Inference-as-a-Service in Gravitational Wave Physics

Alec Gunny[1,*], Ethan Marx[1], William Benoit[2], Saleem Muhammed[2], Ryan Raikman[1], Deep Chatterjee[1], Eric Moreno[1], Dylan Rankin[1], Michael Coughlin[2], Philip Harris[1], Erik Katsavounidis[1]

* - presenter
1 - Massachusetts Institute of Technology
2 - University of Minnesota

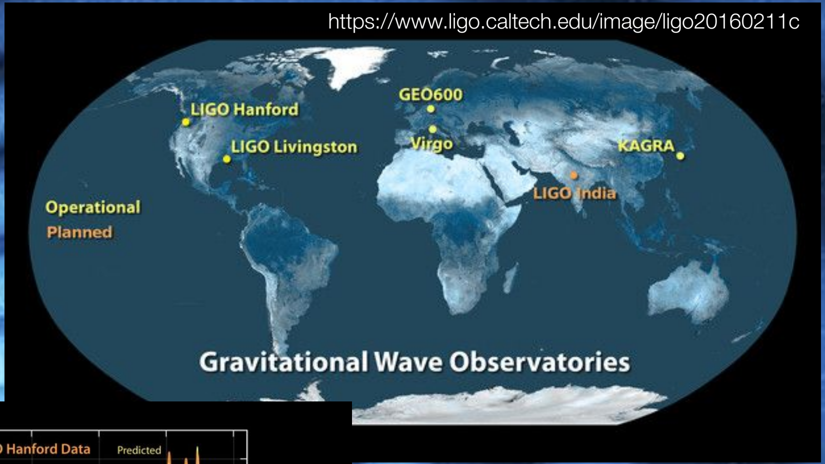
Gravitational Waves

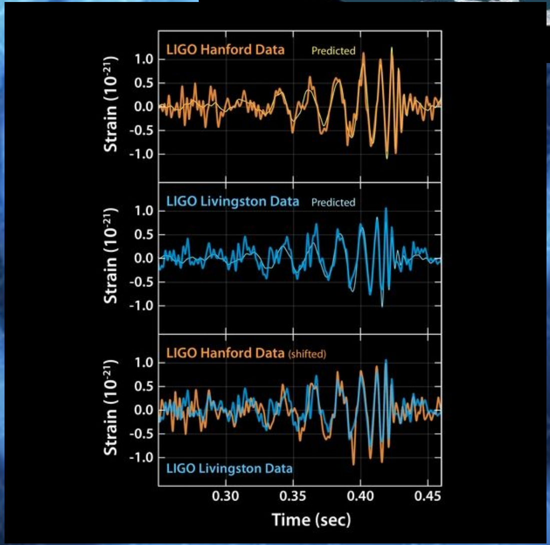
Large scale astrophysical events ripple the fabric of spacetime

International Gravitational Wave Observatory Network (IGWN) set up to detect, locate, and characterize events

Measure timeseries of unitless quantity - gravitational wave **strain**

https://www.ligo.caltech.edu/image/ligo20160211c





https://www.ligo.caltech.edu/image/ligo20160615f

https://www.ligo.caltech.edu/image/ligo20160211a

Detecting a Gravitational Wave

Characterize and remove noise in the gravitational wave strain channel

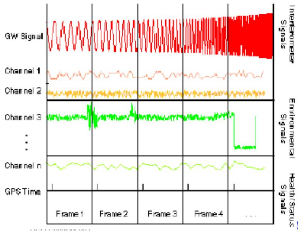
Gravitational-wave Detector Data

Continuous time series (1Hz, 128Hz … 16kHz)

Gravitational Wave channel: ~20GB/day (per instrument)

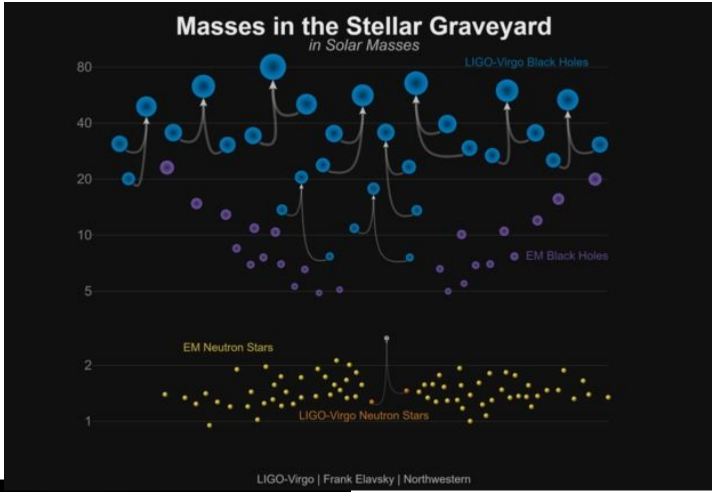
Physical Environment Monitors (seismometers, accelerometers, magnetometers, microphones etc)

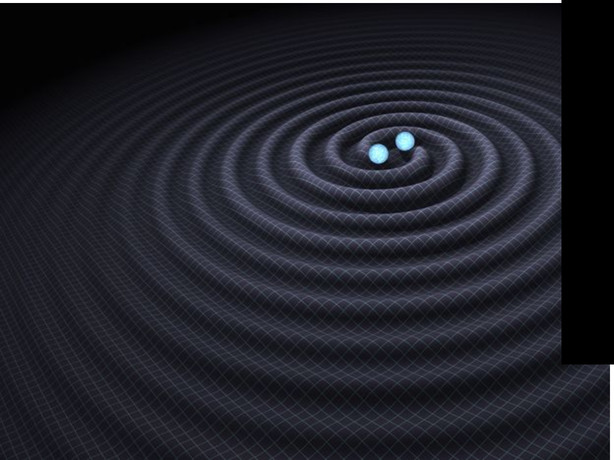
Internal Engineering Monitors (sensing, housekeeping, status etc)

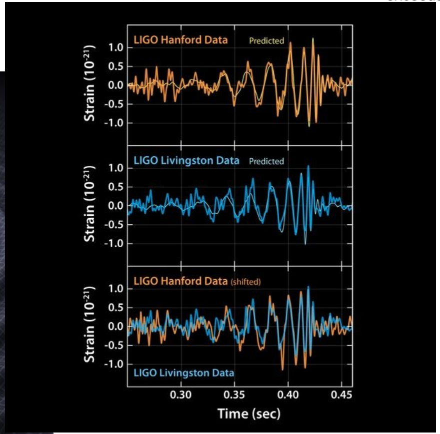


Together with various intermediate data products >2TB/day (per instrument)

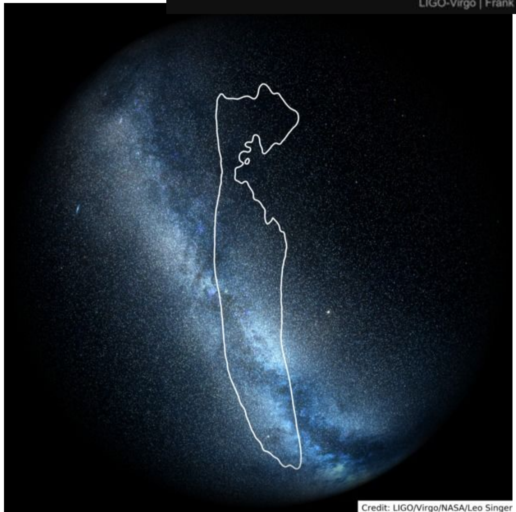
Initial and Enhanced LIGO archive (2002-2010) exceeds 1PB of data







Detect an event



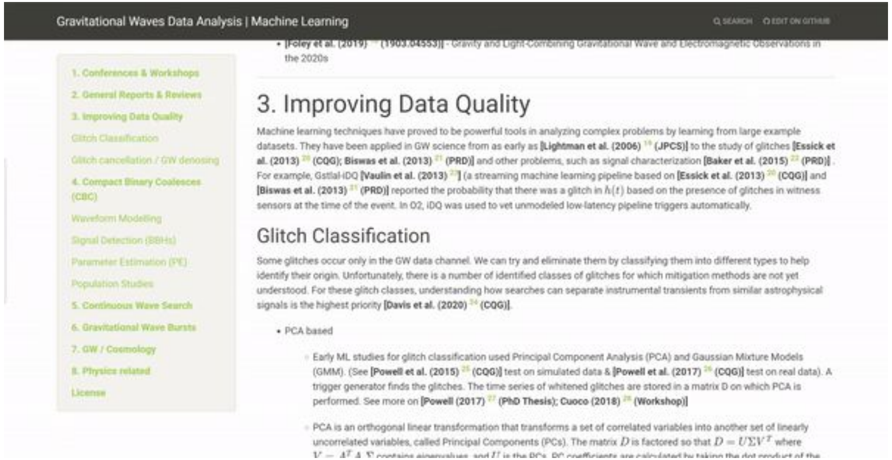
Characterize it:

Where in the sky?

How big?

ML Papers in GW Physics

ML Applications in Production



Gravitational Waves Data Analysis | Machine Learning

SEARCH EDIT ON GITHUB

- [Foley et al. (2019) (1903.04553)] - Gravity and Light-Combining Gravitational Wave and Electromagnetic Observations in the 2020s

1. Conferences & Workshops
2. General Reports & Reviews
3. Improving Data Quality
Glitch Classification
Glitch cancellation / GW denoising
4. Compact Binary Coalesces (CBC)
Waveform Modelling
Signal Detection (BBHs)
Parameter Estimation (PE)
Population Studies
5. Continuous Wave Search
6. Gravitational Wave Bursts
7. GW / Cosmology
8. Physics related
License

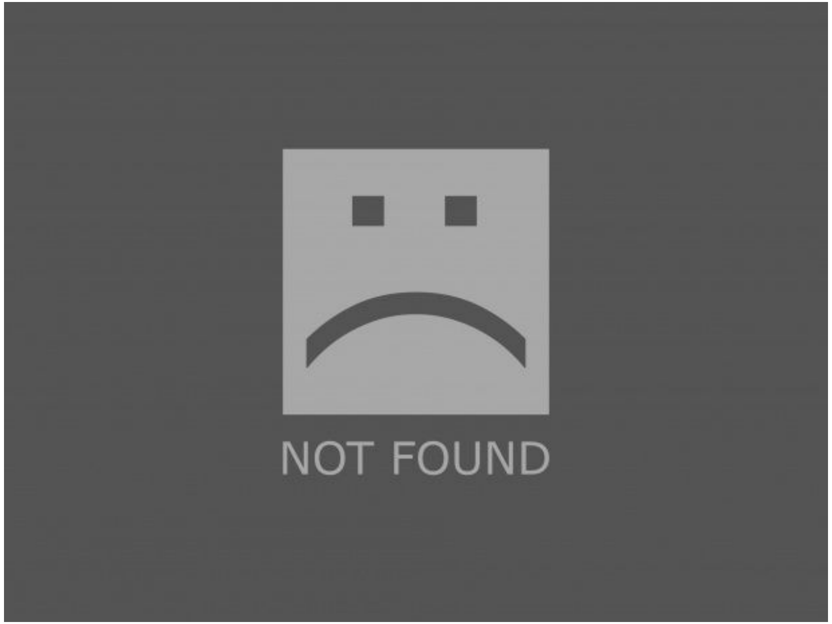
3. Improving Data Quality

Machine learning techniques have proved to be powerful tools in analyzing complex problems by learning from large example datasets. They have been applied in GW science from as early as [Lightman et al. (2006) [19] (JPCS)] to the study of glitches [Essick et al. (2013) [20] (CQG); Biswas et al. (2013) [21] (PRD)] and other problems, such as signal characterization [Baker et al. (2015) [22] (PRD)]. For example, GstlaL-iDQ [Vaulin et al. (2013) [23]] (a streaming machine learning pipeline based on [Essick et al. (2013) [20] (CQG)] and [Biswas et al. (2013) [21] (PRD)] reported the probability that there was a glitch in $h(t)$ based on the presence of glitches in witness sensors at the time of the event. In O2, iDQ was used to vet unmodeled low-latency pipeline triggers automatically.

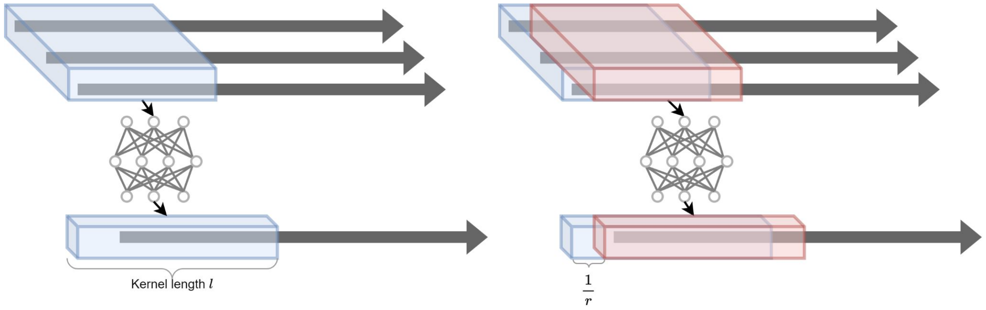
Glitch Classification

Some glitches occur only in the GW data channel. We can try and eliminate them by classifying them into different types to help identify their origin. Unfortunately, there is a number of identified classes of glitches for which mitigation methods are not yet understood. For these glitch classes, understanding how searches can separate instrumental transients from similar astrophysical signals is the highest priority [Davis et al. (2020) [24] (CQG)].

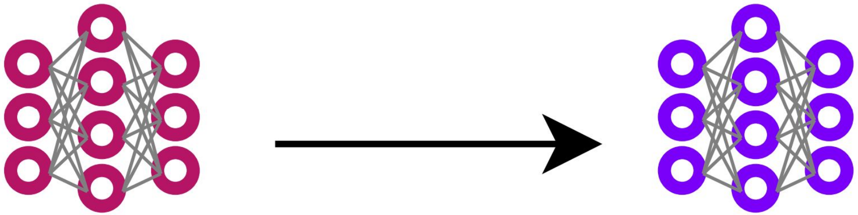
- PCA based
  - Early ML studies for glitch classification used Principal Component Analysis (PCA) and Gaussian Mixture Models (GMM). (See [Powell et al. (2015) [25] (CQG)] test on simulated data & [Powell et al. (2017) [26] (CQG)] test on real data). A trigger generator finds the glitches. The time series of whitened glitches are stored in a matrix D on which PCA is performed. See more on [Powell (2017) [27] (PhD Thesis); Cuoco (2018) [28] (Workshop)]
  - PCA is an orthogonal linear transformation that transforms a set of correlated variables into another set of linearly uncorrelated variables, called Principal Components (PCs). The matrix D is factored so that $D = U\Sigma V^T$ where



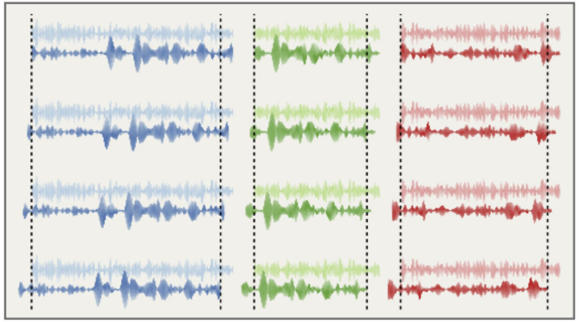
*not entirely true

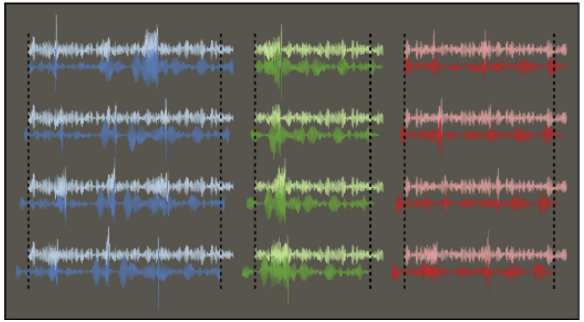


Requires high-throughput, low(-ish)-latency inference on heavily overlapping data



On-the-fly re-training and updating of model weights to reflect non-stationary noise





Need predictions on O(10)-O(1000)yrs of background and simulated events to estimate detection significance and false alarm rates

Deployment - Naive Example

```
import torch
from deepclean.architectures import DeepCleanAE as DeepClean

num_witnesses = 21
weights_path = "/path/to/weights.pt"
nn = DeepClean(num_witnesses).to(“cuda”)
nn.load_state_dict(torch.load(weights_path))

dataset = ...
for X, y in dataset:
    y_hat = nn(X)
    do_some_physics(y, y_hat)
```

Deployment - Naive Example

```
import torch
from deepclean.architectures import DeepCleanAE as DeepClean

num_witnesses = 21
weights_path = "/path/to/weights.pt"
nn = DeepClean(num_witnesses).to("cuda")
nn.load_state_dict(torch.load(weights_path))

dataset = ...
for X, y in dataset:
    y_hat = nn(X)
    do_some_physics(y, y_hat)
```

- DL software stack often unwieldy
- Lots of options - do we need to become experts in all of them?

Deployment - Naive Example

```
import torch
from deepclean.architectures import DeepCleanAE as DeepClean

num_witnesses = 21
weights_path = "/path/to/weights.pt"
nn = DeepClean(num_witnesses).to("cuda")
nn.load_state_dict(torch.load(weights_path))

dataset = ...
for X, y in dataset:
    y_hat = nn(X)
    do_some_physics(y, y_hat)
```

- Access to the model definition and weights
  - Do they match?
  - Do they represent the most up-to-date work?

Deployment - Naive Example

```
import torch
from deepclean.architectures import DeepCleanAE as DeepClean

num_witnesses = 21
weights_path = "/path/to/weights.pt"
nn = DeepClean(num_witnesses).to(“cuda”)
nn.load_state_dict(torch.load(weights_path))

dataset = ...
for X, y in dataset:
    y_hat = nn(X)
    do_some_physics(y, y_hat)
```

- Accessing, using, and saturating the compute capacity of accelerators is non-trivial

Deployment - Naive Example

```
import torch
from deepclean.architectures import DeepCleanAE as DeepClean

num_witnesses = 21
weights_path = "/path/to/weights.pt"
nn = DeepClean(num_witnesses).to("cuda")
nn.load_state_dict(torch.load(weights_path))

dataset = ...
for X, y in dataset:
    y_hat = nn(X)
    do_some_physics(y, y_hat)
```

Just a single function call!

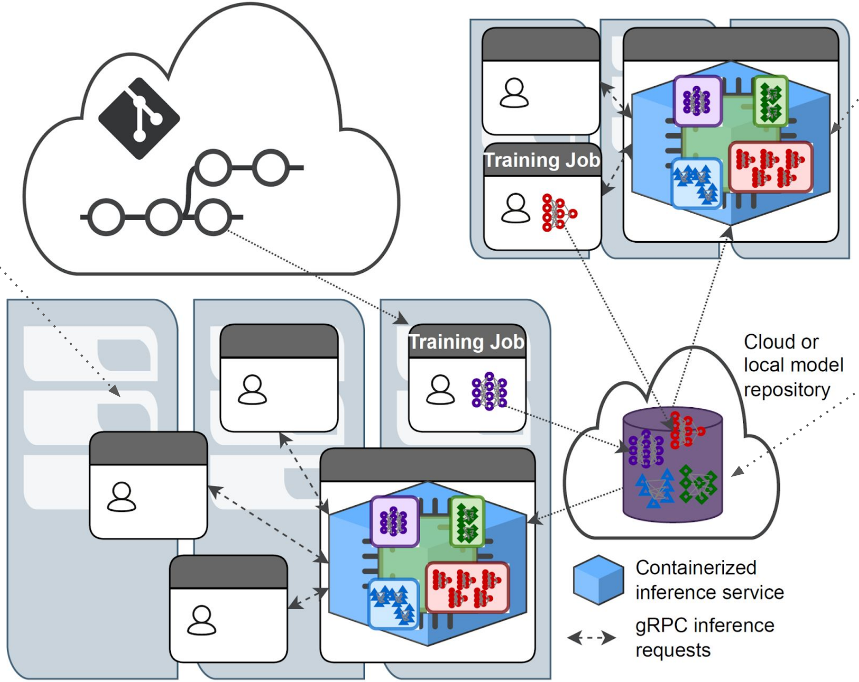
Deployment - Naive Example

```
import torch
from deepclean.architectures import DeepCleanAE as DeepClean

num_witnesses = 21
weights_path = "/path/to/weights.pt"
nn = DeepClean(num_witnesses).to("cuda")
nn.load_state_dict(torch.load(weights_path))

dataset = ...
for X, y in dataset:
    y_hat = nn(X)
    do_some_physics(y, y_hat)
```

This is where you should be spending your energy!

Inference-as-a-Service (IaaS)

Client applications leverage standardize APIs that abstract the details of the hardware, software, or even particular ops used to perform inference



Inference is handled by blackbox application to which users send requests

Models are hosted and versioned in a central model repository from which all deployments read

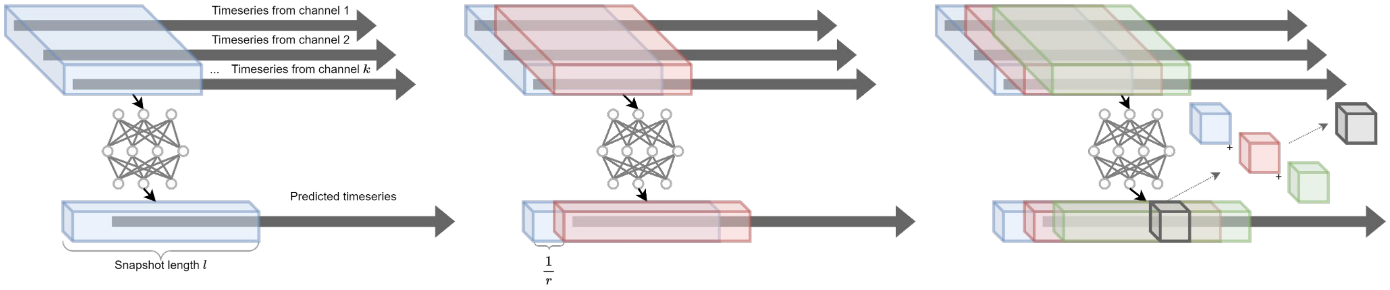
Off-the-shelf solution: Triton Inference Server

- IaaS application built by NVIDIA
  - Parallel and ensemble scheduling
  - Support for heterogeneous hardware and software environments
  - Non-interrupting model updates
  - Metrics endpoint for monitoring throughput and latency
- Drawbacks
  - Lots of boilerplate
  - Non-pythonic protobufs

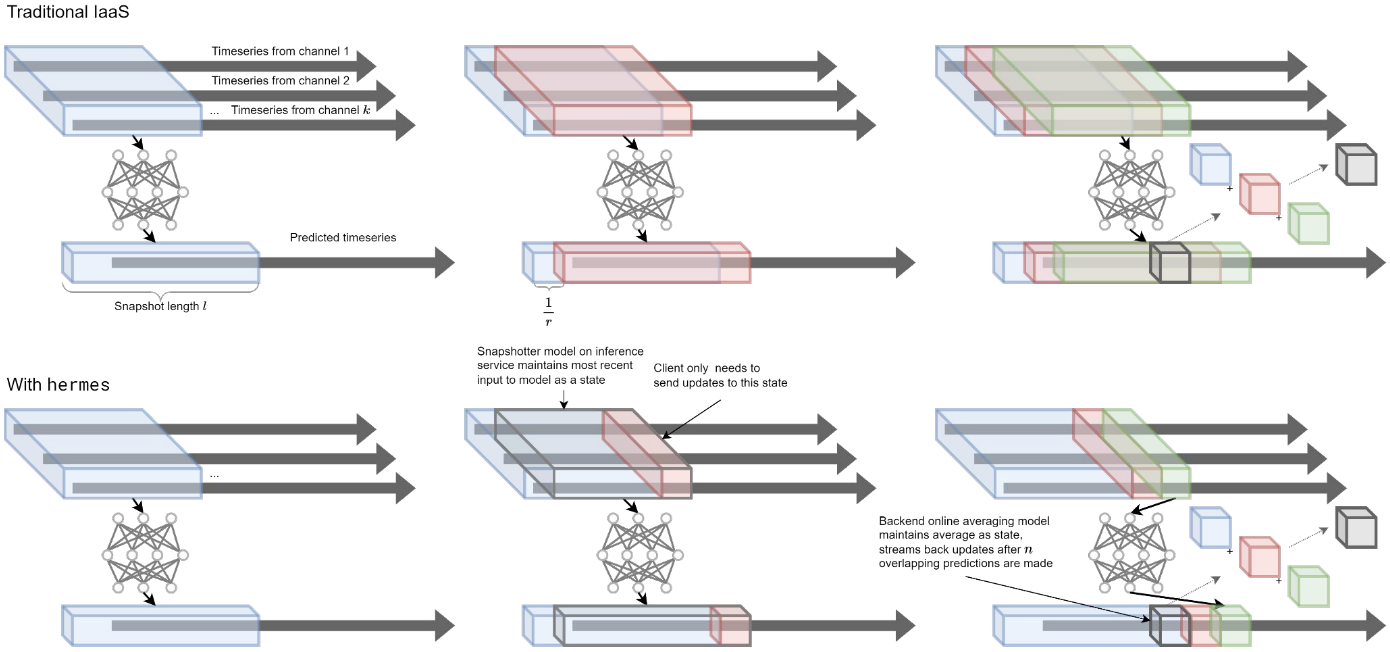
IaaS challenges for streaming timeseries data

Overlapping input/output windows lead to redundant data transfer

Traditional IaaS

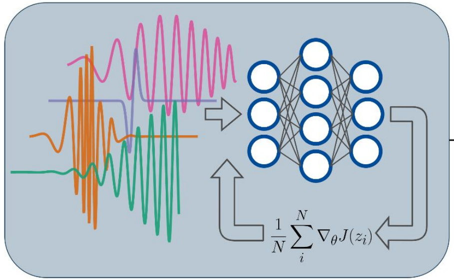


hermes - IaaS deployment utilities



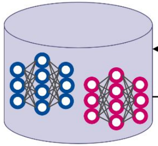
https://github.com/ml4gw/hermes

hermes - IaaS deployment utilities

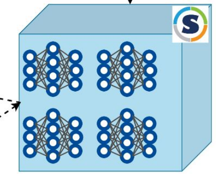


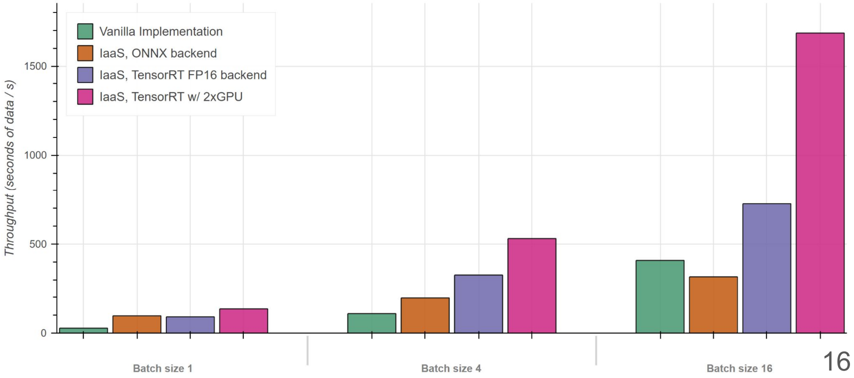

```python
from hermes import quiver as qv

# tf also supported
my_nn = torch.nn.Sequential(...)
my_nn.load(torch.load_state_dict("/path/to/weights.pt"))

repo = qv.ModelRepository("/path/to/repos")
model = repo.add("my-nn", platform=qv.Platform.ONNX)

# tensorflow export infers shapes/names from graph
model.export_version(
    my_nn, input_shapes={"h_of_t": [...]}, output_names=["det_stat"]
)
# model.versions == [1]

# subsequent export calls infer shapes/names from config
model.export_version(my_nn)
# model.versions == [1, 2]

# built-in support for streaming-inference
ensemble = repo.add("my-nn-stream", platform=qv.Platform.ENSEMBLE)
ensemble.add_streaming_inputs(
    model.inputs["h_of_t"],
    stream_size=SAMPLE_RATE // r,
    batch_size=16
)
ensemble.add_output(model.outputs["det_stat"])
ensemble.export_version(None)
```

- Infers information from model graph/config to reduce boilerplate
- Dependencies kept separate to make deployments modular and lightweight
- Built-in support for TensorRT conversion with mixed-precision
- Full (WIP) example: https://alecgunny.github.io/hermes-examples

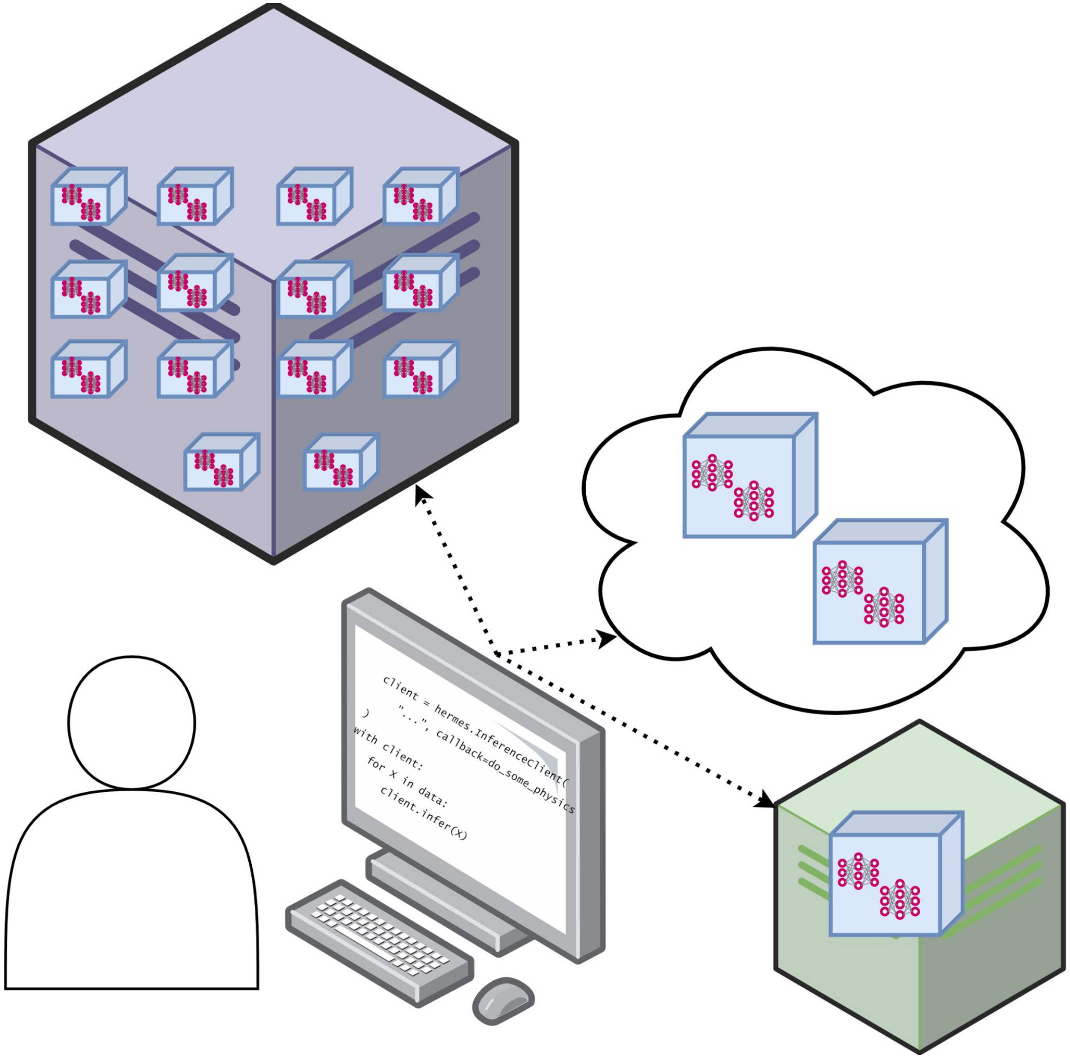


```python
from hermes.aeriel.client import InferenceClient
from hermes.aeriel.serve import serve


def callback(response, request_id, sequence_id):
    return do_some_physics(response)


batch_size = 16
step_size = batch_size * (SAMPLE_RATE // r)
with serve("/path/to/model-repo", gpus=[0, 1, 2, 3]) as instance:
    data = load_data_in_parallel()
    instance.wait()
    client = InferenceClient(
        "localhost:8001", "my-nn-stream", callback=callback
    )
    with client:
        start, stop = 0, step_size
        while stop < len(data):
            update = data[start: stop]
            client.infer(update, sequence_id=1001, sequence_start=start == 0)
            start, stop = start + step_size, stop + step_size
```





True IaaS - extend to large-scale deployment scenarios across heterogeneous computing environments



Thank you!