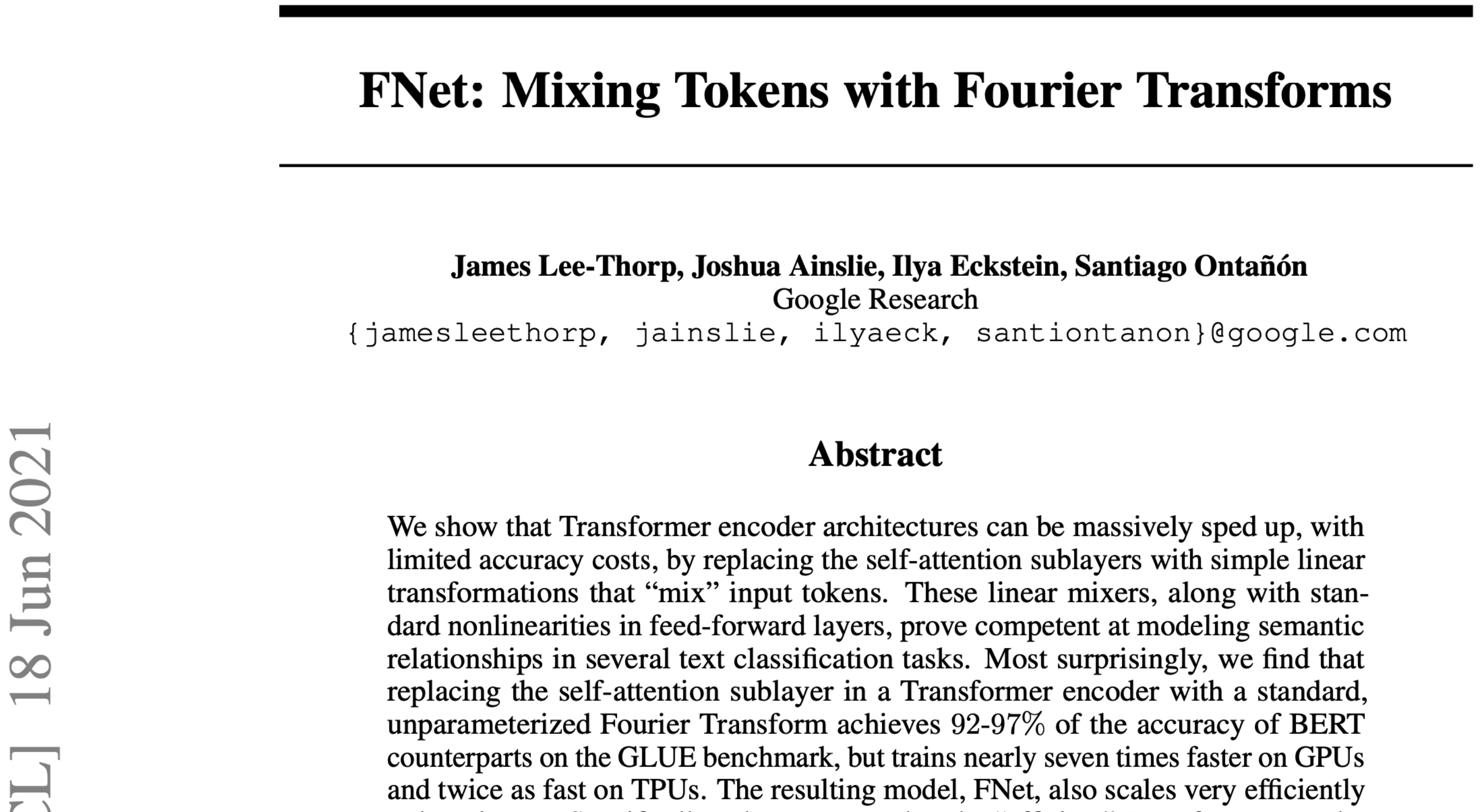
DBDMG reading group
November 26, 2021



FNet: Mixing Tokens with Fourier Transforms

James Lee-Thorp, Joshua Ainslie, Ilya Eckstein, Santiago Ontañón
Google Research
`{jamesleethorp, jainslie, ilyaeck, santiontanon}@google.com`

Abstract

We show that Transformer encoder architectures can be massively sped up, with limited accuracy costs, by replacing the self-attention sublayers with simple linear transformations that "mix" input tokens. These linear mixers, along with standard nonlinearities in feed-forward layers, prove competent at modeling semantic relationships in several text classification tasks. Most surprisingly, we find that replacing the self-attention sublayer in a Transformer encoder with a standard, unparameterized Fourier Transform achieves 92-97% of the accuracy of BERT counterparts on the GLUE benchmark, but trains nearly seven times faster on GPUs and twice as fast on TPUs. The resulting model, FNet, also scales very efficiently

FNet: Mixing Tokens with Fourier Transforms

- We mix parts of sequences to learn semantics
- Since ~2017, in NLP, predominantly:

$O(|S|^2)$

$$\mathrm{Attention}(Q, K, V) = \mathrm{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

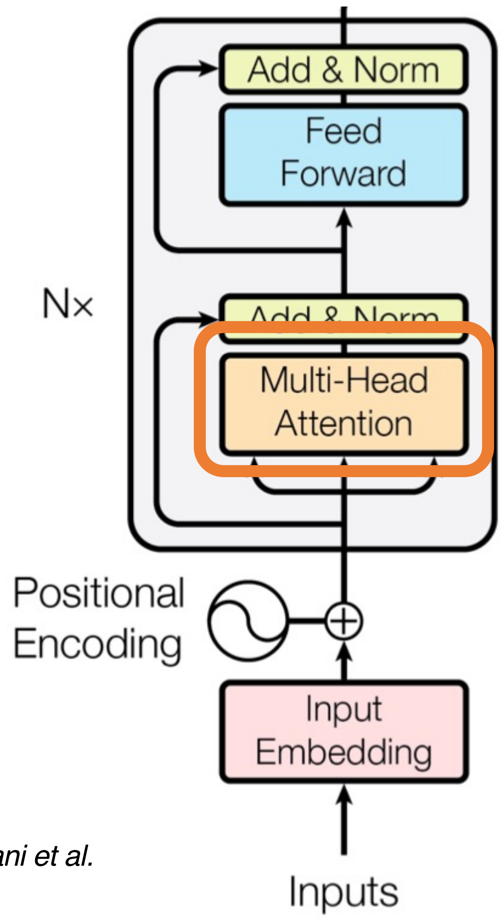
- Recent work propose lighter attention variants (sparsity, randomicity, etc..)

- The authors get rid off attention

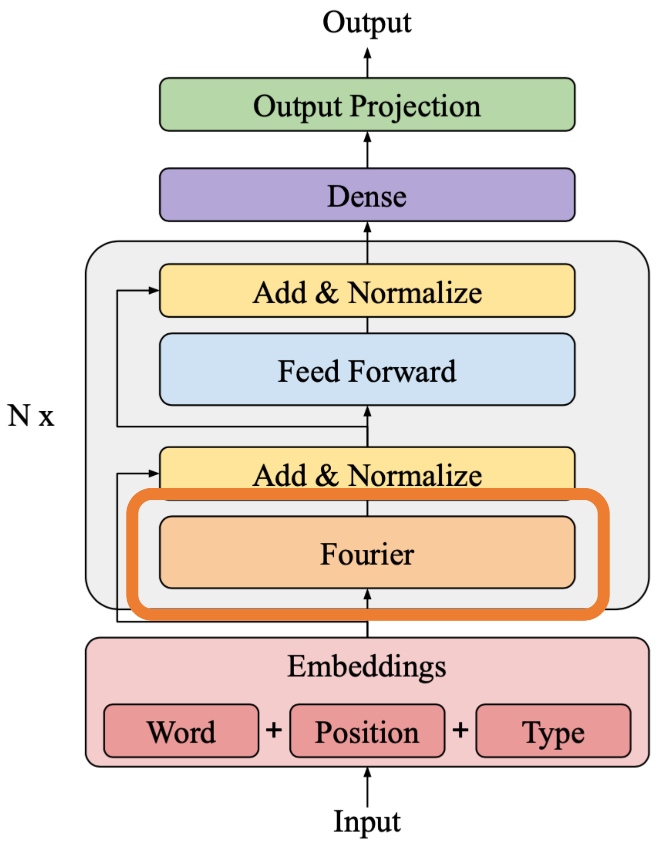
Can we use simpler, lighter mixing strategies
and retain expressiveness (and performance)?

Linear mixing and Fourier Transform are promising avenues

FNet: Mixing Tokens with Fourier Transforms

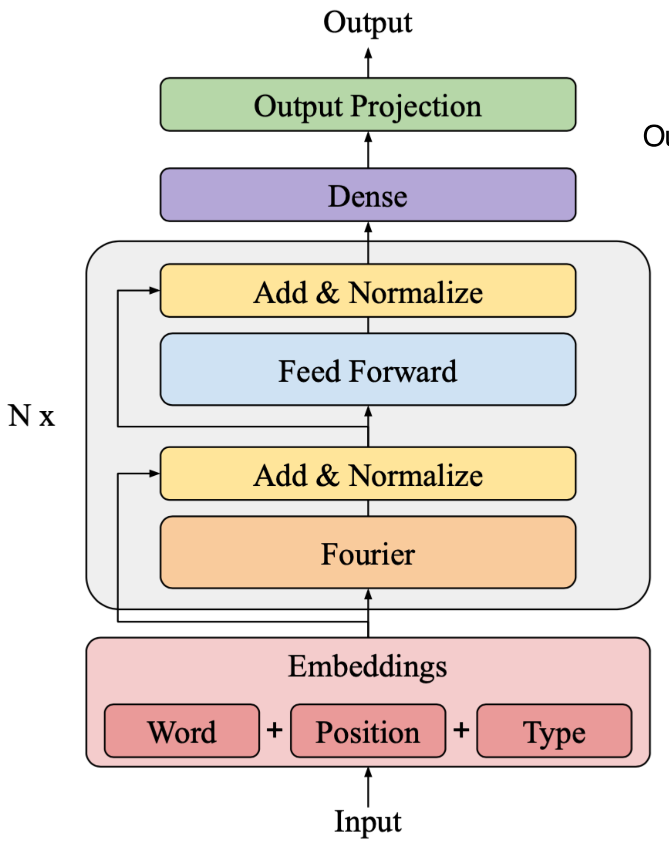


Vaswani et al.



FNet encoder

FNet: Mixing Tokens with Fourier Transforms



FNet encoder

Discrete Fourier Transform

Original sequence

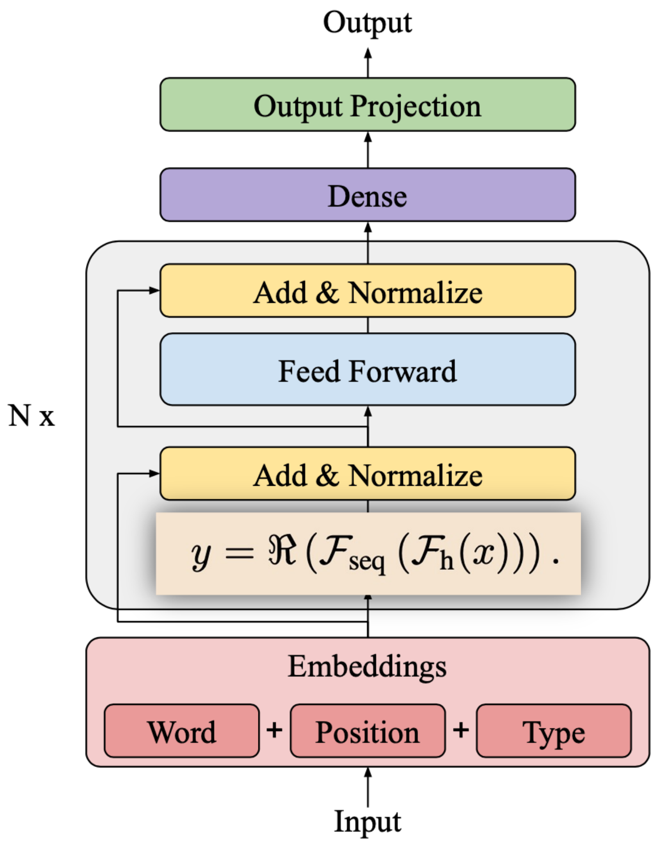
Sinewaves

Output sequence

$$X_k = \sum_{n=0}^{N-1} x_n e^{-\frac{2\pi i}{N} nk}, \quad 0 \le k \le N-1.$$

- Output: (complex) weighted sum of sine waves at different frequencies
- Non-parametric, deterministic
- Each input contributes to each output
- From “time” to “frequency” domain

FNet: Mixing Tokens with Fourier Transforms



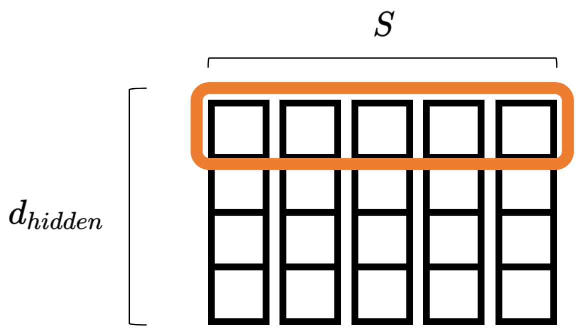
FNet encoder

Discrete Fourier Transform

$$X_k = \sum_{n=0}^{N-1} x_n e^{-\frac{2\pi i}{N} nk}, \quad 0 \leq k \leq N-1.$$

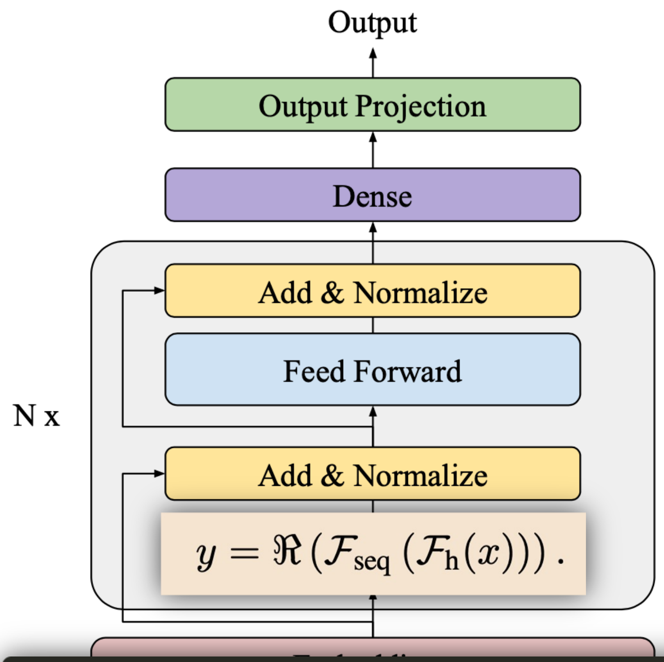
New mixing:

- Two 1D DFT along *hidden* and *sequence*
- Take the Real part



Input tokens

FNet: Mixing Tokens with Fourier Transforms

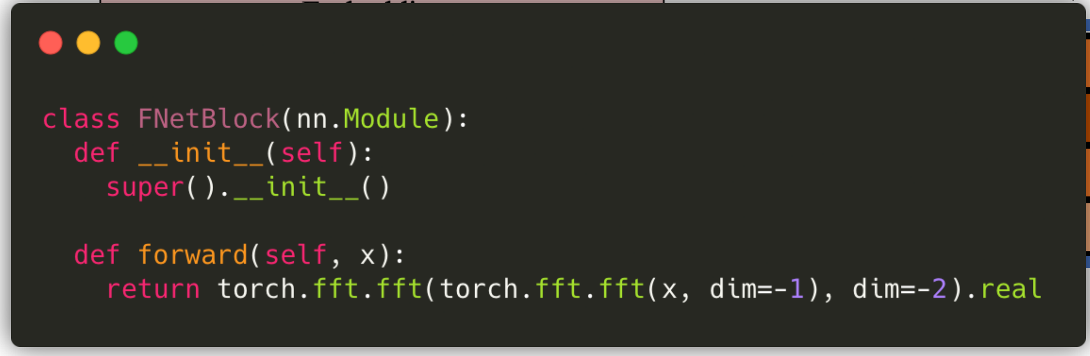

Output
Output Projection
Dense
Add & Normalize
Feed Forward
Add & Normalize
$y = \Re\left(\mathcal{F}_{\text{seq}}\left(\mathcal{F}_{\text{h}}(x)\right)\right).$
N x

Discrete Fourier Transform

$$X_k = \sum_{n=0}^{N-1} x_n e^{-\frac{2\pi i}{N}nk}, \quad 0 \leq k \leq N-1.$$

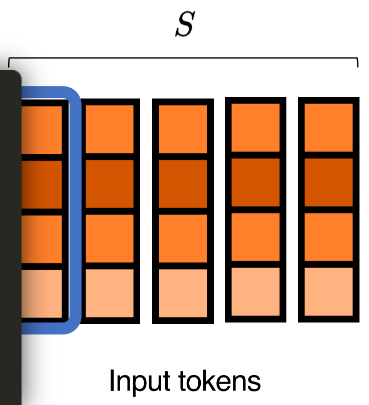
New mixing:

- Two 1D DFT along *hidden* and *sequence*
- Take the Real part

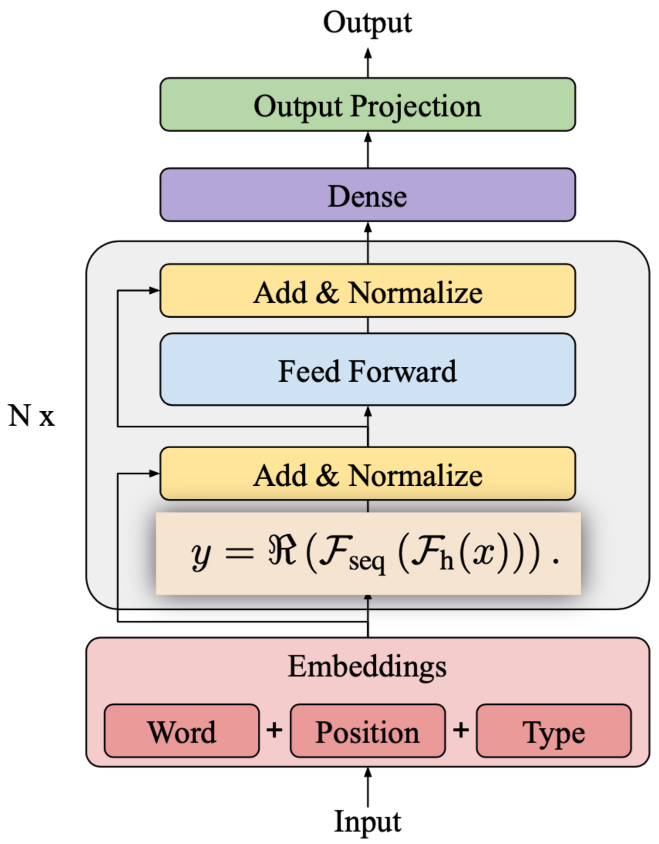


```
class FNetBlock(nn.Module):
  def __init__(self):
    super().__init__()

  def forward(self, x):
    return torch.fft.fft(torch.fft.fft(x, dim=-1), dim=-2).real
```



FNet: Mixing Tokens with Fourier Transforms



FNet

Intuition:

- Going back and forth between "time" and frequency domain*
- In "frequency", the FF sublayer performs a (large kernel) convolution

** almost. The resulting DFT cannot be inverted since only the real part is taken.*

Can we use simpler, lighter mixing strategies and retain expressiveness (and performance)?

Transfer Learning

- MLM & NSP pre-training (dropped in [v2]) 🤔
  - C4 dataset (Raffel et al., 2019)
- GLUE benchmark
- Speed and accuracy trade-off

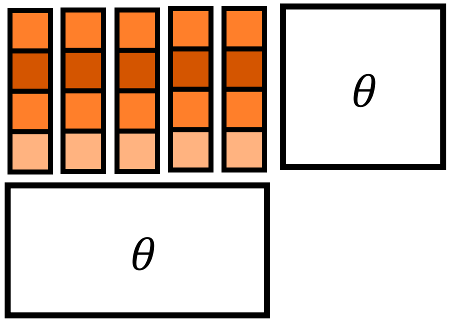
Long-Range Arena benchmark (Tay et al., 2020)

- Tasks with long range dependencies
- Vanilla Transformer is (by a small margin) the second most accurate
- Performer (Choromanski et al., 2020) is the fastest

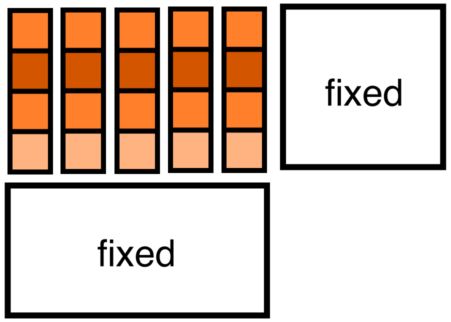
Transfer learning

- Test FNet against 4 models

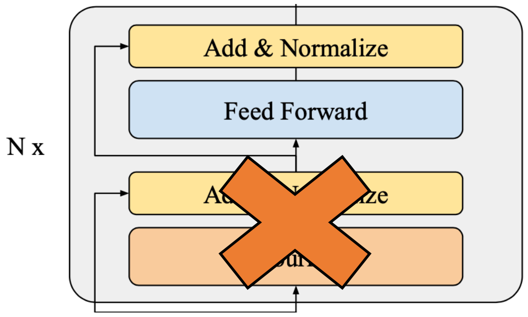
BERT



Linear mixing

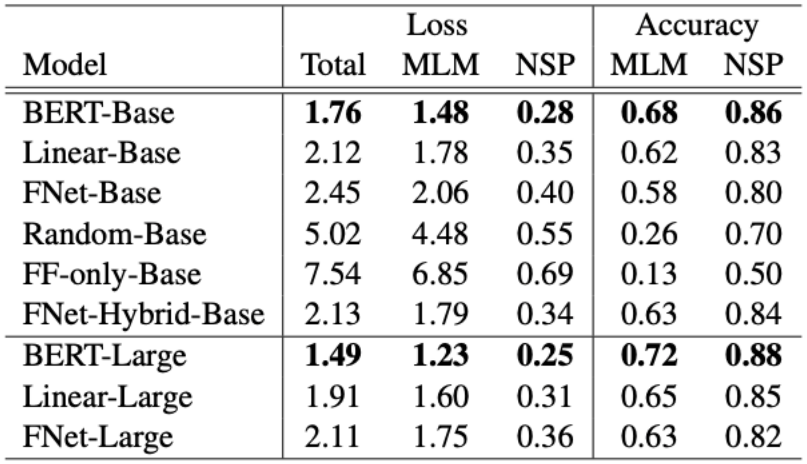


Random mixing



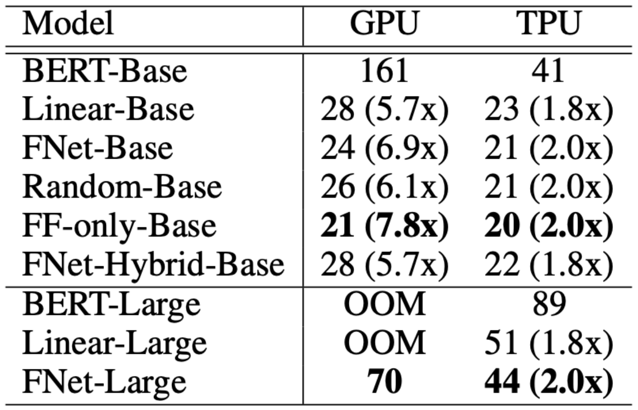
Feed Forward-only

Transfer learning: MLM and NSP pre-training



Model	Loss			Accuracy	
	Total	MLM	NSP	MLM	NSP
BERT-Base	**1.76**	**1.48**	**0.28**	**0.68**	**0.86**
Linear-Base	2.12	1.78	0.35	0.62	0.83
FNet-Base	2.45	2.06	0.40	0.58	0.80
Random-Base	5.02	4.48	0.55	0.26	0.70
FF-only-Base	7.54	6.85	0.69	0.13	0.50
FNet-Hybrid-Base	2.13	1.79	0.34	0.63	0.84
BERT-Large	**1.49**	**1.23**	**0.25**	**0.72**	**0.88**
Linear-Large	1.91	1.60	0.31	0.65	0.85
FNet-Large	2.11	1.75	0.36	0.63	0.82

Results on TPU (4 x 4 v3)



Model	GPU	TPU
BERT-Base	161	41
Linear-Base	28 (5.7x)	23 (1.8x)
FNet-Base	24 (6.9x)	21 (2.0x)
Random-Base	26 (6.1x)	21 (2.0x)
FF-only-Base	**21 (7.8x)**	**20 (2.0x)**
FNet-Hybrid-Base	28 (5.7x)	22 (1.8x)
BERT-Large	OOM	89
Linear-Large	OOM	51 (1.8x)
FNet-Large	**70**	**44 (2.0x)**

Pre-training milliseconds per step with
BS of 64 on GPU and 256 on TPU

- BERT is more expressive thanks to task specific, per token weights
- Linear and FNet are less accurate but much faster
- Not all the mixings are valid…
- … but they are required
- FNet-Hybrid uses attention in the last two layers

Transfer learning: MLM and NSP pre-training

	Dimensions		Parameters (millions)		
Name	d_h	Layers	BERT	Linear	FNet
Large	1024	24	338	268	237
Base	768	12	111	93	83
	512	12	55	49	42
	512	8	42	38	34
	256	8	15	15	13
"Mini"	512	4	30	28	26
	256	4	12	12	11
"Micro"	256	2	10	10	10
	128	2	5	5	4

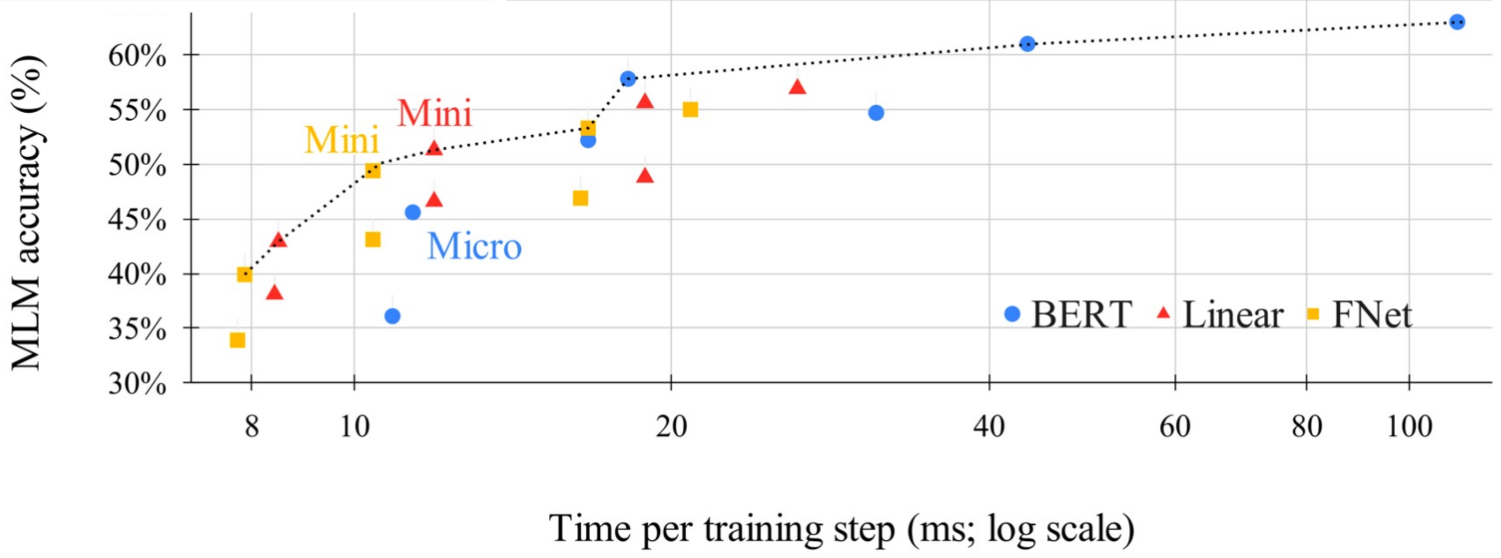


Figure 2: Speed-accuracy trade-offs for GPU pre-training. The dotted line shows the Pareto efficiency frontier. For smaller models, the FNet (yellow squares) and Linear (red triangles) models define the frontier, indicating better speed-accuracy trade-offs.

Transfer learning: GLUE

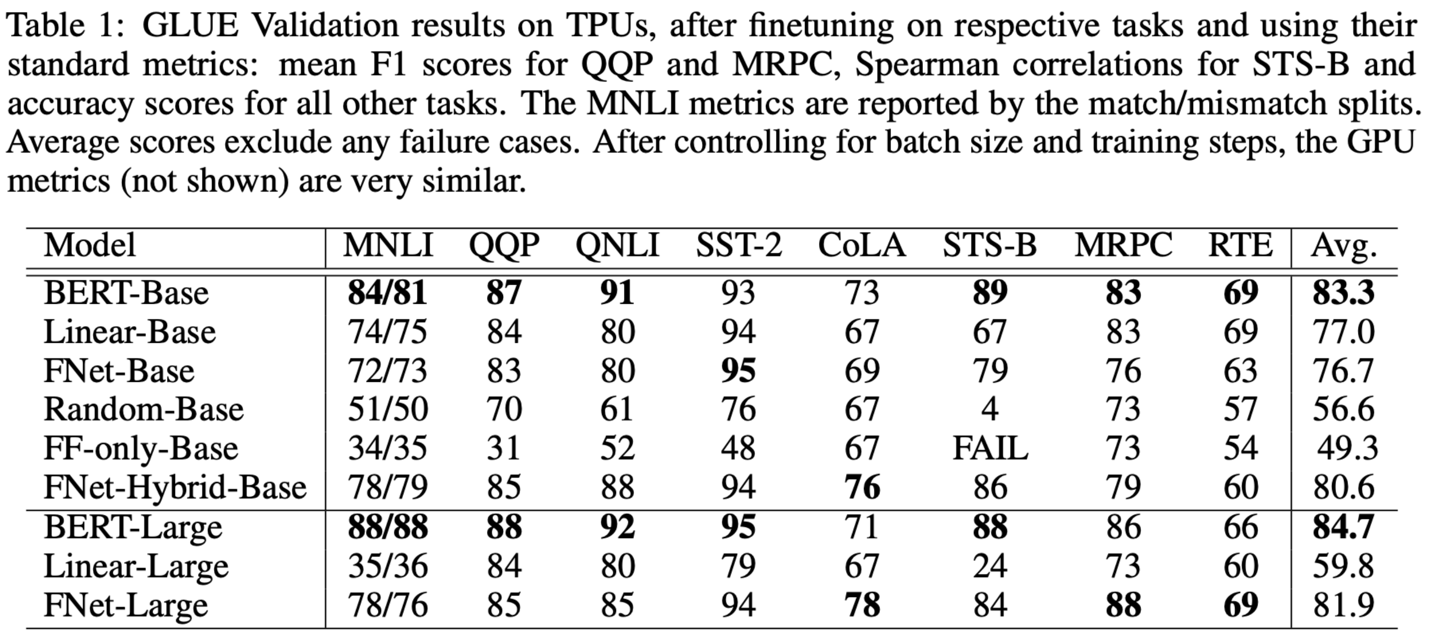


Table 1: GLUE Validation results on TPUs, after finetuning on respective tasks and using their standard metrics: mean F1 scores for QQP and MRPC, Spearman correlations for STS-B and accuracy scores for all other tasks. The MNLI metrics are reported by the match/mismatch splits. Average scores exclude any failure cases. After controlling for batch size and training steps, the GPU metrics (not shown) are very similar.

Model	MNLI	QQP	QNLI	SST-2	CoLA	STS-B	MRPC	RTE	Avg.
BERT-Base	**84/81**	**87**	**91**	93	73	**89**	**83**	**69**	**83.3**
Linear-Base	74/75	84	80	94	67	67	83	69	77.0
FNet-Base	72/73	83	80	**95**	69	79	76	63	76.7
Random-Base	51/50	70	61	76	67	4	73	57	56.6
FF-only-Base	34/35	31	52	48	67	FAIL	73	54	49.3
FNet-Hybrid-Base	78/79	85	88	94	**76**	86	79	60	80.6
BERT-Large	**88/88**	**88**	**92**	**95**	71	**88**	86	66	**84.7**
Linear-Large	35/36	84	80	79	67	24	73	60	59.8
FNet-Large	78/76	85	85	94	**78**	84	**88**	**69**	81.9

- Results match the ones from pre-training
- Linear encoder is still slightly more accurate but slower than FNet
    - With larger models, Linear gets worse

Long-Range Arena benchmark

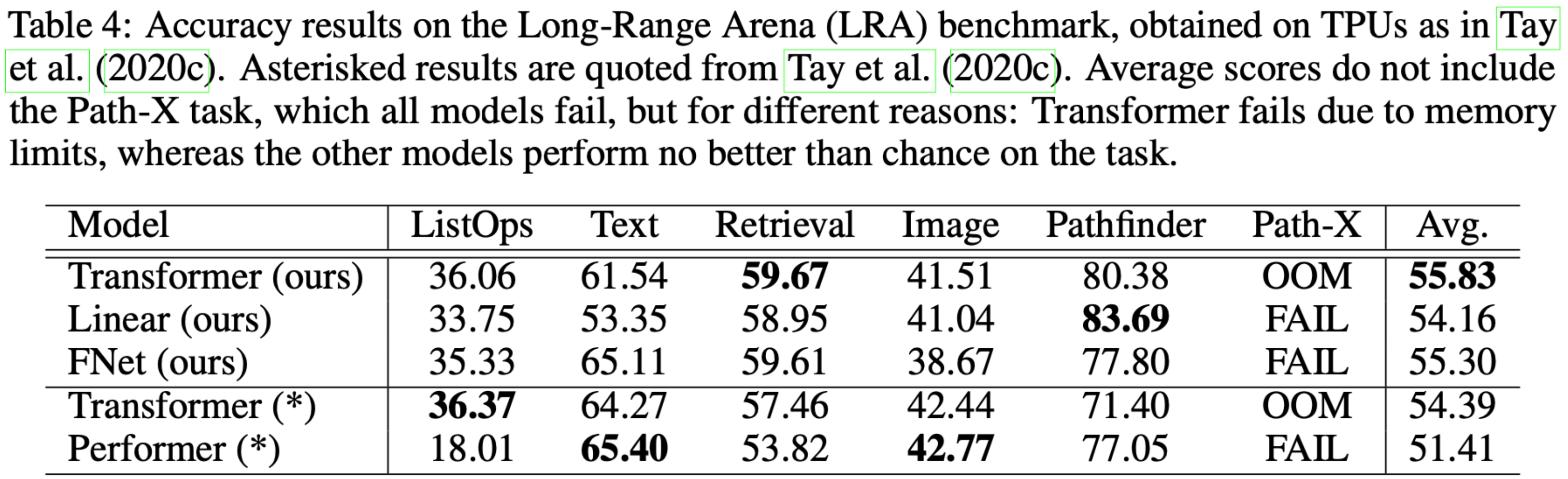


Table 4: Accuracy results on the Long-Range Arena (LRA) benchmark, obtained on TPUs as in Tay et al. (2020c). Asterisked results are quoted from Tay et al. (2020c). Average scores do not include the Path-X task, which all models fail, but for different reasons: Transformer fails due to memory limits, whereas the other models perform no better than chance on the task.

Model	ListOps	Text	Retrieval	Image	Pathfinder	Path-X	Avg.
Transformer (ours)	36.06	61.54	**59.67**	41.51	80.38	OOM	**55.83**
Linear (ours)	33.75	53.35	58.95	41.04	**83.69**	FAIL	54.16
FNet (ours)	35.33	65.11	59.61	38.67	77.80	FAIL	55.30
Transformer (*)	**36.37**	64.27	57.46	42.44	71.40	OOM	54.39
Performer (*)	18.01	**65.40**	53.82	**42.77**	77.05	FAIL	51.41

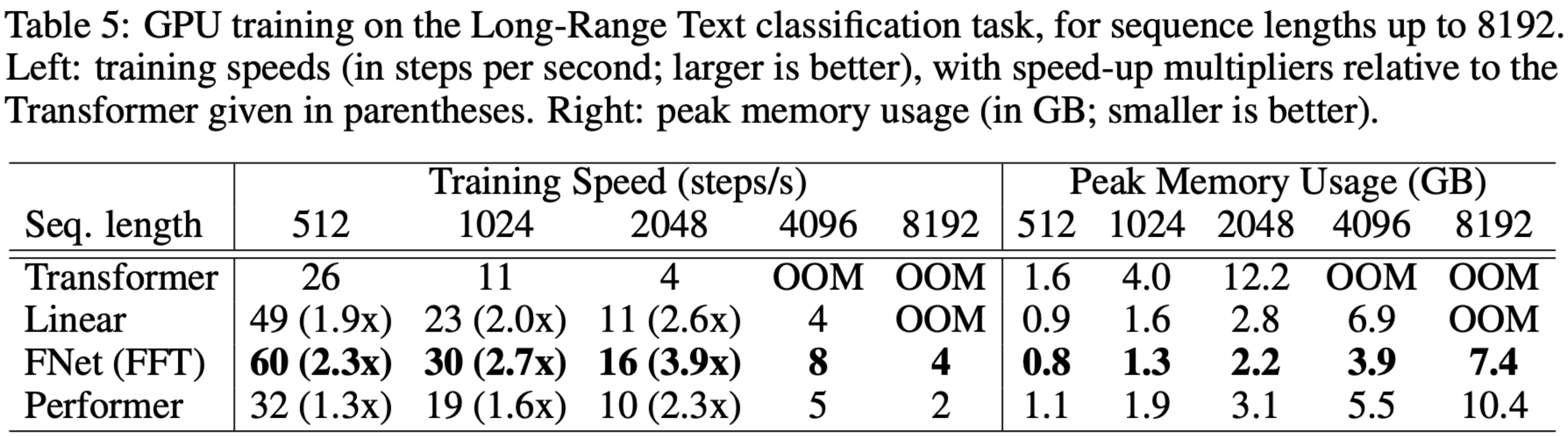


Table 5: GPU training on the Long-Range Text classification task, for sequence lengths up to 8192. Left: training speeds (in steps per second; larger is better), with speed-up multipliers relative to the Transformer given in parentheses. Right: peak memory usage (in GB; smaller is better).

	Training Speed (steps/s)					Peak Memory Usage (GB)				
Seq. length	512	1024	2048	4096	8192	512	1024	2048	4096	8192
Transformer	26	11	4	OOM	OOM	1.6	4.0	12.2	OOM	OOM
Linear	49 (1.9x)	23 (2.0x)	11 (2.6x)	4	OOM	0.9	1.6	2.8	6.9	OOM
FNet (FFT)	**60 (2.3x)**	**30 (2.7x)**	**16 (3.9x)**	**8**	**4**	**0.8**	**1.3**	**2.2**	**3.9**	**7.4**
Performer	32 (1.3x)	19 (1.6x)	10 (2.3x)	5	2	1.1	1.9	3.1	5.5	10.4

The authors’ take

- Linear units may work as a drop-in replacement for the attention mechanism
- FNets achieve 92 and 97% of BERT-Base and BERT-Large counterparts’ accuracy on GLUE but train seven times faster on GPUs and twice on TPUs
- FNet is comparable to efficient Transformers on LRA benchmark, with a lighter memory footprint
- FNet can be a lightweight, distilled student model for resource-constrained settings

My take

- The performance drop is significant
- Linear encoder is as competitive as FNet (see MLP-Mixer, Tolstikhin et al., 2021)
- Results from the hybrid model are interesting: does the scaled dot-product attention act as a pooler of raw, unrouted information?

My take

- The performance drop is significant
- Linear encoder is as competitive as FFT (see MLP-Mixer, Tolstikhin et al., 2021)
- Results from the hybrid model are interesting: does the scaled dot-product attention act as pooler of raw, unrouted information?

Thanks!