Tensor Processing Units (TPUs) are Google's custom-developed application-specific integrated circuits (ASICs) used to accelerate machine learning workloads. TPUs are designed from the ground up with the benefit of Google's deep experience and leadership in machine learning.

Cloud TPU enables you to run your machine learning workloads on Google's TPU accelerator hardware (/tpu/) using TensorFlow (https://www.tensorflow.org/). Cloud TPU is designed for maximum performance and flexibility to help researchers, developers, and businesses to build TensorFlow compute clusters that can leverage CPUs, GPUs, and TPUs. High-level TensorFlow APIs help you to get models running on the Cloud TPU hardware.

Cloud TPU resources accelerate the performance of linear algebra computation, which is used heavily in machine learning applications. TPUs minimize the time-to-accuracy when you train large, complex neural network models. Models that previously took weeks to train on other hardware platforms can converge in hours on TPUs.

Cloud TPUs are available in the following zones:

Cloud TPUs are very fast at performing dense vector and matrix computations. Transferring data between Cloud TPU and host memory is slow compared to the speed of computation—the speed of the PCIe bus is *much* slower than both the Cloud TPU interconnect and the on-chip high bandwidth memory (HBM). This means that partial compilation of a model, where execution 'ping-pongs' between host and device, uses the device in a very inefficient way, as it would be idle most of the time, waiting for data to arrive over the PCIe bus. To alleviate this situation, the programming model for Cloud TPU is designed to execute much of the training on the TPU—ideally the entire training loop.

Following are some salient features of the programming model implemented by TPUEstimator (https://www.tensorflow.org/api_docs/python/tf/estimator/tpu/TPUEstimator):

- All model parameters are kept in on-chip high bandwidth memory.
- The cost of launching computations on Cloud TPU is amortized by executing many training steps in a loop.
- Input training data is streamed to an "infeed" queue on the Cloud TPU. A program running on Cloud TPU retrieves batches from these queues during each training step.
- The TensorFlow server running on the host machine (the CPU attached to the Cloud TPU device) fetches data and pre-processes it before "infeeding" to the Cloud TPU hardware.

- Data parallelism: Cores on a Cloud TPU execute an identical program residing in their own respective HBM in a synchronous manner. A reduction operation is performed at the end of each neural network step across all the cores.

Cloud TPUs are optimized for specific workloads. In some situations, you might want to use GPUs (/compute/docs/gpus/) or CPUs on Compute Engine instances to run your machine learning workloads. In general, you can decide what hardware is best for your workload based on the following guidelines:

- CPUs
  - Quick prototyping that requires maximum flexibility
  - Simple models that do not take long to train
  - Small models with small effective batch sizes
  - Models that are dominated by custom TensorFlow operations written in C++ (https://www.tensorflow.org/extend/adding_an_op)
  - Models that are limited by available I/O or the networking bandwidth of the host system
- GPUs
  - Models that are not written in TensorFlow or cannot be written in TensorFlow
  - Models for which source does not exist or is too onerous to change
  - Models with a significant number of custom TensorFlow operations that must run at least partially on CPUs
  - Models with TensorFlow ops that are not available on Cloud TPU (see the list of available TensorFlow ops (/tpu/docs/tensorflow-ops))
  - Medium-to-large models with larger effective batch sizes
- TPUs
  - Models dominated by matrix computations

- Models with no custom TensorFlow operations inside the main training loop
- Models that train for weeks or months
- Larger and very large models with very large effective batch sizes

Cloud TPUs are **not** suited to the following workloads:

- Linear algebra programs that require frequent branching or are dominated element-wise by algebra. TPUs are optimized to perform fast, bulky matrix multiplication, so a workload that is not dominated by matrix multiplication is unlikely to perform well on TPUs compared to other platforms.
- Workloads that access memory in a sparse manner might not be available on TPUs.
- Workloads that require high-precision arithmetic. For example, double-precision arithmetic is not suitable for TPUs.
- Neural network workloads that contain custom TensorFlow operations written in C++. Specifically, custom operations in the body of the main training loop are not suitable for TPUs.

Neural network workloads must be able run multiple iterations of the entire training loop on the TPU. Although this is not a fundamental requirement of TPUs themselves, this is one of the current constraints of the TPU software ecosystem that is required for efficiency.

A typical TensorFlow training graph consists of multiple overlapping subgraphs which provide a variety of functionality including:

- I/O operations to read training data.
- Input preprocessing stages, often connected via queues.
- Model variables.
- Initialization code for those variables.
- The model itself.
- Loss functions.

- Gradient code (usually automatically generated).
- Summary ops for monitoring training.
- Save/Restore ops for checkpointing.

On Cloud TPU, TensorFlow programs are compiled by the XLA (https://www.tensorflow.org/performance/xla/) just-in-time compiler. When training on Cloud TPU, the *only* code that can be compiled and executed on the hardware is that corresponding to the dense parts of the model, loss and gradient subgraphs. All other parts of the TensorFlow program run on the host machines (Cloud TPU server) as part of a regular distributed TensorFlow session. This typically consists of the I/O operations which read training data, any preprocessing code (for example: decoding compressed images, randomly sampling/cropping, assembling training minibatches) and all of the housekeeping parts of the graph such as checkpoint save/restore.

A single Cloud TPU chip contains 2 cores, each of which contains multiple matrix units (MXUs) designed to accelerate programs dominated by dense matrix multiplications and convolutions (see Hardware Architecture (/tpu/docs/system-architecture#hardware_architecture)). Programs that spend a considerable fraction of their execution time performing matrix multiplications are typically well suited to Cloud TPU. A program whose computation is dominated by non-matrix operations such as add, reshape, or concatenate, will likely not achieve high MXU utilization. Following are some guidelines to help you choose and build models that are suitable for Cloud TPU.

A single Cloud TPU device consists of four chips, each of which has two TPU cores. Therefore, for efficient utilization of Cloud TPU, a program should make use of each of the eight cores. TPUEstimator (https://www.tensorflow.org/api_docs/python/tf/estimator/tpu/TPUEstimator) provides a graph operator to build and run a replicated computation (https://www.tensorflow.org/api_docs/python/tf/contrib/tpu/replicate). Each replica is essentially a copy of the training graph that is run on each core and trains a mini-batch containing 1/8th of the overall batch size.

The XLA compiler performs code transformations, including tiling a matrix multiply into smaller blocks, to efficiently execute computations on the matrix unit (MXU). The structure of the MXU hardware, a 128x128 systolic array (https://en.wikipedia.org/wiki/Systolic_array), and the design of TPU's memory subsystem, which prefers dimensions that are multiples of 8, are used by the XLA compiler for tiling efficiency. Consequently, certain layouts are more conducive to tiling, while others require *reshapes* to be performed before they can be tiled. Reshape operations are often memory bound on the Cloud TPU.

The XLA compiler compiles a TensorFlow graph just in time for the first batch. If any subsequent batches have different shapes, the model doesn't work. (Re-compiling the graph every time the shape changes is too slow.) Therefore, any model that has tensors with dynamic shapes that change at runtime isn't well suited to TPUs.

A high performing Cloud TPU program is one where the dense compute can be easily tiled into 128x128 chunks. When a matrix computation cannot occupy an entire MXU, the compiler pads tensors with zeroes. There are two drawbacks to padding:

- Tensors padded with zeroes under-utilize the TPU core.
- Padding increases the amount of on-chip memory storage required for a tensor and can lead to an out-of-memory error in the extreme case.

While padding is automatically performed by the XLA compiler when necessary, one can determine the amount of padding performed by means of the op_profile (/tpu/docs/cloud-tpu-tools#interpreting_the_results_1) tool. You can avoid padding by picking tensor dimensions that are well suited to TPUs.

Choosing suitable tensor dimensions goes a long way in extracting maximum performance from the TPU hardware, particularly the MXU. The XLA compiler attempts to use either the batch size or the feature dimension to maximally utilize the MXU. Therefore, one of these must

be a multiple of 128. Otherwise, the compiler will pad one of them to 128. Ideally, batch size as well as feature dimensions should be multiples of 8, which enables extracting high performance from the memory subsystem.

See the list of available TensorFlow ops (/tpu/docs/tensorflow-ops).

Artificial intelligence (AI) models trained in the cloud increasingly need to be run "at the edge". For the purposes of this document, "at the edge" means devices running on the edge of the Internet of Things (IoT). These include a wide variety of sensors and smart devices that pick up and communicate real-time data to other devices and to the cloud.

Edge TPU augments Cloud TPU and Cloud IoT to provide an end-to-end (cloud-to-edge, hardware + software) infrastructure to facilitate deploying AI-based solutions.

Edge TPU is currently in early access. For information on this program, see the early access page (/edge-tpu/).

- Read the TPU quickstart (/tpu/docs/quickstart) to get started using Cloud TPU resources.
- Complete one of the Cloud TPU tutorials (/tpu/docs/tutorials) to learn how to run common machine learning workloads on Cloud TPU resources.