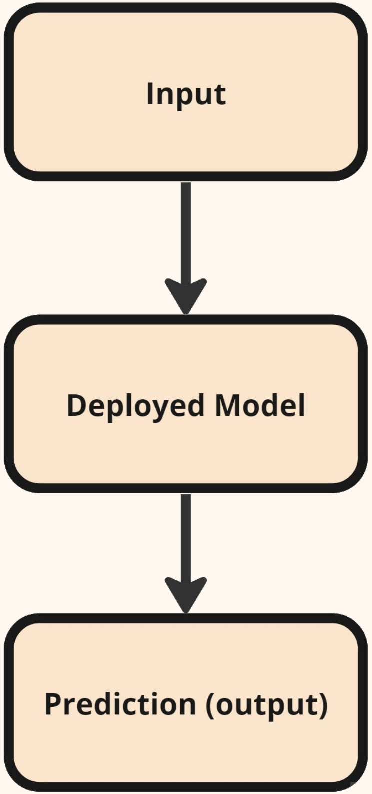
Example Slide - Inference Engines and Audio

This is how big each section of my slides will be. Please move to a location that works for you so you can see!

```
Code blocks are this size
```

Diagrams will look like this —>



Some slides have text this size. All slides will have this background colour (Apart from the title slide)

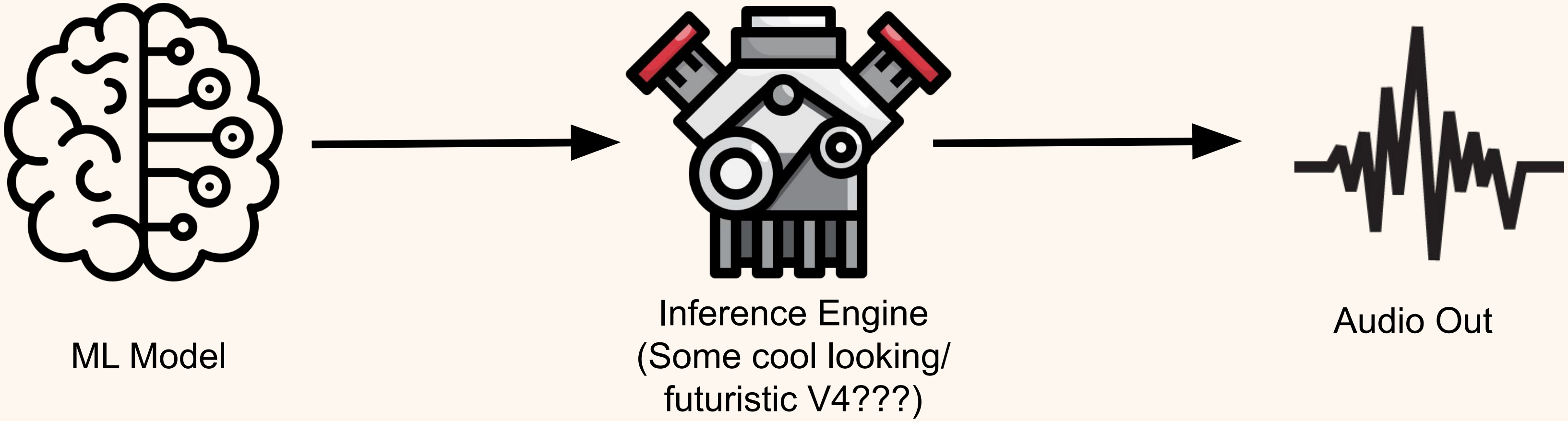
HARRIET DRURY

INFERENCE ENGINES AND AUDIO

Introduction

Hello! My name is Harriet

- Junior dev at Sound Stacks, part of the Native Instruments group (Native Instruments, iZotope, Brainworx, Plugin Alliance).
- Working on the Cmajor platform. A new platform for audio DSP development.



Talk Layout

- Introduction
- What is Inference?
- Deployment Tactics
- Real Time Safety in Inference
- Inference in Cmajor
- Conclusion/ Questions

Github Repo for talk slides/ examples: https://github.com/hazza-music/ADC23

What is Inference?

What is Inference?

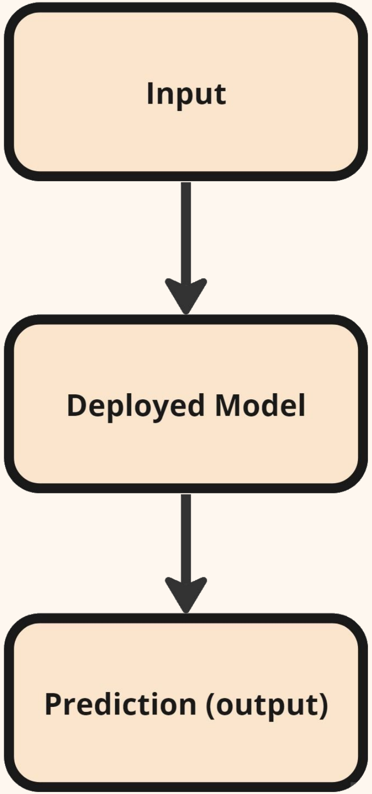
How would you describe machine learning inference to someone who has no experience in inference?

Sure, I'd be happy to explain machine learning inference in a way that's easy to understand for someone with no prior experience:

Imagine you have a smart assistant, like a virtual pet, that you've been teaching to recognize different types of fruits. You've shown it hundreds of pictures of apples, bananas, and oranges and told it what each one is. Now, your smart assistant has learned from all those pictures and instructions.

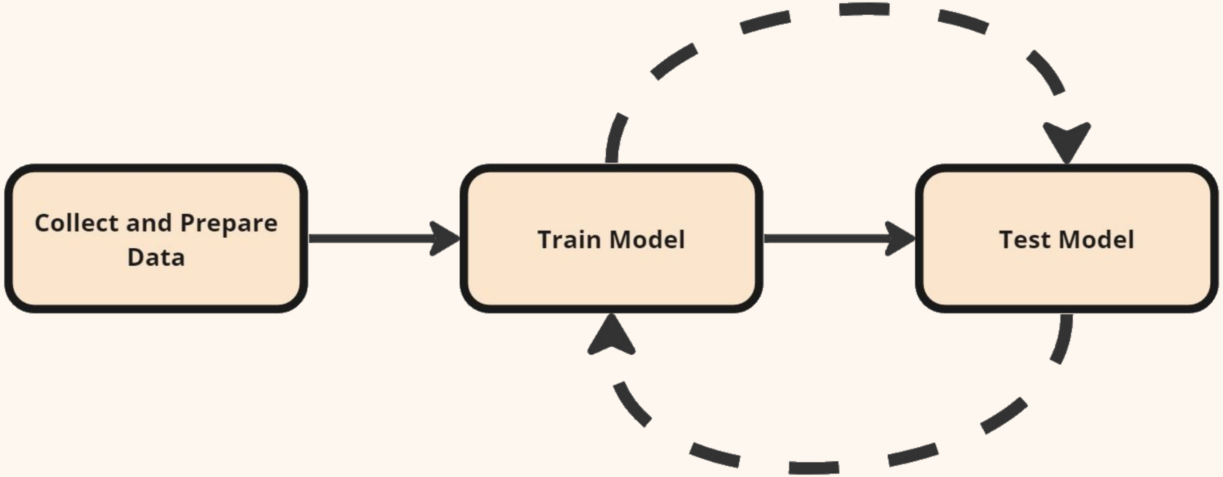
Machine learning inference is like using that smart assistant's knowledge to make quick decisions without having to teach it all over again. It's the part where your assistant takes what it has learned and applies it to new situations.

What is Inference?



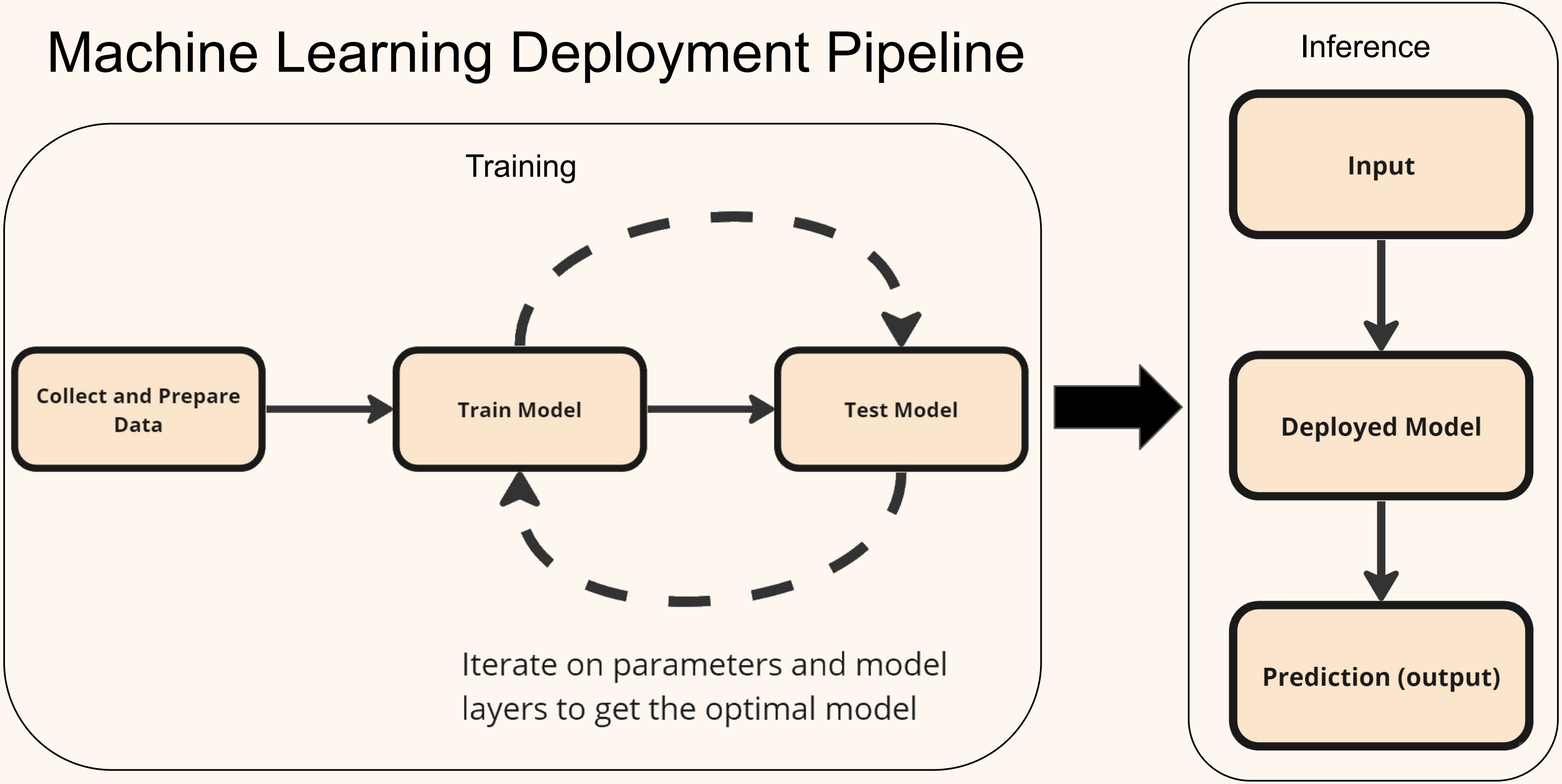
The process of giving a machine learning model unseen data to output (predict) a value.

Training



Iterate on parameters and model layers to get the optimal model

Machine Learning Deployment Pipeline



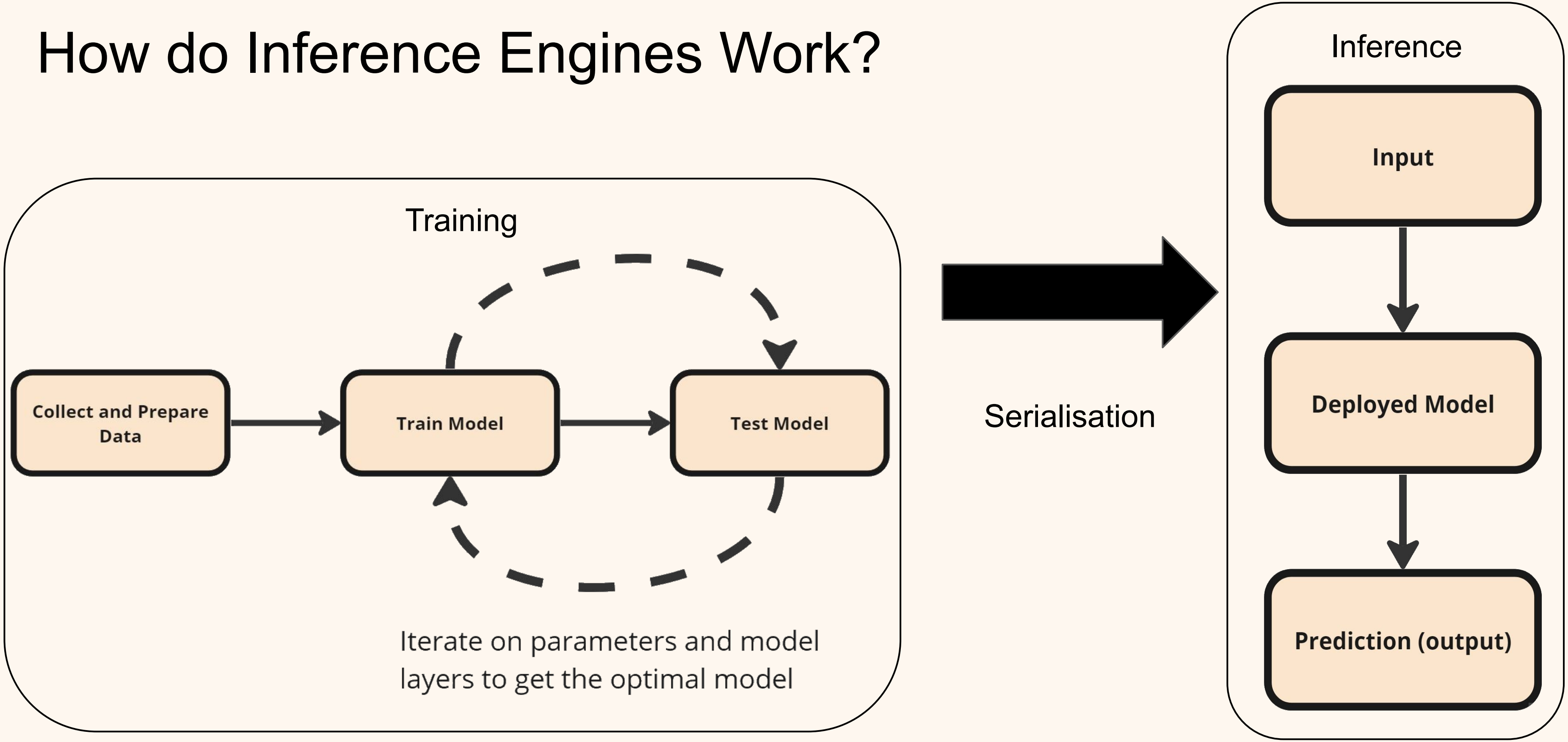
How do Inference Engines work?

Key Concepts

Generally, inference follows these steps:

- Load the Model
- Preprocess Data
- Run Inference
- Interpret Output

How do Inference Engines Work?



Common Saved Model Extensions

.tflite

.h5

.onnx

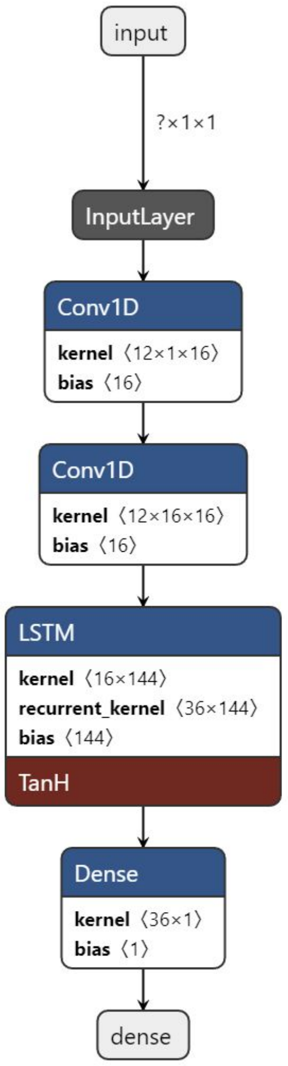
.pt/.pth

saved_weights.json

Topology Graph

Execution of various layers in a machine learning model are described by a topology graph. This will contain details on input/ output sizes, weight sizes and the layers used.

A layer is the highest-level building block in deep learning. A layer is a container that usually receives weighted input, transforms it with a set of mostly non-linear functions and then passes these values as output to the next layer.



Training and Inference Tools

Training Tools

TensorFlow

PyTorch

Keras

Inference Tools

ONNX

TensorFlow → TensorFlow Lite → TensorFlow Micro

PyTorch

TensorRT

RTNEURAL

microTensor

Inference Tools

TensorFlow Micro

TFLite - Overview

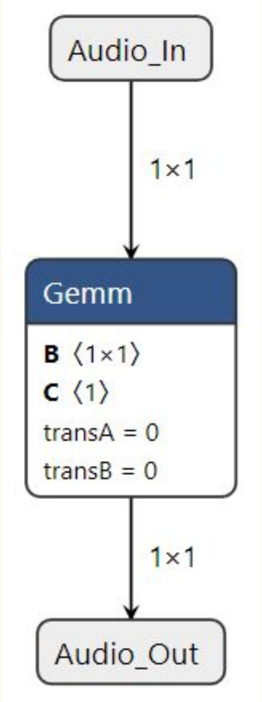
TensorFlow Lite

```
// Load the model
std::unique_ptr<tflite::FlatBufferModel> model = tflite::FlatBufferModel::BuildFromFile(filename);

// Build the interpreter
tflite::ops::builtin::BuiltinOpResolver resolver;
std::unique_ptr<tflite::Interpreter> interpreter;
tflite::InterpreterBuilder(*model, resolver)(&interpreter);

// Resize input tensors, if desired.
interpreter->AllocateTensors();

float* input = interpreter->typed_input_tensor<float>(0);
// Fill `input`.

interpreter->Invoke();

float* output = interpreter->typed_output_tensor<float>(0);
```

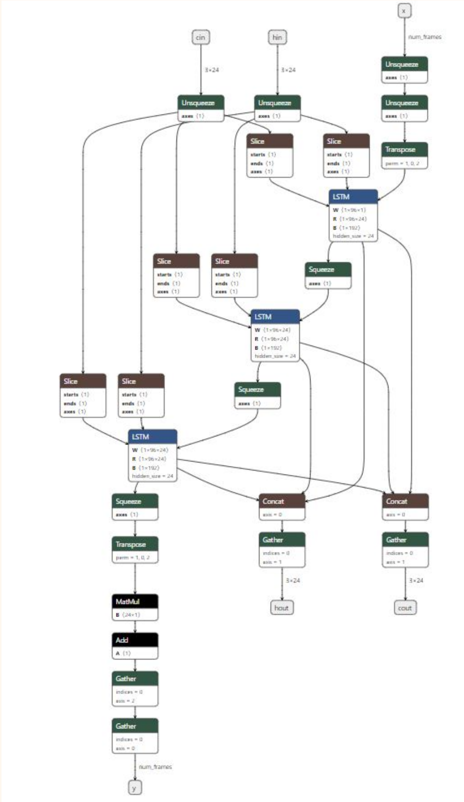
ONNX - Overview

```
std::vector<const char*> inputNames{inputName};
std::vector<const char*> outputNames{outputName};
std::vector<Ort::Value> inputTensors;
std::vector<Ort::Value> outputTensors;

Ort::MemoryInfo memoryInfo = Ort::MemoryInfo::CreateCpu(OrtAllocatorType::OrtArenaAllocator,
                                                        OrtMemType::OrtMemTypeDefault);

inputTensors.push_back(Ort::Value::CreateTensor<float>(memoryInfo, inputTensorValues.data(),
                                                        inputTensorSize, inputDims.data(),
                                                        inputDims.size()));

outputTensors.push_back(Ort::Value::CreateTensor<float>(memoryInfo, outputTensorValues.data(),
                                                         outputTensorSize, outputDims.data(),
                                                         outputDims.size()));

session.Run(Ort::RunOptions{nullptr}, inputNames.data(), inputTensors.data(), 1 /*Number of
inputs*/, outputNames.data(), outputTensors.data(), 1 /*Number of outputs*/);
```

ONNX - Operators



ONNX - Operators



Netron - Shoutout

https://github.com/lutzroeder/netron

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

Netron supports ONNX, TensorFlow Lite, Core ML, Keras, Caffe, Darknet, MXNet, PaddlePaddle, ncnn, MNN and TensorFlow.js.

Netron has experimental support for PyTorch, TorchScript, TensorFlow, OpenVINO, RKNN, MediaPipe, ML.NET and scikit-learn.

RTNeural - Overview

RTNEURAL

```
// define model type
RTNeural::ModelT<double, 8, 1
    RTNeural::DenseT<double, 8, 8>,
    RTNeural::TanhActivationT<double, 8>,
    RTNeural::DenseT<double, 8, 1>
> modelT;

// load model weights from json
std::ifstream jsonStream("model_weights.json", std::ifstream::binary);
modelT.parseJson(jsonStream);

modelT.reset(); // reset state

double input[] = { 1.0, 0.5, -0.1 }; // set up input vector
double output = modelT.forward(input); // compute output
```

Inference Engines - Suitability for Audio

Real Time Safety in Inference Engines

Things to look for in an inference engine to determine whether or not it is RT safe:

- Unbounded Execution Time
  - Locks
  - Memory Allocation
  - Garbage Collection
- No Waiting for External Hardware

Can inference be ran in a real time process using the libraries above?

Real Time Safety in Inference Engines

ONNX

Tflite

RTNeural

Real Time Safety in Inference Engines

ONNX ❌

- Dynamic Memory Allocations

Tflite ❓

- After running inference once, subsequent inferences are real time safe, but there is a couple allocations that need to be changed. Large layer of abstraction involved

RTNeural ✓

- Can be used in a C++ program to run inference on a network in real time

Inference in Cmajor

(Writing an Inference Engine)

Why make another Inference Engine?

Tflite can be tricky to use, but..

- It has a large audience base and good testing
- Easy to implement inference/ good API

RTNeural has a good base…

- We want to be as agnostic to training as possible

ONNX is not RT safe, but…

- It is also training agnostic
- Easier to find performance gains in smaller operations

Cmajor, Briefly

Cmajor is a C-styled language that is designed for DSP signal processing code.

Aims:

- To match/beat the performance of C/C++
- Be simple to learn
- Make code portable across processor architectures (CPUs, GPUs, DSPs, etc)

Uses an LLVM JIT compiler, to optimise and hot reload code

More info: https://cmajor.dev/

The Objective

Write a Cmajor inference engine to plug into existing tooling within the Cmajor platform.

Writing an Inference Engine

Checklist of initial requirements:

- Real time safe
- Training Agnostic (as much as possible)
- Target small models/ matrices and build up
- Plug into existing workflows
- Good performance numbers

Inference in Cmajor

A few options:

- DIY Custom Model
- RTNeural to Cmajor Transpiler
- ONNX to Cmajor Transpiler

Current Feature Request:

- Support for introducing external processors. We can then call pre existing Inference Engines

Example - GuitarLSTM

Github: https://github.com/GuitarML/GuitarLSTM

Deep learning models for guitar amp/pedal emulation using LSTM with Keras.

I’ve added a few lines of code to save the model in various formats, but everything else has stayed the same

RTNeural to Cmajor

You can run cmajor/tools/rtneural/rtneuralToCmajor.py to perform the conversion, with arguments:

```
--model <file>        Specifies the RTNeural model file to convert

--outputDir <folder>  Specifies a folder into which the generated patch will be written

--name <name>         Optional name for the model - defaults to "Model"

--useFloat64          By default we use float32, but this changes the generated model to float64
```

If no patchDir argument is supplied, the output will be a single Cmajor file.

RTNeural to Cmajor

CmajorPatches > RTNeuraltoCmajor > model.cmajor

```
//  //   d8             88,dPba,,adPba,   ,adPPYba,  88     The Cmajor Language
//  //   88             88P'  "88"   "8a         '88  88
//  //   Y8,            88      88      88  ,adPPPP88  88     (c)2022 Sound Stacks Ltd
//  //    Y8a.   .a8P   88      88      88  88,    ,88  88     https://cmajor.dev
//  //      '"Y888Y"'   88      88      88  '"8bbP"Y8  88
//  //                                                ,88
//  //                                              888P"

graph Model [[ main ]]
{
    input stream float<1> in;
    output stream float<1> out;

    namespace rt = rtneural (float);

    node
    {
        l1 = rt::layer::Conv1d (1, 16, 12, 1, l1Weights, l1Biases);
        l2 = rt::layer::Conv1d (16, 16, 12, 1, l2Weights, l2Biases);
        l3 = rt::layer::Lstm (16, 36, rt::ActivationFunction::tanh, l3W, l3U, l3B);
        l4 = rt::layer::Dense (36, 1, l4Weights, l4Biases);
    }

    connection
    {
```

RTNeural to Cmajor

Outputted files:

- `model.cmajor` - Contains all the nodes, connections and weights to describe the model
- `RTNeural.cmajor` - Contains definitions of the RTNeural operators required by the model
- `model.cmajorpatch` - As with all cmajor manifest files, this describes the patch's properties and contains links to the other files in the patch

ONNX to Cmajor

You can run cmajor/tools/onnx/onnxToCmajor.py to do the conversion.

Command-line options are:

```
--model <file>        Specifies the ONNX model file to convert

--patchDir <folder>   Specifies a folder into which the generated patch will be written
```

ONNX to Cmajor

```
graph model [[ main ]]
{
    input stream float32<1> conv1d_input;
    output stream float32<1> dense;

    node
    {
        conv1d_input_Transformer = onnx::graphIn (float32<1>, float32[1,1]);
        dense_Transformer = onnx::graphOut (float32[1], float32<1>);
        sequential_lstm_zeros_1_Const_0 = onnx::operation::Constant (sequential_lstm_zeros_1_Const_0_.type, sequential_lstm_zeros_1_Const_0_);
        sequential_dense_MatMul_ReadVariableOp_0 = onnx::operation::Constant (sequential_dense_MatMul_ReadVariableOp_0_.type, sequential_dense_MatMul_ReadVariableOp_0_);
        sequential_dense_BiasAdd_ReadVariableOp_0 = onnx::operation::Constant (sequential_dense_BiasAdd_ReadVariableOp_0_.type, sequential_dense_BiasAdd_ReadVariableOp_0_);
        sequential_conv1d_1_Conv1D_ExpandDims_1_0 = onnx::operation::Constant (sequential_conv1d_1_Conv1D_ExpandDims_1_0_.type, sequential_conv1d_1_Conv1D_ExpandDims_1_0_);
```

Code Example

Let’s demo our RTNeural transpiler using our VSCode extension….

Summary

We’ve looked at:

- What is Inference?
- Inference vs Training
- Common inference tools
- Real time safety in inference
- Writing a custom inference engine
- Brief Cmajor example

Thank you for Listening

Big thanks to:

Cmajor

QR for slides & code