Group G11

NanoGPT is a minimal GPT-2 implementation by Andrej Karpathy that inspired a community speedrun challenge: reaching a validation loss of 3.28 as fast as possible on 8×H100 GPUs, with a current record of 1.435 minutes. In this work, we port and optimize this training pipeline in JAX for Google TPU v6e hardware, investigating how this architecture competes on a well-defined benchmark. Our implementation incorporates several algorithmic improvements including rotary positional embeddings (RoPE),the Muon optimizer with layer-wise learning rate decay for transformer weights, with Adam used for token embeddings and the language model head each with a warmdown schedule, logit soft-capping based on the Gemma 2 paper, and architecture dimensions specifically tuned to TPU v6e's hardware constraints. Preliminary results show we achieve a validation loss of <3.28 in 24 minutes on 8×TPU v6e.

Project

Based on modded-nanoGPT

Train model to 3.28 validation loss on FineWeb dataset

Use Google TPU v6e-8 chips

Current world record 2026-03-18 on 8 H100 is 1.435 minutes, original time was 45 minutes

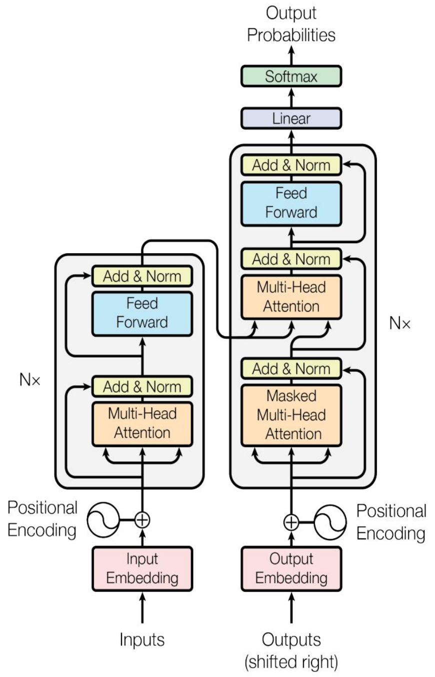


Figure 1: The Transformer - model architecture.

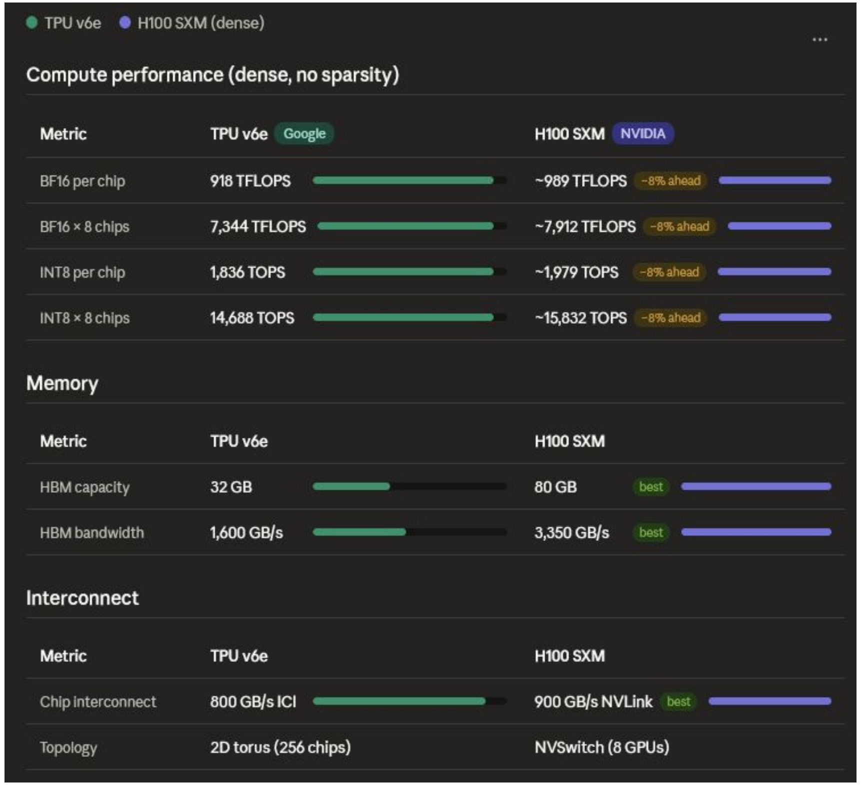
Setup

Each run used the same TPU v6e-8 chips

Same 10 billion token subset each run already tokenized.

Each run had the same key to handel randomness

In order for run to count Val loss < 3.28



● TPU v6e ● H100 SXM (dense)

Compute performance (dense, no sparsity)

Metric	TPU v6e Google	H100 SXM NVIDIA
BF16 per chip	918 TFLOPS	~989 TFLOPS ~8% ahead
BF16 × 8 chips	7,344 TFLOPS	~7,912 TFLOPS ~8% ahead
INT8 per chip	1,836 TOPS	~1,979 TOPS ~8% ahead
INT8 × 8 chips	14,688 TOPS	~15,832 TOPS ~8% ahead

Memory

Metric	TPU v6e	H100 SXM
HBM capacity	32 GB	80 GB best
HBM bandwidth	1,600 GB/s	3,350 GB/s best

Interconnect

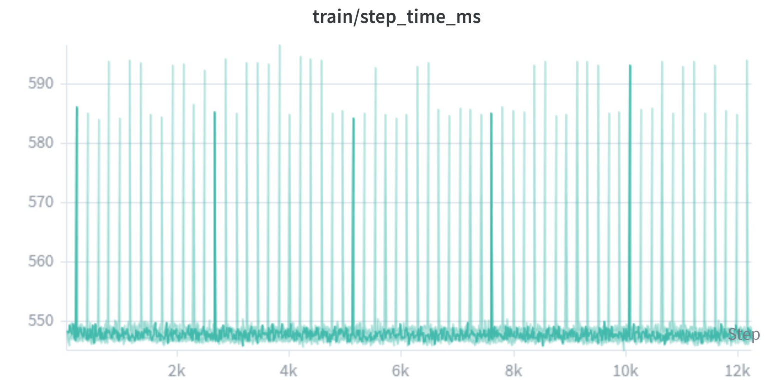
Metric	TPU v6e	H100 SXM
Chip interconnect	800 GB/s ICI	900 GB/s NVLink best
Topology	2D torus (256 chips)	NVSwitch (8 GPUs)

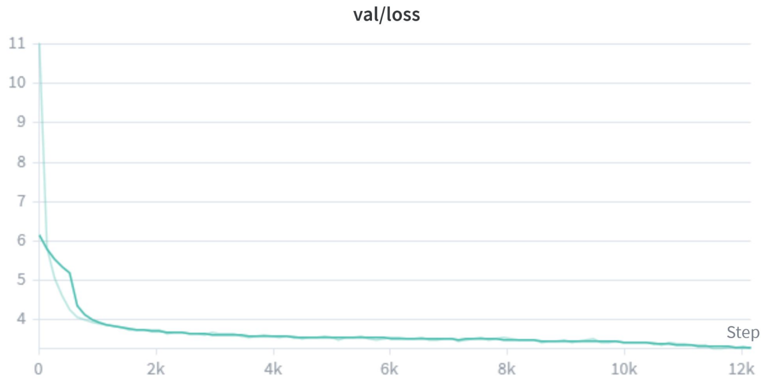
Run 1: Baseline

Total time 105 min vs expected 31 min

Issues: Poor data loading, weights handling

Notable implementations: Rotary Positional Embedding, high learning rate

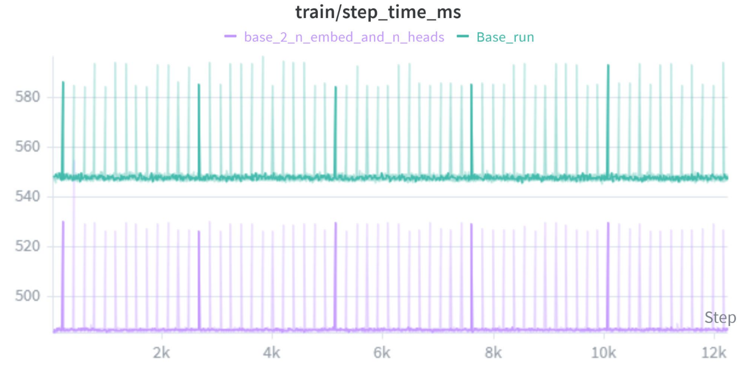


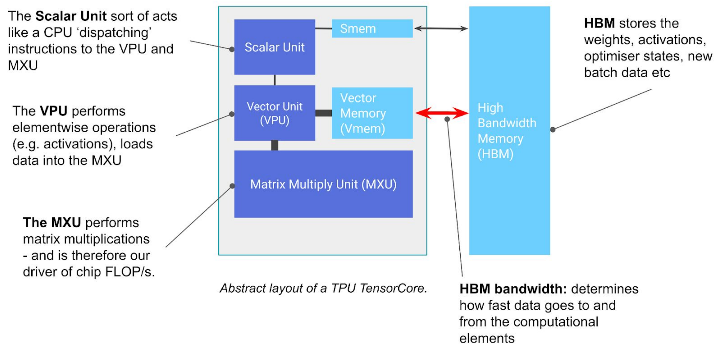


Run 2: Hardware-Aware Optimization

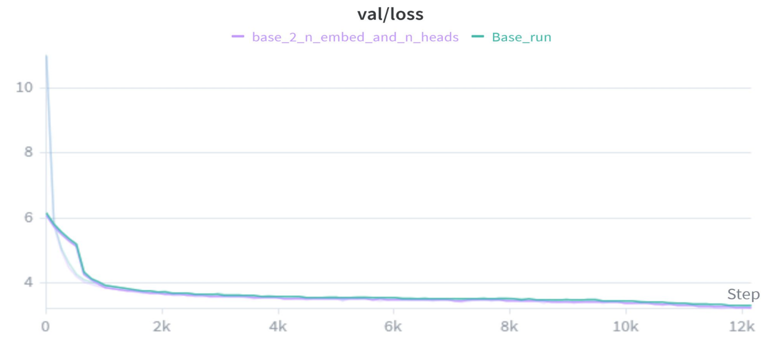
Changed the so that head dim = 256

Improvement 13% over baseline




The **Scalar Unit** sort of acts like a CPU 'dispatching' instructions to the VPU and MXU
The **VPU** performs elementwise operations (e.g. activations), loads data into the MXU
The MXU performs matrix multiplications - and is therefore our driver of chip FLOP/s.
Scalar Unit
Smem
Vector Unit (VPU)
Vector Memory (Vmem)
Matrix Multiply Unit (MXU)
High Bandwidth Memory (HBM)
HBM stores the weights, activations, optimiser states, new batch data etc
HBM bandwidth: determines how fast data goes to and from the computational elements

Abstract layout of a TPU TensorCore.

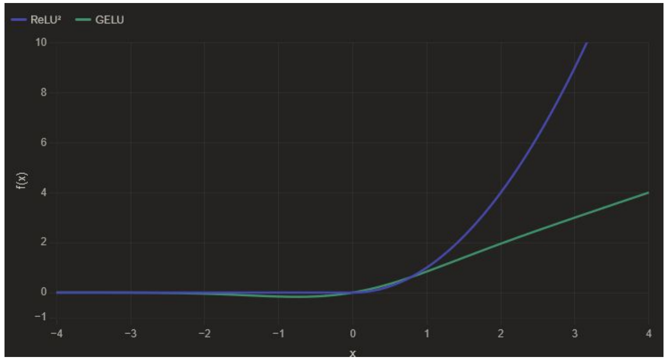


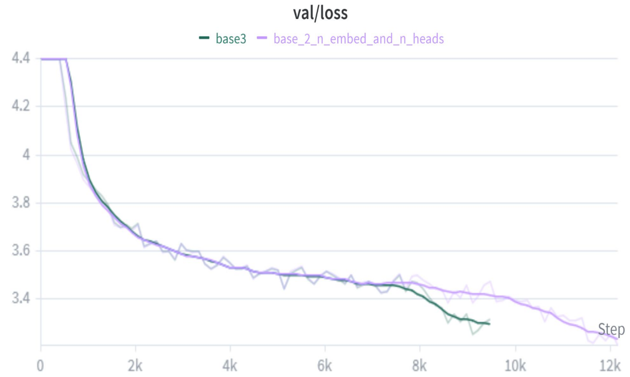
Run 3: Relu^2 and RoPE frequency

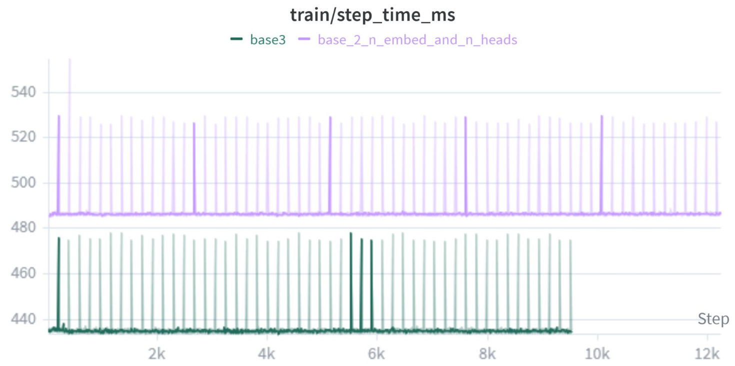
12.5% speed up / step, total time 65 minutes

Changed from Gelu to Relu^2

RoPE frequency from 10000 to 1024 due to lower context length from original paper.





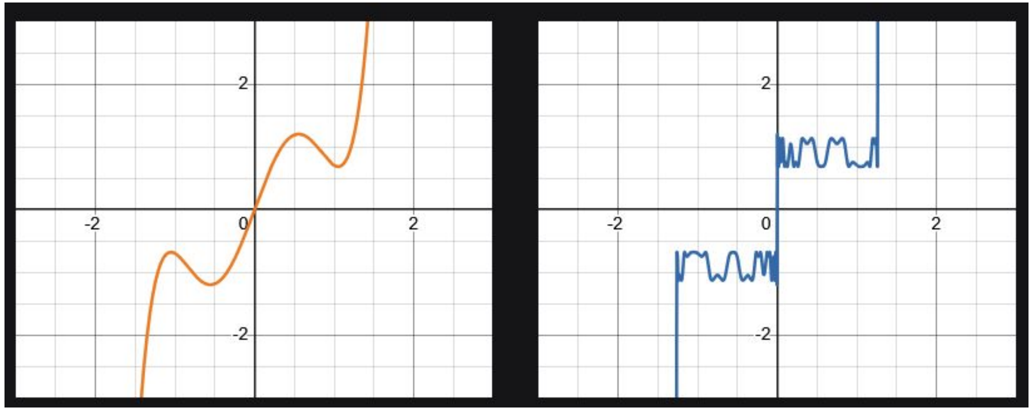


Run 4: Muon

Total time 37 minutes 76% then run 3

```
Algorithm 2 Muon
Require: Learning rate $\eta$, momentum $\mu$
 1: Initialize $B_0 \leftarrow 0$
 2: for $t = 1, \ldots$ do
 3:     Compute gradient $G_t \leftarrow \nabla_\theta \mathcal{L}_t(\theta_{t-1})$
 4:     $B_t \leftarrow \mu B_{t-1} + G_t$
 5:     $O_t \leftarrow \mathrm{NewtonSchulz5}(B_t)$
 6:     Update parameters $\theta_t \leftarrow \theta_{t-1} - \eta O_t$
 7: end for
 8: return $\theta_t$
```

Apply the function left 5 times get right



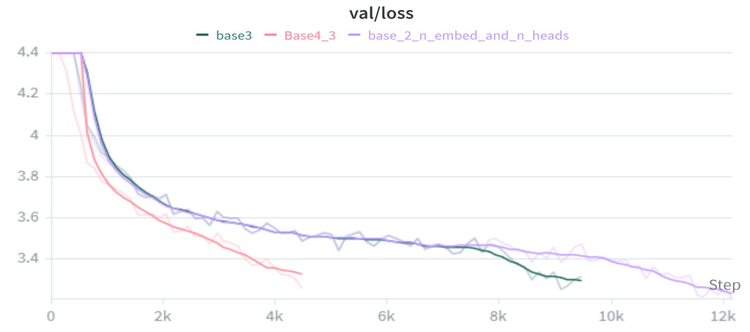
```
Algorithm 2 Muon
Require: Learning rate $\eta$, momentum $\mu$
 1: Initialize $B_0 \leftarrow 0$
 2: for $t = 1, \ldots$ do
 3:     Compute gradient $G_t \leftarrow \nabla_\theta \mathcal{L}_t(\theta_{t-1})$
 4:     $B_t \leftarrow \mu B_{t-1} + G_t$
 5:     $O_t \leftarrow$ SVD (Bt)
 6:     Update parameters $\theta_t \leftarrow \theta_{t-1} - \eta O_t$
 7: end for
 8: return $\theta_t$
```

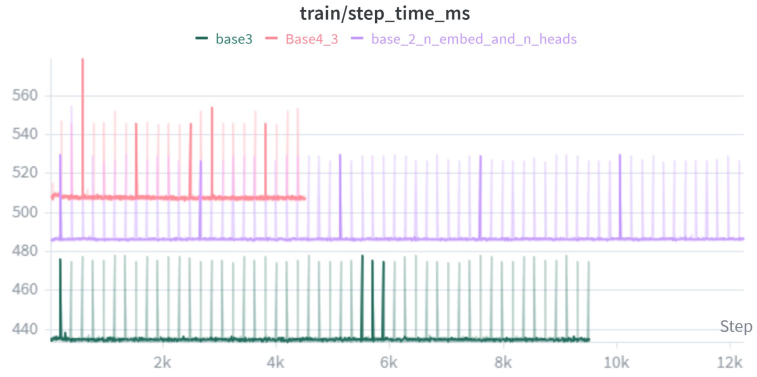
Run 4:

Slower per step but faster convergence compared to adamW as stated in the paper.

New zero-initialised embedding layer for predicting tokens instead 1 layer doing prediction and representation.

QK layer norm stabilizes training and improves performance, as expected based on paper.

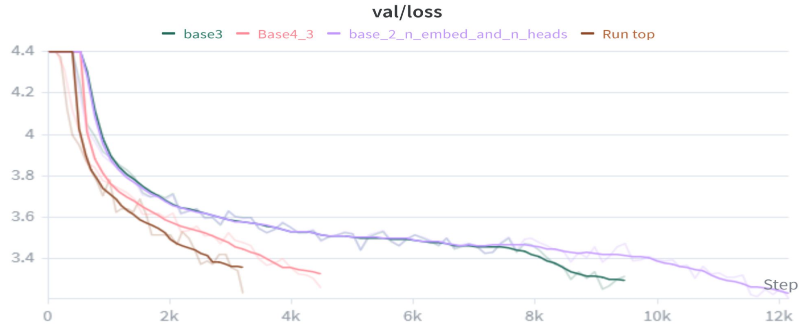


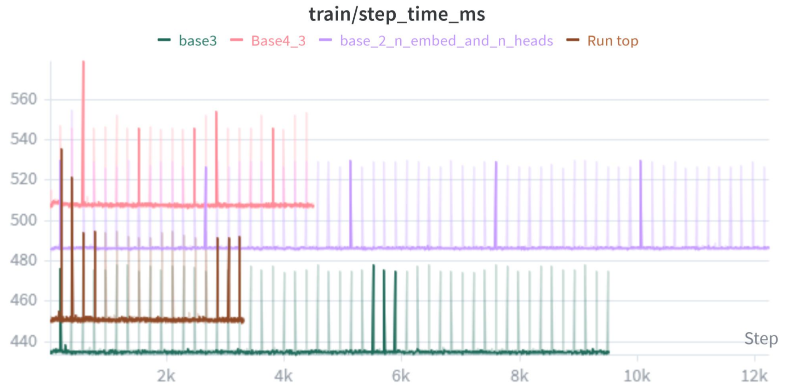


Run 5: Fixes

Fastest run 27 min

Hyper param tuning





Conclusions