FourCastNet - An Example for Deep Learning for Earth Sciences in the HPC Context

Thorsten Kurth*, Jaideep Pathak, Morteza Mardani,
David Hall, Karthik Kashinath (NVIDIA)
Shashank Subramanian, Peter Harrington (LBL)
Sanjeev Raja (U-M), Kamyar Azizzadenesheli (Purdue U)
Ashesh Chattoadhyay, Pedram Hassanzadeh (Rice U)
Zongyi Li, Animashree Anandkumar (CalTech)

MAELSTROM Workshop, 28.03.2022

NVIDIA · Lawrence Berkeley National Laboratory · Purdue University · Rice · Caltech

Opportunities of Deep Learning for Numerical Weather Prediction

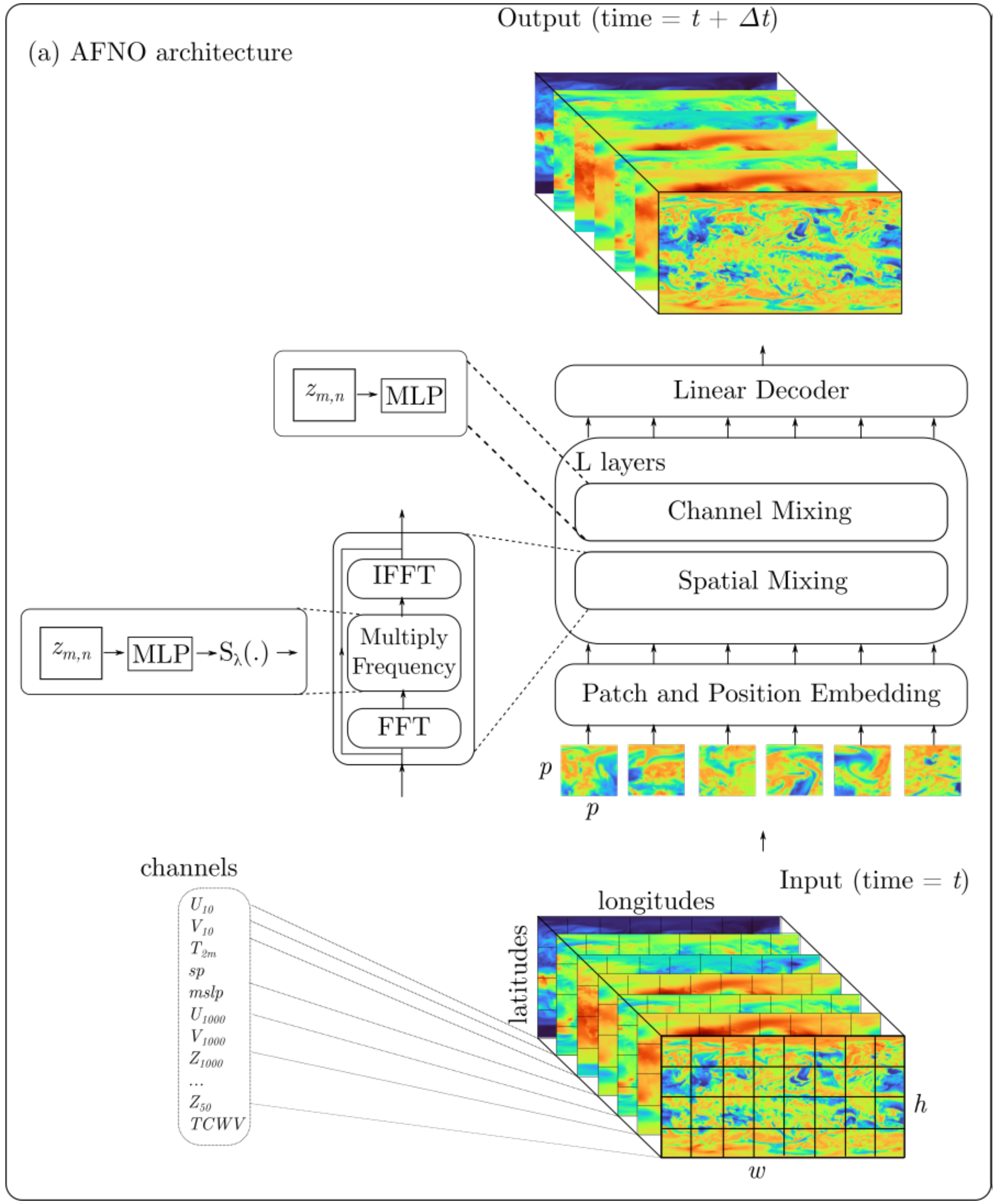
- Data-driven models can ingest any kind of data (observations, reanalyzed data, etc.)
- Data-driven models could overcome model biases of traditional NWP
- Training process can be augmented to better predict extreme events or whatever the practitioner cares about (precipitation, wind, etc.)
- Rapid prediction process:
  generate large ensembles of forecasts

Challenges of Deep Learning for Numerical Weather Prediction

- Large receptive field
  - NN needs to learn to correlate large scale features
- High resolution in space and time
  - ~5° resolution translates into 32x64 pixel grid,
    i.e. best spatial resolution is limited by ~500km
  - Target resolution required for beating state of the art NWP: ~0.1°
- Lot of training samples required
- **Expensive training process:** leverage HPC resources

FourCastNet Architecture

- AFNO: Adaptive Fourier Neural Operator
- Vision Transformer (ViT) Model for NWP
- Tokenization of input data to reduce spatial complexity and overall memory footprint (patch embedding)
- Global continuous mixing of tokens using FFT (large scale)
- Intra-token mixing using fully connected layers (small scale)
- Repetitions of transformer blocks (12x)
- Predictions on full input resolution grid



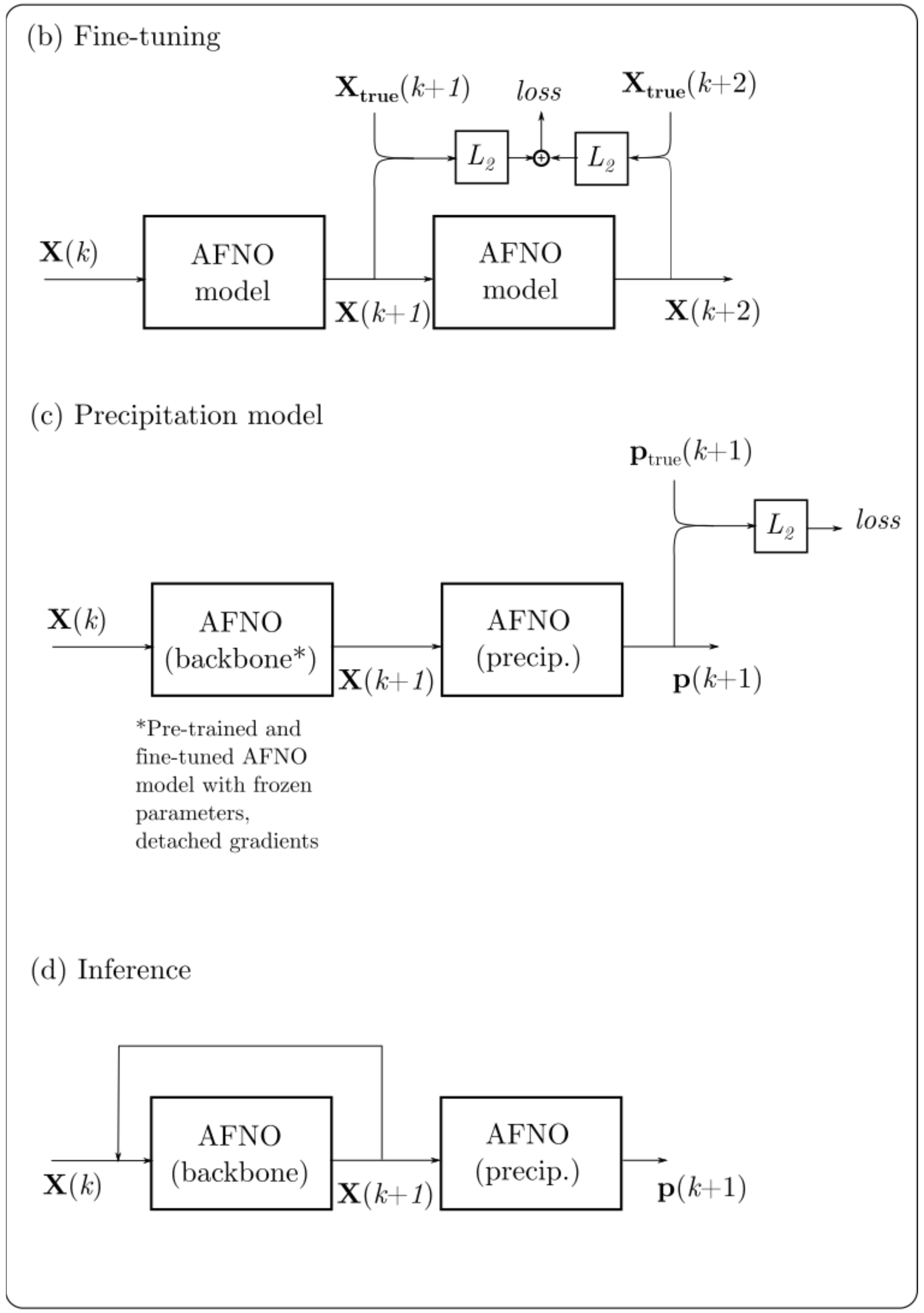
Target Dataset

- ERA5 dataset
- Goal: predict surface wind velocities and precipitation
- Resampling to regular euclidian grid: 720x1440@6h resolution
- Use 20 input features per grid point (cf. table)
- Train: 1979-2015
  Eval: 2016, 2017
  Test: 2018+
- Total size: ~5TB

Level	Variables
Surface	U_{10}, V_{10}, T_{2m}, sp, mlsp
1000hPa	U, V, Z
850hPa	T, U, V, Z, RH
500hPa	T, U, V, Z, RH
50hPa	Z
Integrated	TCWV

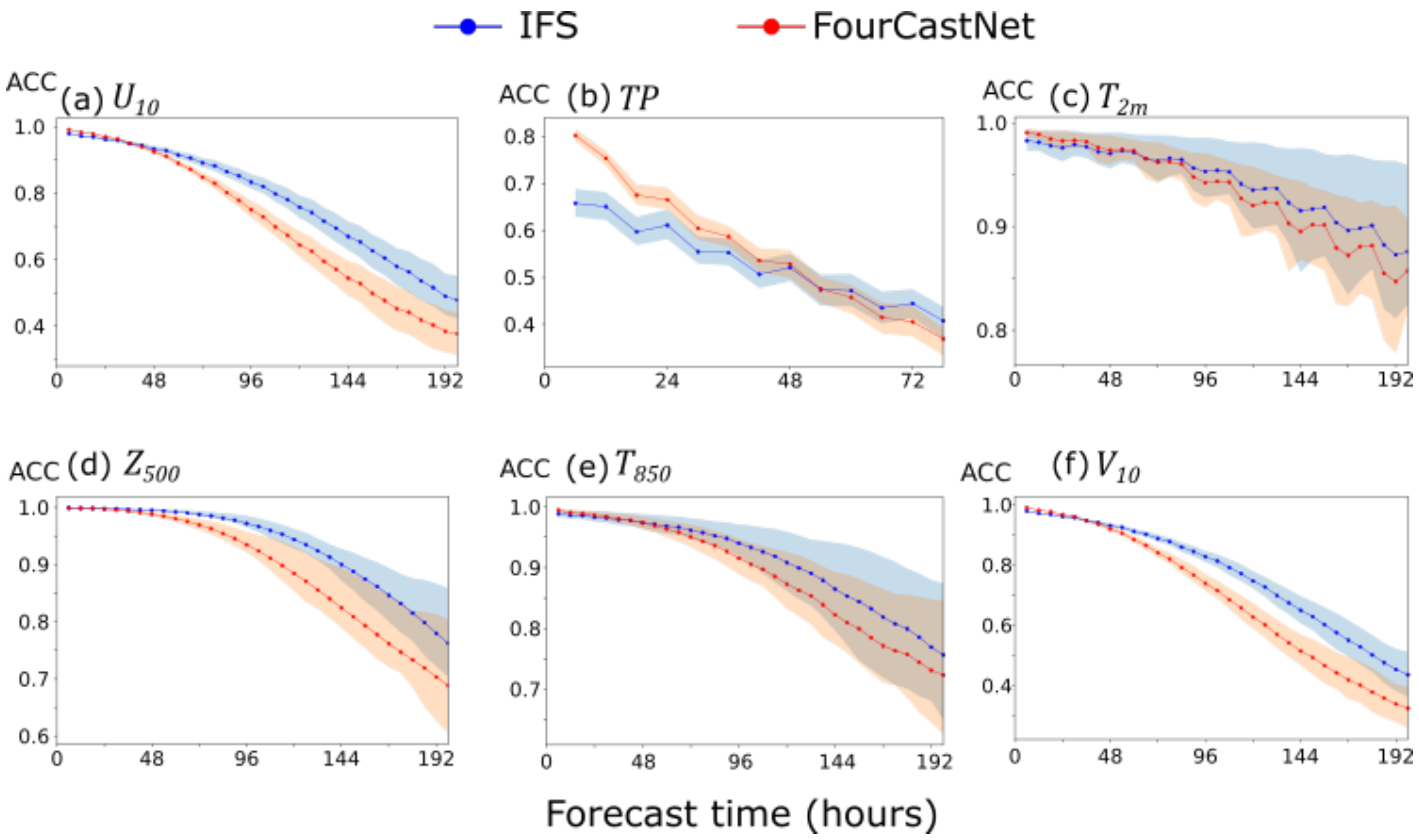
FourCastNet Training Workflow

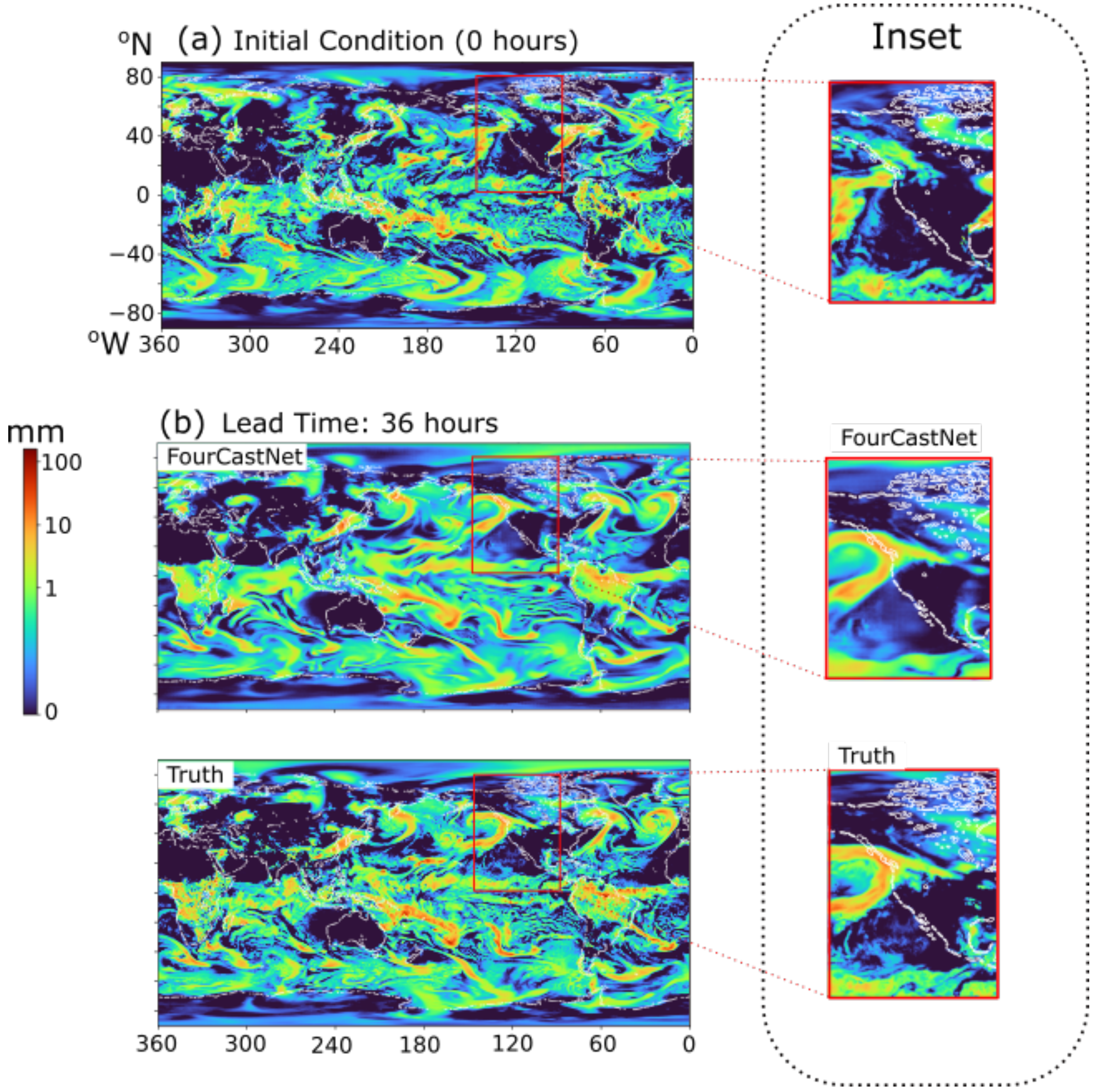
- Pre-training predicting X(k+1) from X(k)
- Fine-tuning with 2-step training: predicting X(k+1) from X(k) and then X(k+2) from the previous prediction
- For precipitation: fine-tuned and frozen AFNO model with precipitation augmentation (integrated precipitation between k+1 and k+2)
- Inference model has autoregressive component



Forecasting Results - General

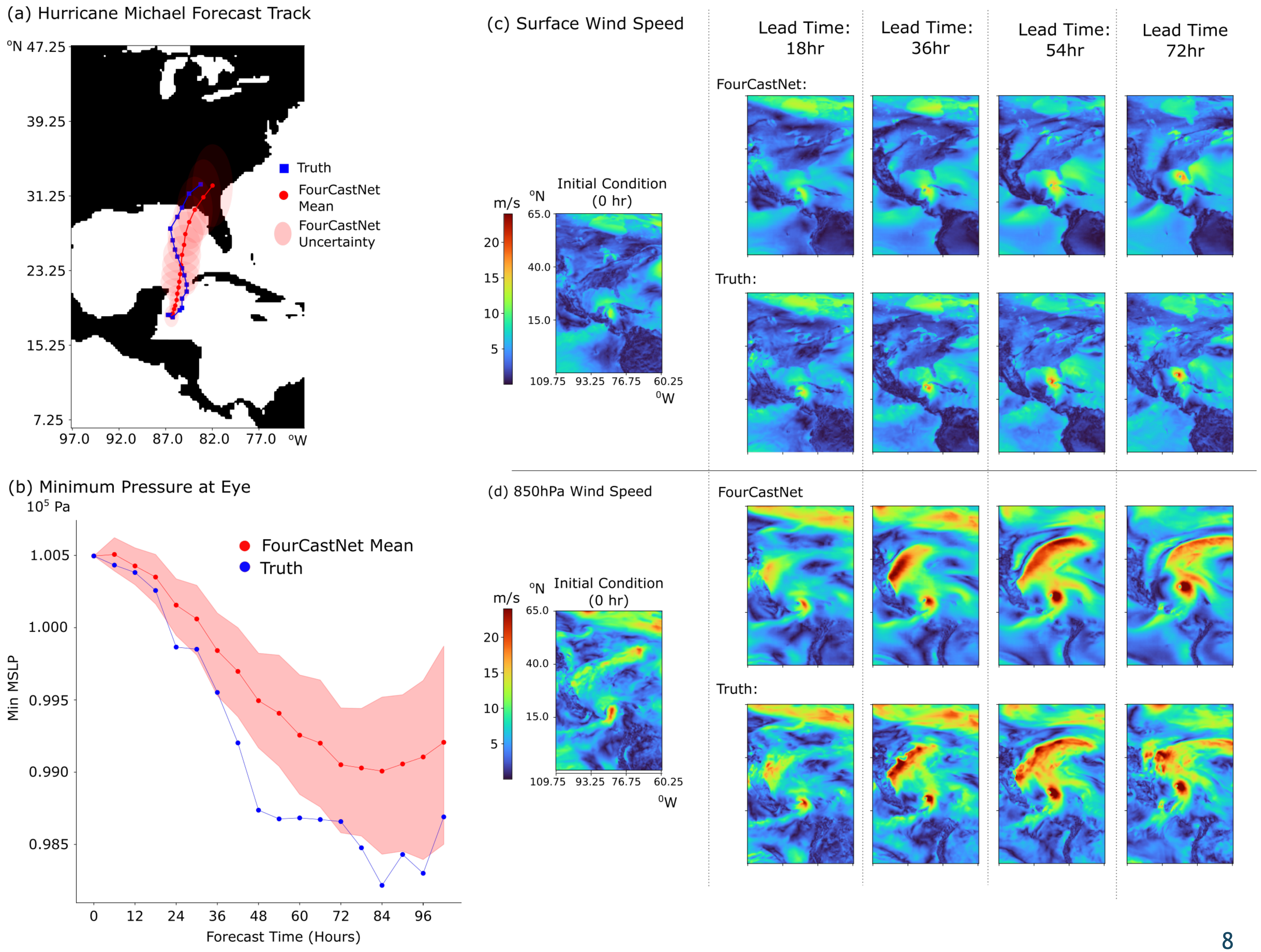
- FourCastNet reproduces structures
- Prediction Skill (ACC) close to IFS up to 8 days into the future





Forecasting Results - Hurricane Michael

- Excellent skill in forecasting fine scale, highly dynamic observables
- Real track within 90% CI of FourCastNets ensemble forecast
- Pressure curve for Eye follows prediction



FourCastNet and HPC

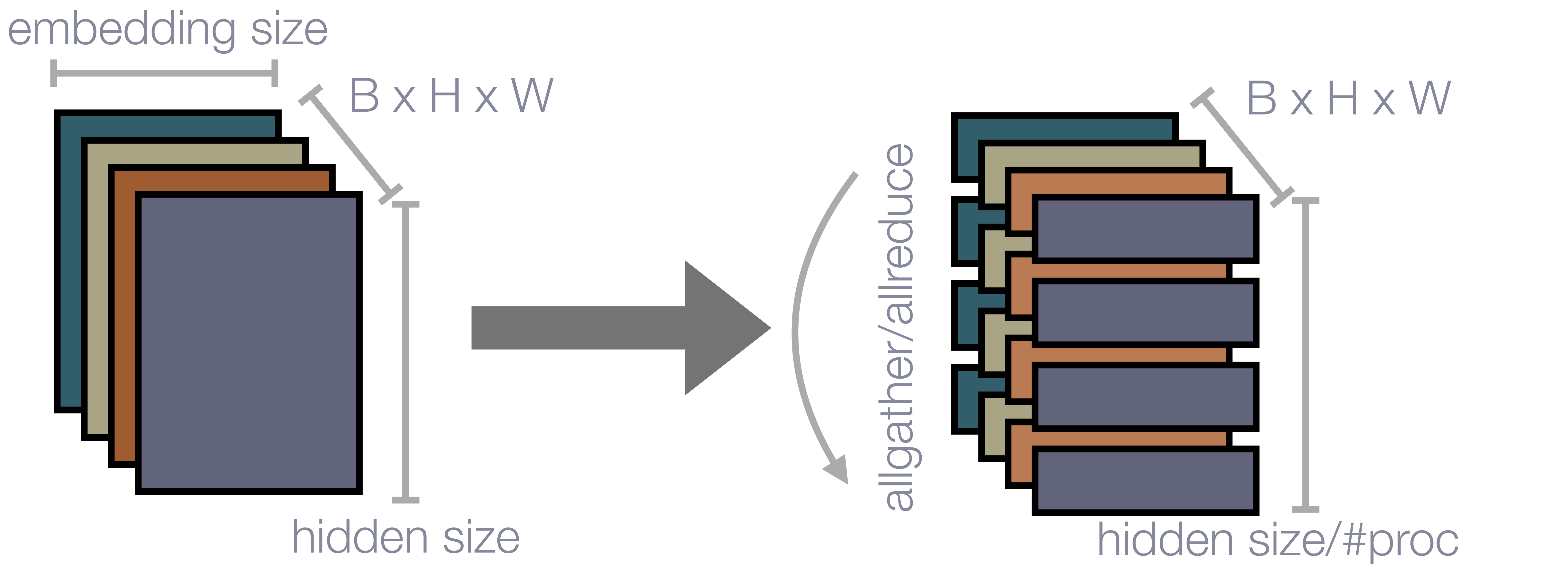
- Results have not reached 0.1° spatial resolution target
- Fine scale resolution also dependent on ViT patch embedding size
- Temporal resolution can be increased as well
- Resolution and patch embedding size scale exponentially with memory footprint
- **Parallelization needed**

Resolution	Patch Size	#Transformer Blocks	Memory Footprint scaffolding/ running [GB]*
0.25°	8	12	9.3 / 3.4
0.25°	4	12	32.4 / 7.0
0.25°	2	12	80+**
0.1°	4	24	405.0** / 87.5**

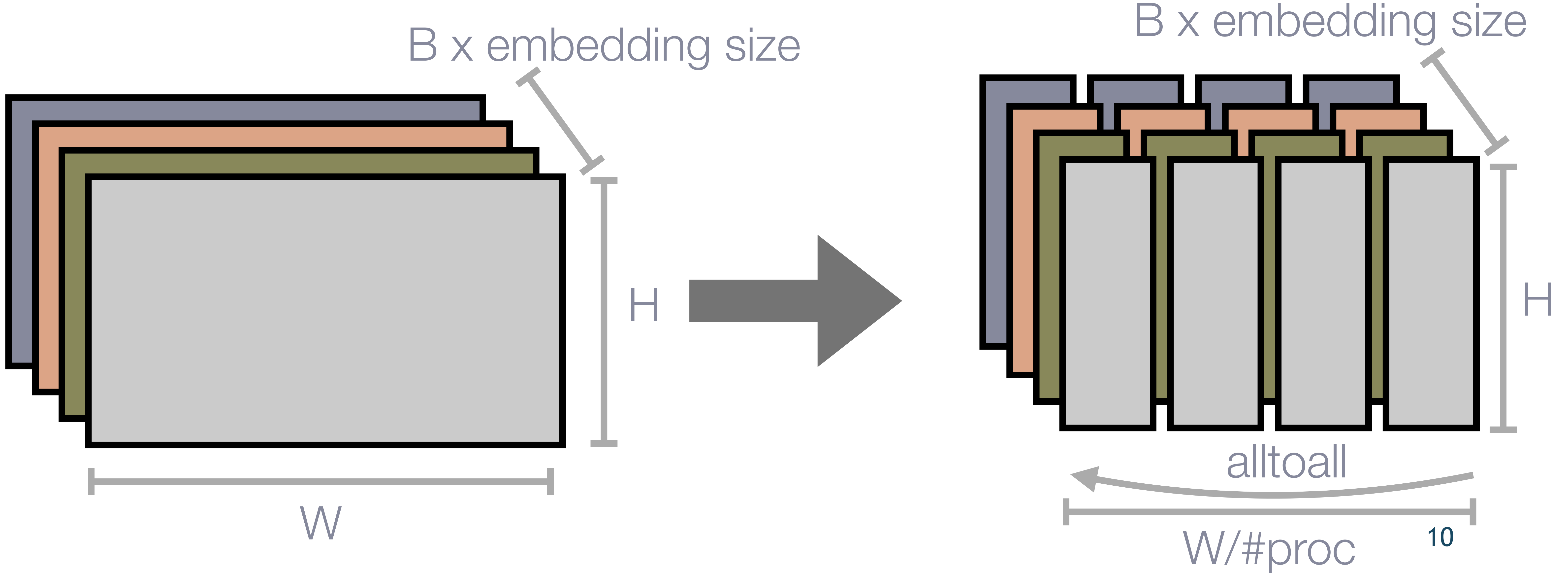
*excluding IO pipeline **extrapolated

Parallelization methods I

- **Feature parallelization**
  splitting channel dim, dense layers become distributed matrix multiplications

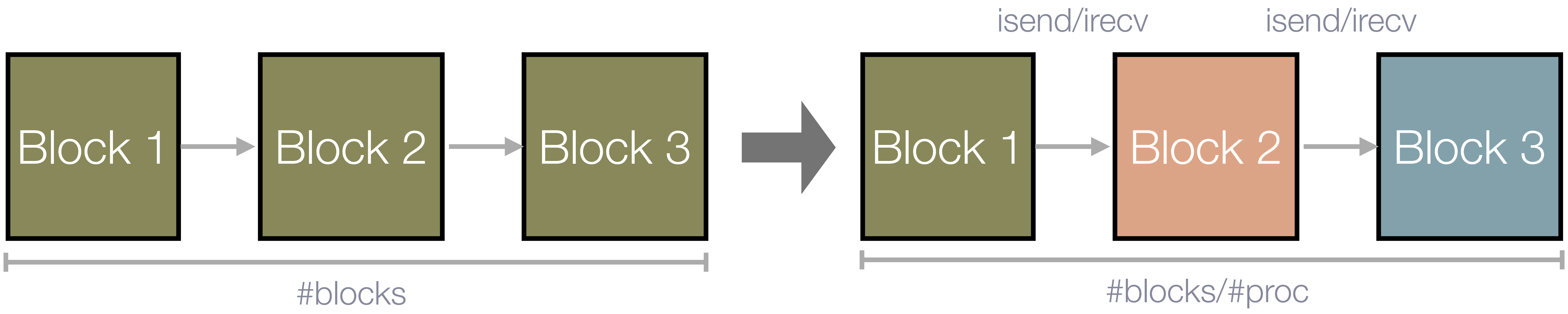


- **Domain decomposition**
  splitting height and width, FFT become distributed, LayerNorm needs to exchange stats



Parallelization methods II

- **Layer pipelining**
  place only a few transformer blocks on each GPU



- **Data parallelism**
  split the global batch across GPU
- **Hybrid parallelism**
  mix and match different types

Beyond Data Parallelism in pytorch

- Think about forward and backward pass
- Some operations might need different communication in forward pass and backward pass
(example row-parallel matrix multiplication: no comm in fwd but allreduce in bwd)
- pytorch: custom autograd functions help to implement such cases
- Examples: NVIDIA [Megatron]

```
# implementation
class _CopyToMatmulParallelRegion(torch.autograd.Function):
    """Pass the input to the matmul parallel region."""

    @staticmethod
    def symbolic(graph, input_):
        return input_

    @staticmethod
    def forward(ctx, input_):
        return input_

    @staticmethod
    def backward(ctx, grad_output):
        return _reduce(grad_output, group=matmul_parallel_group)


# functional wrapper
def copy_to_matmul_parallel_region(input_):
    return _CopyToMatmulParallelRegion.apply(input_)
```

Beyond Data Parallelism in pytorch

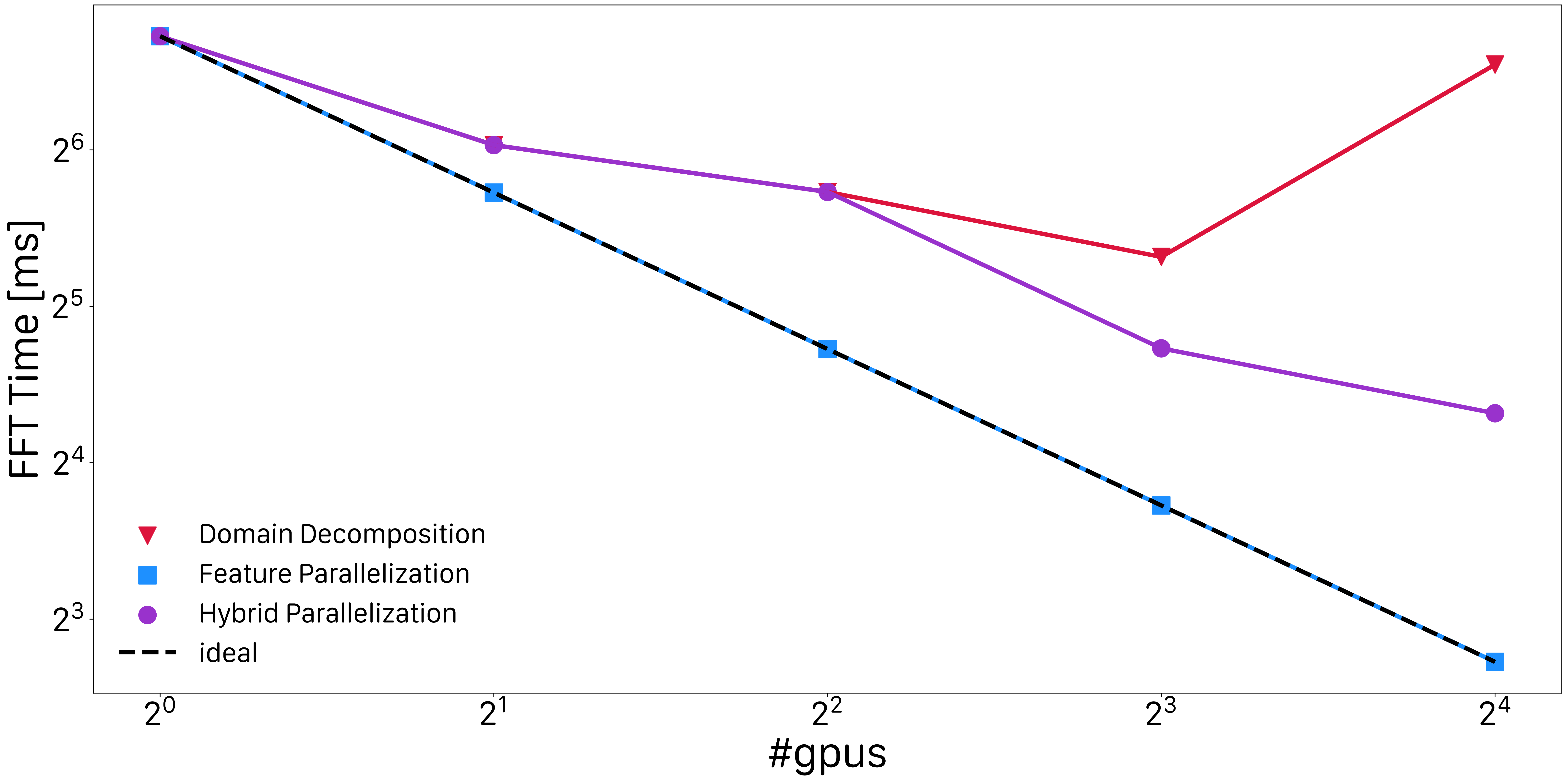
- Think about forward and backward pass
- Some operations might need different communication in forward pass and backward pass
  (example row-parallel matrix multiplication: no comm in fwd but allreduce in bwd)
- pytorch: custom autograd functions help to implement such cases
- Examples: NVIDIA [Megatron]

```
# example usage
def forward(self, x):
    # make sure bwd pass is correct
    x = copy_to_matmul_parallel_region(x)

    # MLP: convs are pointwise, so effectively just matmuls
    x = F.conv2d(x, self.weight1, bias=self.bias1)
    x = self.act(x)
    x = self.drop(x)
    x = F.conv2d(x, self.weight2, bias=None)

    # reduce after distributed matmul
    x = reduce_from_matmul_parallel_region(x)
    x = x + self.bias2

    return x
```

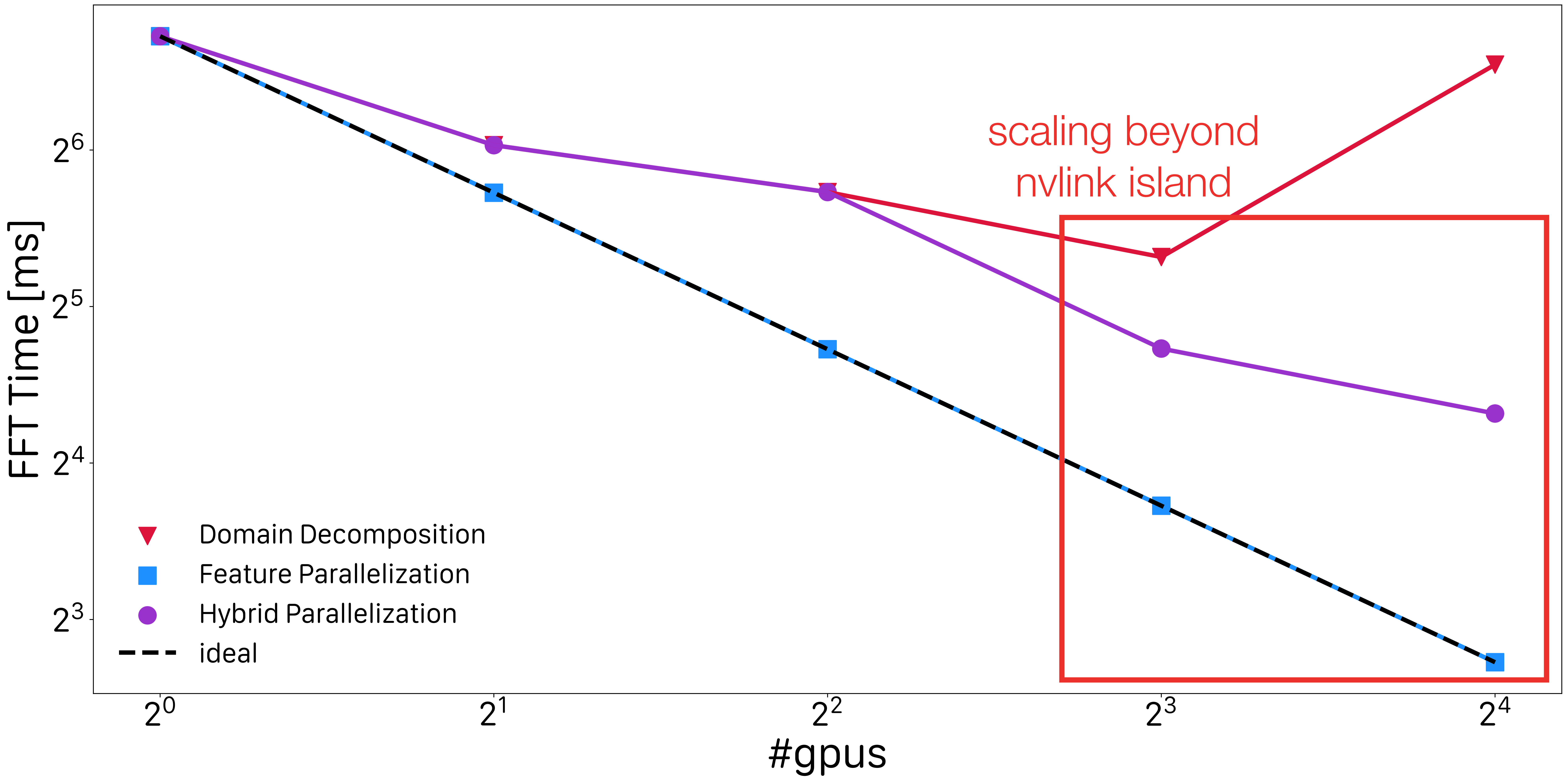
Performance Modeling

- Developed hybrid performance model
- Ops which are hard to model algorithmically: **roofline**
- Ops which are easier to model: use **tailored model** (i.e. FFTs, collective comms)
- Assume SOL execution, no comm overlap
- Implemented compute, memory and communication complexity functions depending on function parameters in pandas notebook: allows for exploring various parameters in cheap fashion
- Verified accuracy on test cases which we can run today (real execution slower because **reality is not running at SOL**)

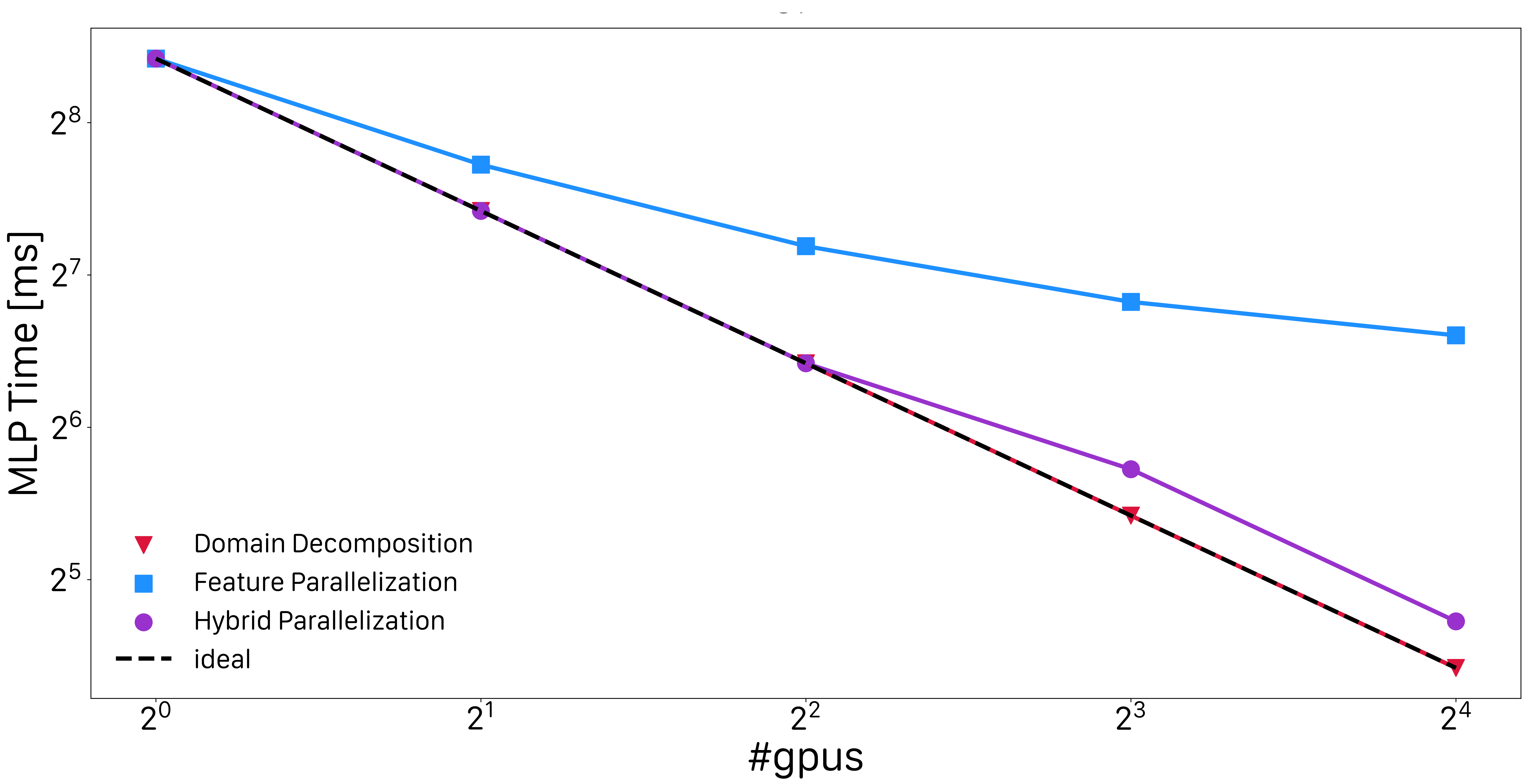
Performance Predictions DGX A100 - Component Scaling



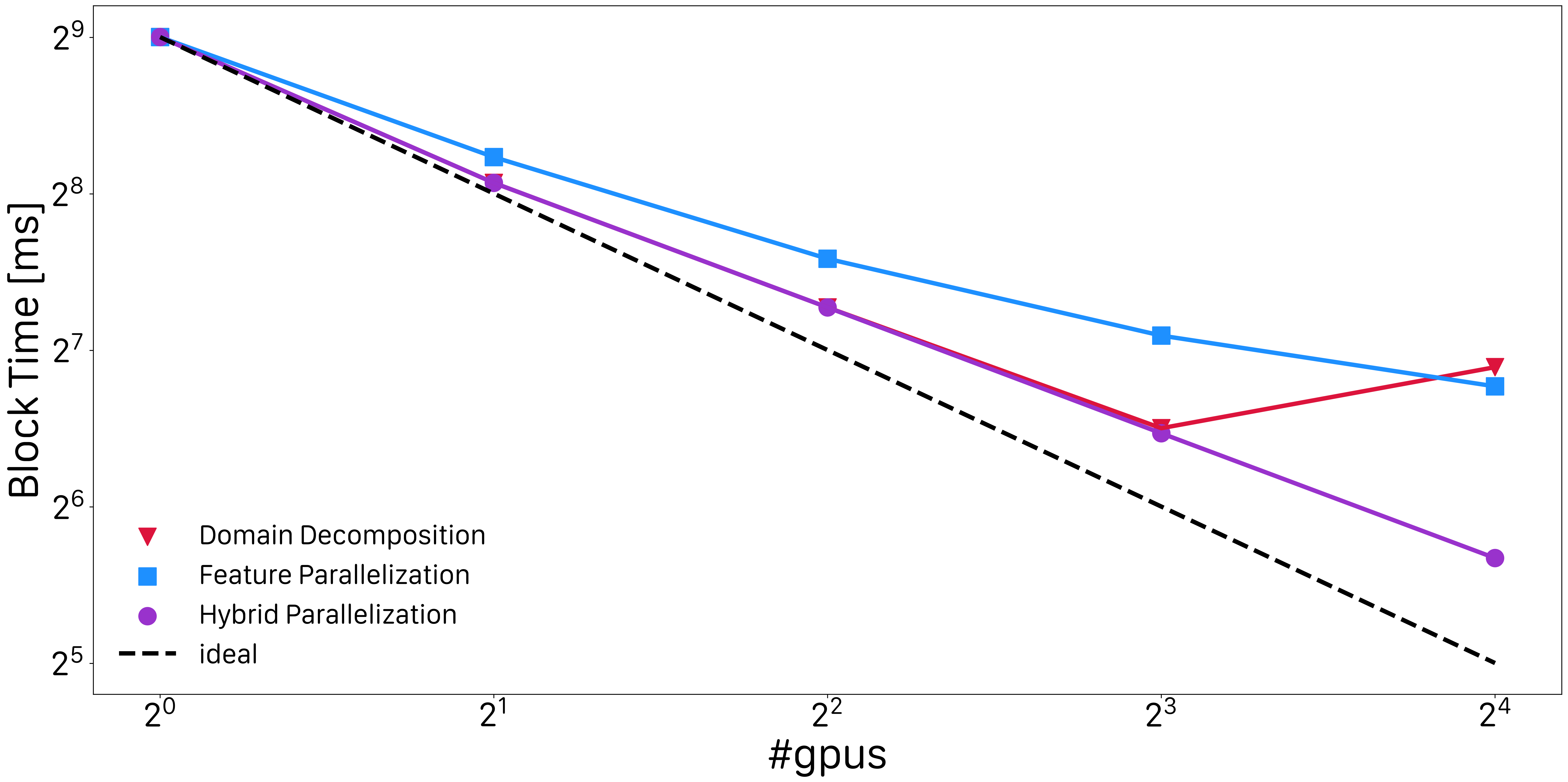
Performance Predictions DGX A100 - Component Scaling



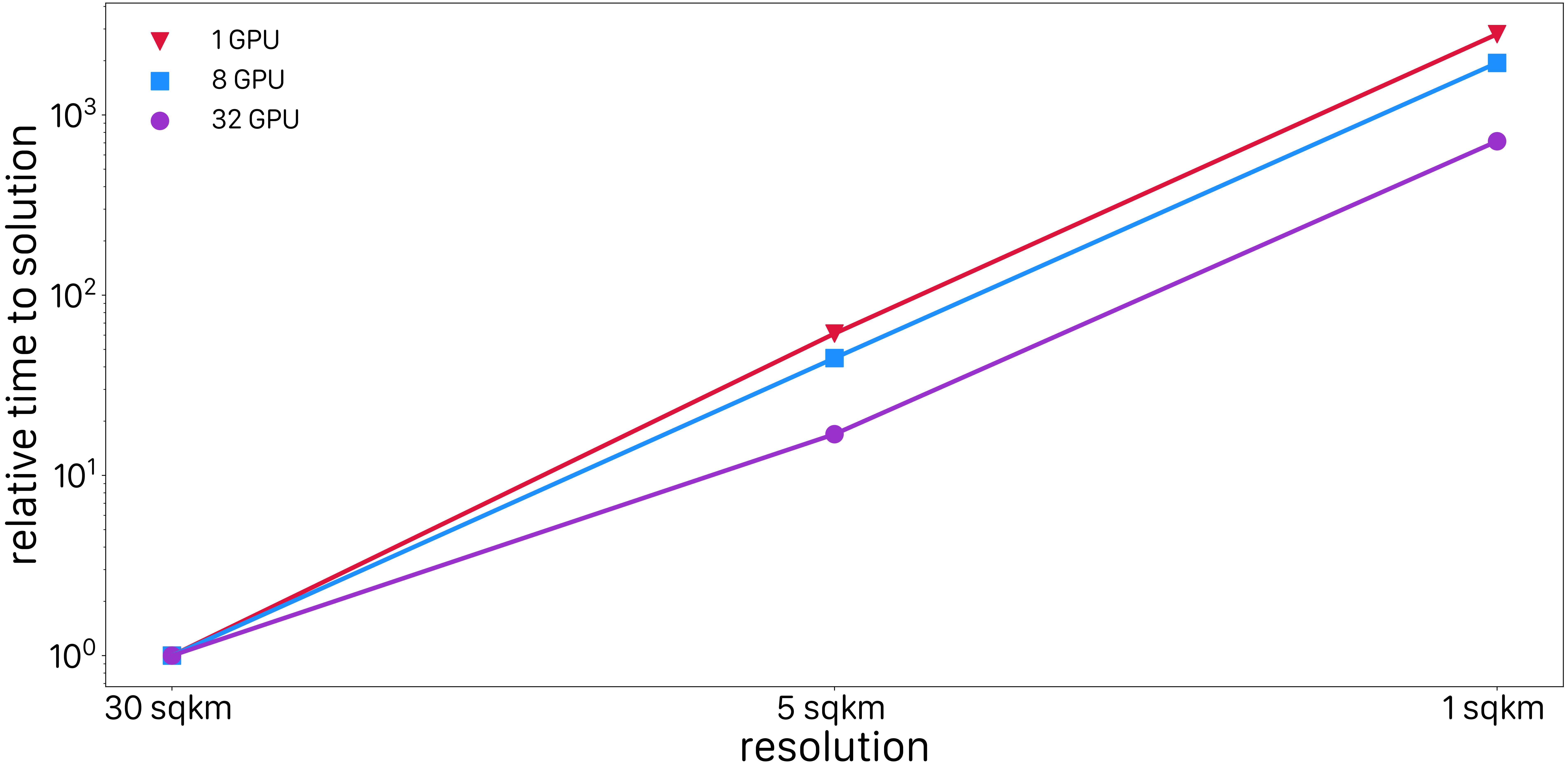
Performance Predictions DGX A100 - Component Scaling



Performance Predictions DGX A100 - Component Scaling



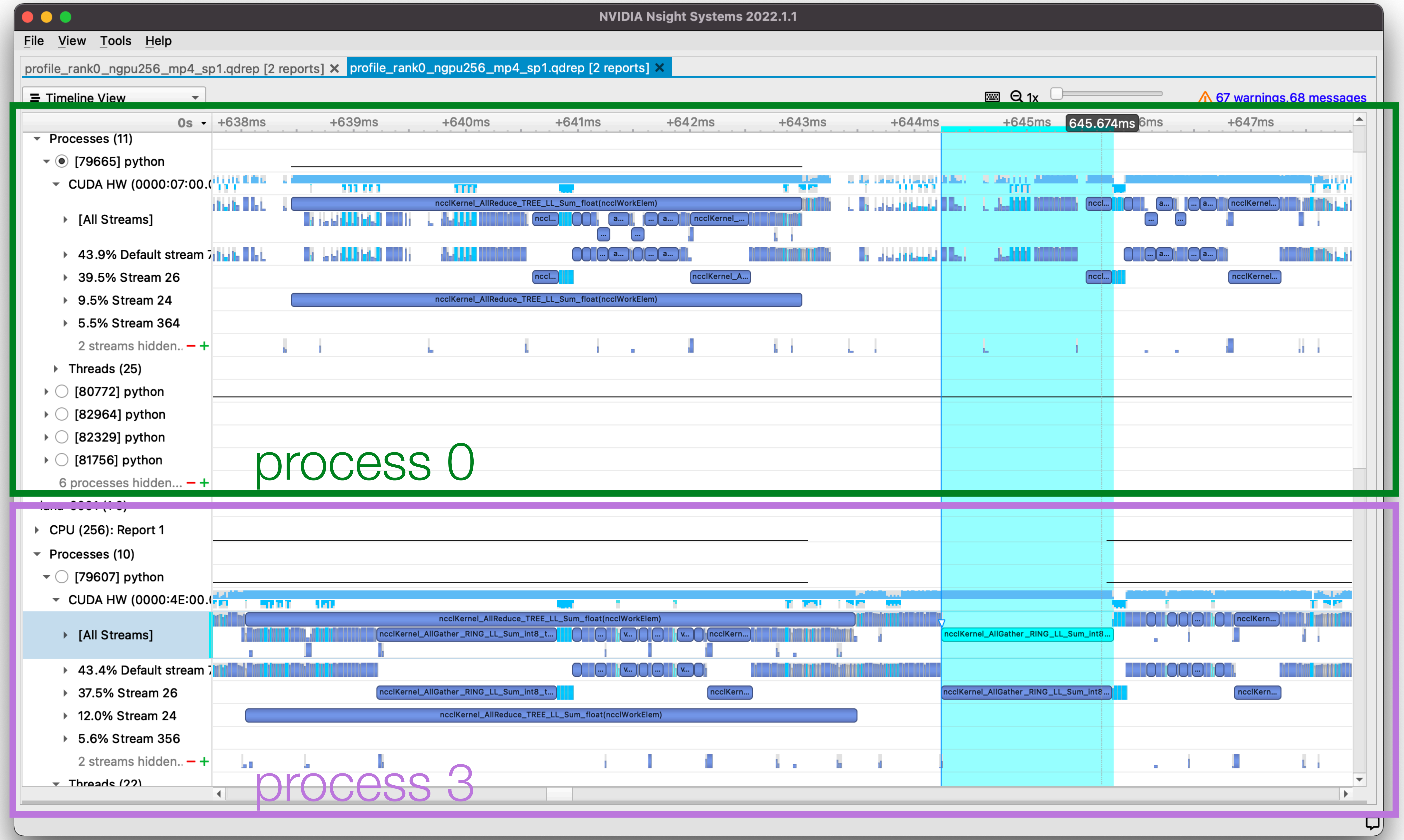
Performance Predictions DGX A100 - Time To Solution



- Exponential growth of TTS with increasing resolution

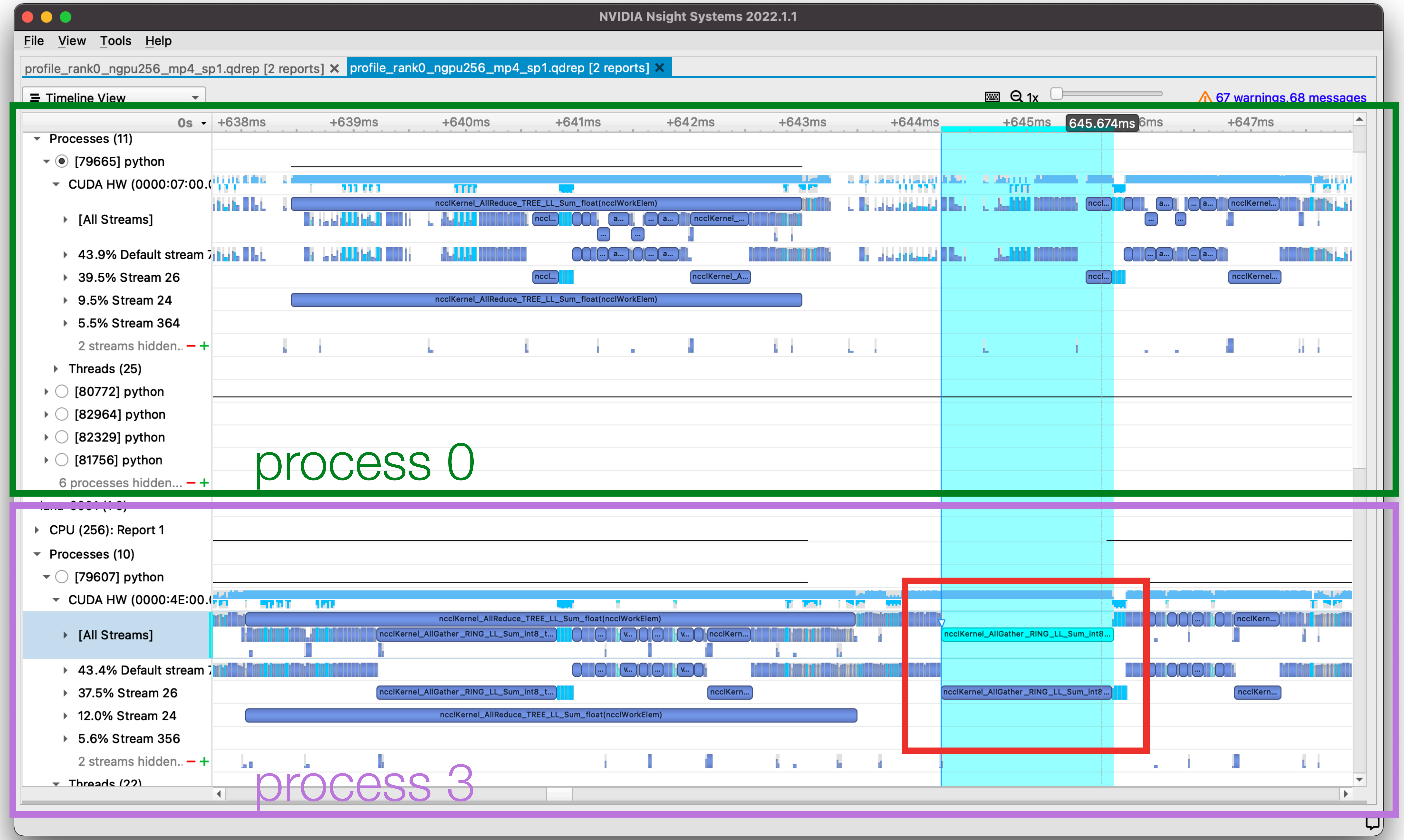
Performance Variability

- Adding communication increases perf variability
- Hybrid parallelism: certain communicators can act independently (i.e. model parallel + data parallel)
- Performance variability can lead to unwanted communication skews
- Take CPU “out of the loop” as much as possible



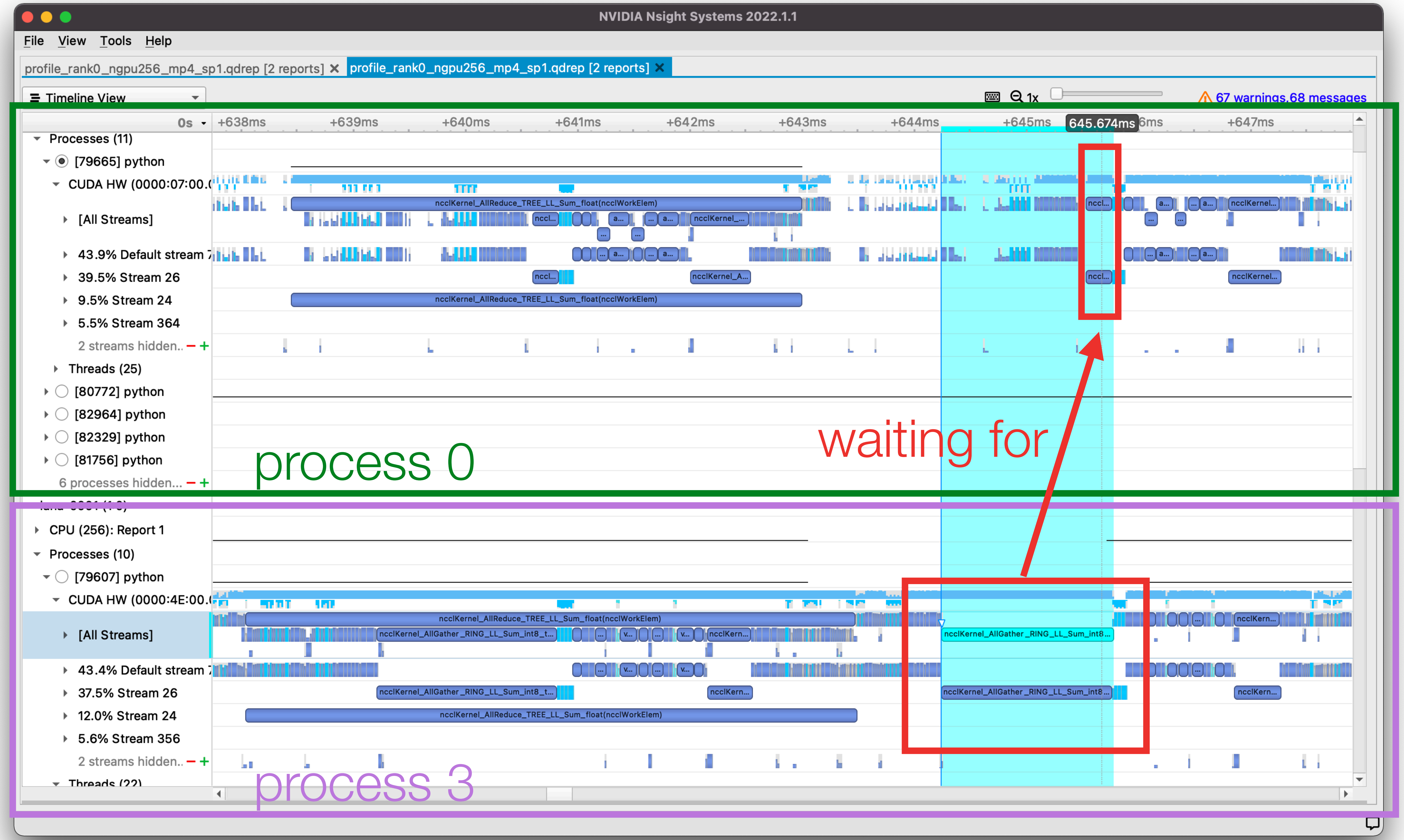
Performance Variability

- Adding communication increases perf variability
- Hybrid parallelism: certain communicators can act independently (i.e. model parallel + data parallel)
- Performance variability can lead to unwanted communication skews
- Take CPU "out of the loop" as much as possible



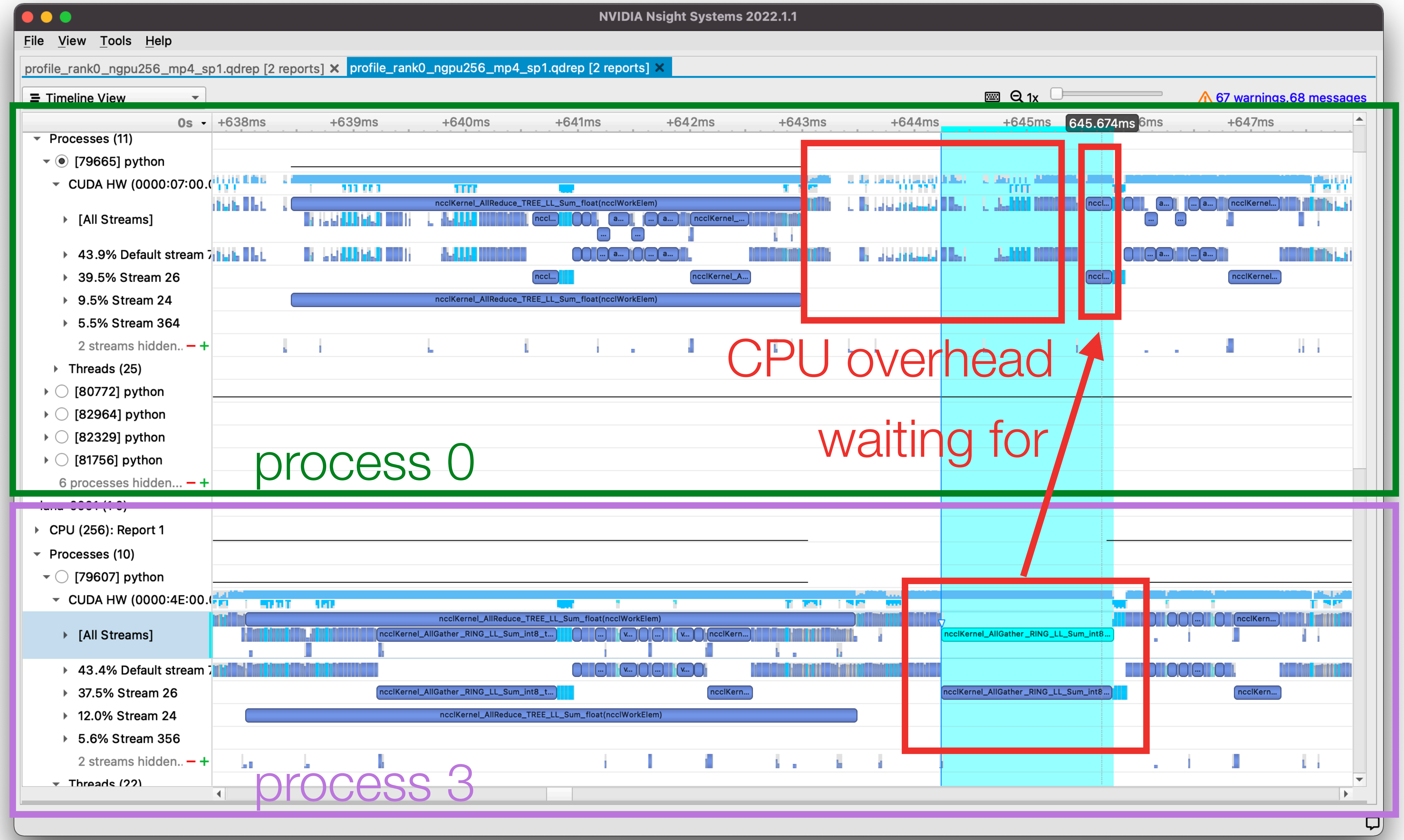
Performance Variability

- Adding communication increases perf variability
- Hybrid parallelism: certain communicators can act independently (i.e. model parallel + data parallel)
- Performance variability can lead to unwanted communication skews
- Take CPU "out of the loop" as much as possible



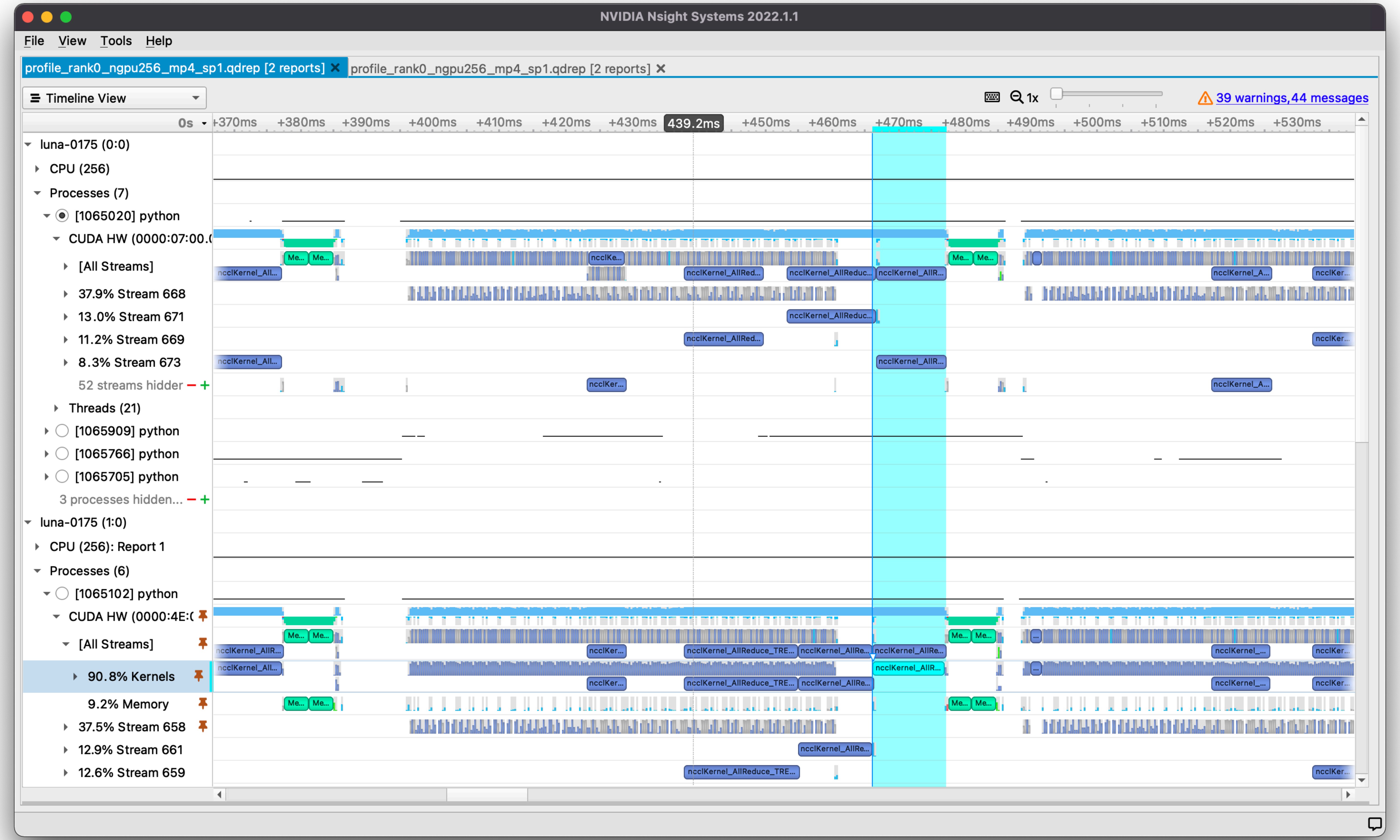
Performance Variability

- Adding communication increases perf variability
- Hybrid parallelism: certain communicators can act independently (i.e. model parallel + data parallel)
- Performance variability can lead to unwanted communication skews
- Take CPU “out of the loop” as much as possible



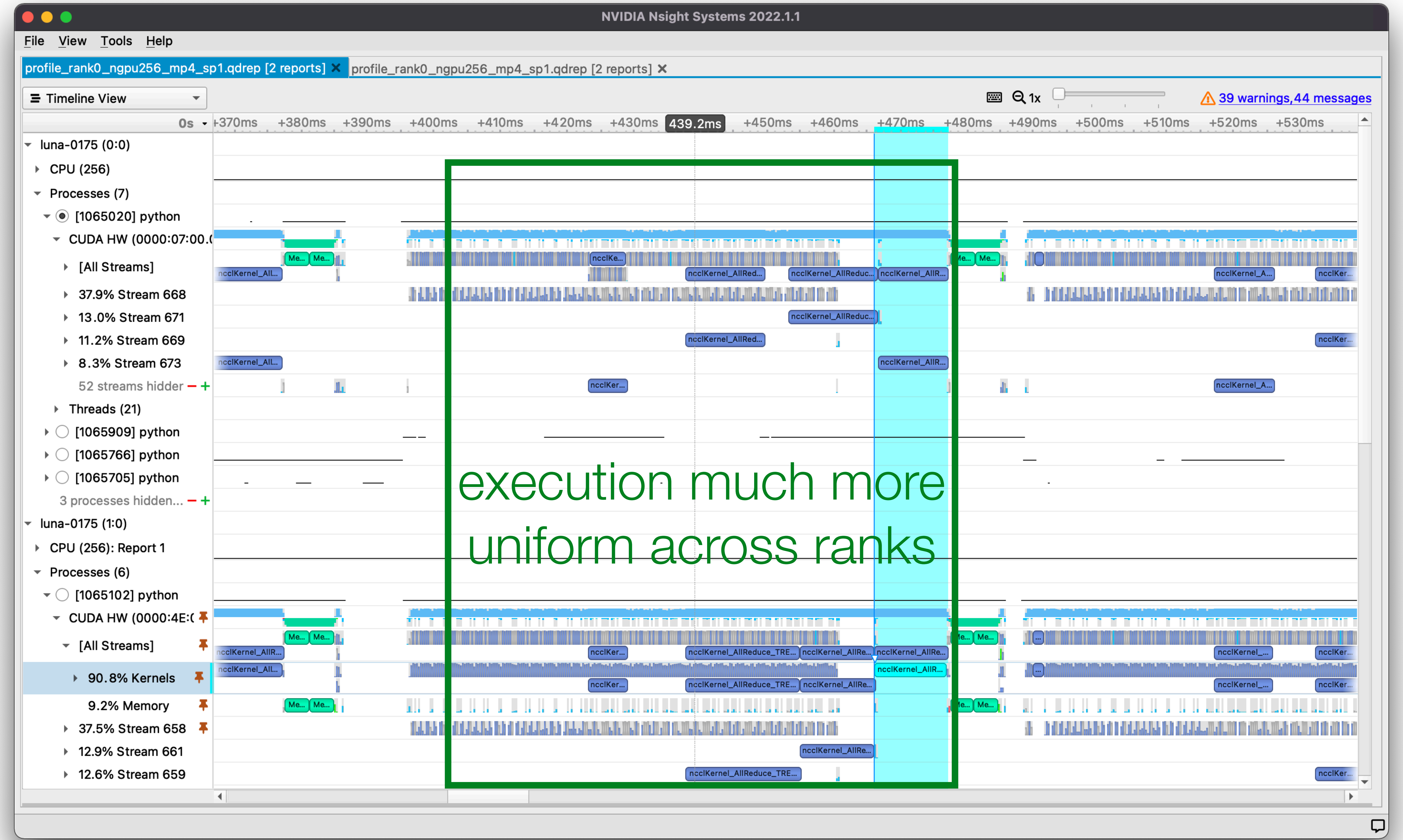
CUDA Graphs

- CPU has to determine what kernels to launch next
- If CPU is busy with something else (OS context switch), it can stall GPU execution
- CUDA graphs allow the user to record a sequence of kernels to execute and replay it as often as desired
- Important: network graph has to be static and nodes of same size (no variable sample or batch size)
- For more info see pytorch documentation



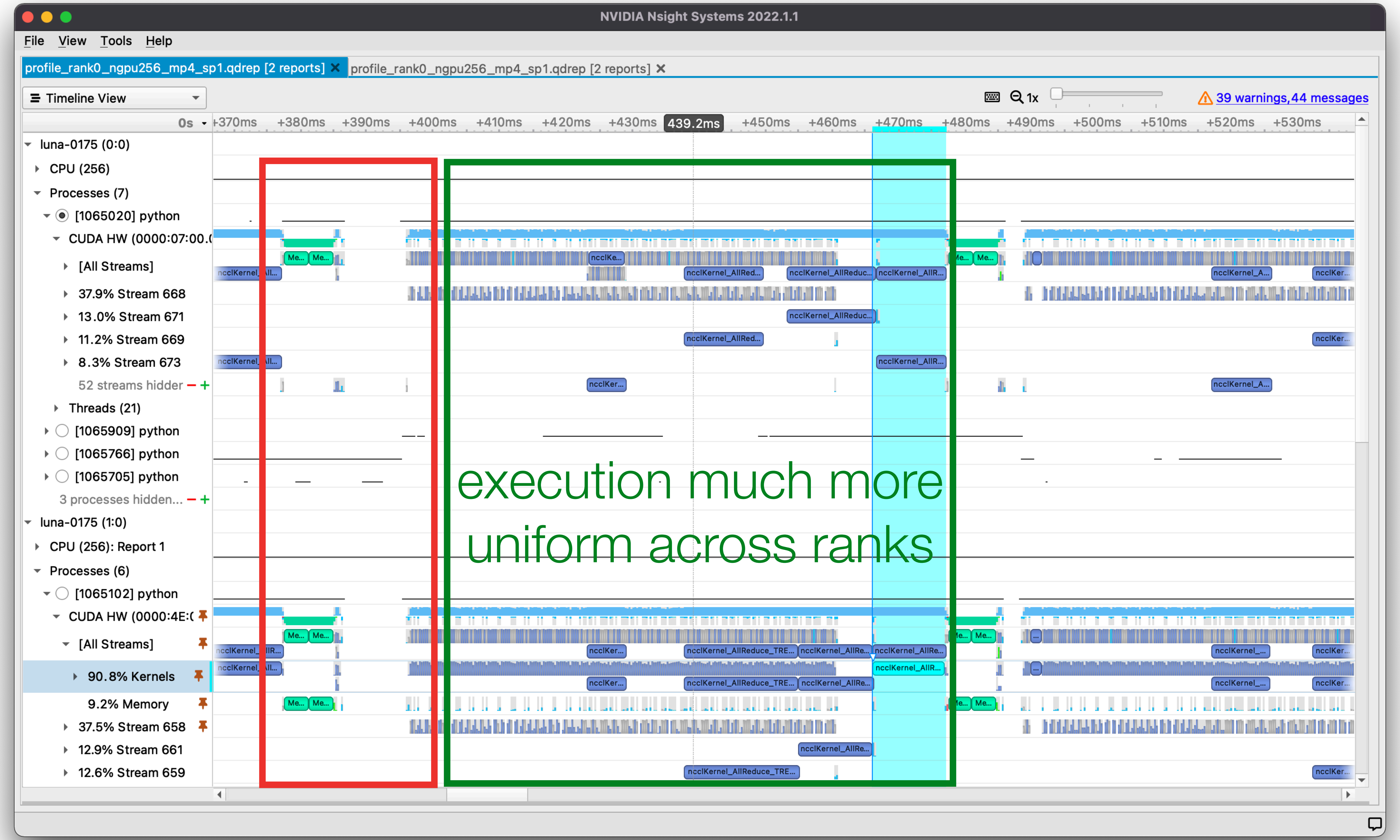
CUDA Graphs

- CPU has to determine what kernels to launch next
- If CPU is busy with something else (OS context switch), it can stall GPU execution
- CUDA graphs allow the user to record a sequence of kernels to execute and replay it as often as desired
- Important: network graph has to be static and nodes of same size (no variable sample or batch size)
- For more info see pytorch [documentation]



CUDA Graphs

- CPU has to determine what kernels to launch next
- If CPU is busy with something else (OS context switch), it can stall GPU execution
- CUDA graphs allow the user to record a sequence of kernels to execute and replay it as often as desired
- Important: network graph has to be static and nodes of same size (no variable sample or batch size)
- For more info see pytorch documentation

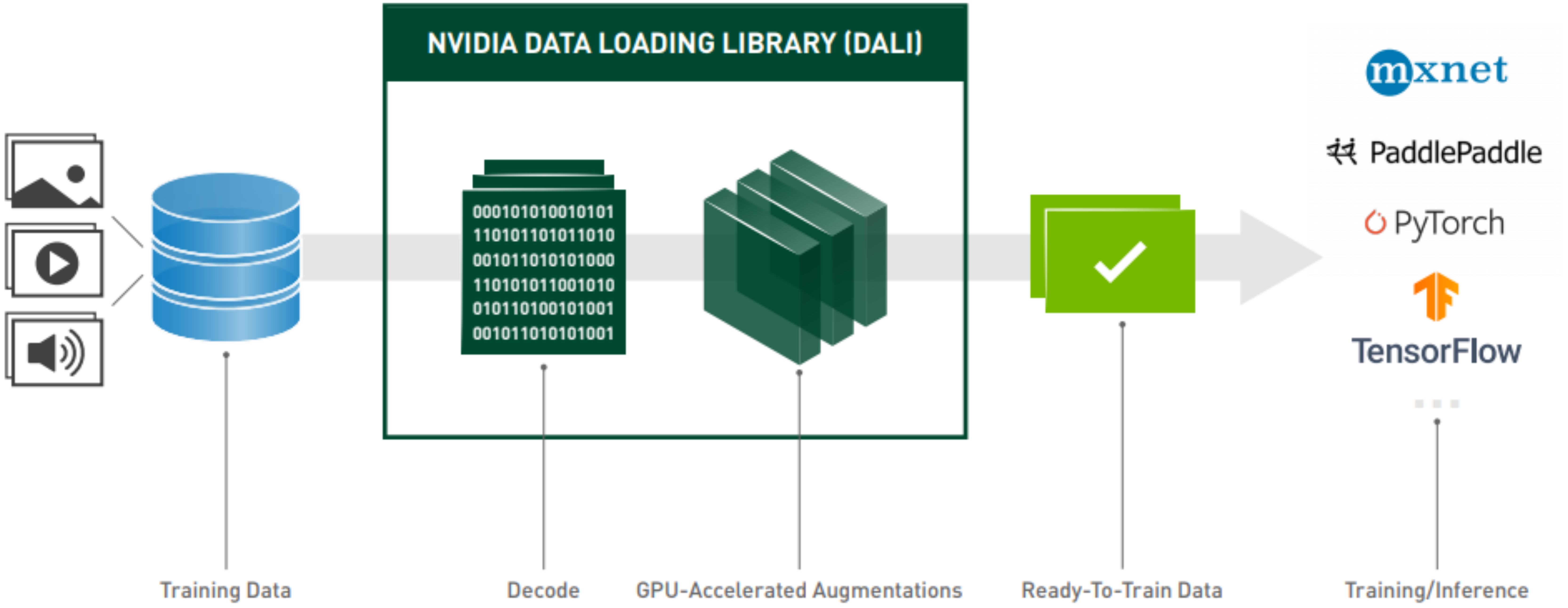


exposed IO

IO Pipeline Challenges

- In DL, data reads are random and potentially small, but predictive
- Caching mechanism essential for large scale training
  (distributed FS characteristics could severely impact performance)
- Good solution should perform IO concurrently to GPU computation
- Expensive post processing operations should be executed on GPU if possible
- Caveat: many available solutions are tailored to industry applications
  (images, audio, text)
- Scientific apps have specific data formats: NetCDF, numpy, HDF5, GRIB, etc.

NVIDIA DALI



- Open Source, framework agnostic, data loading and preprocessing library
- Mainly targeting industry applications but offers a lot of flexibility via external source and custom operators

Summary

- Developed novel neural network architecture for NWP
- Demonstrated excellent skill at good spatio-temporal resolution
- Discussed parallelization opportunities beyond data parallelism
- Discussed scaling challenges of DL solutions on HPC systems: CPU interference and IO
- Demonstrated that hybrid parallelization (channel and domain parallelism on top of data parallelism is required to achieve good scaling)
- 1 sqkm spatial resolution on hourly data is the target. This might be a beyond-exascale problem

Thank You