Google Cloud demonstrates the world’s largest distributed training job for large language models across 50000+ TPU v5e chips

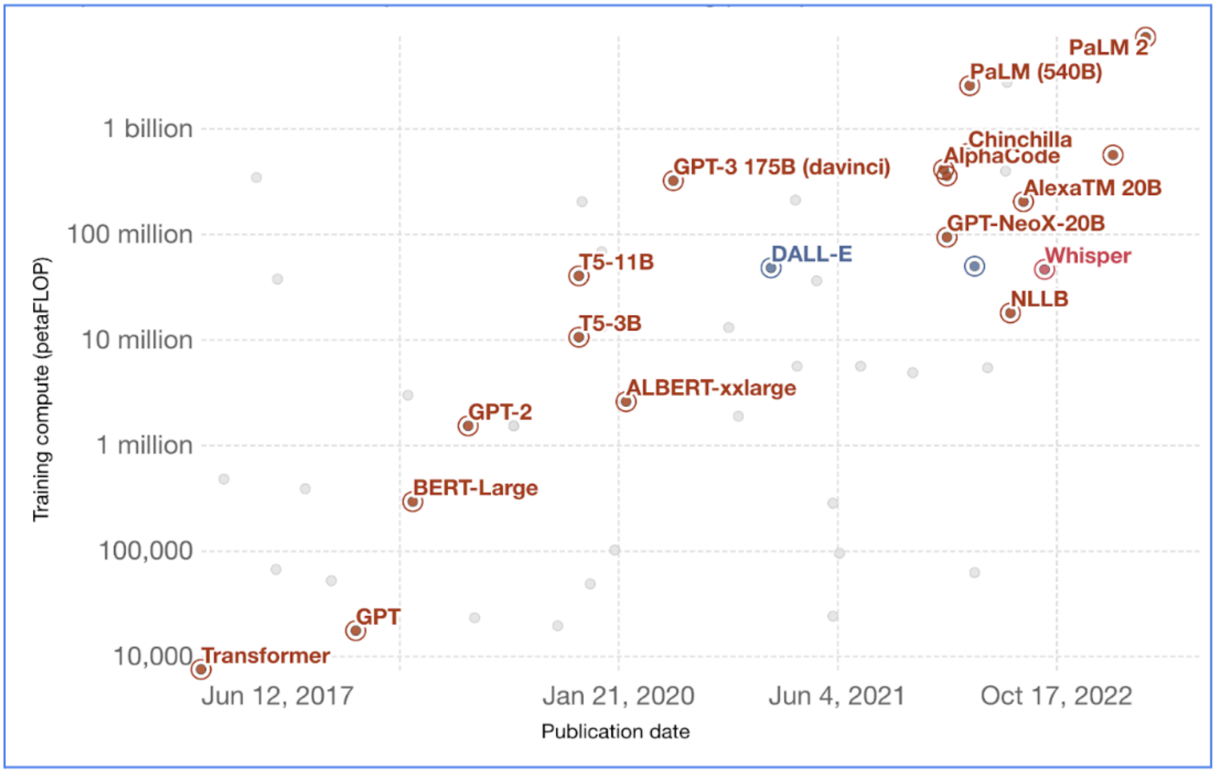
November 9, 2023

Rajesh Anantharaman

Product Management Lead, Cloud TPU

With the boom in generative AI, the size of foundational large language models (LLMs) has grown exponentially, utilizing hundreds of billions of parameters and trillions of training tokens.

LLM model size growth



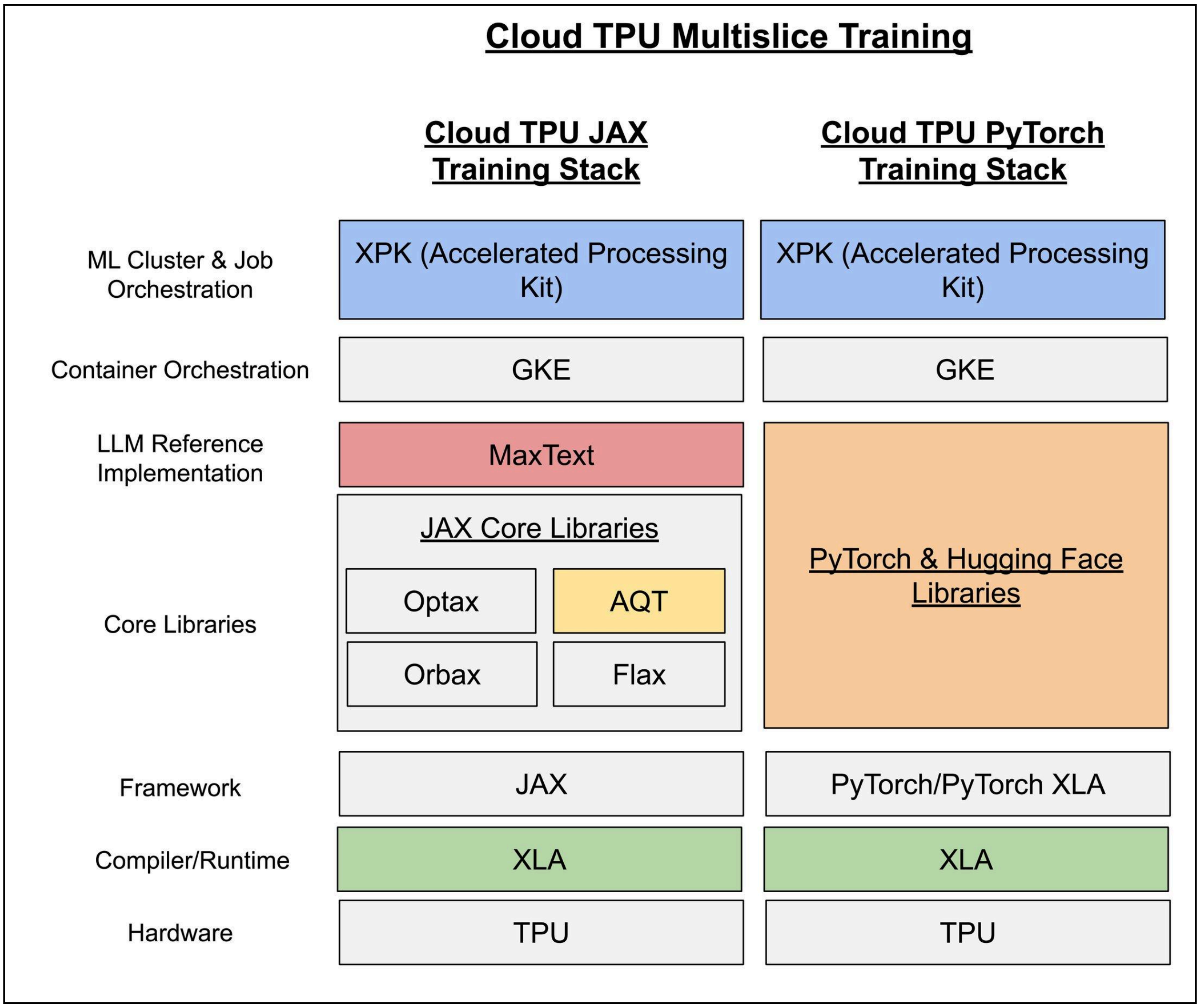
Source: "Computation used to train notable intelligence systems", One World Data

Training these kinds of large LLMs require tens of exa-FLOPs (10^18 FLOPs) of AI supercomputing power, which is typically distributed across large clusters that contain tens of thousands of AI accelerator chips. But utilizing large-scale clusters for distributed machine learning (ML) training presents many common and key technical challenges.

1. **Orchestration:** The software stack that is used for distributed training needs to manage all of these chips and scale as high as possible to accelerate training times. This stack also needs to be reliable, fault tolerant, and resilient in order to ensure training progress.
2. **Compilation**: As training progresses, the computation and communication that happen across the chips need to be managed effectively by a high-performance compiler.
3. **End-to-end optimization**: Distributed training at large scale requires deep expertise throughout both the ML training stack and the end-to-end ML training workflow, from storage and compute to memory and networking.

Google Cloud TPU Multislice Training

To address each of the above distributed training challenges across orchestration, compilation, and end-to-end optimization, today we announced the general availability of Cloud TPU Multislice Training. This full-stack training offering — supporting TPU v4 and v5e — is built from the ground up to be scalable, reliable, and easy-to-use for end-to-end optimization of ML training. With Multislice, you can leverage Google's cost-efficient, versatile, and scalable Cloud TPUs for training ML models efficiently and at large scale.



Cloud TPU Multislice Training

	Cloud TPU JAX Training Stack	Cloud TPU PyTorch Training Stack
ML Cluster & Job Orchestration	XPK (Accelerated Processing Kit)	XPK (Accelerated Processing Kit)
Container Orchestration	GKE	GKE
LLM Reference Implementation	MaxText	PyTorch & Hugging Face Libraries
Core Libraries	JAX Core Libraries Optax AQT Orbax Flax	
Framework	JAX	PyTorch/PyTorch XLA
Compiler/Runtime	XLA	XLA
Hardware	TPU	TPU

Cloud TPU Multislice Training has the following key features:

1. **Robust orchestration and scalability:** Scale large model training across tens of thousands of TPU chips in a reliable and fault-tolerant way across the training workflow.
2. **Performant compilation:**Maximize performance and efficiency using the XLA compiler to automatically manage compute and communication.
3. **Flexible stack for end-to-end training:** Provide first-class support for popular ML frameworks such as JAX and PyTorch, easy-to-use reference implementations and libraries, and support for a wide range of model architectures including LLMs, diffusion models and DLRM.

Here we highlight some of the key components in the overall Multislice Training stack, many of which we have open-sourced in order to continue contributing to the broader AI/ML community:

1. Accelerated Processing Kit (XPK) is an ML cluster and job orchestration tool built to be used with Google Kubernetes Engine (GKE) to standardize the best practices for orchestrating ML jobs. XPK focuses on ML semantics for creating, managing, and running ML training jobs to make it easier for machine learning engineers (MLE) to use, manage, and debug. XPK decouples provisioning capacity from running jobs through separate APIs.
2. MaxText is a performant, scalable, and adaptable JAX LLM implementation. This implementation is built on top of open-source JAX libraries such as Flax, Orbax, and Optax. MaxText is a decoder-only LLM implementation written in pure Python, making it much easier for MLEs to understand, adapt, and modify. MaxText also leverages the XLA compiler heavily, making it easy for MLEs to achieve high performance without needing to build custom kernels. XLA through OpenXLA is an open-source ML compiler for a variety of hardware accelerators, such as TPUs, GPUs, CPUs, and others.

3. Accurate Quantized Training (AQT) is a Google-built training library that uses reduced numerical precision of 8-bit integers (INT8) instead of 16-bit floats (BF16) for training. AQT takes advantage of the fact that ML accelerators have 2X the compute speed when using INT8 operations versus BF16 operations. Using AQT's simple and flexible API, MLEs can attain both higher performance during training and also higher model quality in production.

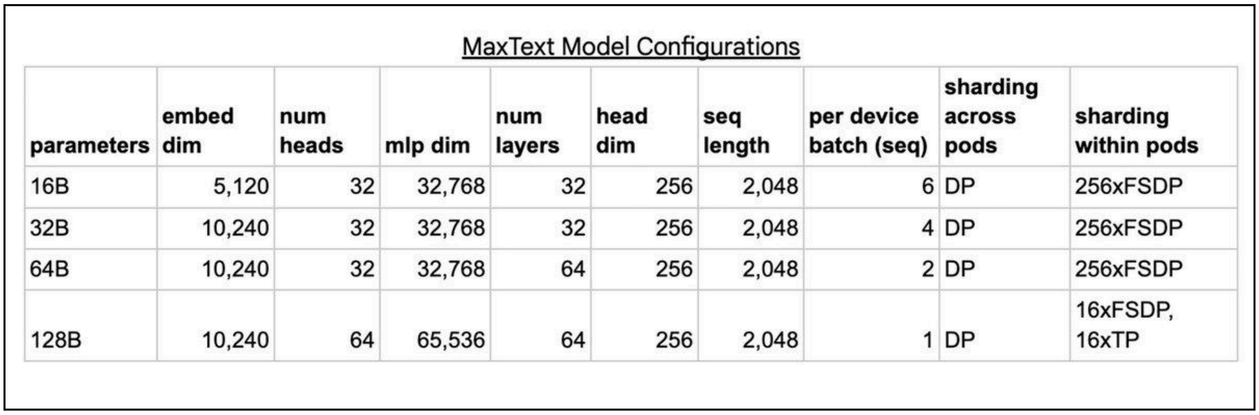
Google Cloud TPU ran the world's largest distributed training job for LLMs across 50,000+ TPU v5e chips

In November 2023, we used Multislice Training to run what we believe to be the world's largest publicly disclosed LLM distributed training job (in terms of the number of chips used for training) on a compute cluster of 50,944 Cloud TPU v5e chips (spanning 199 Cloud TPU v5e pods) that is capable of achieving 10 exa-FLOPs (16-bit), or 20 exa-OPs (8-bit), of total peak performance. To give a sense of scale, this cluster of Cloud TPU v5e chips has more AI accelerators than the TOP1 Supercomputer Frontier at Oak Ridge National Laboratory, which featured 37,888 AMD M1250X GPUs.

Setting up the LLM distributed training job on Cloud TPU v5e

We performed our large-scale LLM distributed training job using Cloud TPU Multislice Training on Cloud TPU v5e. A Cloud TPU v5e pod consists of 256 chips connected via high-speed inter-chip interconnect (ICI). These pods are connected and communicate using Google’s Jupiter data center networking (DCN). We set up this distributed training job on the JAX framework, utilizing XPK, GKE, MaxText, AQT, and other components of the JAX training stack. The rest of this blog focuses on the JAX training stack portion of Cloud TPU Multislice Training.

We trained multiple MaxText models in various sizes of 16B, 32B, 64B, and 128B parameters. For each model, we scaled the training using data parallelism (DP) across pods over DCN, where each pod stores its own replica of the model. Then, each replica of the model is sharded across chips within a pod over ICI using fully sharded data parallelism (FSDP) for the 16B, 32B, and 64B configurations, and a mix of FSDP and tensor parallelism (TP) for the 128B configuration.



MaxText Model Configurations

parameters	embed dim	num heads	mlp dim	num layers	head dim	seq length	per device batch (seq)	sharding across pods	sharding within pods
16B	5,120	32	32,768	32	256	2,048	6	DP	256xFSDP
32B	10,240	32	32,768	32	256	2,048	4	DP	256xFSDP
64B	10,240	32	32,768	64	256	2,048	2	DP	256xFSDP
128B	10,240	64	65,536	64	256	2,048	1	DP	16xFSDP, 16xTP

We managed TPU capacity with Google Kubernetes Engine (GKE) and utilized XPK on top of GKE to orchestrate ML jobs. XPK handles creating clusters, resizes them as necessary, submits jobs into the GKE Kueue system as JobSets, manages those Jobsets, and provides visibility into the state of the cluster.

To accelerate model training, we used the Accurate Quantized Training (AQT) library to train in quantized INT8. As of October 2023, this approach enables a 1.2X to 1.4X acceleration in steps per second while producing a convergence gap smaller than that normally associated with quantizing a model trained in BF16 to INT8.

How we scaled the largest distributed LLM training job

As we scaled our TPU compute cluster, we began to push the limits of the stack.

Orchestration

Managing over 50,000 accelerator chips working on a single training job requires a well-designed orchestration solution that enables both different users to submit smaller jobs for experiments as well as supporting a full-scale job that runs on the entire cluster. This functionality is provided through GKE's Jobset and Kueue features. As we pushed the limits on the number of VMs that GKE could handle, we optimized managing internal IP addresses, precaching docker images, designed clusters for scale, and enabled high-throughput scheduling. We also optimized GKE to push VM scaling limits in areas such as pod IP exhaustion, Domain Name Service (DNS) scalability, and control-plane node limits. We packaged and documented these solutions alongside XPK to make this a repeatable process for customers training at this massive scale.

Performance

JAX is powered by XLA (Accelerated Linear Algebra), a compiler-based linear algebra execution engine that optimizes workloads for ML accelerators like TPUs and GPUs to deliver supercomputer-like performance. The key parallelism technique behind XLA is SPMD (single program, multiple data) where the same computation is run in parallel on different devices. XLA leverages GSPMD which simplifies SPMD programming by allowing the user to program a single giant supercomputer and then automatically parallelizing the computation across devices based on a few user annotations. Running at large scale exposed the need for optimizations that only become necessary with a large number of slices. For example, each worker VM needs to communicate over DCN with the worker VMs of the same rank in other slices. Originally, this caused slowdowns due to excess device-to-host and host-to-device transfers that scaled linearly with the number of slices. By optimizing the XLA runtime, we were able to prevent these transfers from being the bottleneck.

Storage

Interacting with persistent storage is a crucial aspect of training. Our 199-pod cluster had 1 Tb/s to Cloud Storage, 1,270 Tb/s inter-slice DCN, and 73,400 Tb/s intra-slice ICI. When loading Docker images, loading data, and reading/writing checkpoints, we optimized the interaction with persistent storage.

We found that at large scale, data loading from Cloud Storage began to affect performance, starting at 64 pods scale. We have since mitigated this limit with a distributed data loading strategy that alleviates pressure on Cloud Storage by having a subset of hosts load data.

We also found limits due to checkpointing. By default, checkpointing loads the full checkpoint into each data parallel replica from Cloud Storage. Consider checkpoint loading for a 128B model sharded with cross-pod data parallelism. For a traditional optimizer state of three numbers per parameter (4bytes/number), this means loading a checkpoint of size ~1.536 TB separately into each pod (in this case for 199 pods). This would require 199 pods * 1.536TB/pod, or approximately 300TB of bandwidth. For reasonable performance from persistent storage of 1 Tb/s, this would require approximately 2,400 seconds (40 minutes). However, we needed much lower start or restart time, so had to take a different approach.

To alleviate the problem, we added features that enabled a single pod to load the checkpoint and broadcast it to the other replicas. As a result, a single pod can read the checkpoint and then broadcast the optimizer state to the other pods by leveraging the flexibility of JAX. In principle, this should take 1.536TB/1Tb/s = ~12 seconds to load the checkpoint and then (2*1.536TB/pod) / (64 VM/per pod * 100 Gb/s/VM) = ~4 seconds to gather the optimizer state across the cluster, for a total of 16 seconds and a 150x speedup. Similarly, optimizations are needed when writing checkpoint data and loading training data. At write-time, a single leader replica can then write the entire checkpoint to avoid excessive QPS to Cloud Storage.

How did we measure training performance?

Training performance is measured in terms of Model FLOPs Utilization (MFU) and Effective Model FLOPs Utilization (EMFU). For an N-parameter decoder-only model, each token seen requires 6N matmul FLOPs for the learnable weights and 12LHQT matmul FLOPs for attention where L, H, Q, and T are the number of layers, the number of heads, the head dimension, and the sequence length respectively (see Appendix B of the PaLM paper for more details). Knowing the TFLOPs required for each token, we can represent the throughput of a step as observed TFLOP/chip/s which is computed as the total TFLOPs required for all tokens seen in the step for each chip divided by the step time.

$$observed\ TFLOP/chip/s = \frac{learnable\_weight\_tflops + attention\_tflops}{step\_time} = \frac{(6N \times tokens\_per\_chip\_per\_step)/(10^{12}) + (12LHQT \times tokens\_per\_chip\_per\_step)/(10^{12})}{step\_time}$$

We can then compute MFU by dividing the observed TFLOP/chip/s by the peak TFLOP/chip/s of the hardware (197 TFLOP/chip/s for TPU v5e).

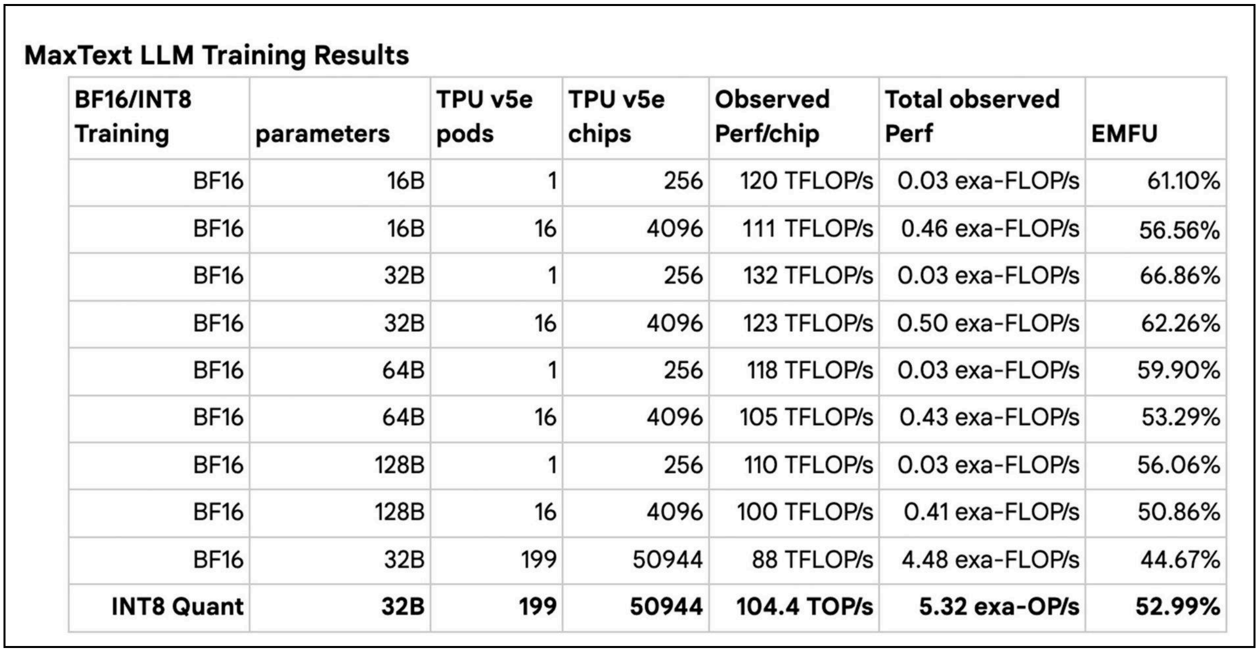
$$MFU = \frac{observed\ TFLOP/chip/s}{peak\ hardware\ TFLOP/chip/s}$$

EMFU broadens observed TFLOP/chip/s to observed TOP/chip/s (tera-operations/chip/s), which encapsulates both quantized operations and floating point operations. However, since the observed TOP/chip/s for quantized operations can be larger than the peak TFLOP/chip/s for floating point operations, it is possible to achieve greater than 100% EMFU.

$$EMFU = \frac{observed\,TOP/chip/s}{peak\ hardware\ TFLOP/chip/s}$$

Largest LLM distributed training job scalability results

For each model size (16B, 32B, 64B, 128B), we ran a sweep of training jobs, scaling the number of TPU v5e pods from 1 to 160. We saw as high as 66.86% MFU with BF16 training on a single TPU v5e pod and strong scaling outcomes when expanding to 160 pods. We also ran jobs exercising the entire 199-pod cluster with both BF16 training and INT8 quantized training (using AQT), achieving an impressive observed 5.32 exa-OP/s with INT8 quantized training. This scaling study was done with limited software optimizations across the Multislice Training JAX stack, and we will continue improving our software stack.



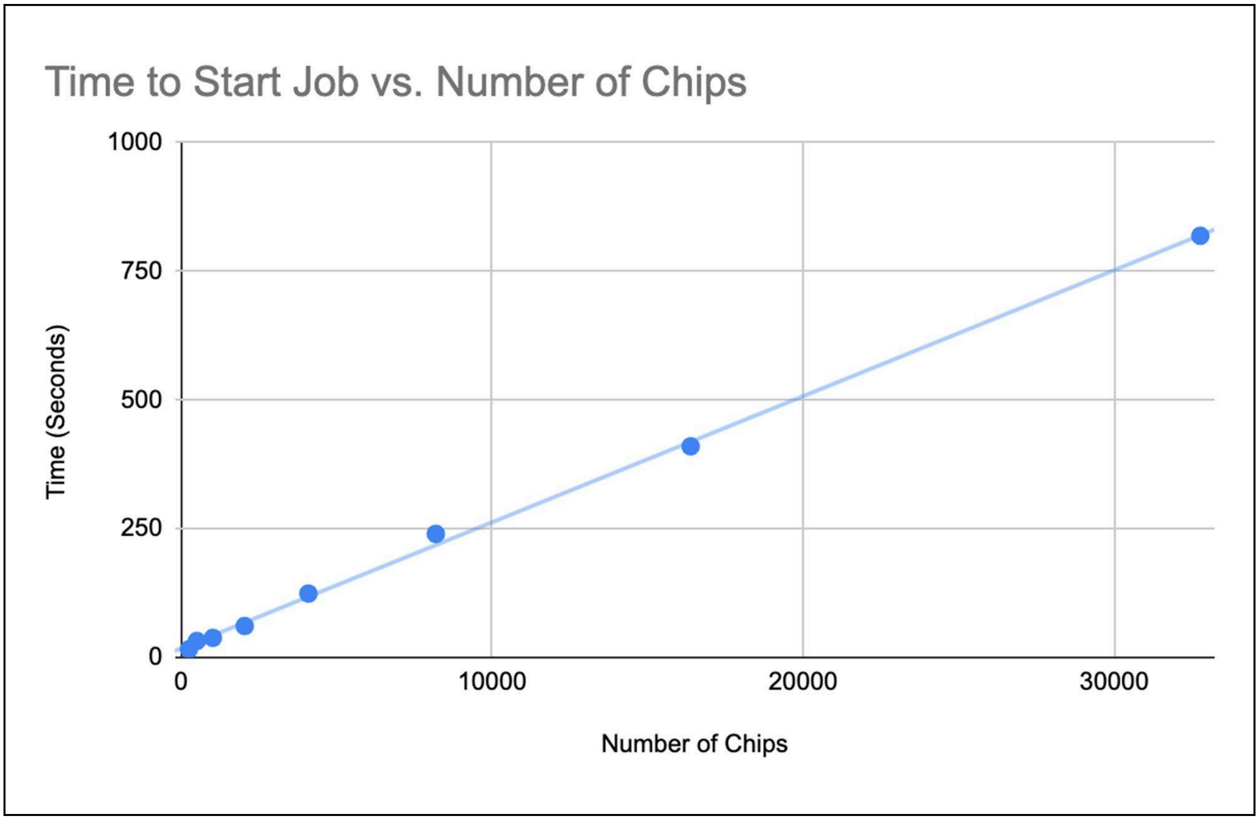
MaxText LLM Training Results

BF16/INT8 Training	parameters	TPU v5e pods	TPU v5e chips	Observed Perf/chip	Total observed Perf	EMFU
BF16	16B	1	256	120 TFLOP/s	0.03 exa-FLOP/s	61.10%
BF16	16B	16	4096	111 TFLOP/s	0.46 exa-FLOP/s	56.56%
BF16	32B	1	256	132 TFLOP/s	0.03 exa-FLOP/s	66.86%
BF16	32B	16	4096	123 TFLOP/s	0.50 exa-FLOP/s	62.26%
BF16	64B	1	256	118 TFLOP/s	0.03 exa-FLOP/s	59.90%
BF16	64B	16	4096	105 TFLOP/s	0.43 exa-FLOP/s	53.29%
BF16	128B	1	256	110 TFLOP/s	0.03 exa-FLOP/s	56.06%
BF16	128B	16	4096	100 TFLOP/s	0.41 exa-FLOP/s	50.86%
BF16	32B	199	50944	88 TFLOP/s	4.48 exa-FLOP/s	44.67%
INT8 Quant	**32B**	**199**	**50944**	**104.4 TOP/s**	**5.32 exa-OP/s**	**52.99%**

Future work

Job start-up time

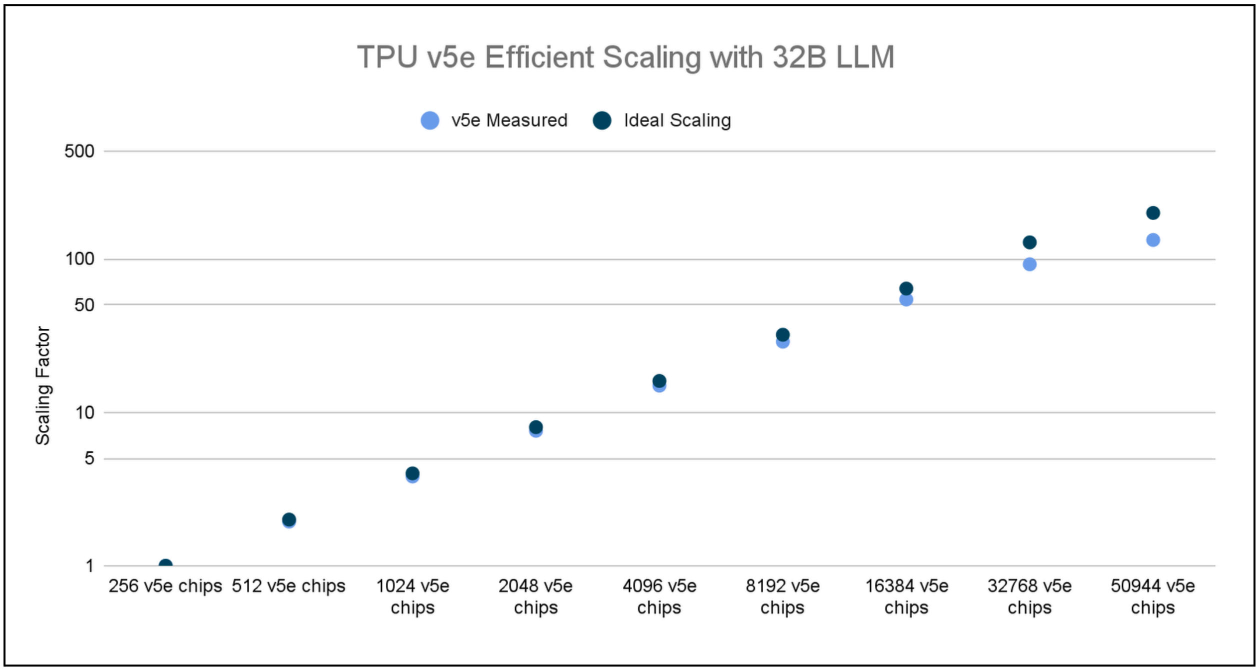
Along with measuring training performance, we also measured the start-up time of our ML jobs on the cluster, which scaled near-linearly with the number of chips.



The start times we observed were impressive, but we believe we can improve these even further. We are working on areas such as optimizing scheduling in GKE to increase throughput and enabling ahead-of-time compilation in MaxText to avoid just-in-time compilations on the full cluster.

Scaling efficiency

We achieved excellent scaling across 50,944 TPU v5e chips, but we believe we can improve this even further. We've identified and are currently working on changes in the compiler and MaxText to improve stability and performance at large scale. We are considering scalable solutions such as hierarchical DCN collectives and further optimizing compiler scheduling in multipod regimes.



*Google Internal data for TPU v5e As of November, 2023: All numbers normalized per chip. seq-len=2048 for 32 billion parameter decoder only language model implemented using MaxText. *2*

Conclusion

Google Cloud TPU Multislice Training was built from the ground up to address the challenges of distributed ML training in orchestration, compilation, and end-to-end optimization. We demonstrated the benefits of Cloud TPU Multislice Training with what we believe is the largest (as of November 2023) publicly disclosed LLM distributed training job in the world (in terms of number of chips used for training) on a compute cluster of 50,944 Cloud TPU v5e chips on the JAX ML framework, utilizing both BF16 and INT8 quantized training.

As generative AI continues to trend towards bigger and bigger LLMs, we will continue to push the boundaries of innovation needed to further scale and improve our software stack. We have open-sourced all the code used in this project. Please check out our open-source repositories for MaxText, XPK, AQT and XLA. To learn more about Google

Cloud TPU Multislice Training and how to use it with Cloud TPUs to accelerate your generative AI projects, please contact your Google Cloud account representative.

This work constituted a massive effort across Google Cloud as well as multiple teams within Google. Special thanks to Raymond Zou, Rafi Witten, Lukasz Lew, Victor Barr and Andi Gavrilescu for their immense contributions to developing all of the components that made this project a success.