Integration of NuGraph GNN into LArSoft

Giuseppe Cerati (FNAL)

NuGraph All Hands Meeting

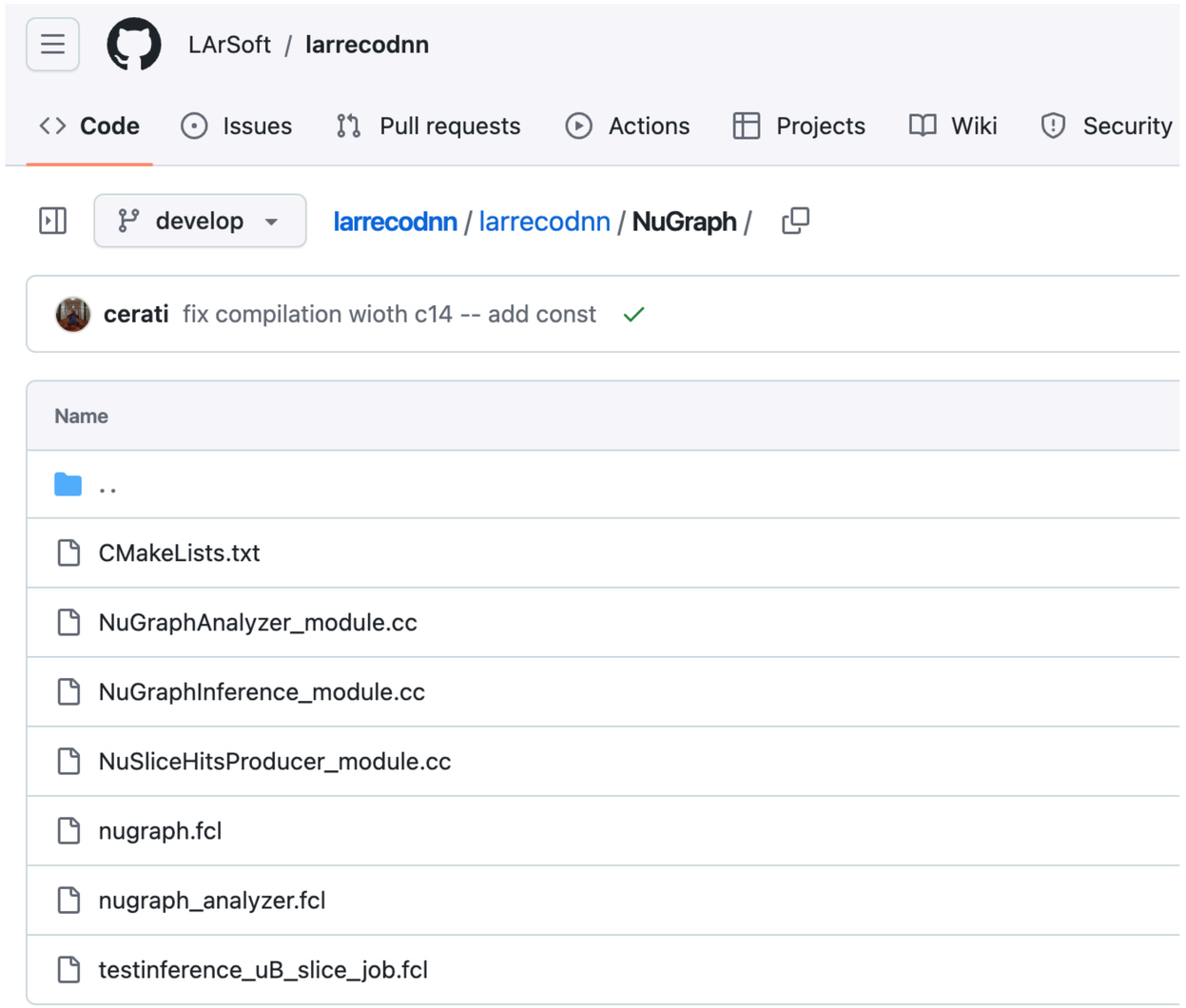
Feb. 19, 2024

Breaking news from last week

- NuGraph2 model has been integrated in LArSoft release:
  - https://github.com/LArSoft/larsoft/releases/tag/v09_83_01
  - It's located in the larrecodnn/NuGraph package

- New dependencies introduced for NuGraph:
  - delaunator v0_4_0
    - we considered using ROOT::Math::Delaunay2D but it was not giving identical output
  - libtorch v2_1_1
  - torch_scatter v2_1_2

Snapshot of the integrated version



Fermilab

NuGraphInference_module

https://github.com/LArSoft/larrecodnn/blob/develop/larrecodnn/NuGraph/NuGraphInference_module.cc

```
vector<vector<Edge>> edge2d(planes.size(), vector<Edge>());
for (size_t p = 0; p < planes.size(); p++) {
  if (debug) std::cout << "Plane " << p << " has N hits=" << coords[p].size() / 2 << std::endl;
  if (coords[p].size() / 2 < 3) continue;
  delaunator::Delaunator d(coords[p]);
  if (debug) std::cout << "Found N triangles=" << d.triangles.size() / 3 << std::endl;
  for (std::size_t i = 0; i < d.triangles.size(); i += 3) {
    //create edges in both directions
    Edge e;
    e.n1 = d.triangles[i];
    e.n2 = d.triangles[i + 1];
    edge2d[p].push_back(e);
    e.n1 = d.triangles[i + 1];
    e.n2 = d.triangles[i];
    edge2d[p].push_back(e);
```

(1) create Delaunay edges

```
auto x = torch::Dict<std::string, torch::Tensor>();
auto batch = torch::Dict<std::string, torch::Tensor>();
for (size_t p = 0; p < planes.size(); p++) {
  long int dim = nodeft[p].size() / 4;
  torch::Tensor ix = torch::zeros({dim, 4}, torch::dtype(torch::kFloat32));
```

```
  for (size_t n = 0; n < nodeft[p].size(); n = n + 4) {
    ix[n / 4][0] = nodeft[p][n];
    ix[n / 4][1] = nodeft[p][n + 1];
    ix[n / 4][2] = nodeft[p][n + 2];
    ix[n / 4][3] = nodeft[p][n + 3];
  }
  x.insert(planes[p], ix);
  torch::Tensor ib = torch::zeros({dim}, torch::dtype(torch::kInt64));
  batch.insert(planes[p], ib);
}
```

(2) fill input torch Tensors

```
std::vector<torch::jit::IValue> inputs;
inputs.push_back(x);
inputs.push_back(edge_index_plane);
inputs.push_back(edge_index_nexus);
inputs.push_back(nexus);
inputs.push_back(batch);
if (debug) std::cout << "FORWARD!" << std::endl;
auto outputs = model.forward(inputs).toGenericDict();
```

(3) collect inputs, run inference

```
if (vertexDecoder) {
  torch::Tensor v = outputs.at("x_vertex").toGenericDict().at(0).toTensor();
  double vpos[3];
  vpos[0] = v[0].item<float>();
  vpos[1] = v[1].item<float>();
  vpos[2] = v[2].item<float>();
  vertcol->push_back(recob::Vertex(vpos));
}

if (filterDecoder) { e.put(std::move(filtcol), "filter"); }
if (semanticDecoder) {
  e.put(std::move(semtcol), "semantic");
  e.put(std::move(semtdes), "semantic");
}
if (vertexDecoder) { e.put(std::move(vertcol), "vertex"); }
```

(4) get output, put into event

Fermilab

Storing the output in event record

- Use already available classes to store GNN output in event record:
  - FeatureVector<N> and MVADescription<N> for hit-level predictions
    - both defined in lardataobj/AnalysisBase/MVAOutput.h
    - FeatureVector contains the prediction scores, one entry per hit
      - result stored in same order as input hit collection (no Assns)
    - MVADescription contains the input label of the hit collection and a description of each entry in the FeatureVector. One entry per event.
  - Use recob::Vertex for vertex prediction

Configuration files

https://github.com/LArSoft/larrecodnn/blob/develop/larrecodnn/NuGraph/nugraph.fcl

- Define input collections
- Feature normalization values
- Choice of decoders
- File name of compiled model

```
BEGIN_PROLOG

NuGraph: {
   planes: ["u","v","y"]
   hitInput: "nuslhits"
   spsInput: "sps"
   minHits: 10
   debug: false
   avgs_u: [389.00632, 173.42017, 144.42065, 4.5582113]
   avgs_v: [3.6914261e+02, 1.7347592e+02, 8.5748262e+08, 4.4525051e+00]
   avgs_y: [547.38995, 173.13017, 109.57691, 4.1024675]
   devs_u: [148.02893, 78.83508, 223.89404, 2.2621224]
   devs_v: [1.4524565e+02, 8.1395981e+01, 1.0625440e+13, 1.9223815e+00]
   devs_y: [284.20657, 74.47823, 108.93791, 1.4318414]
   filterDecoder:   true
   semanticDecoder: true
   vertexDecoder:   false
   modelFileName: "model.pt"
   module_type: "NuGraphInference"
}

END_PROLOG
```

Actual fcl file for running the job is:

https://github.com/LArSoft/larrecodnn/blob/develop/larrecodnn/NuGraph/testinference_uB_slice_job.fcl

NuGraphAnalyzer_module

https://github.com/LArSoft/larrecodnn/blob/develop/larrecodnn/NuGraph/NuGraphAnalyzer_module.cc

- Simple analyzer that reads back the GNN output from the event record and fills a root TTree with the results
  - leverages larsoft proxies and MVADescription for user-friendly access of results

```
void NuGraphAnalyzer::analyze(art::Event const& e)
{

  art::Handle<anab::MVADescription<5>> GNNDescription;
  e.getByLabel(art::InputTag("NuGraph", "semantic"), GNNDescription);

  auto const& hitsWithScores = proxy::getCollection<std::vector<recob::Hit>>(
    e,
    GNNDescription->dataTag(), //tag of the hit collection we ran the GNN on
    proxy::withParallelData<anab::FeatureVector<1>>(art::InputTag("NuGraph", "filter")),
    proxy::withParallelData<anab::FeatureVector<5>>(art::InputTag("NuGraph", "semantic")));

  std::cout << hitsWithScores.size() << std::endl;
  for (auto& h : hitsWithScores) {
    const auto& assocFilter = h.get<anab::FeatureVector<1>>();
    const auto& assocSemantic = h.get<anab::FeatureVector<5>>();
    _event = e.event();
    _subrun = e.subRun();
    _run = e.run();
    _id = h.index();
    _x_filter = assocFilter.at(0);
    _MIP = assocSemantic.at(GNNDescription->getIndex("MIP"));
    _HIP = assocSemantic.at(GNNDescription->getIndex("HIP"));
    _shower = assocSemantic.at(GNNDescription->getIndex("shower"));
    _michel = assocSemantic.at(GNNDescription->getIndex("michel"));
    _diffuse = assocSemantic.at(GNNDescription->getIndex("diffuse"));
    _tree->Fill();
  }
}
```

Recipe for Testing

- source /cvmfs/uboone.opensciencegrid.org/products/setup_uboone.sh
- setup uboonecode v09_83_01 -q e26:prof
- cp /path/to/model.pt .
- export FW_SEARCH_PATH=$FW_SEARCH_PATH:./
- lar -c testinference_uB_slice_job.fcl -n 1 --nskip 31 -s inputfile.root
- lar -c nugraph_analyzer.fcl -n -1 -s inputfile_*-testinference.root
- root -l NuGraphTree.root

- As input file you can use files from the MicroBooNE public dataset, e.g.
  - xroot://fndca1.fnal.gov:1095//pnfs/fnal.gov/usr/uboone/persistent/PublicAccess/prodgenie_bnb_nu_uboone_overlay_mcc9.1_v08_00_00_26_filter_run1_reco2_reco2/PhysicsRun-2016_5_20_12_58_43-0006343-00065_20160521T005619_ext_unbiased_20160521T032257_merged_gen_20190510T072402_g4_detsim_mix_r1a_r1b_postdlmctrut_d96f7194-172b-4803-94c4-15bcb8121a21.root

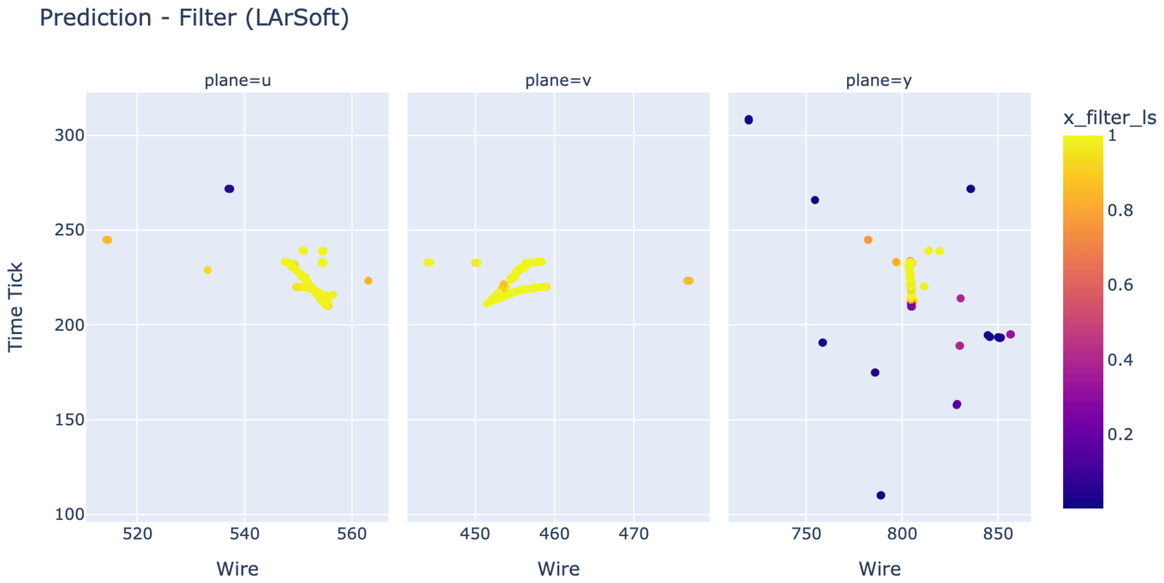
Resource usage

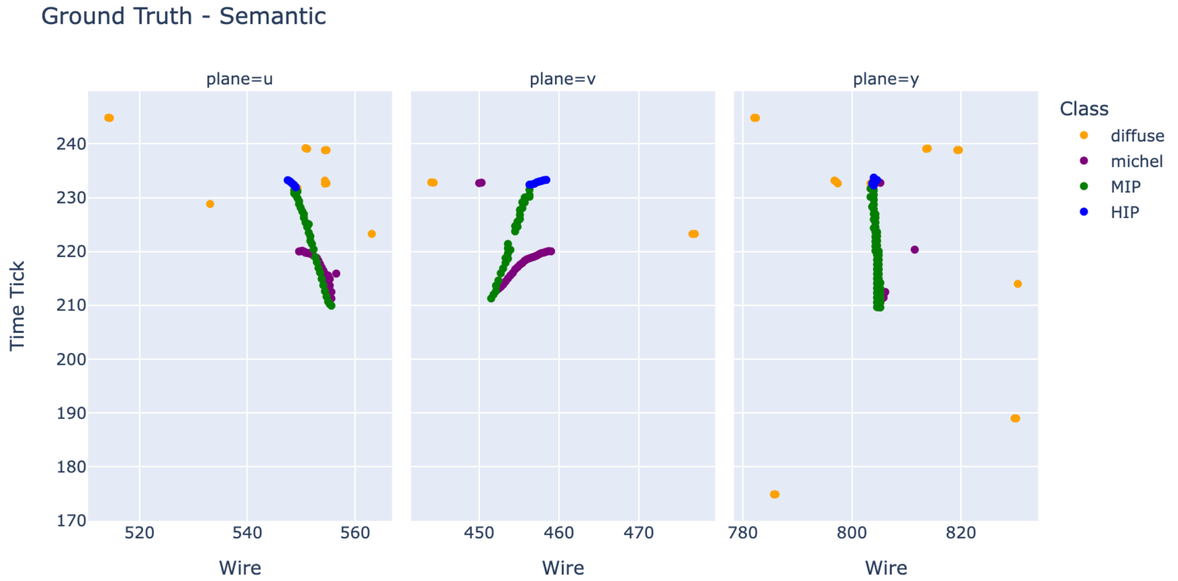
- Inference module takes 0.75 s per event event, including graph construction
  - however about half of the events are skipped due to too few hits
- Other upstream modules are also fast
  - However, from experience with creating the training dataset, SpacePointSolver may take longer and use significant memory for busy events.
    - Exploring alternatives (e.g. Cluster3D) or a better tuning of the parameters may be required.
- Peak resident set size usage: 2570.68 MB
  - would be nice if we could get it below 2 GB

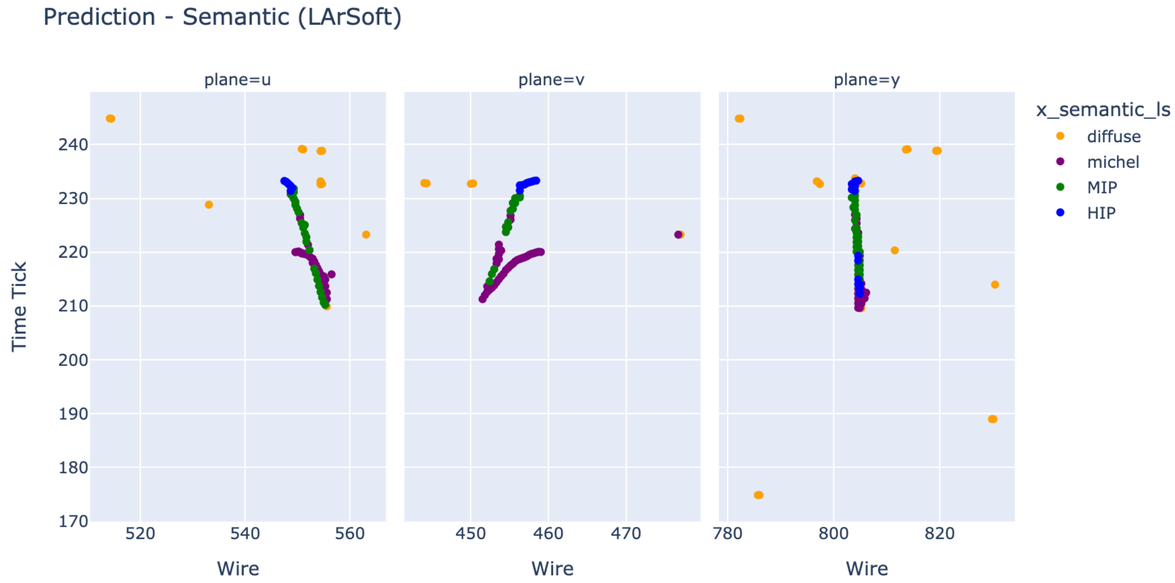
TimeTracker printout (sec)	Min	Avg	Max	Median	RMS	nEvts
Full event	0.0450458	3.36097	87.7468	0.237533	12.1151	74
source:RootInput(read)	0.000725606	0.00255304	0.019421	0.00131291	0.00392539	74
reco:nuslhits:NuSliceHitsProducer	0.0411265	0.116099	0.55599	0.0900547	0.0817036	74
reco:sps:SpacePointSolver	0.000110578	2.48479	85.3879	0.000217748	11.6239	74
reco:NuGraph:NuGraphInference	4.7356e-05	0.74844	5.22709	8.83935e-05	1.14997	74
[art]:TriggerResults:TriggerResultInserter	1.4952e-05	2.38511e-05	6.7179e-05	2.1032e-05	9.54137e-06	74
end_path:rootOutput:RootOutput	2.915e-06	4.5257e-06	1.9485e-05	3.9445e-06	2.18303e-06	74
end_path:rootOutput:RootOutput(write)	0.000867838	0.008697	0.0783238	0.00176224	0.0132931	74

Some examples of event displays from the output tree

r/s/e : 6343/70/3506

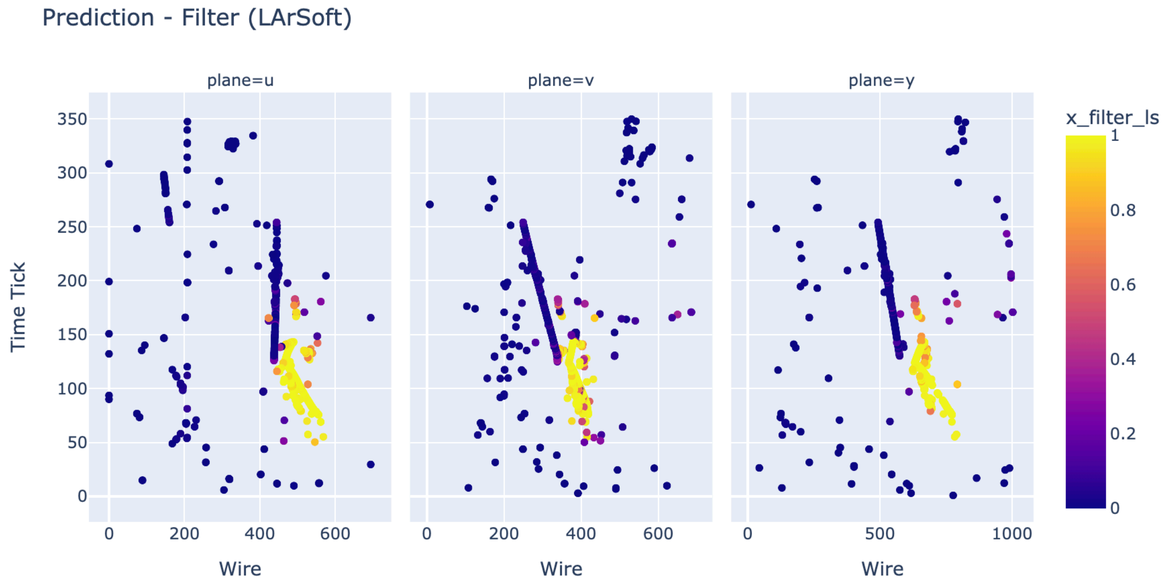


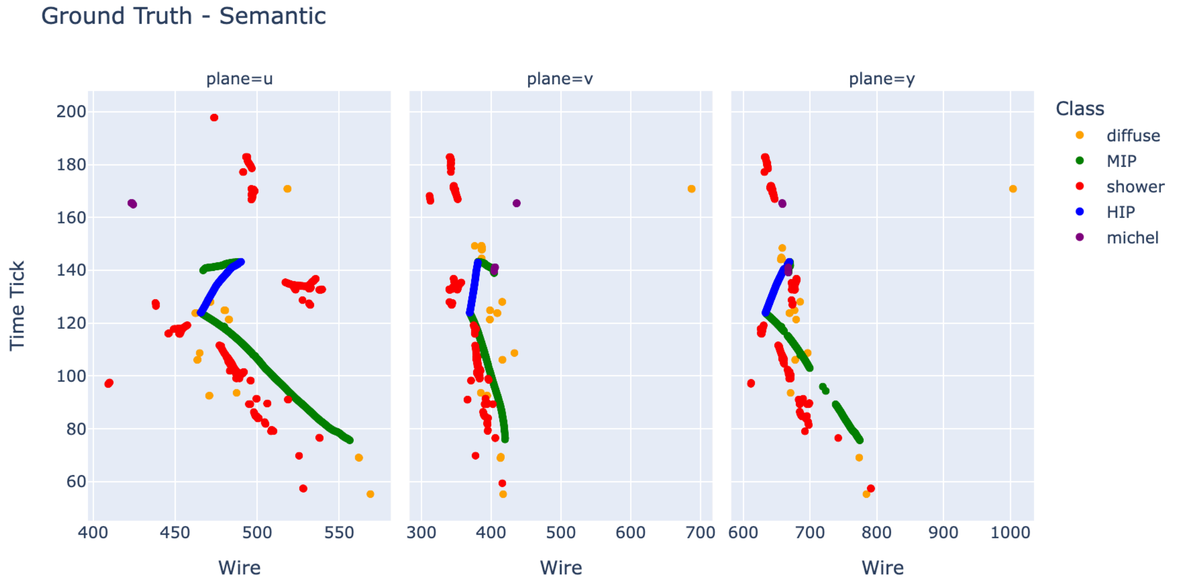


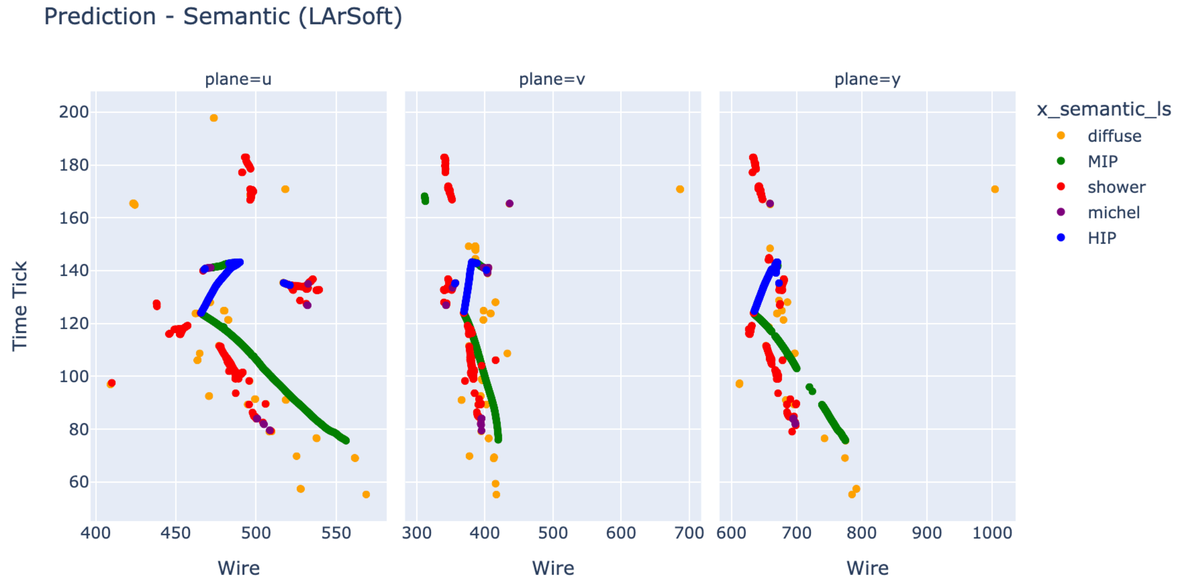


Some examples of event displays from the output tree

r/s/e : 6343/72/3506







Testing other experiments than MicroBooNE

- MicroBooNE specific items in the recipe above:
  - model file has been trained with MicroBooNE samples
    - default feature normalization values in nugraph.fcl also correspond to the training sample above
  - configuration file needs MicroBooNE services (for SpacePointSolver, not for NuGraph)
  - if other experiments need features other than WireID, PeakTime, Integral, RMS those need to be added
  - if other models need decoders other than semantic, filter, and vertex those need to be added

- Unless you want to run NuGraph2 from the uB open dataset you need to export your model
- We need a way to customize the feature and decoder list

From my talk at the November workshop

Saving the model in TorchScript with JIT

- In order to load the module in C++ we need to compile and save it with JIT
  - https://pytorch-geometric.readthedocs.io/en/latest/advanced/jit.html
- A modified (hacked) version of NuGraph does this:
  - https://github.com/exatrkx/NuGraph/compare/feature/cerati_jit-brute-force

- Thanks to V's work NuGraph was almost compatible with TorchScript already. Summary of changes required:
  - Remove checkpointing (not used in inference)
  - Avoid relying on inheritance of decoder classes from common base class
  - And some other changes:
    - add/fix annotations about return types
    - add ".jittable()" to various object instantiations
    - initializing objects to the correct type (i.e. not to None)
    - add "propagate_type" to MessagePassing
    - add "@torch.jit.unused" to functions that are not used
    - loop over self.net.items instead of self.net (with self.net = nn.ModuleDict())

scripts/test.py

```
 7    import nugraph as ng
 8    import pynuml
 9    import tqdm
10  + import torch
11
12    Data = ng.data.H5DataModule
13    Model = ng.models.NuGraph2

29        print('using checkpoint =',args.checkpoint)
30        model = Model.load_from_checkpoint(args.checkpoint, map_location='cpu')
31
32  +     script = model.to_torchscript()
33  +     print(script)
34  +     torch.jit.save(script, "model.pt")
35  +
```

Fermilab

3 2023/11/13

We need to work on a proper merge of the JIT-table version in the main branch.

Fermilab

Factorizing the Inference Module

- A possible solution is to make use of art tools to factorize parts of the module and customize the feature extraction
  - https://cdcvs.fnal.gov/redmine/projects/art/wiki/General_design_considerations#When-more-flexibility-is-required
  - “There are times when there are aspects of a module that cannot (or should not) be broken apart into separate modules, and for which the author of the code desires to allow users of the module to specify their own behavior. “

An example of how this could be accomplished in a module is:

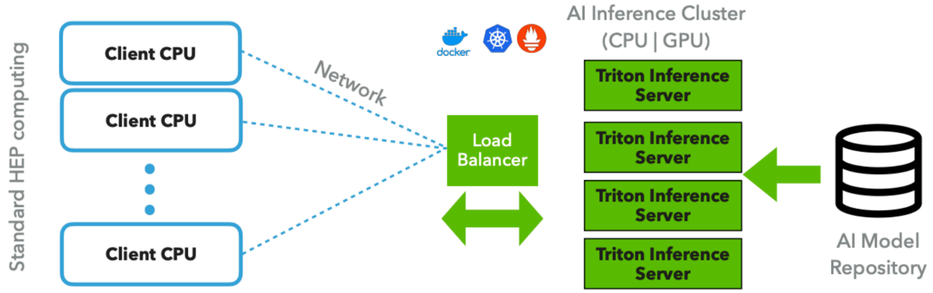
```
class TracksFromHits : public art::EDProducer {
public:

  explicit TracksFromHits(fhicl::ParameterSet const& ps) :
    hitsTag_{ps.get<art::InputTag>("hitsTag")},
    clusteringAlgo_{art::make_tool<Clusters(Hits const&)>(ps.get<fhicl::ParameterSet>("clusteringAlgorithm"), "Clustering")}
  {
    produces<Tracks>();
  }

  void produce(art::Event& e)
  {
    auto const& hits = *e.getValidHandle<Hits>(hitsTag_);
    auto const& clusters = clusteringAlgo_(hits);
    e.put(makeTracks(clusters));
  }

};
```

Future…

From my talk at the November workshop

Other options: Triton/NuSonic

- Using libtorch directly in LArSoft works well for CPU-based workflows (fast inference time on CPU)
- Triton and NuSonic are an option for GPU workflows, or for CPU workflows in case we want to be independent from the distributed pytorch version
- How NuSonic works (arXiv:2009.04509):



- NuSonic supports pytorch models. And the LHC Exa.TrkX network was deployed on Triton as well. However due to constraints on the input format, some changes may be required on our code:
  - all tensors need to be in a single dictionary, while we currently have several different ones
    - see https://github.com/triton-inference-server/server/blob/main/docs/user_guide/model_configuration.md#inputs-and-outputs
- Sonic also should be able to support non-ML algorithms (e.g. Delaunay?) and is also working to support portability languages: see talk by K. Pedro

- NuSonic. Please reach out if you are interested!