Call AI models in Gadgetron

Hui Xue

NHLBI, NIH

July 2020

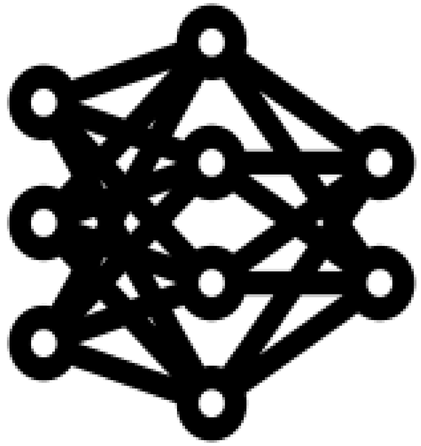
National Heart, Lung, and Blood Institute

In this session

- Motivation to bring in AI models
- Inline vs. offline
- Gadgetron InlineAI
- Example: Run Grappa as an AI model
- Example: Run cine landmark detection
- Demo with inline walk-through
- Final remarks

Why we need to do something to run AI models

- **Because vender provided software lacks support**



Trained NN

- ✓ Rich in computing power
- ✓ Flexible and updated software configuration
- ✓ Computing time matters less

Model inference

Not so easy to deploy NN

At this moment of 2020

- Limited computing power
- Often outdated software configuration
- Clinical relevant computing time

NIH National Heart, Lung, and Blood Institute

Inline vs Offline

data

Images, report

Gadgetron

Computing engine

- **Automated processing**
- **End-to-end from imaging to reporting**
- **Can be duplicated and on cloud**

images

Software

Operator

- **Interactive processing**
- **Established workflow**
- **Expert time**

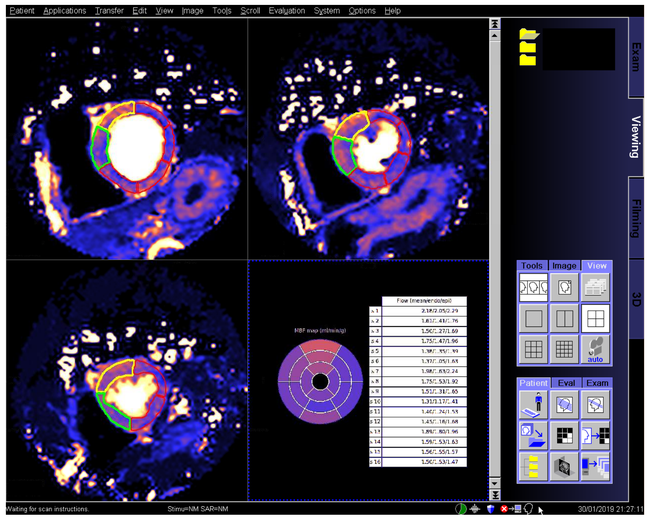
Example of Inline AI: Cine Analysis and reporting

- Deployed to hospitals
- Currently 85% acceptance rate (every 1000 patients, clinicians decided to use the reported values in ~850 cases)

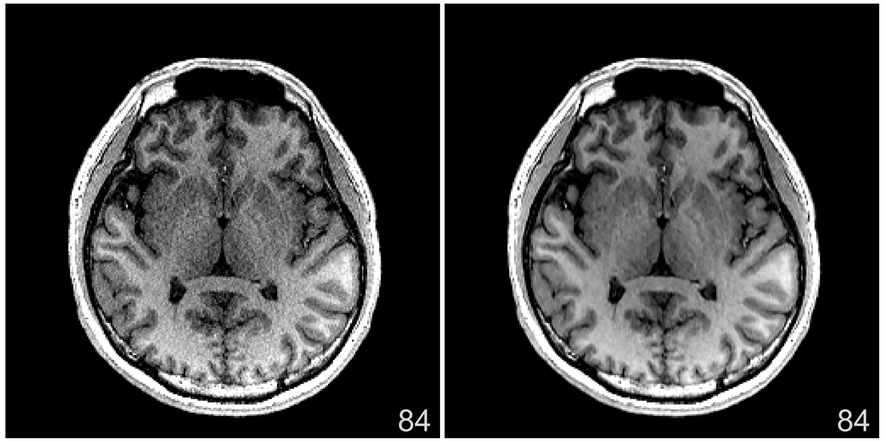
Collaboration with Barts Heart Centre, London, UK

It is the time to do this

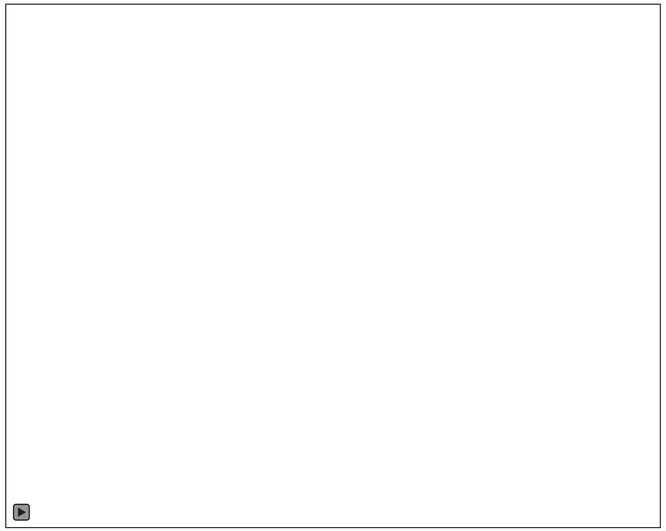
- **Deep learning - new hope to promote Inline workflow**



Cardiac perfusion



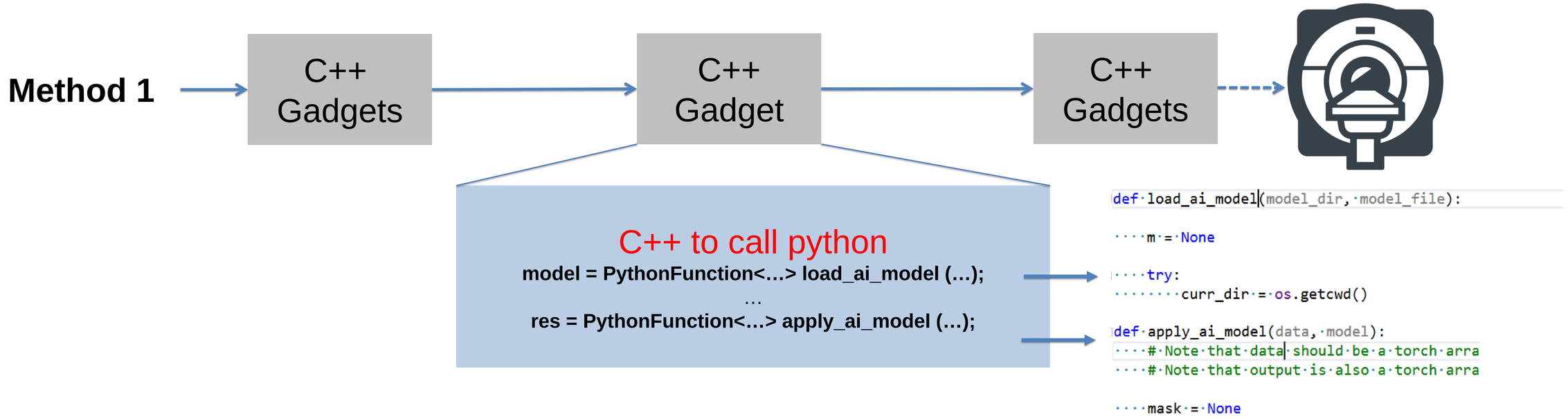
Denoising, 1mm2 MPRAGE



On scanner

1. https://arxiv.org/abs/1911.00625
2. https://onlinelibrary.wiley.com/doi/full/10.1002/mrm.28291

Use Gadgetron to call AI models



Gadgetron Inline AI: Effective Model inference on MR scanner
http://archive.ismrm.org/2019/4837.html

Use Gadgetron to call AI models

Python only

C++ only

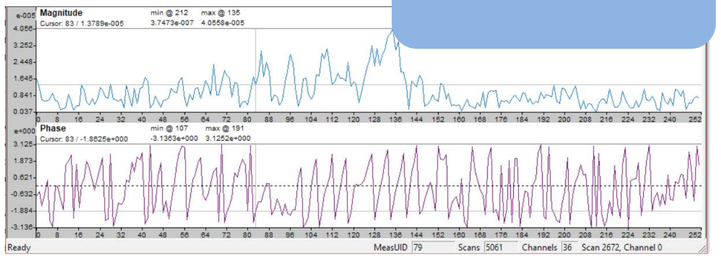
Support major AI framework

Tensorflow, Pytorch, Pytorch C++

Gadgetron Inline AI: Effective Model inference on MR scanner
http://archive.ismrm.org/2019/4837.html

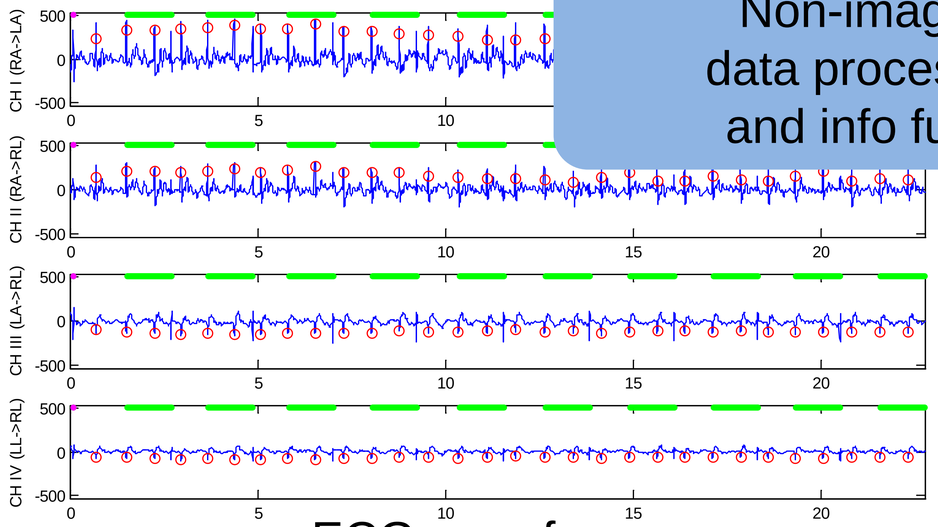
Use Gadgetron to collect data for AI: rich data types for different applications

Reconstruction



MR Kspace data

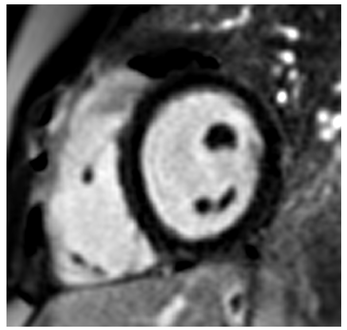
Non-imaging data processing and info fusion



ECG waveform

- Support all steps of MR imaging
- New data types can be added
- Designed to support AI R&D

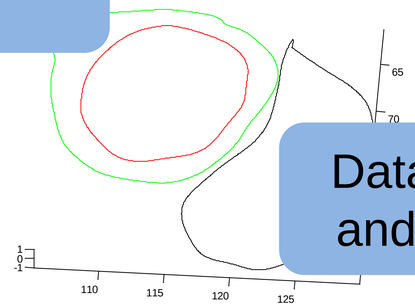
Diagnosis and feedback



Image

ML meta data

Data labelling and collection



Contours and landmarks

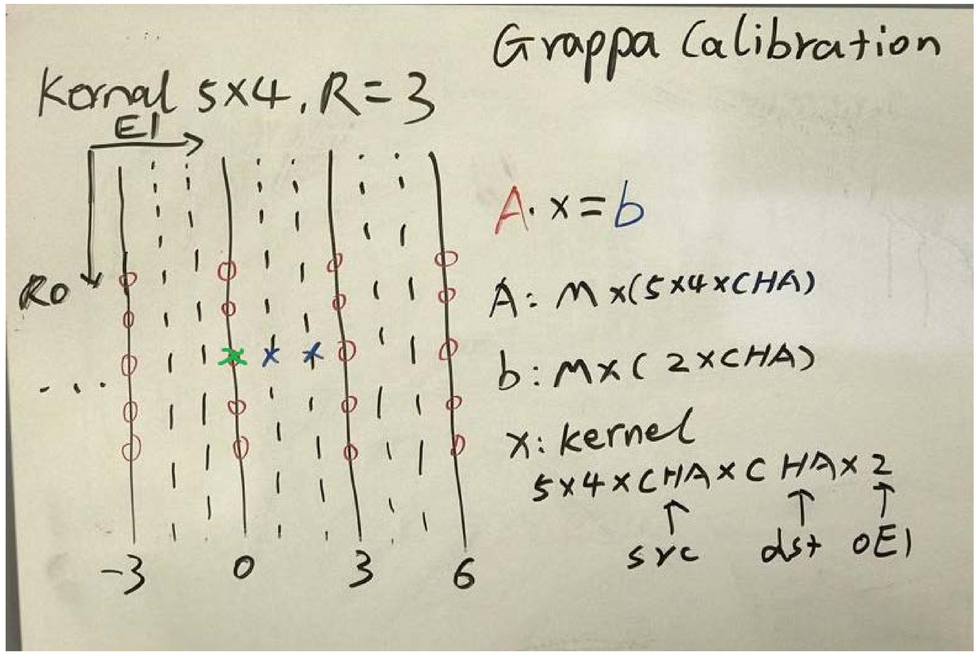
Image analysis

National Heart, Lung, and Blood Institute

Example: Run Grappa in Pytorch

- **Let's look at Grappa**

Recall Grappa is to estimate kspace interpolation kernel to fill in unacquired phase encoding line:

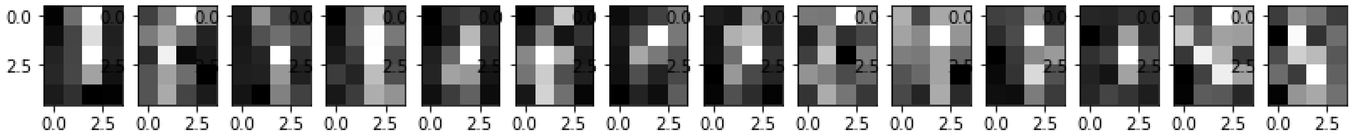


Perform Grappa reconstruction by calling the Gadgetron python wrapper for Grappa:

Kernel size 5x4xSrcCHAxDstCHA

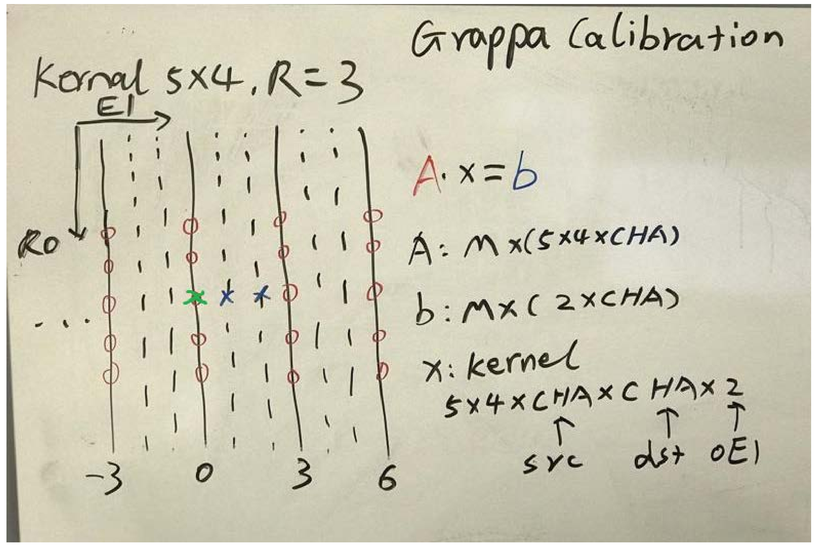
Calibration is solved by :

1) Assmeble data matrix A and RHS b
2) Solve Ax=b for kernel x



Example: Run Grappa in Pytorch

- **Let's look at Grappa**



Grappa recon involves to apply kernel to acquired data

for every missing kspace point in data:

1. Extract the 5x4xCHA acquired dataA

2. Multiply dataA with ker : res = np.dot(dataA, ker) here ker is (5x4xCHA, 2xCHA) res is (1, 2xCHA)

3. Fill res to the data array for missing data

Example: Run Grappa as AI model

Wrap grappa to call in Python

```
namespace Gadgetron
{
    class EXPORTMRICORE grappa2D
    {
    public:

        typedef std::complex<float> T;

        grappa2D();
        ~grappa2D();

        int initialize(size_t accelFactor, size_t kRO, size_t kNE1, bool fitItself, double thres);
        int calib(hoNDArray<T> acsSrc, hoNDArray<T> acsDst);

        // kspace [RO, E1, srcCHA], perform recon using ker_
        hoNDArray<T> recon(hoNDArray<T> kspace, bool periodic_boundary_condition);

        // if kspace is already converted to data matrix (dataA and dataAInd), perform recon using ker_
        hoNDArray<T> recon_data_matrix(hoNDArray<T> dataA, hoNDArray<unsigned short> dataAInd, size_t RO, size_t E1, bool periodic_boundary_condition);
        // if recon = dataA*ker_, fill back recon points to result kspace
        hoNDArray<T> recon_fill_back_kspace(hoNDArray<T> recon, hoNDArray<unsigned short> dataAInd, size_t RO, size_t E1);

        hoNDArray<T> get_A()
        {
            return A_;
        }
```

toolboxes\mri_core\mri_core_grappa_python.h

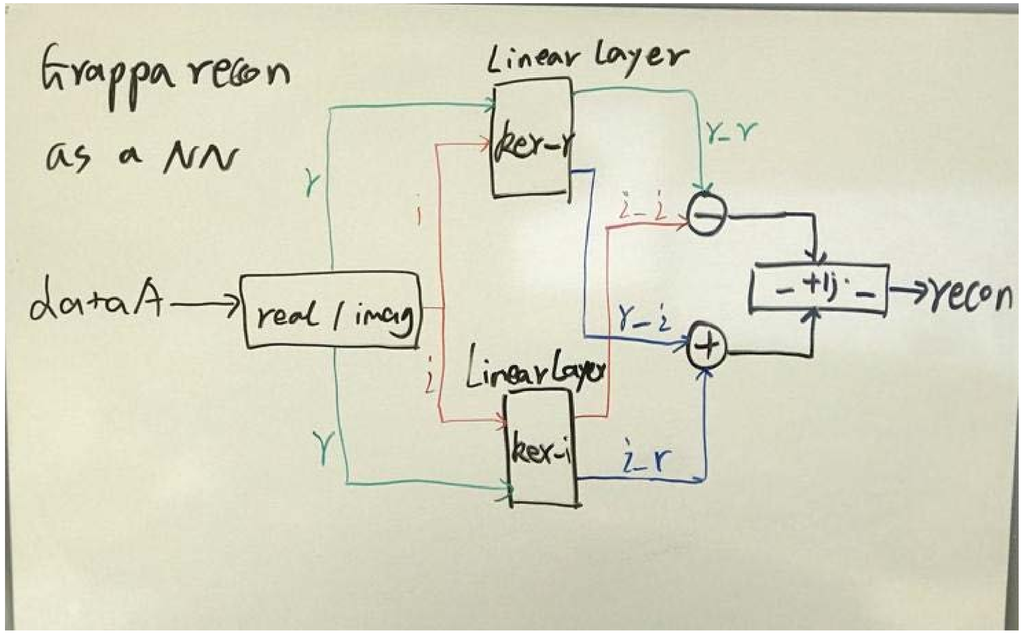
```
BOOST_PYTHON_MODULE(gadgetron_toolbox_mri_core_python)
{
    // for test purpose
    def("hello", hello);

    class_<Gadgetron::grappa2D>("grappa2D")
        .def("initialize", &Gadgetron::grappa2D::initialize)
        .def("calib", &Gadgetron::grappa2D::calib)
        .def("recon", &Gadgetron::grappa2D::recon)
        .def("recon_data_matrix", &Gadgetron::grappa2D::recon_data_matrix)
        .def("recon_fill_back_kspace", &Gadgetron::grappa2D::recon_fill_back_kspace)
        .def("get_A", &Gadgetron::grappa2D::get_A)
        .def("get_B", &Gadgetron::grappa2D::get_B)
        .def("get_ker", &Gadgetron::grappa2D::get_ker)
        .def("get_data_A", &Gadgetron::grappa2D::get_data_A)
        .def("get_data_A_index", &Gadgetron::grappa2D::get_data_A_index)
        .def("help", &Gadgetron::grappa2D::help)
        .def("status", &Gadgetron::grappa2D::status)
        ;
}
```

toolboxes\mri_core\mri_core_python.h

Example: Run Grappa as AI model

- **Implement Grappa recon as NN**




```python
class GrappaAI(nn.Module):
    def __init__(self, Din, Dout, params):
        super(GrappaAI, self).__init__()

        self.verbose = params['verbose']
        self.with_bias = False
        self.grappa_weights_r = params['grappa_weights_r']
        self.grappa_weights_i = params['grappa_weights_i']

        #for real and imag
        self.input_layer_r = torch.nn.Linear(int(Din/2), int(Dout/2), bias=self.with_bias)
        self.input_layer_i = torch.nn.Linear(int(Din/2), int(Dout/2), bias=self.with_bias)

        if self.verbose:
            print("    GrappaAI : input size (%d), output size (%d), with bias %s" % (Din, Dout, self.with_bias))

        print("---> Set grappa weights")
        self.input_layer_r.weight.data = self.grappa_weights_r
        self.input_layer_i.weight.data = self.grappa_weights_i
        
    def forward(self, x):

        #split x to real and img
        M, N = x.shape

        N_h = int(N/2)

        x_r = x[:, 0:N_h]
        x_i = x[:, N_h:N]

        r_r = self.input_layer_r(x_r)
        i_i = self.input_layer_i(x_i)

        r_i = self.input_layer_r(x_i)
        i_r = self.input_layer_i(x_r)

        res_r = r_r - i_i
        res_i = r_i + i_r

        out = torch.cat([res_r, res_i], dim=1)
```

gadgets/python/models/grappa_ai.py

Example: Run Grappa as AI model

- **Integrate GrappaAI into Generic chain**

Generic_Cartesian_Grappa_AI.xml

```
#pragma once

#include "GenericReconCartesianGrappaGadget.h"
#include "python_toolbox.h"

namespace Gadgetron {

    class EXPORTGADGETSMRICORE GenericReconCartesianGrappaAIGadget : public GenericReconCartesianGrappaGadget
    {
    public:
        GADGET_DECLARE(GenericReconCartesianGrappaAIGadget);

        typedef GenericReconCartesianGrappaGadget BaseClass;
        typedef typename BaseClass::ReconObjType ReconObjType;
        typedef std::complex<float> T;

        GenericReconCartesianGrappaAIGadget();
        ~GenericReconCartesianGrappaAIGadget();

    protected:

        //----------------------------------------------------
        // gadget functions
        //----------------------------------------------------
        /// [Nor1 Sor1 SLC]
        std::vector< std::vector< hoNDArray<T> > > kernels_;
        std::vector< std::vector<boost::python::object> > models_;
        /// [RO E1 E2 1 N S SLC]
        std::vector<IsmrmrdImageArray> recon_res_grappa_ai_;

        // gadgetron home
        std::string gt_home_;

        // default interface function
        virtual int process_config(ACE_Message_Block* mb);
        virtual int process(Gadgetron::GadgetContainerMessage< IsmrmrdReconData >* m1);

        // calibration, if only one dst channel is prescribed, the GrappaOne is used
        virtual void perform_calib(IsmrmrdReconBit& recon_bit, ReconObjType& recon_obj, size_t encoding);

        // unwrapping or coil combination
        virtual void perform_unwrapping(IsmrmrdReconBit& recon_bit, ReconObjType& recon_obj, size_t encoding);

        virtual int close(unsigned long flags);
    };
}
```

- Extend the generic recon
- Implement a new Recon gadget
- Derived from CartesianGrappa gadget to reuse components
- Call Python GrappaAI module

Demo and code walk through

Grappa AI recon

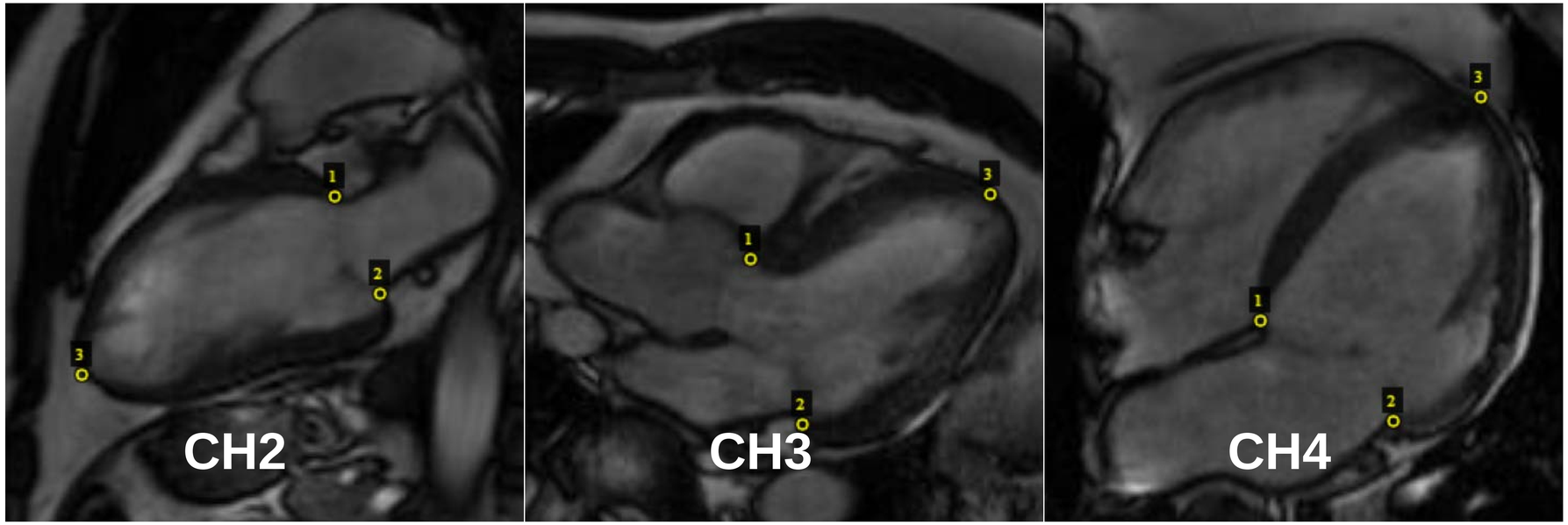
a) Grappa python binding
toolboxes\mri_core\mri_core_grappa_python.h
toolboxes\mri_core\mri_core_python.h

b) Gappa python module
gadgets/python/models/grappa_ai.py

- Compare C++ Grappa with this AI Grappa

Example: AI for image analysis

- **Call pre-trained AI model, report results on line**



Detect landmarks in cardiac images

Example: AI for image analysis

- **A Res-Unet CNN**

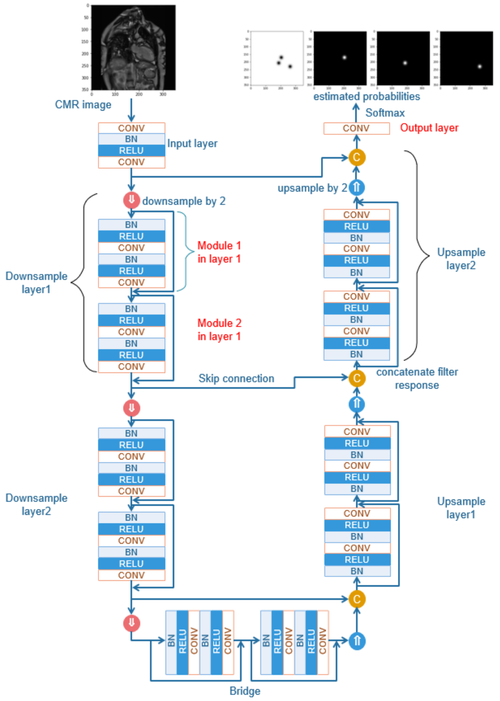


Figure 4. The backbone CNN network developed for landmark detection has a U-net structure. More layers can be inserted to both downsampling and upsampling branches. More modules can be inserted into each layer. The output layer can be a convolution for semantic segmentation or fully connected layer for regression task.

Trained on 11,426 images from 1,905 patients

25 CONV layers

Pytorch 1.5.0

https://onlinelibrary.wiley.com/doi/full/10.1002/mrm.28291

Example: AI for image analysis

CMR_RTCine_LAX_AI.xml

gadgets/cmr/CmrRealTimeLAXCineAIAnalysisGadget.h/cpp

```
·//·load·model
·PythonFunction<boost::python::object>·load_model_cmr_landmark_detection("gadgetron_cmr_landmark_detection",·"load_model_cmr_landmark_detection");
·boost::python::object·model·=·load_model_cmr_landmark_detection(this->gt_home_,·this->lax_landmark_detection_model.value());
·bp::incref(model.ptr());

·//·apply·model
·{
····boost::python::object*·pModel·=·&model;
····PythonFunction<·hoNDArray<float>,·hoNDArray<float>·>·perform_cmr_landmark_detection("gadgetron_cmr_landmark_detection",·"perform_cmr_landmark_detection");
····std::tie(pts,·probs)·=·perform_cmr_landmark_detection(lax_images,·*pModel,·1.0,·8,·0.1,·this->oper_RO.value(),·this->oper_E1.value());
·}
```

gadgets/cmr/config/gadgetron_cmr_landmark_detection.py

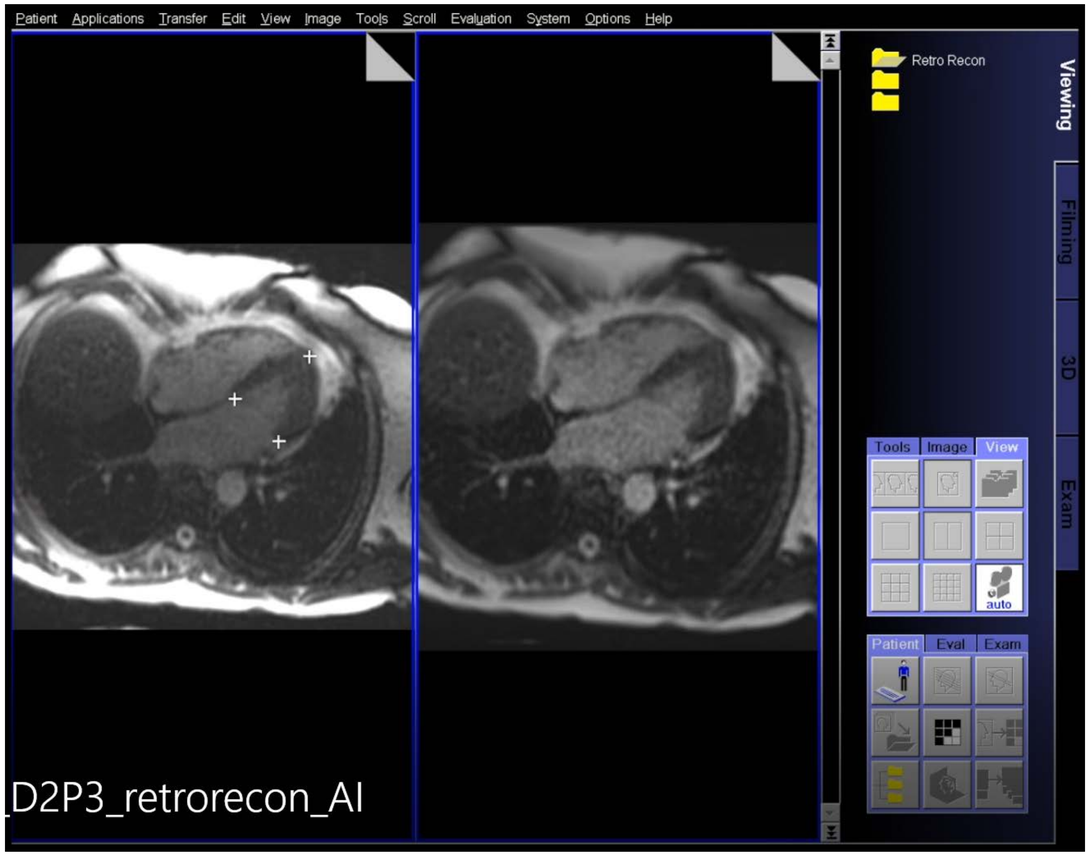
```
def·load_model_cmr_landmark_detection(model_dir,·model_file):

····m·=·None

····try:
········print("Load·cmr·landmark·model·:·%s"·%·os.path.join(model_dir,·model_file),·file=sys.stderr)
········t0·=·time.time()
········m·=·torch.jit.load(os.path.join(model_dir,·model_file))
········t1·=·time.time()
········print("Model·loading·took·%f·seconds·"·%·(t1-t0),·file=sys.stderr)
········sys.stderr.flush()
····except·Exception·as·e:
········print("Error·happened·in·load_model_cmr_landmark_detection·for·%s"·%·model_file,·file=sys.stderr)
········print(e)

····return·m
```

Example: AI for image analysis



National Heart, Lung, and Blood Institute

Demo: AI for CMR landmark detection

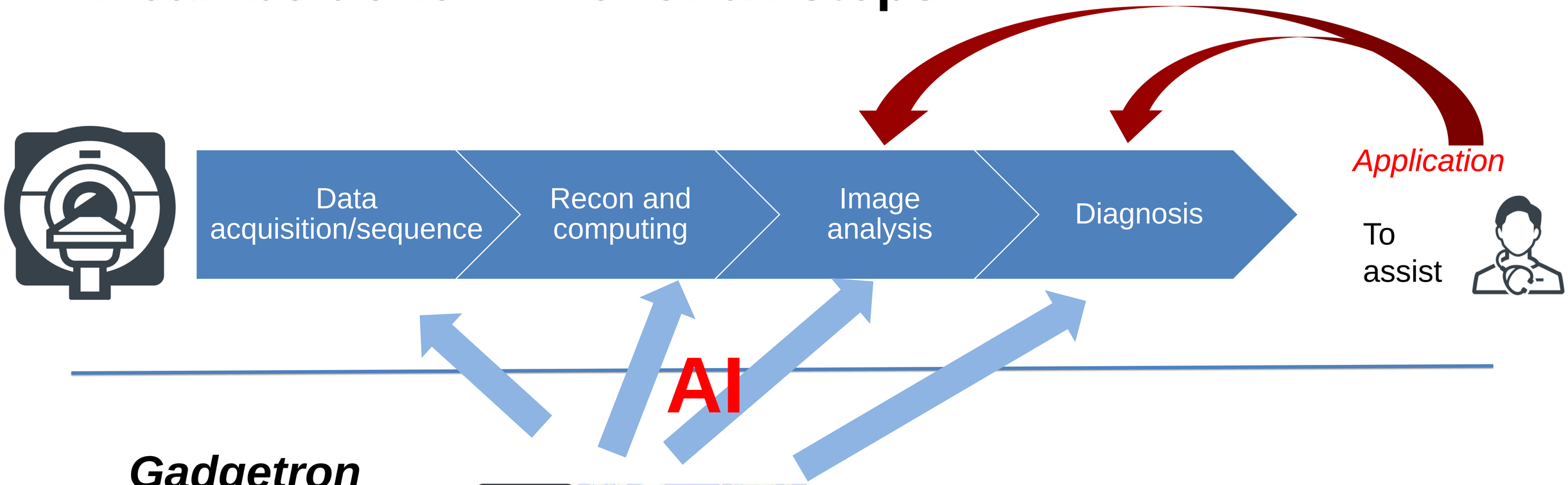
- Code walk through
  - C++ side
    - gadgets/cmr/CmrRealTimeLAXCineAIAnalysisGadget.h/cpp
  - Python side
    - gadgets/cmr/config/gadgetron_cmr_landmark_detection.py

To run this demo:

```
gadgetron_ismrmrd_client -f 2ch_RT_cine.h5 -c CMR_RTCine_LAX_AI.xml -a localhost -p 9002 -F hdr -G
CMR_RTCine_LAX_AI.xml -o ref_20200626_084537.h5
```

Final remarks: exciting time ahead

- **Imaging, computing, analysis and diagnosis**
- **AI can be done inline for all steps**



Gadgetron computing infrastructure

National Heart, Lung,
and Blood Institute