- Significantly reduced hardware cost
- Reduced maintenance cost

Enable new product innovation

Example: Predictive maintenance

Classic solution: None

AI solution:
- Predict end-of-life of industrial machine and avoid production line outage

Embedded AI holds significant business potential in a variety of applications

Embedded AI Coder

Reduce effort to deploy AI code in series

Safe code

- Numerically correct behavior of generated code
- Memory safety
- MISRA compliance

Automatic verification

- Integrated test suite for automated testing of generated code
- Provisioning of test report and safety manual for qualification

Reparameterization

- Build once and calibrate easily for multiple variants
- Supports known automotive formats like DCM

Embedded AI Coder

Universal code generator

Works on microcontrollers and microprocessors from any vendor

Embedded AI Coder generated code can be deployed to any device

Hardware-agnostic optimizations yielding strong performance

Network sized range from hundreds to millions of parameters

Optimizations for vendor-specific architectures available

Multiple vendor-specific architectures supported
- ARM Cortex-M
- ARM Cortex-A
- Synopsys DSPs
- Renesas

Further vendor-specific optimizations possible on demand

Useful applications of neural networks start at ~200 trained parameters with ~600 byte of RAM

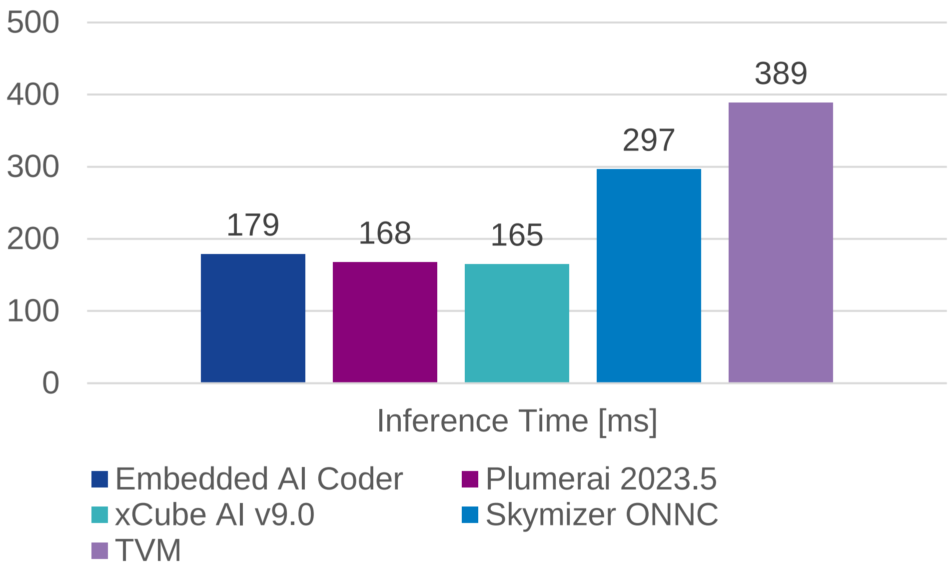
Embedded AI Coder

High performance & resource efficiency

State-of-the-art performance

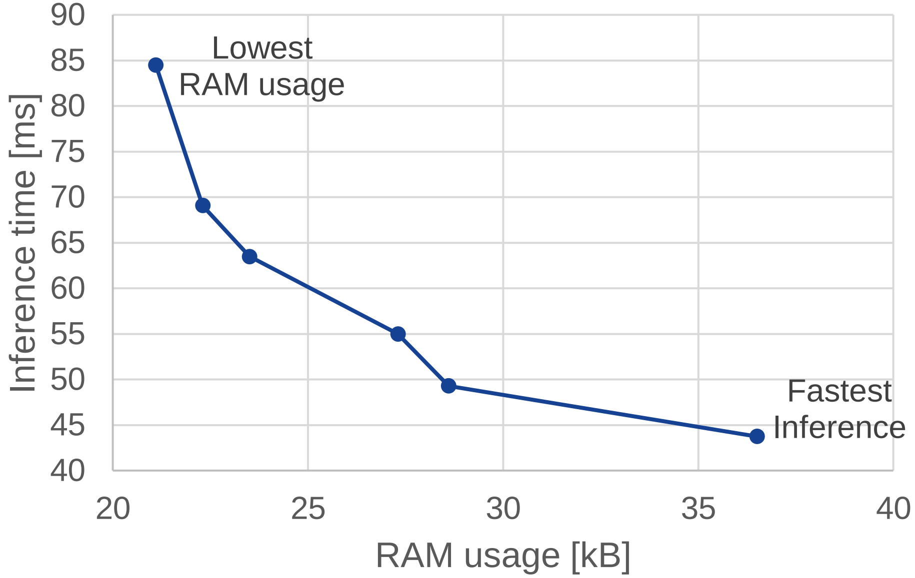
On par performance with other commercial tools and hardware-vendor specific solutions

Significantly higher performance than open-source tools



Flexible Trade-offs for Memory and Runtime

Choose from various implementation variants with different trade-offs between RAM usage and inference speed



All results available on MLCommons® (consortium that publishes AI benchmarks)

Thoughts on SONNX

Embedded AI Coder

Tools for automotive software development need to be reproducible for 8-15 years

Tensorflow <=2.12	Tensorflow 2.15
Critical vulnerabilities in the code base	Implementation of quantized LSTMs changed

Tensorflow 2.18	Tensorflow Lite Micro
Fully Connected uses per-channel quantization	Numerical differences to tensorflow lite, especially with quantized networks

Embedded AI Coder @ SONNX

Float32 vs float64 discrepancies between tools

Perfect agreement between tools is very hard to achieve

Varying float precision in ML tools

```
TfLiteStatus GetQuantizedConvolutionMultipler(TfLiteContext* context,
                                              const TfLiteTensor* input,
                                              const TfLiteTensor* filter,
                                              TfLiteTensor* output,
                                              double* multiplier) {
  const double input_product_scale =
      static_cast<double>(input->params.scale * filter->params.scale);
  TF_LITE_ENSURE(context, input_product_scale >= 0);
  *multiplier = input_product_scale / static_cast<double>(output->params.scale);

  return kTfLiteOk;
}
```

tflite-micro/tensorflow/lite/kernels/kernel_util.cc

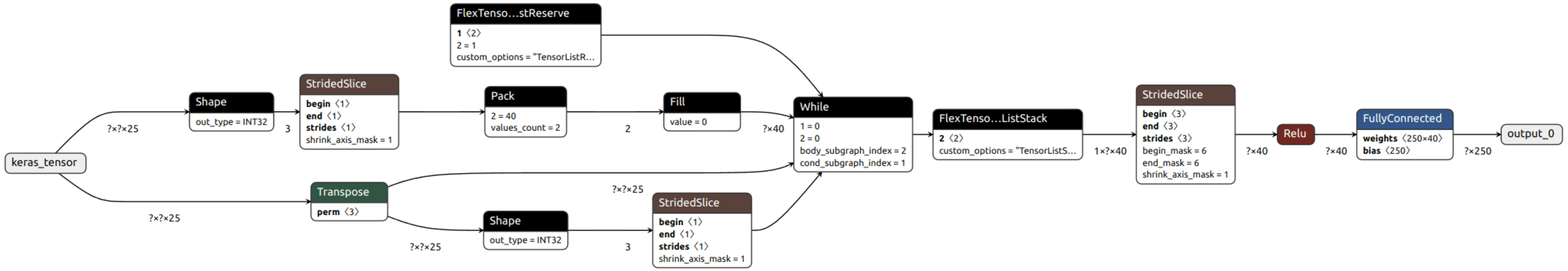
Multiplication is done in float32 precision and then cast to float64

EmbeddedAICoder casts to float64 (python) first and then multiplies

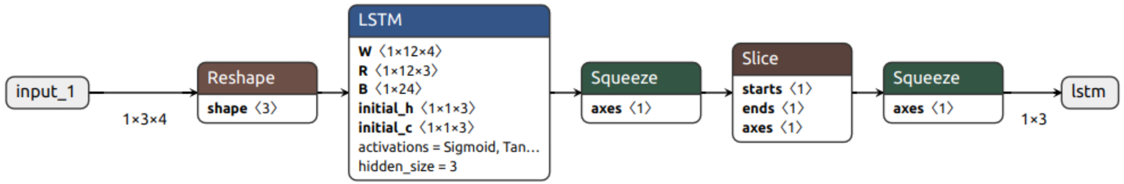
Embedded AI Coder @ SONNX

Neural Network Representations

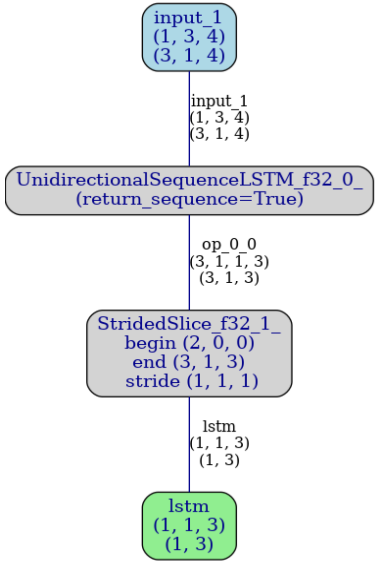
Representation of equivalent models varies between frameworks



Broken LSTM conversion to tflite



ONNX

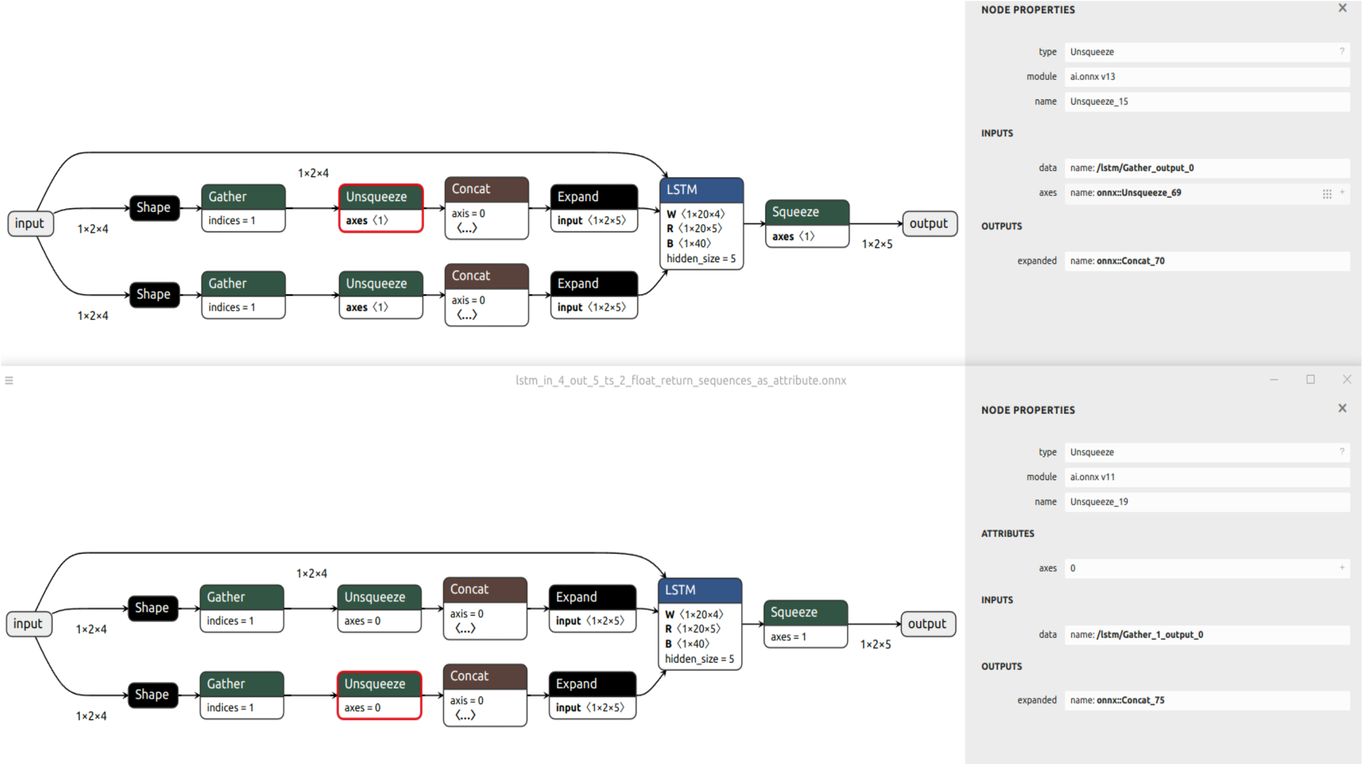


EmbeddedAICoder

Embedded AI Coder @ SONNX

Ambiguities in ONNX Operators

Some Operators in ONNX have several variants of property handling



Simple LSTM models can be very complicated in ONNX

Unsqueeze *axis* can either be an attribute or an input tensor

```python
def convert_unsqueeze(op, subgraph, tensor_collection, index):
    input_shape_tensor = None

    try:
        (
            (input_tensor, input_shape_tensor),
            (output_tensor,),
        ) = get_input_and_output_tensors_from(
            op,
            tensor_collection,
            num_expected_inputs=2,
            num_expected_outputs=1,
            allow_parameter=[1],
            allow_not_found=[1],
        )
    except (ParserError, ValueError):
        (
            (input_tensor,),
            (output_tensor,),
        ) = get_input_and_output_tensors_from(
            op,
            tensor_collection,
            num_expected_inputs=1,
            num_expected_outputs=1,
        )

    output_shape = input_shape_tensor.data
        if input_shape_tensor else output_tensor.shape
    return Unsqueeze(
            input_tensors=(input_tensor,),
            output_tensors=(output_tensor,),
            shape=output_shape,
            index=index,
    )
```

Embedded AI Coder @ SONNX

Data layout transformations for hardware deployment

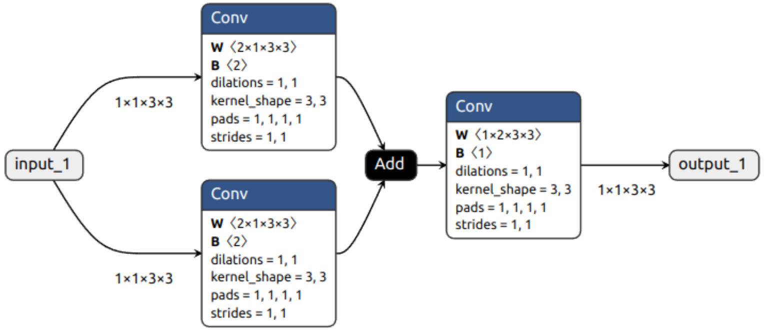
ONNX data layout not optimal for all hardware targets

Inputs

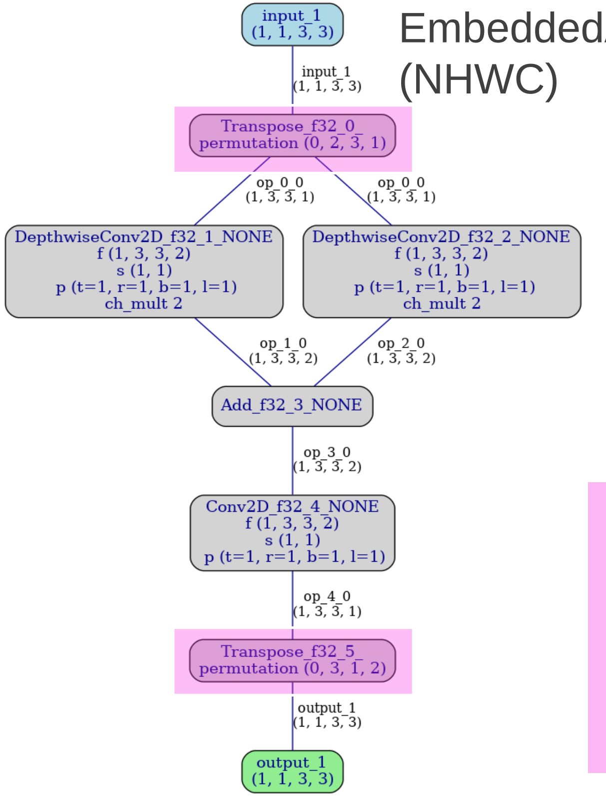
Between 2 and 3 inputs.

- **X** (heterogeneous) - **T**:

  Input data tensor from previous layer; has size (N x C x H x W), where N is the batch size, C is the number of channels, and H and W are the height and width. Note that this is for the 2D image. Otherwise the size is (N x C x D1 x D2 … x Dn). Optionally, if dimension denotation is in effect, the operation expects input data tensor to arrive with the dimension denotation of [DATA_BATCH, DATA_CHANNEL, DATA_FEATURE, DATA_FEATURE …].



ONNX (NCHW)



EmbeddedAICoder (NHWC)

Microcontrollers typically profit from NHWC layout.
Other hardware might require more special layouts.
Requires verification outside of the SONNX standard.

Embedded AI Coder @ SONNX

Goal

SONNX as a file standard

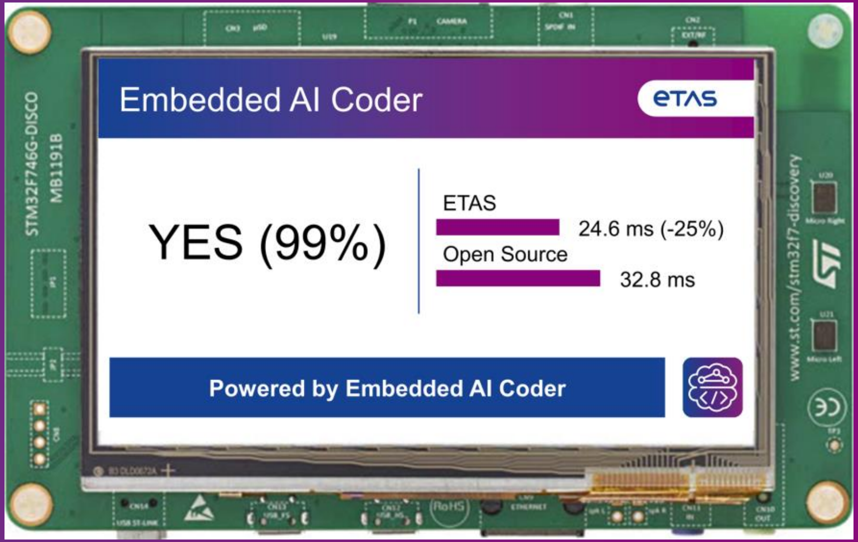
Tools can test ONNX models on whether they comply with the standard and if not, give a reason for it.

Tools like our code generator can give stronger guarantees with models complying with SONNX.

A simple script that checks ONNX files for compliance could be developed in the working group.

Would ensure that the first deployment step DL framework to ONNX model worked correctly.

Empowering Developers to Deploy AI Today



Meet us at

CES Las Vegas (Jan. 7th – Jan 10th, 2025)
Embedded World Nürnberg (March 11th – March 13th, 2025)

www.etas.com/embeddedai
Support.embeddedai@etas.com