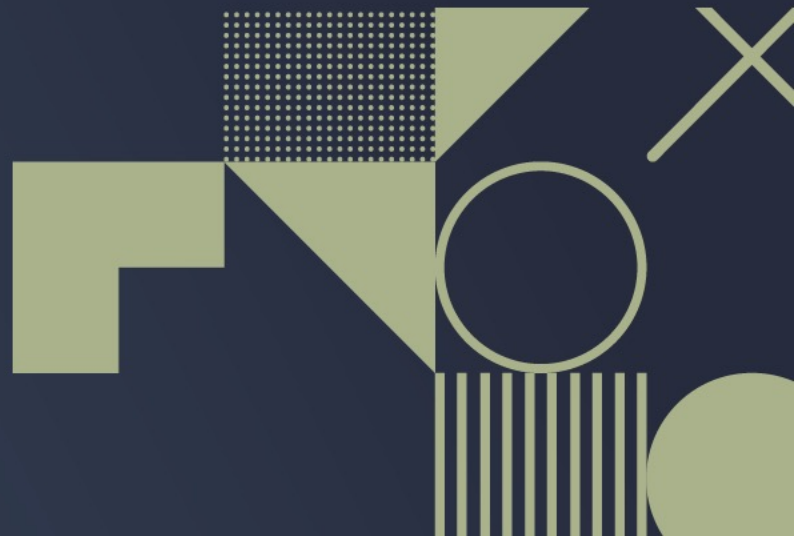
DATA-AWARE STORAGE TIERING FOR DEEP LEARNING

Cong Xu*, Suparna Bhattacharya*, Martin Foltin*, Suren Byna✦, Paolo Faraboschi*

*Hewlett Packard Labs

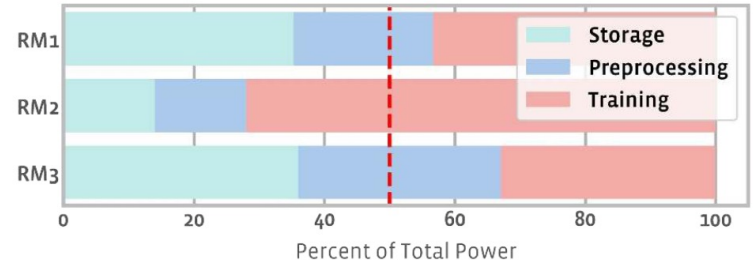
✦ Lawrence Berkeley National Lab



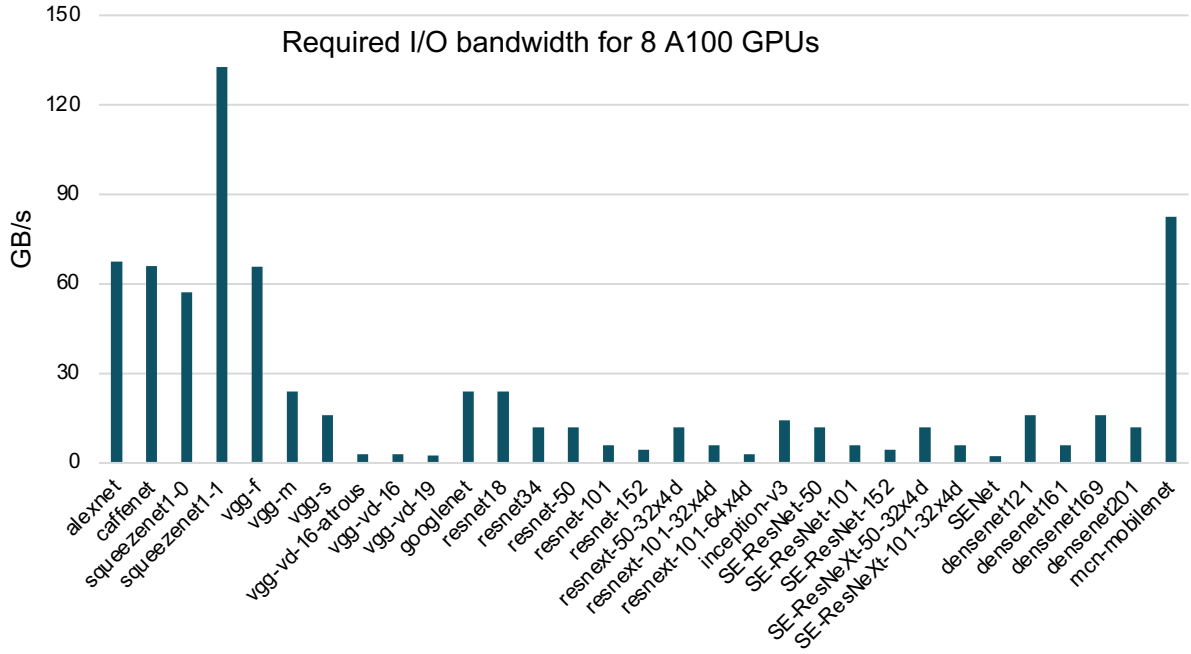
FEEDING DATA TO DNN TRAINING

- economically infeasible to keep the whole dataset in DRAM during the training process
  - ~ 95 million photos are uploaded to **Instagram** every day
    -> ~ 100 TB / day
  - 720,000 hours of video uploaded every day to **YouTube**
    -> ~ 1 PB/day
  - the sensors in an autonomous vehicle record between 1.4 to 19 TB per hour, multiplied by ~600 Waymo fleets
    -> 6 to 90 PB sensor data per day
  - huge recommendation models. i.e. 2-25 PB at Facebook
- fast growth in ML compute capability

Model	Table Size (PB)	Partition Size (PB)	Used Partition Size (PB)
RM1	13.45	0.15	11.95
RM2	29.18	0.32	25.94
RM3	2.93	0.07	1.95



Data ingestion for recommendation models at Facebook

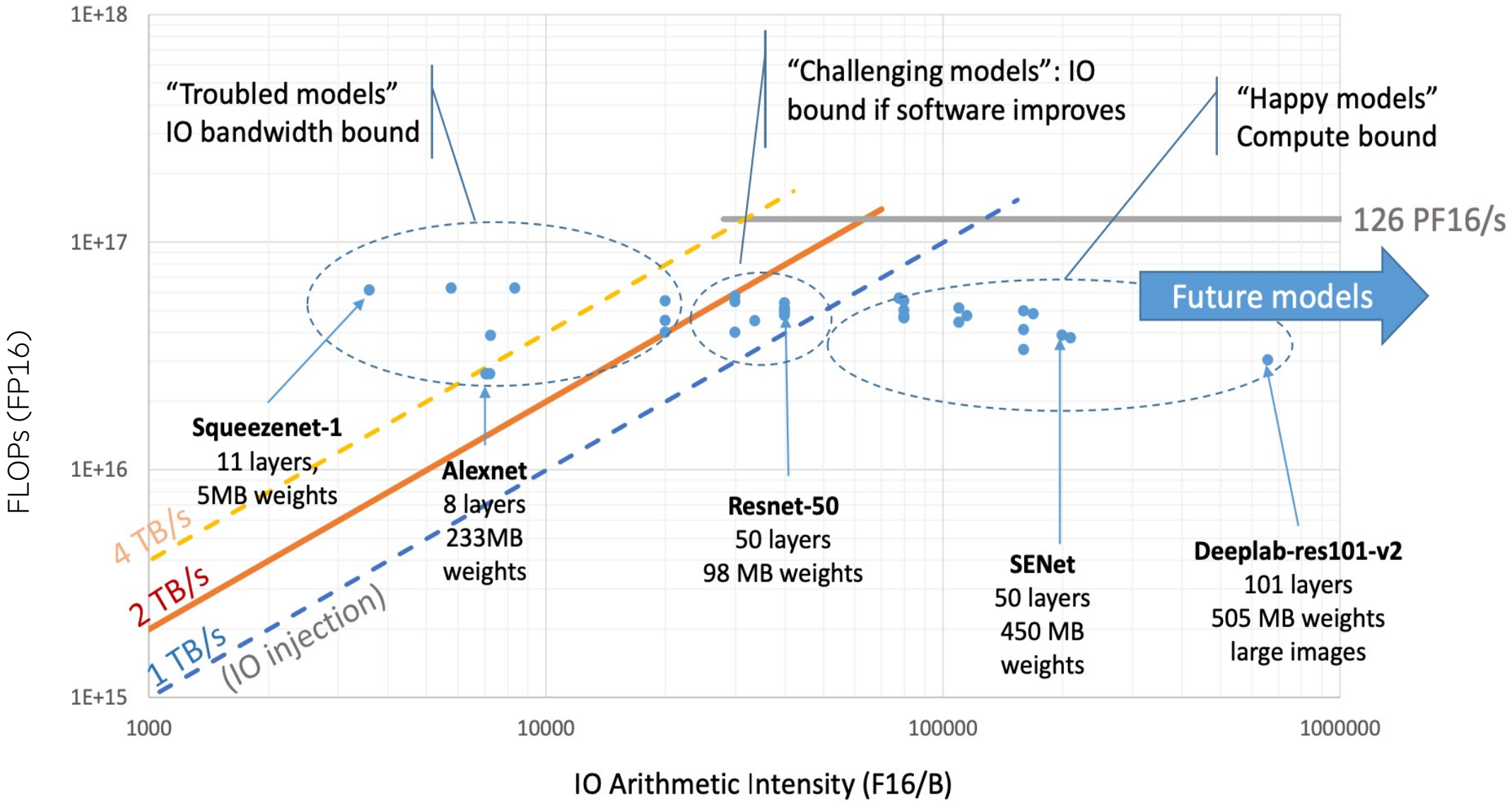


YouTube-100M DataSet

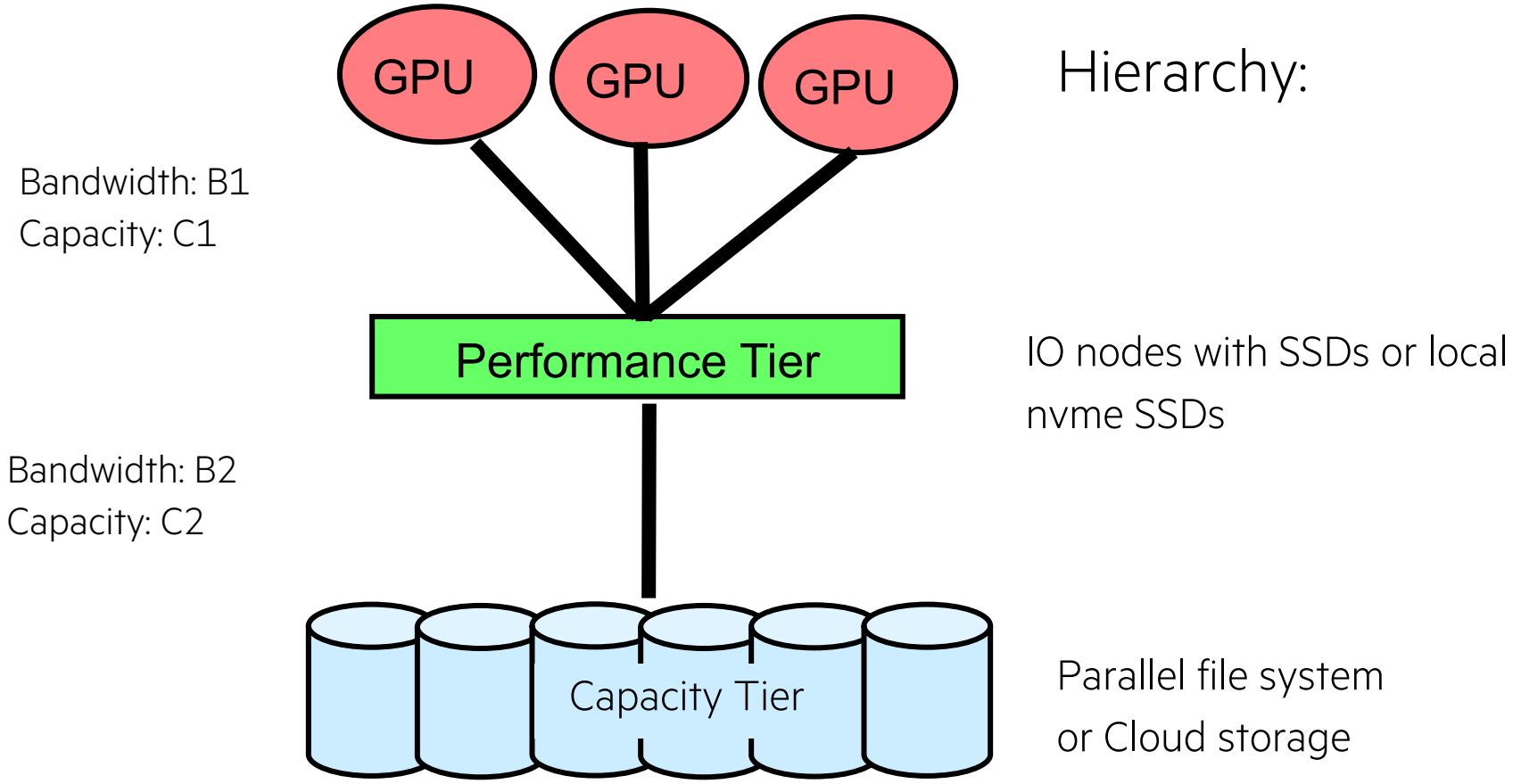
- Size
  - ~100M videos with video-level labels (~5 million hours, 600 years)
  - 20 billion input examples
- Labels
  - 30K labels (not all obviously acoustically relevant)
  - ~3 labels per videos

It's BIG

I/O ROOFLINE MODEL



MULTI-TIER STORAGE IN HPC

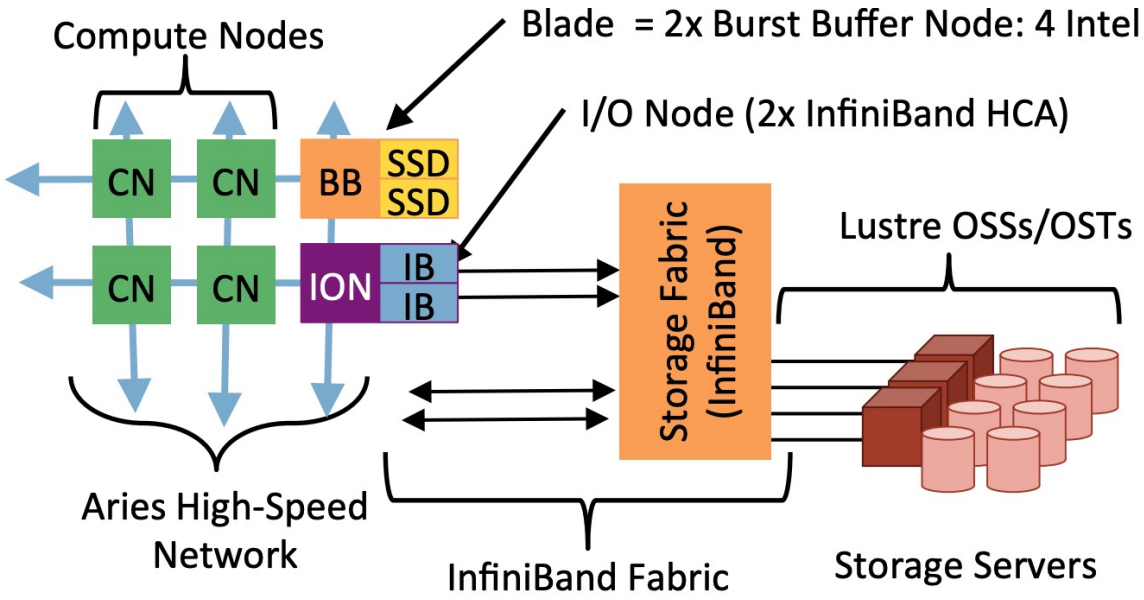


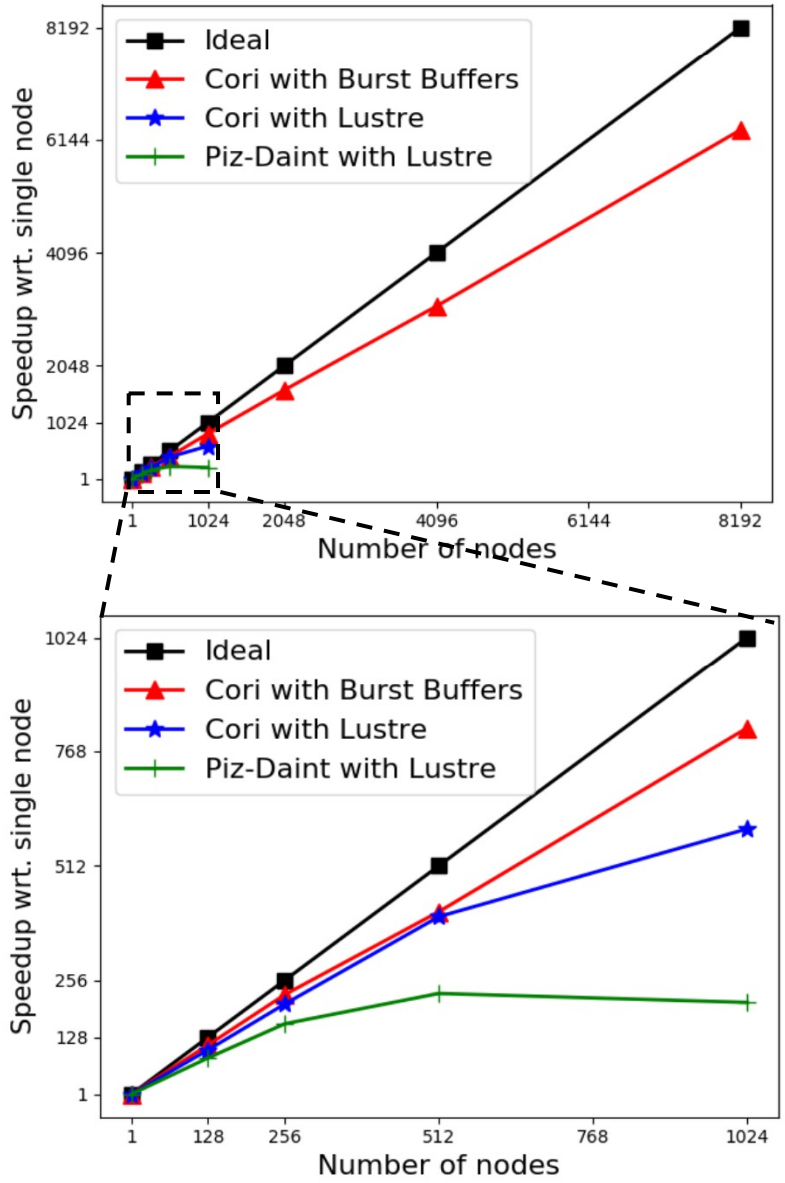
Summit 4068 nodes aggregated:

- B1 = 26.1 TB/s, B2 = 2.5 TB/s, b_ratio = B1 / B2 = 10.4
- C1 = 7.4 PB, C2 = 250 PB, c_ratio = C1/C2 = 3%

Ref: https://www.osti.gov/servlets/purl/1619016

EXAMPLE - NERSC CORI





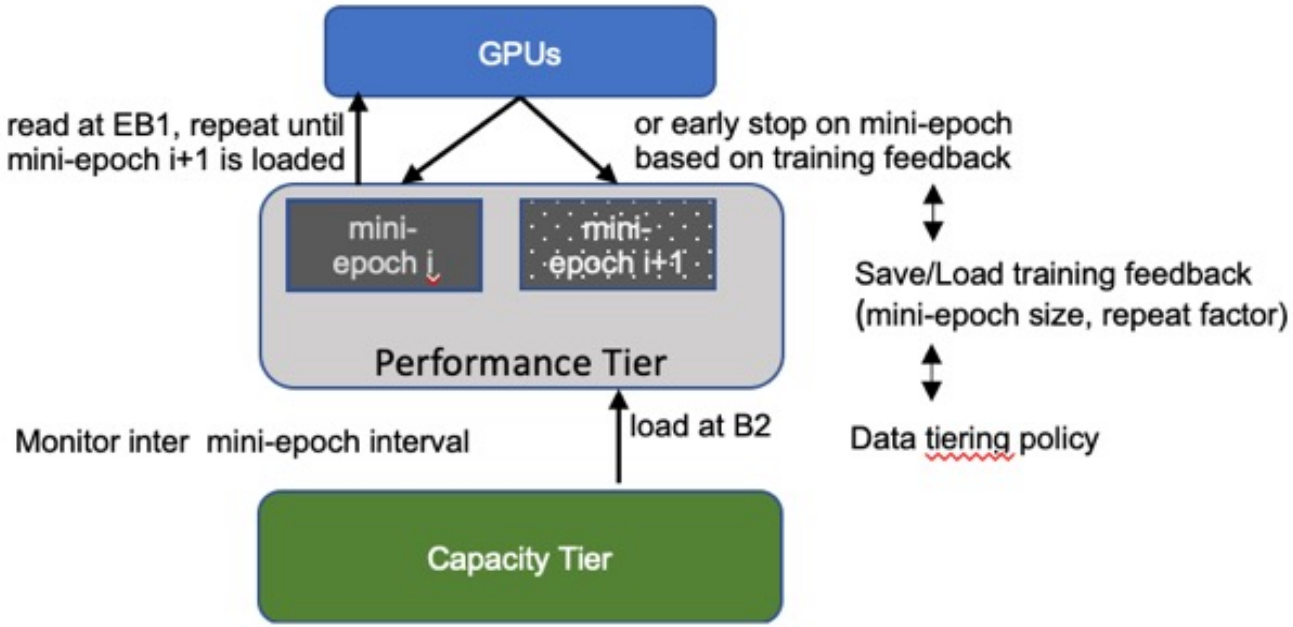
- b_ratio = 1.700 TB/s / 700 GB/s = 2.4
- c_ratio = 1.8 PB / 30 PB = 6%

Time taken to extract 1000 random spectra	From one hdf5 file	From individual fits files
From Lustre	44.1s	160.3s
From BB	1.3s	44.0s
Speedup:	**33x**	**3.6x**

[ref] CosmoFlow: Using Deep Learning to Learn the Universe at Scale

BANDWIDTH-AWARE MINI-EPOCH TRAINING

- Bandwidth aware iteration
  - Split the entire epoch into <ne> mini-epochs s.t. each mini-epoch is smaller than 0.5 * C1 for non-overlapping mini-epochs
  - while GPUs are iterating over mini-epoch i, prefetch another mini-epoch i+1 from the capacity tier to performance tier at B2.
  - assuming GPUs are consuming data at (EB1), iterate over mini-epoch i <rf> number of times before the next mini-epoch is completely loaded



The required code change to the ML applications is minimum.

- almost the same code applies to ImageNet, Youtube-8M, ML-20M dataset

```
epoch = tfds.load(<dataset_name>, split=[f'train[{k}%:{k+10}%]' for k in range(0, 100, 12.5)])
mini-epochs = [mini-epoch.repeat(16) for mini-epoch in epoch]
```

If the size of mini-epoch is not uniform, Dataset APIs are used

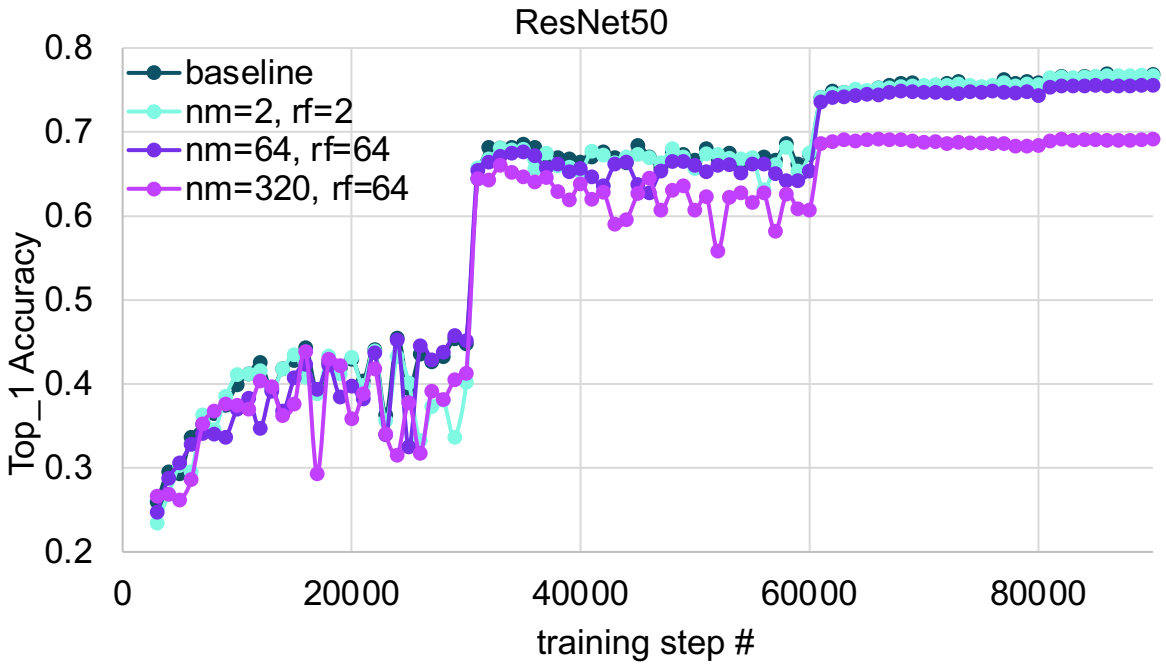
```
epoch = tfds.load(<dataset_name>)
mini_epoch_sizes = [me1_size, me2_size, …] # store the sizes for each mini-epoch
mini-epochs = []
for size in mini_epoch_size:
        mini-epoch.append(epoch.take(size))
        epoch = epoch.skip(size)
```

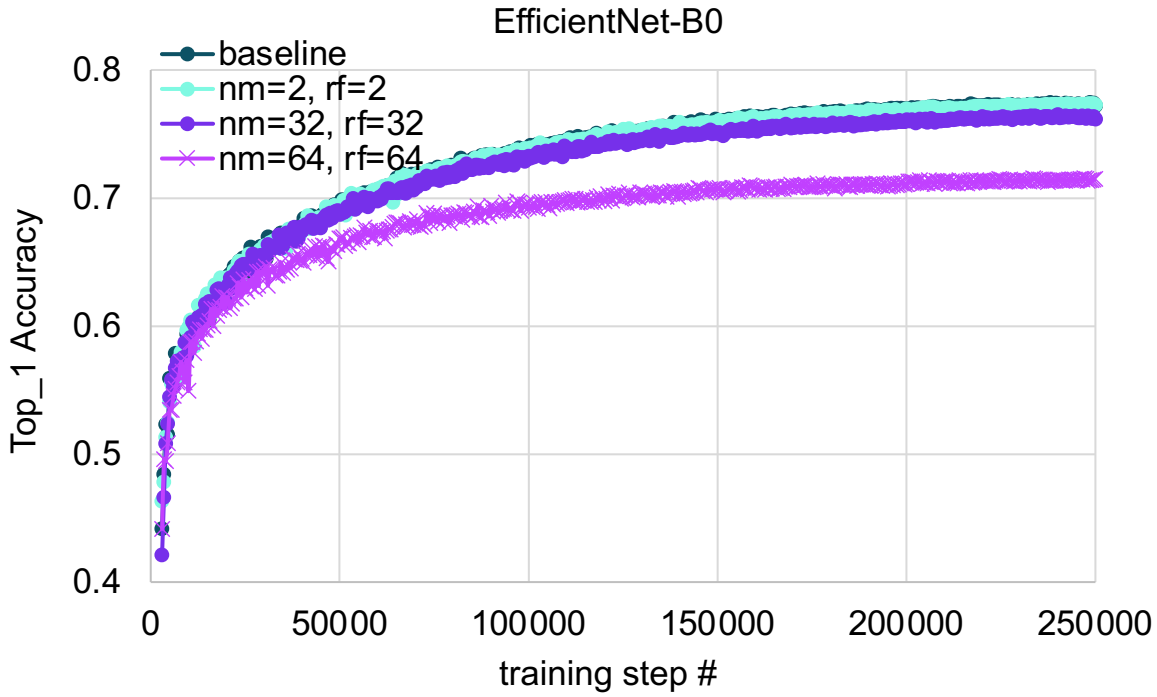
Epoch vs. Mini-Epoch vs. Mini-batch example

- epoch: 100m - billions images
- mini-batch: a few thousand images
- mini-epoch: ~ 10m images

IMAGENET RESULTS

- No accuracy drop if <num_mini_epochs> and <repeating_factors> are within reasonable ranges, i.e. < 32





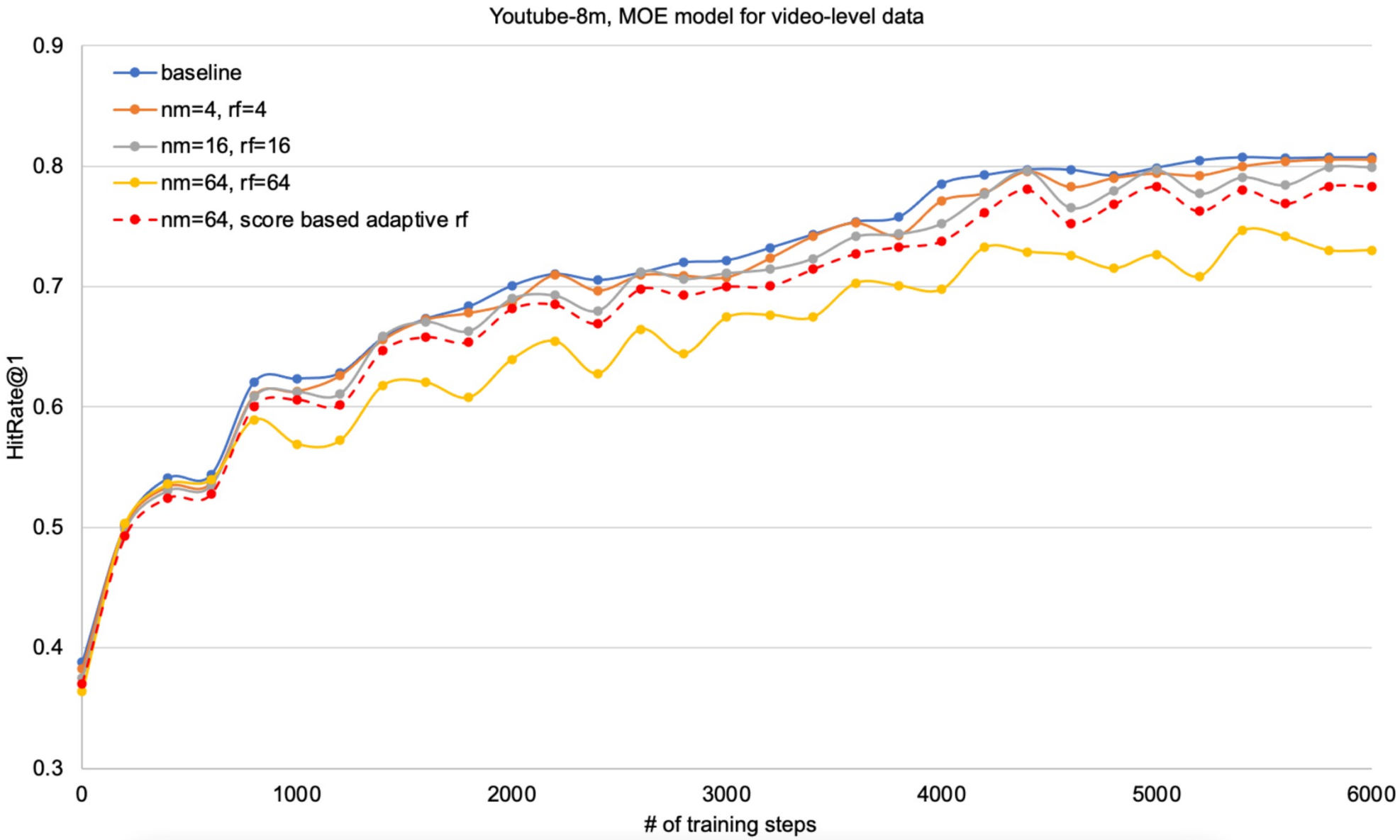
MODEL CONVERGENCE FEEDBACK

- As the repeating factor at which convergence is affected depends on the dataset and the model, a feedback mechanism is implemented as an TF callback
  - Record the loss/accuracy and other metrics at the end of each mini-epoch. A score is calculated based on a combination of the monitored metrics.
  - If the score does not improve for a fixed number of times, early stop on this mini-epoch and wait until mini-epoch i+1 is fully loaded then move forward.

- *Bollinger band-based adaptive repeating factor:*
  - consist of an N-period moving average MA of the score, a lower band at K times an N-period standard deviation below the moving average $MA - K * std$.
  - Move to next mini-epoch if score goes below lower band

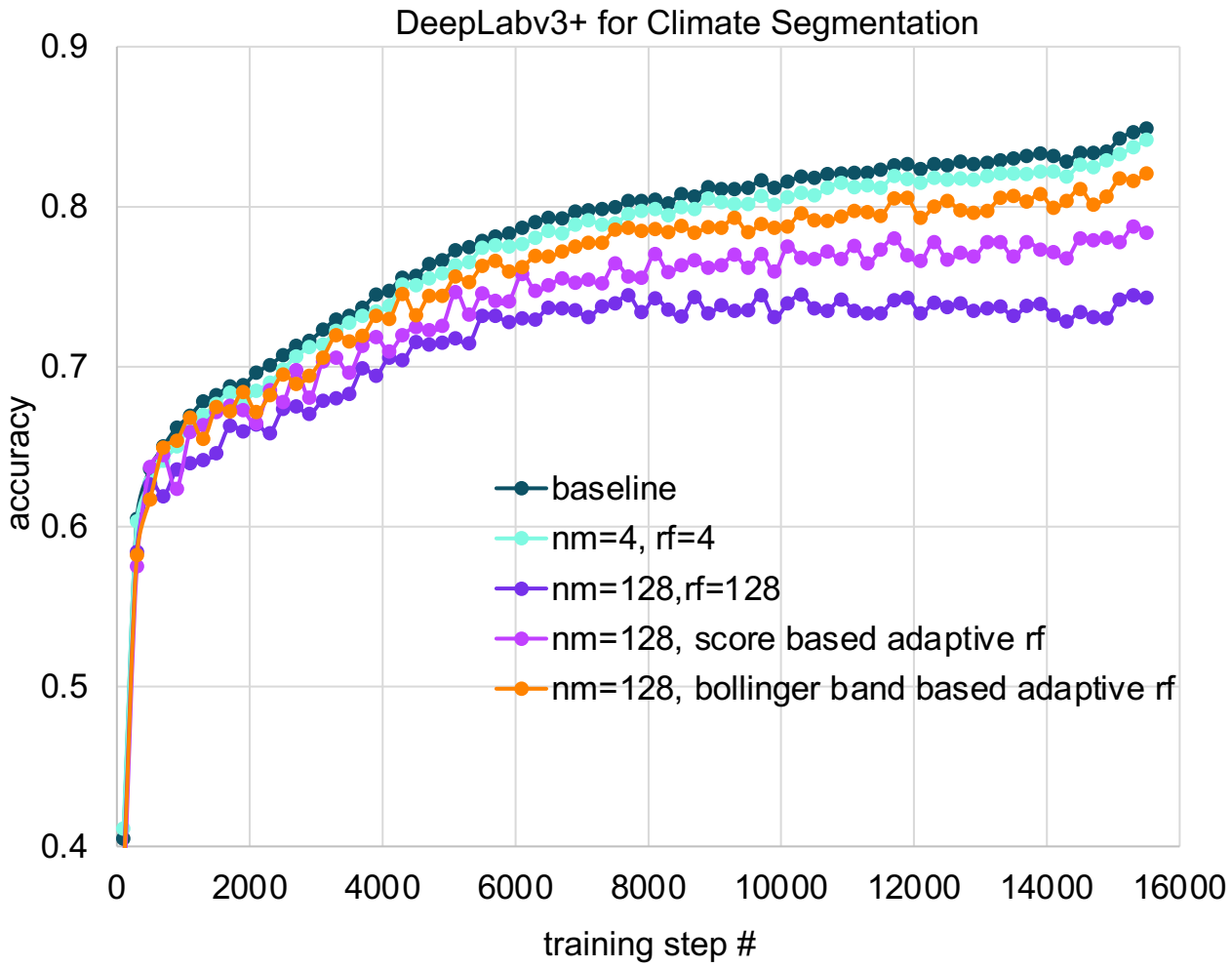
```
loss = logs.get("loss")
train_err = 1 - logs.get("accuracy")
val_err = 1 - logs.get("val_accuracy")
metrics = np.array([loss, train_err, val_err])
current_score = np.dot(self.score_weights, metrics)

if np.less(current_score, self.score):
        self.score = current_score
        self.wait = 0 # Record the weights if current results is better (less).
else:
        self.wait += 1
        if self.wait >= self.wait_th:
                # stop iterating over this mini-epoch and
                # move forward to next mini-epoch when it's loaded
```

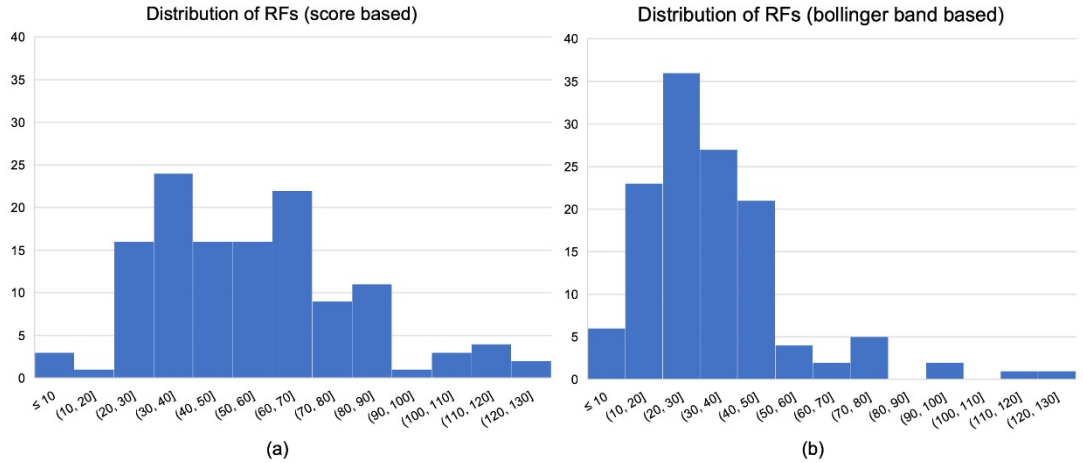
YOUTUBE-8M RESULTS



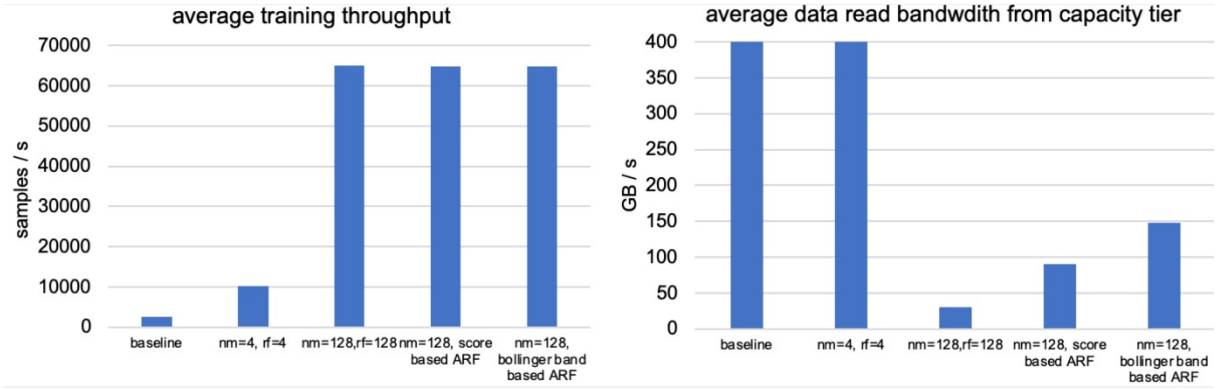
DEEPCAM RESULTS

- Exascale Deep Learning for Climate Analytics
  - Gordon bell price winner, 2018





distributions of RFs over 128 mini-epochs for ARF based on: (a) score and (b) bollinger band



modeling results for average (a) training throughput, and (b) data read bandwidth (EB2) with MET on the Summit system

CONCLUSIONS

- evaluated three different applications with mini-epoch training andmost of the them worked out-of-the-box with modest parameters of <num_mini_epochs> and <repeating_factor>
- 5% to 11% accuracy drop with fixed <repeating_factor> design compared to the baseline, while the adaptive repeating factor was able close most of the accuracy gap
- significantly reduce bandwidth required to capacity tier and improve time-to-convergence of I/O bounded training
- Mini-epoch training is a simple but effective solution with minimum code change required