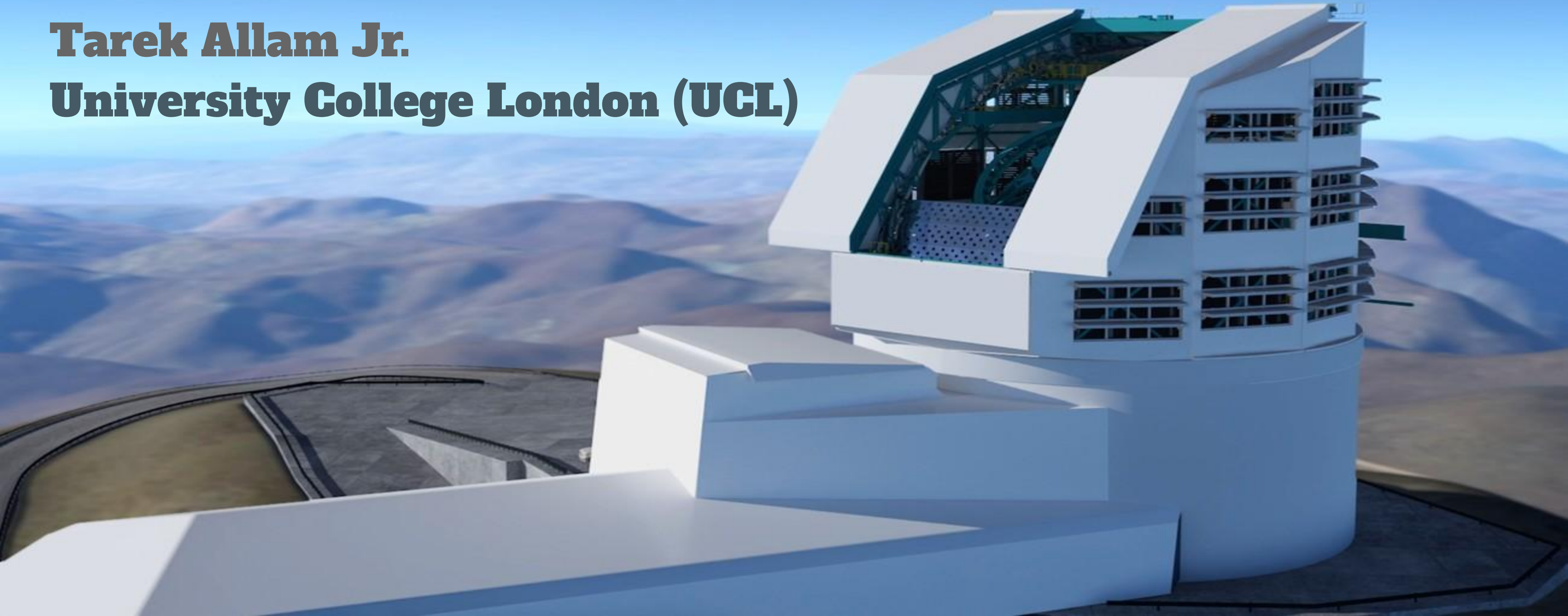
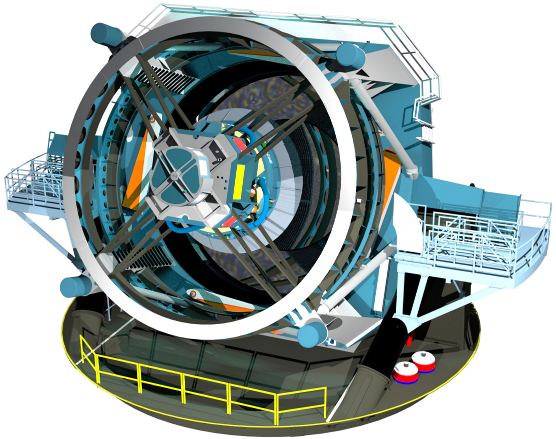
Low-latency High-throughput Classification using Deep Model Compression

Tarek Allam Jr.
University College London (UCL)

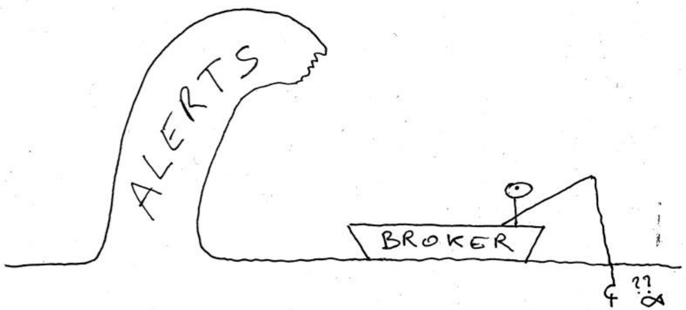


Motivation: A Deluge of Data



Overview

- 10 million alerts, per night!
- Machine Learning methods are now critical
- Accurate and fast classification required for follow up
- Desire for fast re-training of models



Peloton et al. 2020

Deep Learning to the Rescue!?

The death of feature engineering?

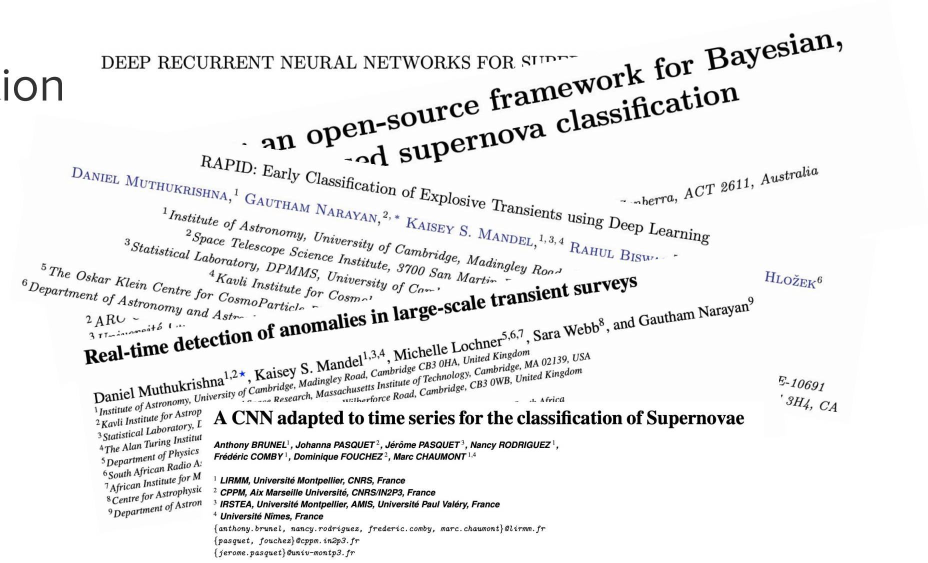
- Exploiting inherent time-series information

RNNs (inc. SuperNNova, RAPID)

CNNs (inc. TCN)

Transformers (inc. t2)

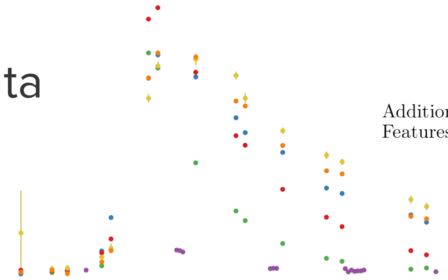
...

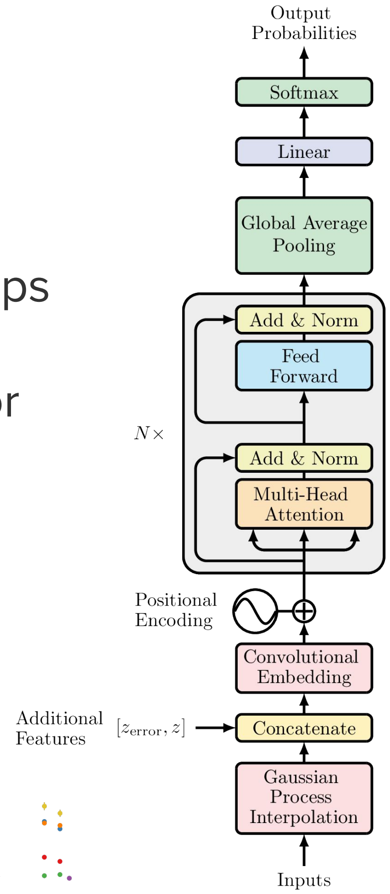


The Time-Series Transformer [t2]

Encoder++

- *Global Average Pooling* to allow for Class Activation Maps
- *Convolutional Embedding* maps time-series into a vector space
- *Concatenate* additional features
- *GP Interpolation* to handle irregular data

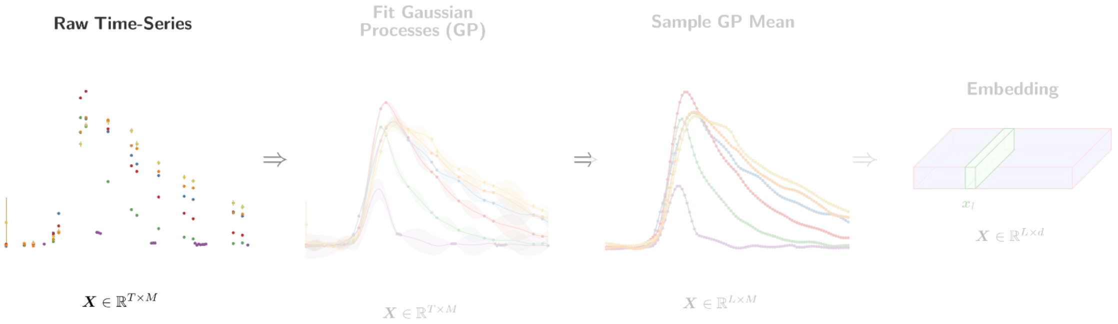


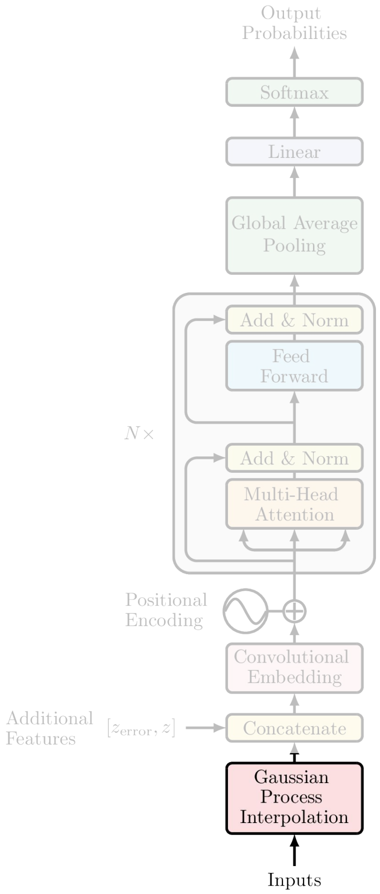


ML Pipeline

Gaussian Process Interpolation

- 2D-Matern kernel for Gaussian Process interpolation
- Evaluate at regular period, 100 points in our case

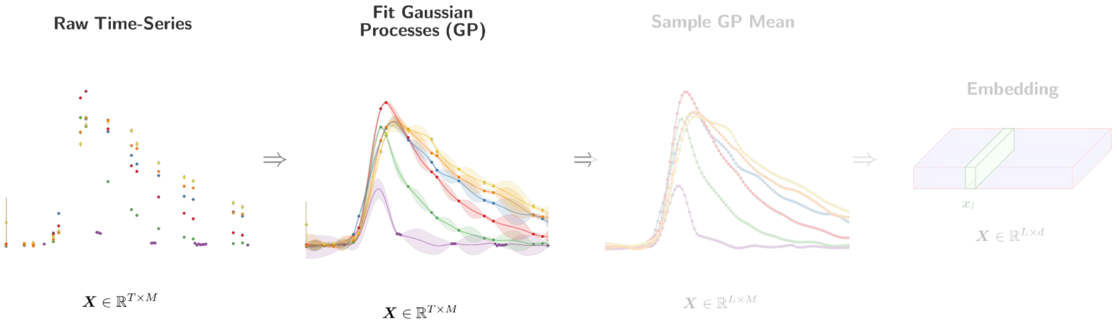


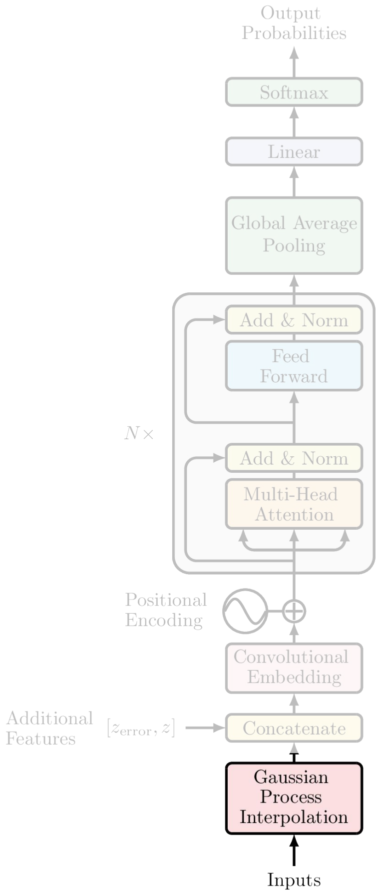


ML Pipeline

Gaussian Process Interpolation

- 2D-Matern kernel for Gaussian Process interpolation
- Evaluate at regular period, 100 points in our case

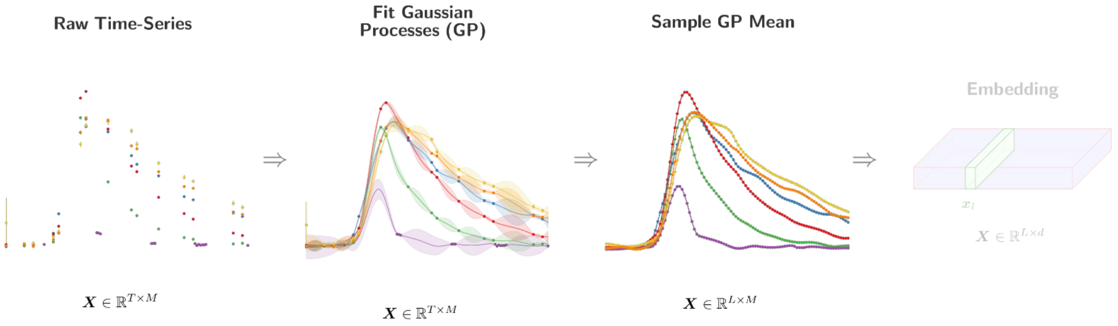


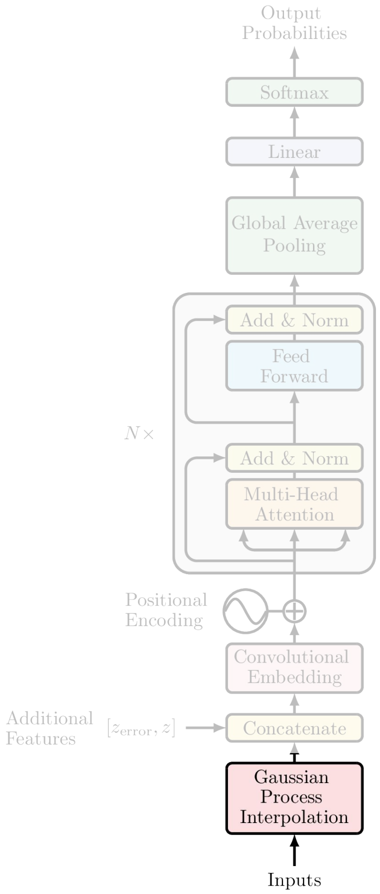


ML Pipeline

Gaussian Process Interpolation

- 2D-Matern kernel for Gaussian Process interpolation
- Evaluate at regular period, 100 points in our case

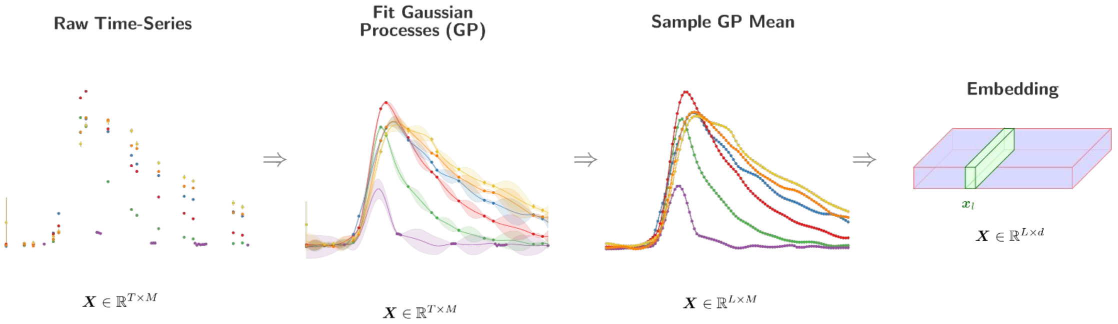


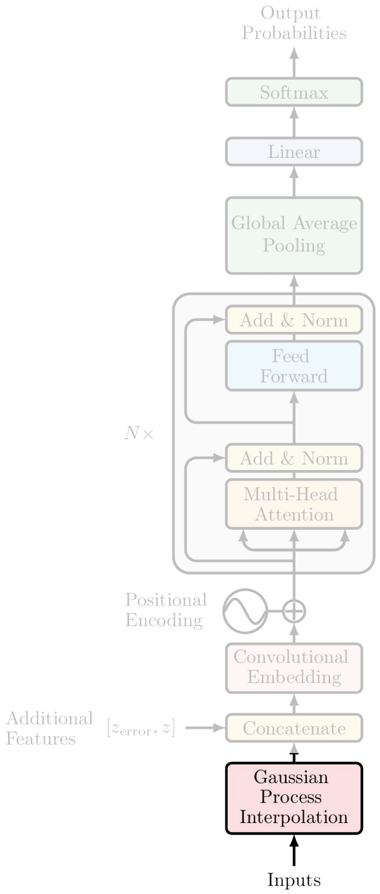


ML Pipeline

Gaussian Process Interpolation

- 2D-Matern kernel for Gaussian Process interpolation
- Evaluate at regular period, 100 points in our case



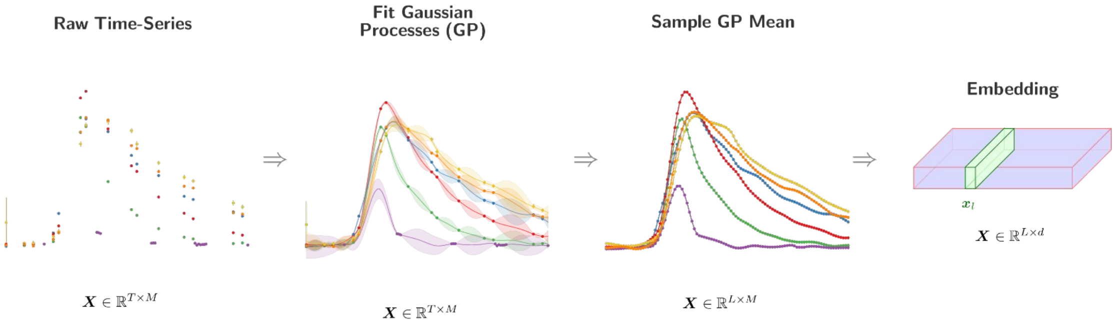


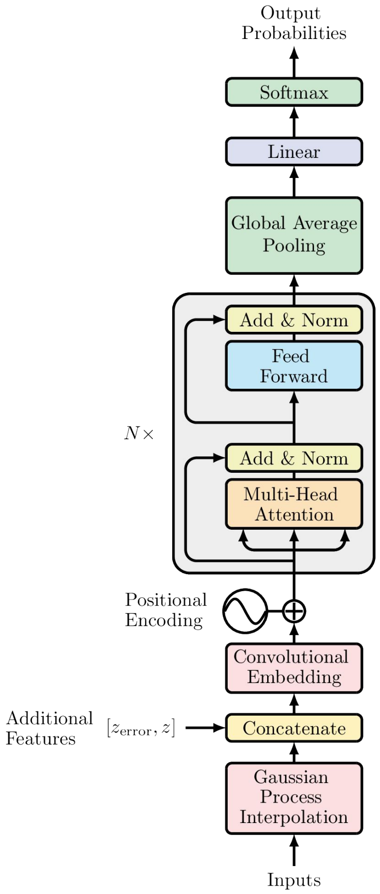
ML Pipeline

Gaussian Process Interpolation

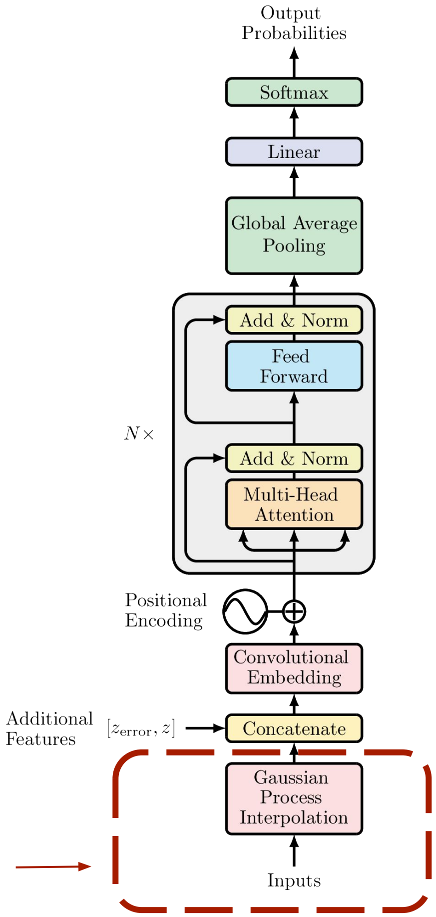
- 2D-Matern kernel for Gaussian Process interpolation
- Evaluate at regular period, 100 points in our case

Deep Learning Magic ...





Understanding Bottlenecks

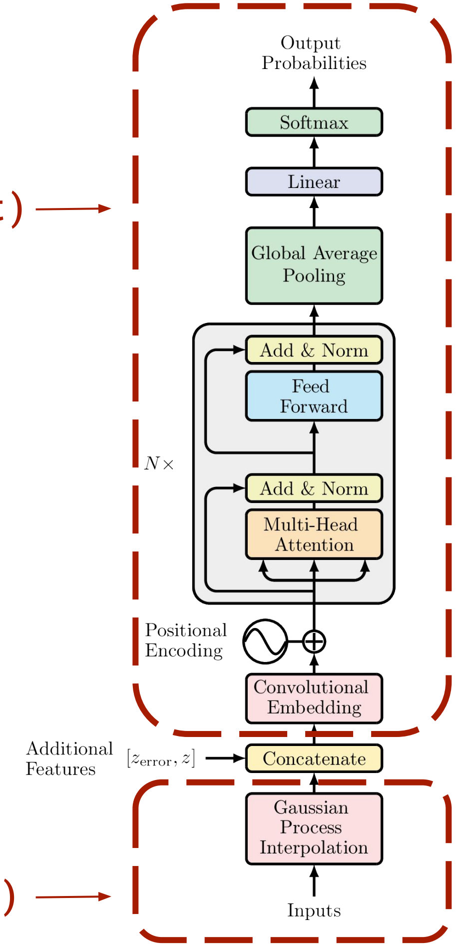


`fit_gps(raw_alert)` →

Understanding Bottlenecks

`model.predict(processed_alert)` →

`fit_gps(raw_alert)` →



Understanding Bottlenecks

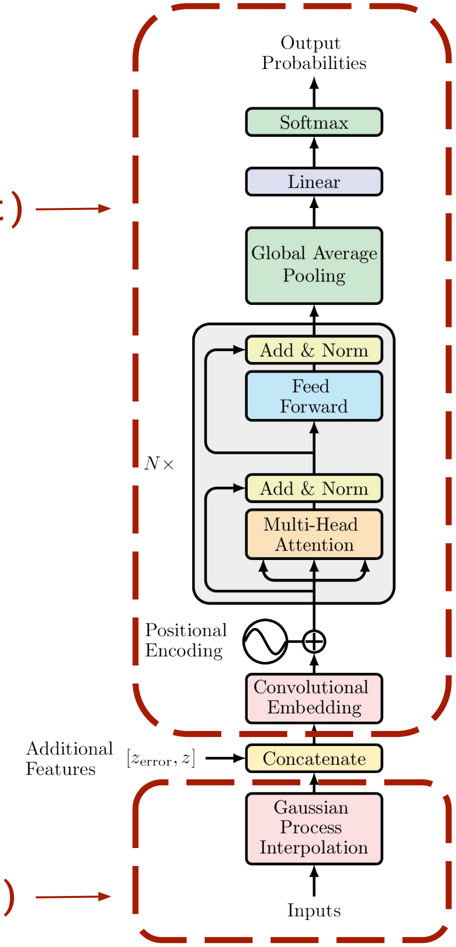
```
Total time: 5.85664 s
File: get_models.py
Function: get_model at line 29

Total time: 1.47076 s
File: ztf-load-run-lpa.py
Function: t2_probs at line 55

Line #    Hits        Time Per Hit \% Time Line Contents
==============================================================
...
206       16      139.6    8.7     9.5     df_gp_mean = generate_gp_all_objects()
...
...
...
...
...
212        8      180.8   22.6    12.3     X = df_gp_mean[cols]
213        8       12.3    1.5     0.8     X = rs(X)
...
...
...
217        8     1101.7  137.7    74.9     y_preds = model.predict(X)
```

`model.predict(processed_alert)` →

`fit_gps(raw_alert)` →



Understanding Bottlenecks

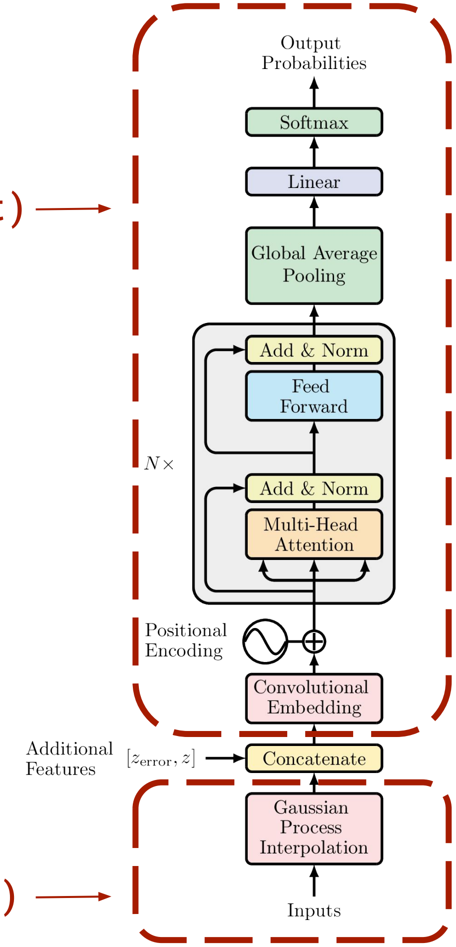
```
Total time: 5.85664 s
File: get_models.py
Function: get_model at line 29

Total time: 1.47076 s
File: ztf-load-run-lpa.py
Function: t2_probs at line 55

Line #    Hits       Time Per Hit \% Time Line Contents
==============================================================
...
206       16       139.6    8.7     9.5     df_gp_mean = generate_gp_all_objects()
...
...
...
...
...
212       8        180.8    22.6    12.3    X = df_gp_mean[cols]
213       8         12.3     1.5     0.8    X = rs(X)
...
...
...
217       8        1101.7   137.7   74.9    y_preds = model.predict(X)
```

`model.predict(processed_alert)` →

`fit_gps(raw_alert)` →



Understanding Bottlenecks

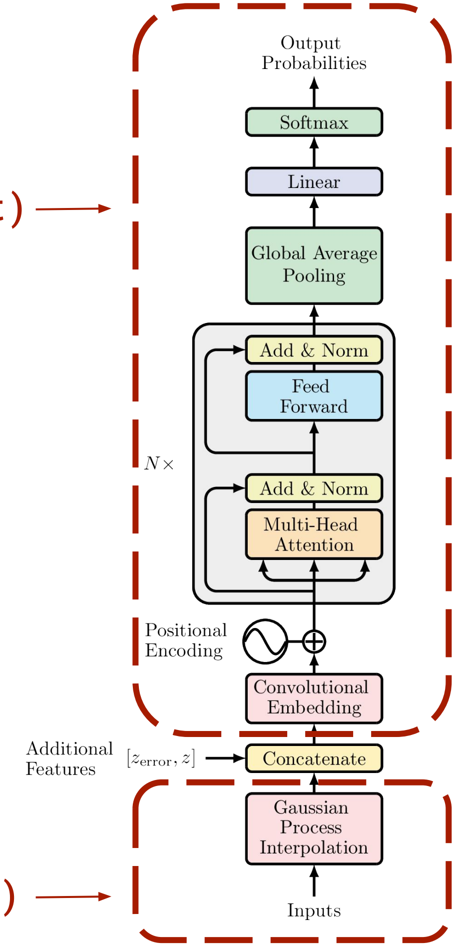
```
Total time: 5.85664 s
File: get_models.py
Function: get_model at line 29

Total time: 1.47076 s
File: ztf-load-run-lpa.py
Function: t2_probs at line 55

Line #    Hits        Time Per Hit \% Time Line Contents
==============================================================
...
206      16       139.6    8.7     9.5     df_gp_mean = generate_gp_all_objects()
...
...
...
...
...
212       8       180.8   22.6    12.3     X = df_gp_mean[cols]
213       8        12.3    1.5     0.8     X = rs(X)
...
...
...
217       8      1101.7  137.7    74.9     y_preds = model.predict(X)
```

model.predict(processed_alert) →

fit_gps(raw_alert) →



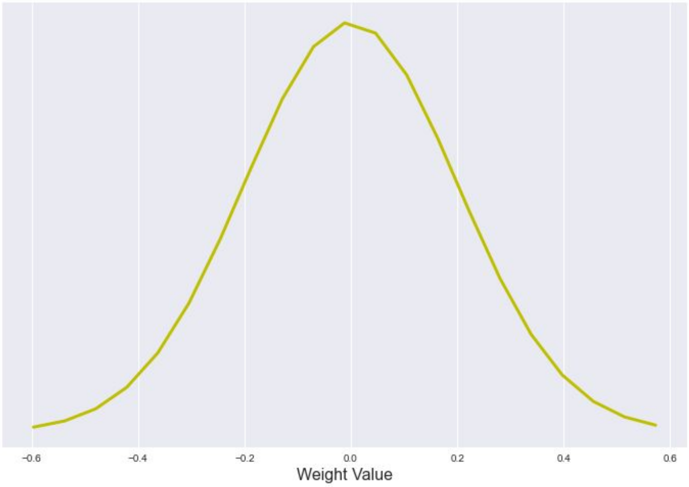
Deep Model Compression: An *Almost* Free Lunch

- Standout paper: Han et al. 2015: *Deep Compression*
    - Clustering
    - Pruning
    - Quantization
- Lossy process – expect some loss in accuracy (but improved compute and storage costs!)

Deep Model Compression: Clustering

Weight Clustering

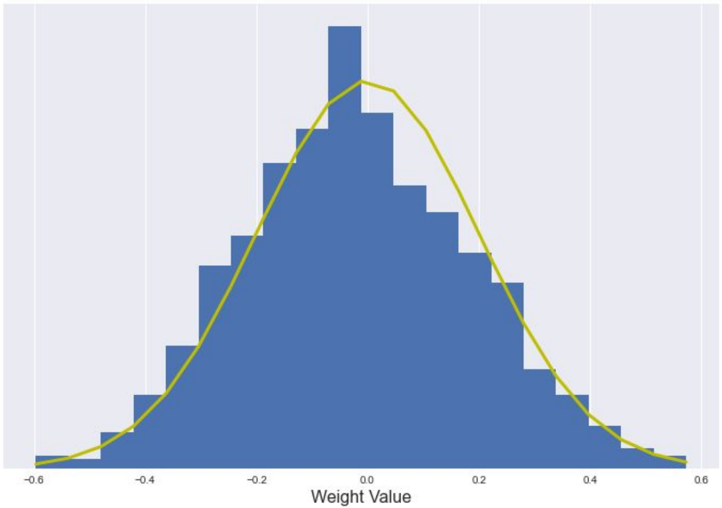
- Reduce number of unique weights with shared weight values
- Can then leverage Huffman encoding
- Note: best to avoid clustering early layers!



Deep Model Compression: Clustering

Weight Clustering

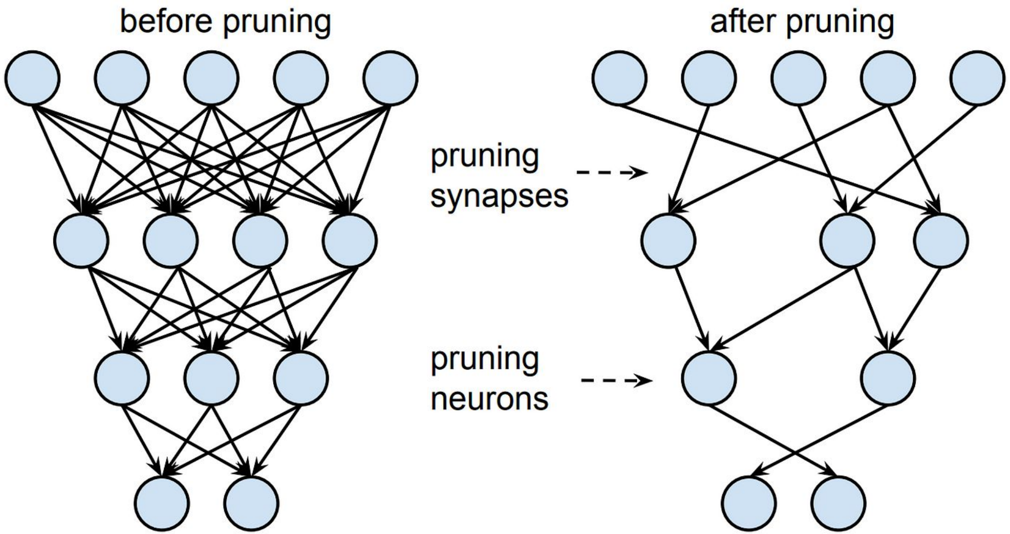
- Reduce number of unique weights with shared weight values
- Can then leverage Huffman encoding
- Note: best to avoid clustering early layers!



Deep Model Compression: Pruning

Weight Pruning

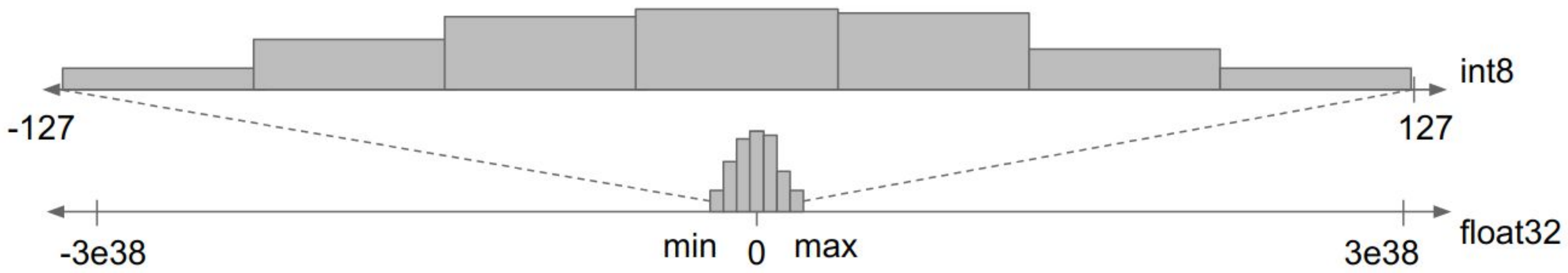
- Set low magnitude weights to zero
- Can then leverage Huffman encoding (again)
- Done *during* training



Deep Model Compression: Quantization

Weight Quantization

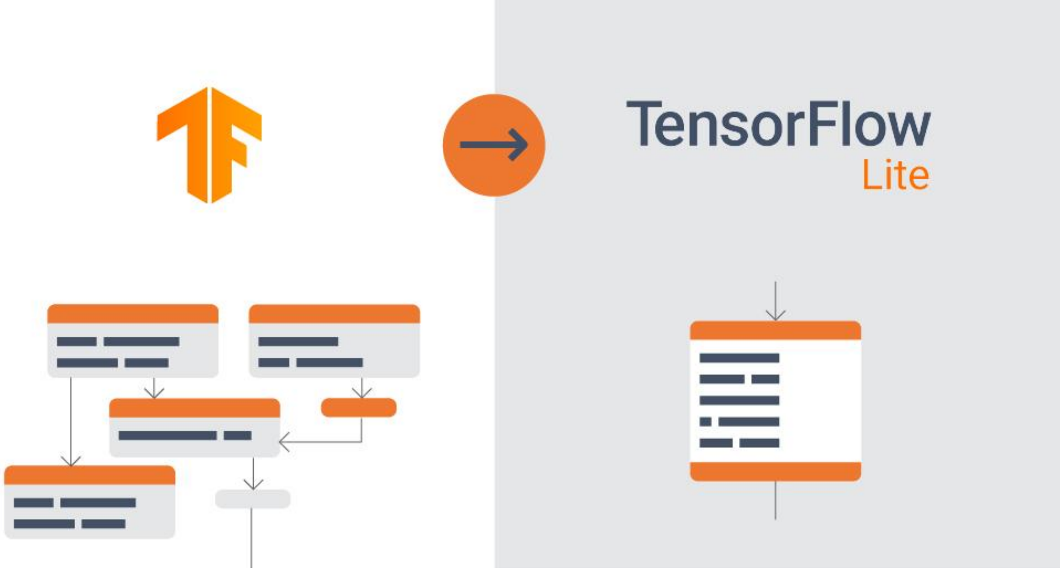
- Reduce precision of stored weight values
- Computations are still done at `float32`

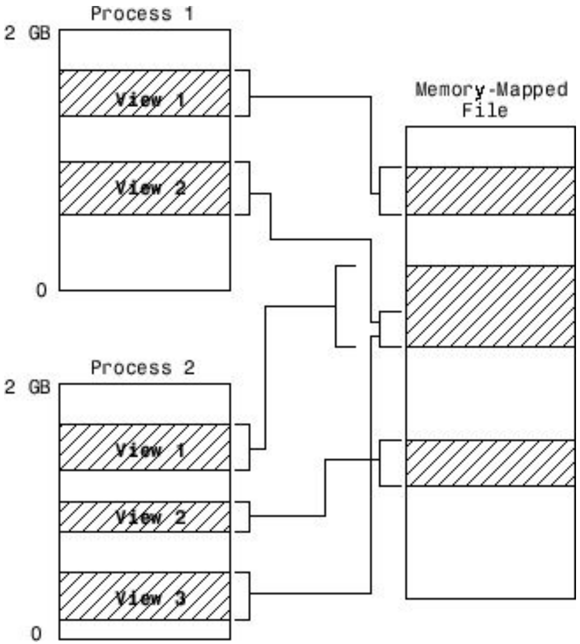


Efficient Frameworks & File Formats

From TensorFlow to TensorFlow-Lite

- Operator Fusion
- `ProtocolBuffers → FlatBuffers`





Local Tests

Compression Method	Model Size (kb)	Load Latency (s^{-3})	Inference Latency (s)	Loss
Baseline	1100	6324.145	0.333	0.968

Local Tests

Compression Method	Model Size (kb)	Load Latency (s^{-3})	Inference Latency (s)	Loss
Baseline	1100	6324.145	0.333	0.968
Baseline + Huffman	244	6015.565	0.224	0.968

Local Tests

Compression Method	Model Size (kb)	Load Latency (s^{-3})	Inference Latency (s)	Loss
Baseline	1100	6324.145	0.333	0.968
Baseline + Huffman	244	6015.565	0.224	0.968
Clustering	892	5559.868	0.227	0.836

Local Tests

Compression Method	Model Size (kb)	Load Latency (s^{-3})	Inference Latency (s)	Loss
Baseline	1100	6324.145	0.333	0.968
Baseline + Huffman	244	6015.565	0.224	0.968
Clustering	892	5559.868	0.227	0.836
Clustering + Pruning	688	5721.021	0.230	1.017

Local Tests

Compression Method	Model Size (kb)	Load Latency (s^{-3})	Inference Latency (s)	Loss
Baseline	1100	6324.145	0.333	0.968
Baseline + Huffman	244	6015.565	0.224	0.968
Clustering	892	5559.868	0.227	0.836
Clustering + Pruning	688	5721.021	0.230	1.017
Clustering + Huffman	240	4991.857	0.223	0.836

Local Tests

Compression Method	Model Size (kb)	Load Latency (s^{-3})	Inference Latency (s)	Loss
Baseline	1100	6324.145	0.333	0.968
Baseline + Huffman	244	6015.565	0.224	0.968
Clustering	892	5559.868	0.227	0.836
Clustering + Pruning	688	5721.021	0.230	1.017
Clustering + Huffman	240	4991.857	0.223	0.836
Clustering + Pruning + Huffman	128	5251.288	0.228	1.017

Local Tests

Compression Method	Model Size (kb)	Load Latency (s^{-3})	Inference Latency (s)	Loss
Baseline	1100	6324.145	0.333	0.968
Baseline + Huffman	244	6015.565	0.224	0.968
Clustering	892	5559.868	0.227	0.836
Clustering + Pruning	688	5721.021	0.230	1.017
Clustering + Huffman	240	4991.857	0.223	0.836
Clustering + Pruning + Huffman	128	5251.288	0.228	1.017
†Clustering	92	0.426	0.046	0.836

Local Tests

Compression Method	Model Size (kb)	Load Latency (s^{-3})	Inference Latency (s)	Loss
Baseline	1100	6324.145	0.333	0.968
Baseline + Huffman	244	6015.565	0.224	0.968
Clustering	892	5559.868	0.227	0.836
Clustering + Pruning	688	5721.021	0.230	1.017
Clustering + Huffman	240	4991.857	0.223	0.836
Clustering + Pruning + Huffman	128	5251.288	0.228	1.017
†Clustering	92	0.426	0.046	0.836
†Clustering + Quantization	**60**	**0.271**	**0.043**	**0.834**

Local Tests

Compression Method	Model Size (KB)	Load Latency (s^{-3})	Inference Latency (s)	Loss
Baseline	1100	6324.145	0.333	0.968
Baseline + Huffman	244	6015.565	0.224	0.968
Clustering	892	5559.868	0.227	0.836
Clustering + Pruning	688	5721.021	0.230	1.017
Clustering + Huffman	240	4991.857	0.223	0.836
Clustering + Pruning + Huffman	128	5251.288	0.228	1.017
†Clustering	92	0.426	0.046	0.836
†Clustering + Quantization	**60**	**0.271**	**0.043**	**0.834**

- **18 x** reduction in model size
- **24,000 x** load time improvement
- **8 x** inference latency improvement

Production Results

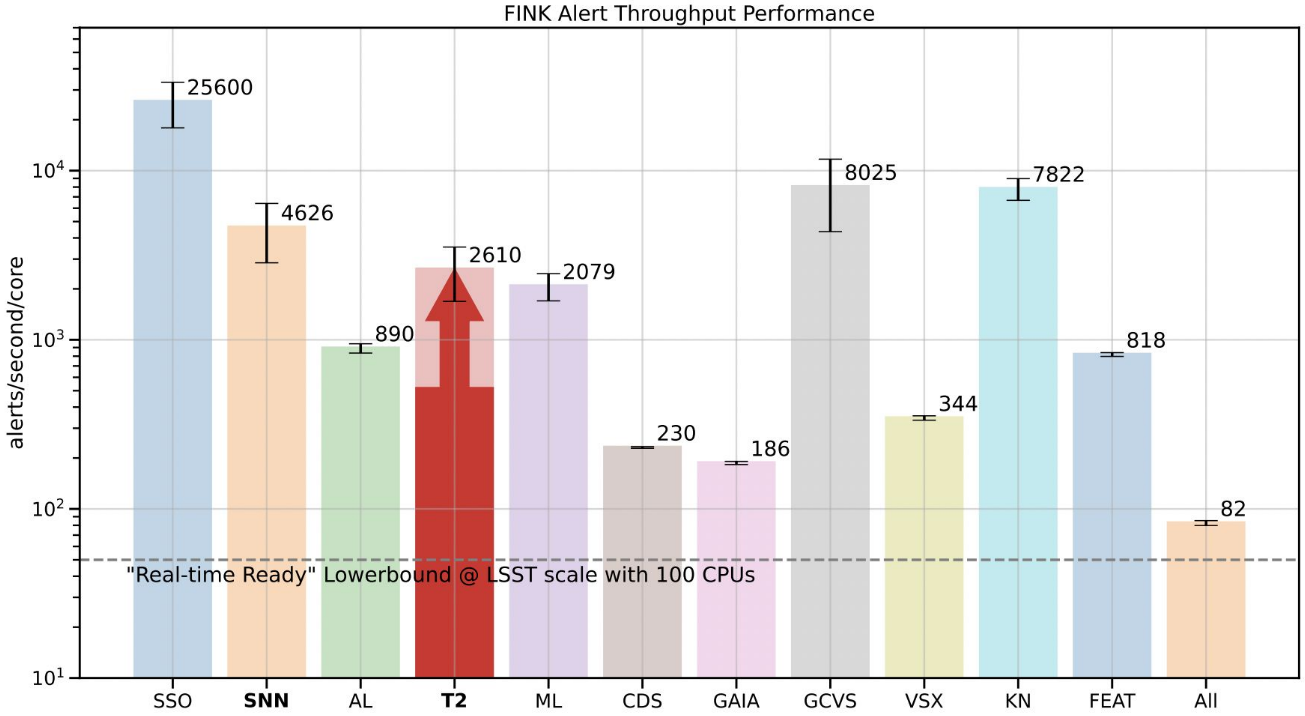
- One full night of ZTF alerts ~ 200K

- Require at least 2 points on light curve

- Averaged over 20 processing runs

Production Results

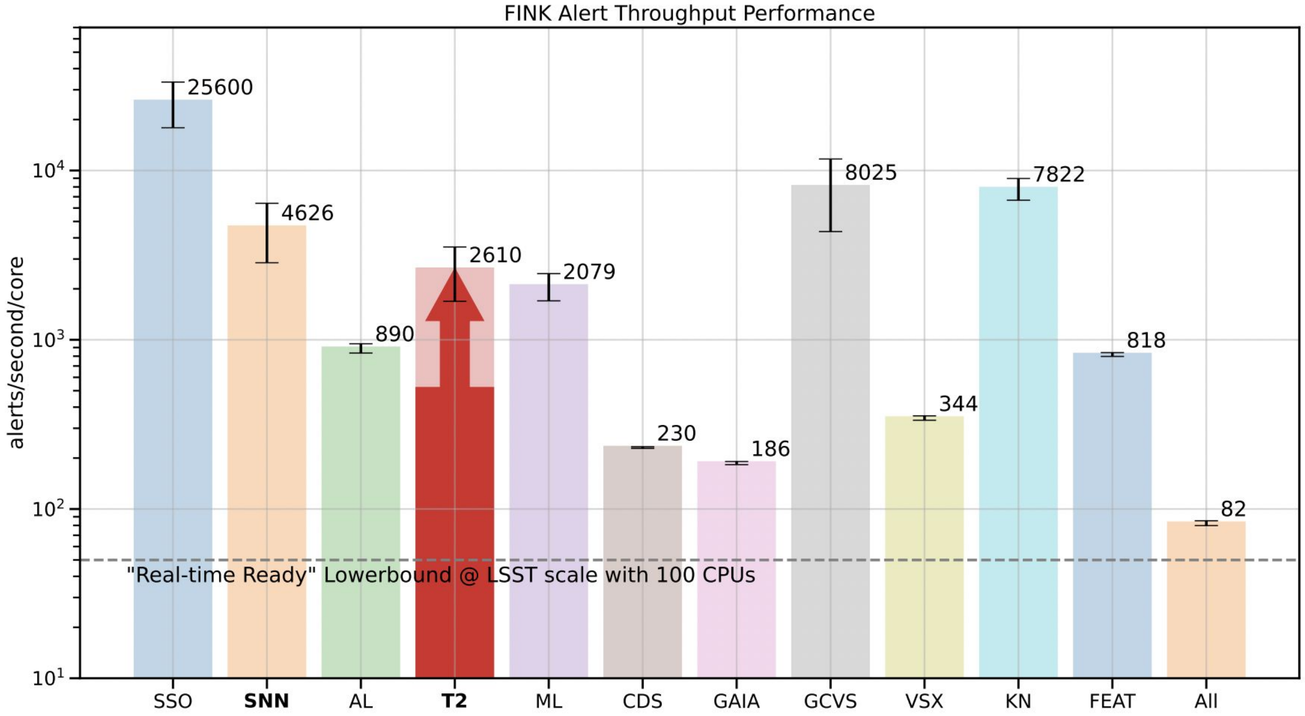
- One full night of ZTF alerts ~ 200K
- Require at least 2 points on light curve
- Averaged over 20 processing runs
- **5x** throughput



Takeaways...

- Use of deep compression can be your friend, *but be careful*

- Use FlatBuffers where possible



Questions?