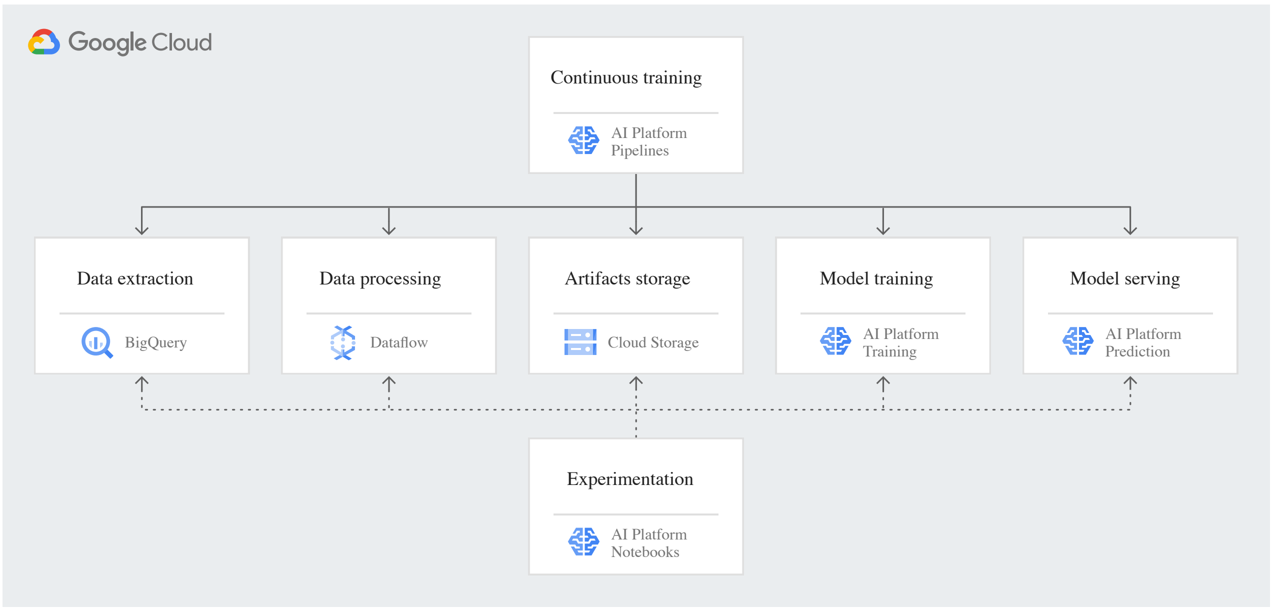
Best practices for performance and cost optimization for machine learning

This guide collates some best practices for how you can enhance the performance and decrease the costs of your machine learning (ML) workloads on Google Cloud, from experimentation to production.

The following diagram shows a typical view of an ML environment for experimentation and operationalization.



The environment uses various smart analytics (/solutions/smart-analytics) and Cloud AI (/solutions/ai) services in different phases of the ML process, namely the following:

- Experimentation with AI Platform Notebooks (/ai-platform-notebooks).
- Data preparation with BigQuery (/bigquery) and Dataflow (/dataflow).
- Training with AI Platform Training (/ai-platform/training/docs/overview).
- Serving with AI Platform Prediction (/ai-platform/prediction/docs).
- Orchestration with AI Platform Pipelines (/ai-platform/pipelines/docs).
- Cost-management best practices for ML projects on Google Cloud across all the services that you use.

In this example, Cloud Storage (/storage) is used by these services to store and retrieve ML artifacts.

Although this guide focuses on the environment outlined in the diagram, BigQuery ML (/bigquery-ml/docs/bigqueryml-intro) can be more performant and cost-effective than using AI Platform if your training data is in BigQuery and you're using models for batch predictions. With BigQuery ML, you can create and execute ML models using standard SQL queries, without managing any infrastructure.

You can use BigQuery ML to train regression models using linear regression, classification models using logistic regression, tree models using XGBoost (https://xgboost.readthedocs.io/en/latest/), recommendation models using matrix factorization, time series using ARIMA, and clustering using k-means. You can also use the BigQuery `CREATE MODEL` (/bigquery-ml/docs/reference/standard-sql/bigqueryml-syntax-create-tensorflow) statement to import models from TensorFlow. Furthermore, BigQuery ML provides a set of `TRANSFORM` (/bigquery-ml/docs/bigqueryml-transform) functions for feature engineering.

Experimentation with AI Platform Notebooks

AI Platform Notebooks (/ai-platform-notebooks) lets you manage JupyterLab instances through a protected, publicly available URL. You can create an instance using one of the existing Deep Learning VM (/deep-learning-vm) instances that has the latest ML and data science libraries preinstalled, along with the latest accelerator drivers. Alternatively, you can create an AI Platform Notebooks instance based on a custom container (/ai-platform/notebooks/docs/custom-container) to customize the Notebooks environment for your needs. AI Platform Notebooks uses TensorFlow Enterprise (/tensorflow-enterprise), an optimized distribution that contains custom-built TensorFlow binaries and related packages.

pache Spark (https://spark.apache.org/) is commonly used by companies that want to explore large am
a and perform ML-related tasks at scale. Dataproc Hub (/dataproc/docs/tutorials/dataproc-hub-users) bri
eractivity of AI Platform Notebooks to Spark Dataproc (/dataproc) clusters in a secure and centrally mana
or more information, see Combining the power of Apache Spark and AI Platform Notebooks with Datapro
/products/data-analytics/administering-jupyter-notebooks-for-spark-workloads-on-dataproc).

Start with a sample of your data

To iterate quickly at low cost, start with a sample of your data on a small AI Platform Notebooks instance. This lets you validate assumptions, confirm your hypotheses, and identify your modeling approach. After this phase, you can scale up to train your model with the full dataset, using more powerful compute instances and accelerators.

Select the machine type with the right balance between cost and performance

You can choose any machine type (/compute/docs/machine-types) for your AI Platform Notebooks instance. Machine types belong to different families, each curated for specific workloads, from general purpose compute (E2, N1, N2, and N2D), to memory-optimized (M1 and M2), to compute-optimized (C2). Typical ML training workloads fit N1 machine types, where you can attach many types of GPUs. The A2 VM family (https://cloud.google.com/blog/products/compute/announcing-google-cloud-a2-vm-family-based-on-nvidia-a100-gpu) is based on the NVIDIA Ampere A100 Tensor Core GPU (https://www.nvidia.com/en-us/data-center/a100/) and optimized for large ML and high performance computing (HPC) workloads. The A2 VMs also support NVIDIA MIG (https://www.nvidia.com/en-us/technologies/multi-instance-gpu/), which lets you run multiple isolated workloads on the same hardware.

Create derivative containers from the standard Deep Learning Container images

AI Platform Deep Learning Containers images (/ai-platform/deep-learning-containers/docs/choosing-container) provide optimized data science environments for the selected framework (such as PyTorch or TensorFlow). The images also support the latest NVIDIA® CUDA-X AI libraries and drivers for GPU images (CUDA, cuDNN, NCCL2), and the Intel® Math Kernel Library. For more information about how to create custom data science environments, see derivative containers (/ai-platform/deep-learning-containers/docs/derivative-container).

Leverage accelerators effectively

You can add and remove up to 8 NVIDIA® Tesla® GPUs (/compute/docs/gpus) in your AI Platform Notebooks instances in order to accelerate your ML workloads. While P100 (https://www.nvidia.com/en-us/data-center/tesla-p100/) and V100 (https://www.nvidia.com/en-us/data-center/v100/) are powerful accelerators, using K80 (https://www.nvidia.com/en-gb/data-center/tesla-k80/) accelerators can dramatically lower costs—especially if your AI Platform Notebooks instance is running for a long time but not utilizing the GPU for most of that time.

Improve the performance of multi-GPU workloads

Use higher network bandwidths (/compute/docs/gpus/optimize-gpus#high-bandwidth) to train a large model on a large dataset. The V100 GPUs are offered with high-speed NVLink™ (https://www.nvidia.com/en-us/design-visualization/nvlink-bridges/) connections for communication between GPUs. For more information, see Optimizing GPU performance (/compute/docs/gpus/optimize-gpus).

Treat your AI Platform Notebooks as ephemeral instances

Don't treat your instances as long-living ones. Unless you're running experiments, the instance should be switched off or deleted. If you switch off your instances, you pay only for disk storage. If you delete your instances, make sure that you store your data in Cloud Storage (/storage) or BigQuery, rather than locally in the instance, while maintaining your Notebooks and code in a source control (/ai-platform/notebooks/docs/save-to-github) system.

Avoid paying for unutilized resources

Make sure that you don't use resources that you don't need while you run ephemeral AI Platform Notebooks instances. You can configure an automatic shutdown routine (https://blog.kovalevskyi.com/aiplatform-notebooks-and-vms-auto-shutdown-on-idle-dd94ed3d4724) when your instance is idle, or you can use the idle VM recommender (/recommender) to identify inactive VMs and persistent disks, based on usage metrics. Because testing and development tend to be confined to business hours, you can also automate and manage VM shutdown at scale by scheduling Compute Engine instances (/blog/products/storage-data-transfer/save-money-by-stopping-and-starting-compute-engine-instances-on-schedule)
.

Consider using preemptible VMs

Preemptible VMs (/compute/docs/instances/preemptible) run at a much lower price (/compute/vm-instance-pricing) than normal instances, and are suitable for long-running (batch) large experiments (for example, Monte Carlo simulations (/solutions/analyzing-portfolio-risk-using-htcondor-and-compute-engine)). You can also add GPUs to your preemptible Deep Learning VM instances at lower preemptible prices for the GPUs. Note that preemptible VMs are not recommended for interactive experimentation.

Monitor and improve your GPU utilization

For long-living experiments, set up the GPU metrics reporting script (/compute/docs/gpus/monitor-gpus#setup-script) and view logs in Cloud Monitoring (/compute/docs/gpus/monitor-gpus#review-metrics). Get alerts about GPU usage by setting up notifications from Cloud Monitoring (/monitoring/support/notification-options). When you know the GPU usage rates, you can perform tasks such as setting up managed instance groups to autoscale (/compute/docs/autoscaler) resources based on your needs.

Data preparation with BigQuery

BigQuery (/bigquery) is a serverless, highly scalable, and cost-effective cloud data warehouse. Typically, you organize and store your data in BigQuery, which you use for training your ML models. You can use BigQuery for querying and processing during exploratory data analysis (EDA), as well as for data preparation. In addition to the recommendations in this section for optimizing your BigQuery storage and query processing costs, see Cost optimization best practices for BigQuery (/blog/products/data-analytics/cost-optimization-best-practices-for-bigquery).

Training-serving data skew is a common pitfall in building ML systems. It's due to the discrepancy betwee
; prepared for training the model and how it's received during inference. When you prepare your data in
ery, you use SQL scripts to implement preprocessing logic. For batch prediction on data in BigQuery, you c
e same preprocessing SQL scripts to prepare the serving data. The model receives the serving data for
tion in the expected *transformed* form.

other hand, for online prediction, the model receives the serving data from a different source (for exampl
'ub/Sub) in its *raw* form, which isn't expected by the model. Therefore, you need to reimplement the
cessing logic in a different service to handle these online prediction requests, which leads to training-serv
For more information, see Data preprocessing for machine learning: options and recommendations
tions/machine-learning/data-preprocessing-for-ml-with-tf-transform-pt1).

Take advantage of BigQuery pricing

BigQuery flat rate pricing (/bigquery/pricing#flat_rate_pricing) is the cheapest option if you're spending at least US$10k each month; the next cheapest option is flexible slots (/blog/products/data-analytics/introducing-bigquery-flex-slots). For large ML preprocessing workloads, we suggest not using on-demand pricing because the quantity of data is high but compute overhead is typically low.

Use BigQuery to explore and preprocess large amounts of data

During EDA, data is usually retrieved from BigQuery and sent to an AI Platform Notebooks instance. However, if you have a large dataset, this might not be possible. Therefore, it's better to execute the analytics and data processing in BigQuery and use an AI Platform Notebooks instance to retrieve and visualize the results. Similarly, we recommend that you preprocess data in BigQuery before you retrieve it for training your model.

Use the BigQuery Storage API

Use the BigQuery Storage API (/bigquery/docs/reference/storage) to load the data into pandas (/bigquery/docs/bigquery-storage-python-pandas) DataFrames in memory. The API provides fast access to BigQuery-managed storage by using an RPC-based protocol, and it's optimized for retrieving the data. If you're using the Python `pandas` (https://pandas.pydata.org/) library, make sure that you set `use_bqstorage_api` to `True` when you call the `pandas.read_gbq` (https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.read_gbq.html) function.

Use BigQueryClient in TensorFlow to read the data

If you're training a TensorFlow model and no preprocessing is needed, read data using the `tfio.bigquery.BigQueryClient` (https://www.tensorflow.org/io/api_docs/python/tfio/bigquery/BigQueryClient) class. This class establishes a connection to BigQuery and initiates a read session. For more information, see How to read BigQuery data from TensorFlow 2.0 efficiently (https://towardsdatascience.com/how-to-read-bigquery-data-from-tensorflow-2-0-efficiently-9234b69165c8)
.

Partition tables to improve performance and reduce cost

When you prepare data for ML training, you can partition a table based on ingestion time, date, or any timestamp column (/bigquery/docs/partitioned-tables), so that you use the data in only a specific partition to train the model. Each partition is considered separately for long-term storage, so this approach reduces both the cost of storage and the size of query processing.

Keep your data only as long as you need it

Configure default dataset, table, or partition expiration (/bigquery/docs/updating-datasets#table-expiration) and use DDL statements (/bigquery/docs/reference/standard-sql/data-definition-language#alter_table_set_options_statement) to alter your existing tables to avoid incurring costs for storing data that you don't need to preserve.

Data preparation with Dataflow and Apache Beam

You can use Dataflow (/dataflow) to execute a wide variety of data processing pipelines that are implemented in Apache Beam (https://beam.apache.org/). Dataflow enables data analytics at scale and removes operational overhead from data engineering workloads. For more information, see How to efficiently process both real-time and aggregate data with Dataflow
(/blog/products/data-analytics/how-to-efficiently-process-both-real-time-and-aggregate-data-with-dataflow)
.

An alternative to transforming and loading the data in Dataflow is to use Dataproc (/dataproc). This is a
ged service for running Apache Spark and Hadoop clusters in a simpler, more cost-efficient way than runni
wn dedicated Hadoop cluster. For more information, see Optimize Dataproc costs using VM machine type
/products/data-analytics/optimize-dataproc-costs-using-vm-machine-type).

The following recommendations focus on Dataflow batch jobs for the data preparation step in ML.

Experiment with interactive Apache Beam on AI Platform Notebooks

Before you launch a Dataflow job at scale, use the interactive Apache Beam runner (/dataflow/docs/guides/interactive-pipeline-development) (beta) with JupyterLab notebooks. This lets you iteratively develop pipelines on a small dataset and cost-effectively validate the correctness of the pipeline.

Leverage Dataflow FlexRS

Dataflow Flexible Resource Scheduling (FlexRS) (/dataflow/docs/guides/deploying-a-pipeline#dataflow-flexible-resource-scheduling) reduces batch processing costs by using advanced scheduling techniques, the Dataflow Shuffle service, and a combination of preemptible virtual machine (VM) instances and regular VMs.

Leverage Dataflow Shuffle

The service-based Dataflow Shuffle (/dataflow/docs/guides/deploying-a-pipeline#dataflow-shuffle) operation enables faster execution; more efficient consumption of CPU, memory and persistent disk resources; better autoscaling; and better fault tolerance. Shuffle-bound jobs that don't use Dataflow Shuffle can result in increased runtime and job cost (/dataflow/docs/guides/specifying-exec-params).

Use the right machine type for your workload

If you're loading large modules (for example, TensorFlow Hub models) within the worker nodes in Dataflow, consider increasing the size of the worker machines to avoid out-of-memory issues. You can set the machine type (/compute/docs/machine-types) using the `workerMachineType` (/dataflow/docs/guides/specifying-exec-params) parameter. Otherwise, use a smaller machine type to reduce costs.

Disable public IP addresses

To help secure your data processing infrastructure and to reduce networking costs, disable public IP addresses. By default, Dataflow assigns both public and private IP addresses to workers. To change this assignment, use the `usePublicIps` (/dataflow/docs/guides/specifying-exec-params#setting-other-cloud-dataflow-pipeline-options) option when you create a pipeline.

Load modules for preprocessing only once

Load your modules only once, not each time they are called to process a data point. When you prepare data for ML, you might need to download external modules for feature extraction. For example, you can apply this technique when you use TensorFlow Hub (TF-Hub) (https://www.tensorflow.org/hub) modules to extract text embeddings, as described in Building a real-time embeddings similarity matching system (/solutions/machine-learning/building-real-time-embeddings-similarity-matching-system). To load your model only once, download and instantiate the model in the setup function in the class that implements the `beam.DoFn` (https://beam.apache.org/documentation/programming-guide/) transformation.

Choose coders that provide good performance

Use coders like ProtoCoder (https://beam.apache.org/releases/javadoc/2.22.0/org/apache/beam/sdk/extensions/protobuf/ProtoCoder.html)
or Schemas

(https://beam.apache.org/releases/javadoc/2.22.0/org/apache/beam/sdk/schemas/Schema.html). Encoding and decoding can be a big source of overhead. Therefore, if you have a large BLOB but need only part of it to be structured, you can selectively decode that part.

Avoid setting `num_shards` on output transforms

When you use [`beam.io.WriteToText`](https://beam.apache.org/releases/pydoc/2.22.0/apache_beam.io.textio.html#apache_beam.io.textio.WriteToText) (https://beam.apache.org/releases/pydoc/2.22.0/apache_beam.io.textio.html#apache_beam.io.textio.WriteToText), leave `num_shards` set to 0. This lets Dataflow decide on the number of shards to write the output data to. If you fix the number of shards to a small number (for example, 1), all of the data will be aggregated in one worker node to be written as one file. This can cause the job to slow down.

Create a batch of data points and process it as a whole

To create a batch of data points, we recommend that you use `beam.BatchElements` (https://beam.apache.org/releases/pydoc/2.22.0/apache_beam.transforms.util.html#apache_beam.transforms.util.BatchElements), which improves vectorization. For example, imagine that you're using a TF-Hub module to extract embeddings from text as part of your Dataflow job. It's more efficient to get output for a batch of data points all at once using the module rather than invoking the module with each individual data point. For more information, see examples of calling AI Platform APIs in the `apache_beam.ml.gcp package` (https://beam.apache.org/releases/pydoc/2.22.0/apache_beam.ml.gcp.html) documentation.

Preprocess the data once and save it as a TFRecord file

If you plan to train a TensorFlow model, create a TFRecord file (https://www.tensorflow.org/tutorials/load_data/tfrecord#tfrecords_format_details). A TFRecord file contains a sequence of records, where each record is encoded as a byte string. TFRecord files are optimized for training TensorFlow models. You can use TensorFlow Transform (TFT) (https://www.tensorflow.org/tfx/transform/get_started) to prepare the data as TFRecords for training TensorFlow models. TFT is implemented using Apache Beam and runs at scale on Dataflow.

Log only what will be useful

In the Dataflow runner, logs from all workers are sent to a central location in Cloud Logging (/dataflow/docs/guides/logging). Too much logging can decrease performance and increase costs, so consider what you are logging and the level of granularity you need. Then override the logging settings (/dataflow/docs/guides/logging#SettingLevels) accordingly. For more information, see Building production-ready data pipelines using Dataflow: Monitoring data pipelines (/solutions/building-production-ready-data-pipelines-using-dataflow-monitoring).

Monitor your Dataflow jobs

Use the Dataflow monitoring interface (/dataflow/docs/guides/using-monitoring-intf), which provides an overview of your jobs and shows details about their status and execution. The job-monitoring charts illustrate step-level visibility to help identify what might be causing lag. They also show statistical information that highlights anomalous behavior, and provide metrics to help you find bottlenecks and sinks. For more information about how to improve performance, see Profiling Dataflow Pipelines (https://medium.com/google-cloud/profiling-dataflow-pipelines-ddbbef07761d).

Training with AI Platform Training

AI Platform Training (/ai-platform/training/docs/overview) is a simple, fully managed service that brings the power and flexibility of TensorFlow, scikit-learn (https://scikit-learn.org/stable/), XGBoost, and custom containers to the cloud. You can use AI Platform to train your ML models and tune their hyperparameters at scale using a serverless distributed environment and powerful accelerators.

Make the right trade-off between model accuracy and size for your task

If your task requires high accuracy, you might need to train a large and complex model, like Inception_v4 (https://arxiv.org/abs/1602.07261) for vision applications. However, if you plan to serve your model on edge devices that have limited storage and compute resources, it's better to train a smaller model that has less precision. Smaller models are also faster to train and produce predictions faster than larger models. For example, MobileNets (https://arxiv.org/abs/1704.04861) are optimized for mobile vision applications.

Choose the right machine configuration for your training characteristics

You can choose arbitrary machine types (/ai-platform/training/docs/machine-types#machine_type_table) and various GPU types (/ai-platform/training/docs/using-gpus). The machine configuration that you choose depends on your data size, model size, and algorithm selection. For example, deep learning frameworks like TensorFlow and PyTorch benefit from GPU acceleration, while frameworks like scikit-learn and XGboost don't. On the other hand, when you're training a large scikit-learn model, you need a memory-optimized machine.

Don't use large machines for simple models

Simple models might not train faster with GPUs or with distributed training, because they might not be able to benefit from increased hardware parallelism. Because the scikit-learn framework doesn't support distributed training, make sure that you use only the scale-tier or custom machine-type configurations that correspond to a single worker instance.

For small datasets, use a single large machine

If you can fit all of your data in memory, it can be more cost-effective to use a large machine type (high CPUs and memory), rather than performing distributed training or streaming the data from Cloud Storage to your machine.

Scale up before scaling out

Scaling up instead of scaling out while experimenting can help you identify the configurations that are performant and cost-effective. For example, start by using a single worker that has a single GPU, and then try a more powerful GPU before you use multiple GPUs. After that, try distributed training, as discussed later in this section. Scaling up is faster than scaling out because network latency is much slower than the GPU interconnect.

For large datasets, use distributed training

Distributed training (/ai-platform/training/docs/training-jobs#formatting_your_configuration_parameters) performs data parallelism on a cluster of nodes to reduce the time required to train a TensorFlow model when you use a large dataset. Make sure that you adjust the number of iterations with respect to the distribution scale—that is, take the total number of iterations that are required and divide that total by the number of GPUs multiplied by the number of worker nodes.

Increase the size of the parameter server to increase the bandwidth

Asynchronous distributed training with powerful GPUs requires a lot of bandwidth to the parameter server (https://www.tensorflow.org/guide/distributed_training#parameterserverstrategy). However, the bandwidth available is proportional to the size (number of vCPUs) of the parameter server. Therefore, we recommended that you over-provision the compute capacity of the parameter server in order to increase the bandwidth and take full advantage of GPU workers.

Use MultiWorkerMirroredStrategy for distributed training of TensorFlow models

The `MultiWorkerMirroredStrategy` (https://www.tensorflow.org/guide/distributed_training#multiworkermirroredstrategy) class implements synchronous distributed training across multiple workers, each potentially with multiple GPUs. Each variable in the model is mirrored across all of the replicas, and efficient all-reduce algorithms are used to communicate the variable updates across the devices. This strategy is scalable, performant, and doesn't need a parameter server, so it reduces the training infrastructure cost. For more information, see Distributed training with TensorFlow (https://www.tensorflow.org/guide/distributed_training).

Use Cloud TPU for supported models

If you're training one of the official supported models (/tpu/docs/tutorials/support-matrix) for TensorFlow and PyTorch, use Cloud TPU (/tpu). Cloud TPU is built around Google-designed custom ASIC chips (https://wikipedia.org/wiki/Application-specific_integrated_circuit) and is specifically built to accelerate deep learning computations. You can run your training jobs on AI Platform Training using Cloud TPU (/ai-platform/training/docs/using-tpus), which offers pricing that can significantly reduce the cost (/blog/products/ai-machine-learning/mlperf-benchmark-establishes-that-google-cloud-offers-the-most-accessible-scale-for-machine-learning-training)
. For more information about training custom models, see Use TPUs (https://www.tensorflow.org/guide/tpu).

Use the `tf.data` API to best utilize your accelerators

Achieving peak training performance on GPUs and TPUs requires an efficient input pipeline that delivers data for the next step before the current step has finished. To help you build flexible and efficient data input pipelines for TensorFlow models, the `tf.data` (https://www.tensorflow.org/guide/data) API provides a set of functions like `map` (https://www.tensorflow.org/api_docs/python/tf/data/Dataset#map), `batch` (https://www.tensorflow.org/api_docs/python/tf/data/Dataset#batch), `cache` (https://www.tensorflow.org/api_docs/python/tf/data/Dataset#cache), `prefetch` (https://www.tensorflow.org/api_docs/python/tf/data/Dataset#prefetch), `shuffle` (https://www.tensorflow.org/api_docs/python/tf/data/Dataset#shuffle), `repeat` (https://www.tensorflow.org/api_docs/python/tf/data/Dataset#repeat), and `interleaves` (https://www.tensorflow.org/api_docs/python/tf/data/Dataset#interleave). In addition, using larger batch sizes improves the utilization of your accelerators. For more information, see Better performance with the tf.data API (https://www.tensorflow.org/guide/data_performance).

Stream data from Cloud Storage for training scikit-learn models

When you train a scikit-learn model on large datasets (/ai-platform/training/docs/training-at-scale#large_datasets), downloading the entire dataset into the training worker and loading it into memory doesn't scale. In these cases, consider using TensorFlow's stream-read/file_io API (https://github.com/tensorflow/tensorflow/blob/r2.1/tensorflow/python/lib/io/file_io.py), which is preinstalled on the worker VM.

Consider using distributed XGBoost with large datasets

When you train an XGBoost model on large datasets, you can benefit from the built-in distributed XGBoost algorithm (/ai-platform/training/docs/algorithms/distributed-xgboost) because the model runs at scale using multiple virtual machines in parallel.

Control the lifetime of your training job

In your AI Platform Training job, make sure that you set `max_running_time` (/ai-platform/training/docs/reference/rest/v1/projects.jobs#scheduling) to limit the running time of the job. If the training job is still running after this duration, AI Platform Training cancels it so that you no longer incur costs.

Use early stopping in training your models

To limit the cost of an ML training job, we recommend that you implement an early stopping (https://wikipedia.org/wiki/Early_stopping) behavior, either when your model reaches a certain predictive performance level or when there is no improvement in the predictive performance of your model.

Prepare your environment in a container image

If your training environment requires a lot of dependencies that take time to install, use a container image. Installing these dependencies adds considerable time overhead in the beginning of the training job. Using a prepared container image instead can save time and reduce cost, and you can submit your training container image using AI Platform Training with custom containers (/ai-platform/training/docs/containers-overview).

Use automatic hyperparameter tuning

AI Platform provides a blackbox optimization service (/ai-platform/training/docs/hyperparameter-tuning-overview) that helps you automatically tune hyperparameters in complex ML models. When you're configuring a hyperparameter tuning job (/ai-platform/training/docs/using-hyperparameter-tuning), we recommend that you set `enableTrialEarlyStopping` to `True`; this helps limits the cost of the hyperparameter tuning job. If you have a previous hyperparameter tuning job, you can set `resumePreviousJobId` to `True` to start from a state that is partially optimized. This makes it possible to reuse the knowledge gained in the earlier hyperparameter tuning job. In addition, we recommended that you set `maxParallelTrials` to be between 2 and `maxTrials`/2 in order to converge faster to good hyperparameter values, which in turn reduces cost of hyperparameter tuning.

Clean up artifacts that are produced by old jobs

Running many training jobs for a long period of time can produce a considerable number of artifacts, like logs and checkpoints. Retaining these artifacts incurs unnecessary storage cost if you no longer need them.

Monitor the resource utilization of training jobs

You can monitor your training jobs (/ai-platform/training/docs/monitor-training) using the Google Cloud Console to review resource consumption (/ai-platform/training/docs/monitor-training#monitoring_resource_consumption), including CPU, GPU, and memory utilization, as well as network usage. This helps you decide whether to scale your resources up for performance improvement or down for cost optimization. Furthermore, you can use sizing recommendations (/compute/docs/instances/apply-sizing-recommendations-for-instances) to effectively downsize your machine types based on changes in vCPU and RAM usage.

Serving with AI Platform Prediction

AI Platform Prediction (/ai-platform/prediction/docs) is a fully managed, scalable service that you can use to host your trained ML models in the cloud and serve them as REST APIs for online inference. In addition, when you want to store a large amount of data offline, you can use the deployed model for batch prediction requests.

Use reduced-precision floating-point types

Smaller models lead to lower serving latency. When you build a TensorFlow model for online serving, we recommend that you use 16-bit floating-point types (half precision) rather than 32-bit floating-point types (full precision) to represent your data and the weights of your model. You can also use mixed-precision training (https://www.tensorflow.org/guide/mixed_precision) to maintain numerical stability, to speed up training, and to produce smaller models that have lower inference latency.

Reduce model size using post-training quantization

Post-training quantization (https://www.tensorflow.org/lite/performance/post_training_quantization) is a conversion technique that can reduce your TensorFlow model size while also improving CPU and hardware accelerator latency, with little degradation in model accuracy. Options for post-training quantization include dynamic range quantization (https://www.tensorflow.org/lite/performance/post_training_quant), full integer quantization (https://www.tensorflow.org/lite/performance/post_training_integer_quant), and float16 quantization (https://www.tensorflow.org/lite/performance/post_training_float16_quant).

Take advantage of the TensorFlow Model Optimization Toolkit

Before you deploy your TensorFlow models to AI Platform Prediction, use the TensorFlow Model Optimization Toolkit (https://blog.tensorflow.org/2019/06/tensorflow-integer-quantization.html). This toolkit provides a suite of tools for optimizing ML models deployment and execution. You can use the tools for post-training quantization (https://blog.tensorflow.org/2019/06/tensorflow-integer-quantization.html), quantization aware training (https://blog.tensorflow.org/2020/04/quantization-aware-training-with-tensorflow-model-optimization-toolkit.html)
, and model pruning (https://blog.tensorflow.org/2019/05/tf-model-optimization-toolkit-pruning-API.html).

Use manual scaling for highly variable request volume

If the number of requests that your model receives inherently fluctuates faster than automatic scaling can keep up with, it can be more efficient to use manual scaling. If you have a predictable workload (for example, a high load on weekends and a low load on weekdays), we recommend that you schedule scaling to match expected demand.

Use the N1 machine type with manual scaling for low latency

If your traffic has regular steep spikes and if reliably low latency is important to your application, use N1 machine types, which provide much lower and predictable latency compared to other machine types.

Choose the right machine type for serving your model

You can customize the type of virtual machine (/ai-platform/prediction/docs/machine-types-online-prediction) that AI Platform Prediction uses to host your model. For serving large models with high traffic, you can choose one of the N1 machines (standard, memory-optimized, or CPU-optimized). Note that N1 machines do not support scaling down to zero nodes. If your model requires scaling to zero nodes, use `mls1-c1-m2` and `mls1-c4-m2` machines.

Use autoscaling and set `minNodes` to zero when low latency isn't critical

If your use case isn't latency sensitive, you can use autoscaling and set `minNodes` (/ai-platform/prediction/docs/reference/rest/v1/projects.models.versions#autoscaling) to zero. This can help reduce cost when your model service isn't receiving any requests. For autoscaling, you can use `mls1-c1-m2` and `mls1-c4-m2` machines; the `mls1-c4-m2` (quad-core) machines can improve latency.

Check whether you need GPUs for online serving

When you use N1 machine types, AI Platform Prediction lets you attach GPU accelerators (/ai-platform/prediction/docs/machine-types-online-prediction#gpus) to your online prediction service. However, you only benefit from GPU accelerators if you're serving TensorFlow models, not if you're using scikit-learn or XGBoost. In addition, if you don't have a large TensorFlow model that processes large data points for online prediction, like images or video streams, you might not need the full power of a GPU.

Use TF-TRT with NVIDIA GPU accelerators

When you use NVIDIA GPU accelerators for serving, we recommend that you use TensorFlow with TensorRT (TF-TRT) (https://docs.nvidia.com/deeplearning/frameworks/tf-trt-user-guide/index.html#capabilities). This tool performs several important transformations and optimizations to the neural network graph. These include eliminating layers with unused outputs; fusing convolution, bias, and ReLU to form a single layer; layer aggregation; horizontal layer fusion (layer aggregation); and quantization. For more information, see Optimizing TensorFlow Serving performance with NVIDIA TensorRT (https://medium.com/tensorflow/optimizing-tensorflow-serving-performance-with-nvidia-tensorrt-6d8a2347869a)
.

Reuse the authenticated service object in the client object

To call a model version (/ai-platform/prediction/docs/deploying-models#create_a_model_version) that's deployed to AI Platform Prediction, you need to create a service object that interacts with the model API, using the `googleapiclient.discovery.build` (https://googleapis.github.io/google-api-python-client/docs/epy/googleapiclient.discovery-module.html)
method. The method uses the `GOOGLE_APPLICATION_CREDENTIALS` environment variable for authentication. Calling this method every time you request a prediction adds overhead to the response time. Therefore, create this service object once and then reuse it in subsequent prediction calls.

Use base64 encoding when sending images

When you send images to AI Platform Prediction for inference, use base64 (https://www.base64decode.org/) encoding rather than an array of floats. AI Platform Prediction service expects a JSON object that represents the prediction request. However, binary data, like images, cannot be formatted as the UTF-8 encoded strings that JSON supports. Therefore, use base64 encoding to represent it, because this minimizes the size of the request payload compared to using an array of floats, and consequently reduces the response time.

Send multiple data points in one request

Model versions that are deployed to AI Platform Prediction accept a list of instances (/ai-platform/prediction/docs/online-predict#formatting_instances_as_json_strings). You can take advantage of this if you have multiple data points (less than 100), and if you want to get a prediction by sending them in one request payload. However, the overall data size must be less than 1.5 MB.

For high traffic, reduce sample size for request-response logging

AI Platform provides request-response logging (/ai-platform/prediction/docs/online-predict#requesting_logs_for_online_prediction_requests) to BigQuery. Online prediction that's working at a high rate of queries per second (QPS) can produce a substantial number of logs. These are subject to BigQuery pricing; by reducing the sample size, you can reduce the quantity of logging and therefore potentially reduce your cost. To configure the volume of request-response logging to BigQuery, specify the **`samplingPercentage`** (/ai-platform/prediction/docs/reference/rest/v1/projects.models.versions#requestloggingconfig) value when you're deploying your model version to AI Platform Prediction.

Use AI Platform batch prediction jobs for scoring large datasets

When you do offline prediction on a large number of instances, and you don't need your predictions right away, you can use the batch predictions (/ai-platform/prediction/docs/batch-predict) service. This service runs a distributed data processing job at scale for better throughputs and lower cost compared to online prediction.

Increase the batch size for batch prediction jobs

If you use batch prediction jobs, configure the batch prediction job (/ai-platform/prediction/docs/batch-predict#configuring_a_batch_prediction_job) to maximize the use of vectorization when the model is scoring the data. This increases the throughput of the batch prediction job, and it reduces the running time and consequently the cost.

Delete unused model versions when you use manual scaling

If your model versions are set for manual scaling, or if the `minNodes` parameter in autoscaling isn't set to zero, delete unused model versions. Unused model versions incur unnecessary cost, depending on the size of the machine and how long they've been running.

Monitor your AI Platform Prediction metrics

You should monitor your model's traffic patterns, error rates, latency, and resource utilization to help you spot problems with your models, and to help find the right machine type to optimize latency and cost. You can also use Cloud Monitoring (/monitoring/docs) to configure alerts based on the ML metrics (/monitoring/api/metrics_gcp#gcp-ml). For example, you can receive alerts if the model prediction latency gets too high. For more information, see Monitoring model versions (/ai-platform/prediction/docs/monitor-prediction).

Orchestration with AI Platform Pipelines

AI Platform Pipelines is a hosted Kubeflow Pipelines (KFP) (https://www.kubeflow.org/docs/pipelines/overview/pipelines-overview/) service on Google Kubernetes Engine (GKE) (/kubernetes-engine). It provides a platform where you can orchestrate the steps in your ML workflow, like data extraction, validation, preparation, model training, tuning, evaluation, and serving. You can implement your ML pipelines using TensorFlow Extended (TFX) (https://www.tensorflow.org/tfx) or the KFP SDK (https://www.kubeflow.org/docs/pipelines/sdk/sdk-overview/), and then deploy them to AI Platform Pipelines for scalable and reliable execution.

Use managed services for pipeline steps

To scale out your ML training pipeline execution, especially when you work with large datasets and big models, we recommend that you use Google Cloud managed services. We also recommend that you offload the different workloads to the right services. For example, use Dataflow for data validation and transformation steps, and use AI Platform (distributed) training for model training and tuning. However, when you work with relatively small datasets, it can be more efficient to perform all of the steps locally on the GKE cluster of AI Platform Pipelines.

For large datasets, use Dataflow for the model evaluation step

When you work with large datasets, Dataflow is more scalable and cost-effective than AI Platform Training. For example, see TensorFlow Model Analysis (https://github.com/tensorflow/model-analysis), which is implemented using Apache Beam, and which can run on Dataflow to evaluate TensorFlow SavedModels.

Perform incremental training with a warm start (if possible)

In continuous training pipelines, you train your model regularly on new data. If your model implementation doesn't change from one training iteration to another, you can start the current training iteration using the model that was trained in the previous iteration and tune it using the new data. This reduces the time (and consequently the cost) of training your model every time from scratch using all of the data. It also converges faster than training a randomly initialized model using only the new data. TFX Pipelines (https://github.com/tensorflow/tfx/blob/master/tfx/examples/chicago_taxi_pipeline/taxi_pipeline_warmstart.py#L109) has built-in support for warm starts.

Optimize the frequency of retraining the model

You need to retrain the model at times, but it can be inefficient and costly to retrain it too frequently. On the other hand, the model's predictive performance decays if it's not retrained often enough. Monitoring the deployed model for data drift and concept drift (https://wikipedia.org/wiki/Concept_drift) gives you indications that the model might need to be retrained.

Clean up artifacts produced by the pipeline steps

Running a pipeline produces artifacts like data splits, transformed data, validation output, and evaluation output. These artifacts accumulate quickly and incur unnecessary storage cost, so you should periodically clean up the artifacts that you don't need.

Build your own GKE cluster for KFP

AI Platform Pipelines makes it easier to set up and use Kubeflow Pipelines (KFP) by creating a GKE cluster for you and deploying KFP onto the cluster. Alternatively, you can build your own GKE cluster (/ai-platform/pipelines/docs/setting-up) so that you have more granular control over the cluster. This lets you customize the configuration to your workload's requirements.

Configure autoscaling and node auto-provisioning for the GKE cluster

Autoscaling (/kubernetes-engine/docs/how-to/cluster-autoscaler) and node auto-provisioning (/kubernetes-engine/docs/how-to/node-auto-provisioning) help you to control the number of pipelines that run in parallel compared to those that are queued while waiting for resources. Automatic GPU node (/kubernetes-engine/docs/how-to/gpus) provisioning can reduce costs when you're not using GPUs.

Balance the benefits of persisted storage

Persistent disks (/persistent-disk) can have higher performance when they're attached to GKE nodes. However, the data must be copied to Cloud Storage when other services need to use the data. If the data is used only by ML processes within the GKE cluster, use persistent disks. Otherwise, use Cloud Storage to share data and artifacts between services.

Configure Cloud SQL for the pipeline metadata storage

When you use Cloud SQL (/sql) as the ML metadata store, pipelines can have higher throughput and better scalability than with the in-cluster metadata storage. The metadata is also preserved even if the cluster is deprovisioned.

Monitoring performance and managing the cost of ML proje

Cloud Operations (/products/operations) provides a suite of tools to monitor, troubleshoot, and improve the performance of your ML training and serving systems on Google Cloud. At the project level, Google Cloud offers several ways in which you can capture, report, manage, and forecast costs. For more information about cost optimization on Google Cloud, see the whitepaper Understanding the principles of cost optimization (/resources/principles-of-cost-optimization-whitepaper), which provides an overview of Google Cloud's cost management products (/cost-management), and see the related posts on cost management (/blog/topics/cost-management) in the Google Cloud blog.

Investigate your system using application logs

Cloud Logging (/logging) is a fully managed service that performs at scale and that can ingest application and system log data. This lets you analyze and export selected logs to long-term storage in real time.

Use Cloud Monitoring to track ML applications

Cloud Monitoring (/monitoring/docs) collects metrics, events, and metadata from Google Cloud services, and then generates insights through dashboards and charts (/monitoring/dashboards) and through alerts (/monitoring/alerts). You can use this information to track performance, uptime, and the overall health of your ML applications. You can see how your ML applications are functioning and troubleshoot them if needed by using Cloud Trace (/trace), Cloud Debugger (/debugger), and Cloud Profiler (/profiler).

Capture your spend data in a consistent format

When you capture information about your costs and spend, use tools and techniques that aggregate the data into a consistent format. This lets you work with the data in a consistent way and infer appropriate information from it to effectively manage your costs. You can specify formatting for your cost data by setting up a billing export (/billing/docs/how-to/export-data-bigquery) to BigQuery to host the data set for retention and analysis (/blog/products/data-analytics/analyzing-gcp-costs-using-folders-and-bigquery-billing-export). The billing export provides a more detailed view of your usage and costs than the console reports, which you can visualize through tools like Data Studio (https://datastudio.google.com/).

Attribute resource usage by using labels

Labels (/resource-manager/docs/creating-managing-labels) are key-value pairs that can be attached to resources. You can use them to identify (and therefore track the costs for) a team, environment, or any other way you organize your work for a group of resources. We recommend that you make label values simple and useful from both a business and technical perspective, as well as apply them consistently (programmatically if possible). For a list of products the support labels, see supported services (/resource-manager/docs/creating-managing-labels#label_support).

Understand and monitor your projects costs

To stay informed about your project costs, use the Billing Reports (/billing/docs/how-to/reports) page and custom dashboards (/billing/docs/how-to/visualize-data). The Billing Reports page shows you information about your spend, which you can filter by project, labels, product, and so on. You can build custom dashboards using Data Studio or other visualization tools. You can add different views to ensure that all users have access to the insights that they need in order to make informed decisions about cost optimizations.

Control costs through budgets and alerting

Use budgets (/billing/docs/how-to/budgets) and programmatic notifications (/billing/docs/how-to/budgets-programmatic-notifications) (alerts) to help control your costs by setting spend levels and specifying that you want to be alerted when your costs get to those levels. (Be aware that even when an alert has been triggered, resources continue to function as normal.) You can also integrate alerts with email (/billing/docs/how-to/budgets-notification-recipients) or third-party solutions like Slack

(/billing/docs/how-to/notify). For research and development projects, students, researchers, and developers, by setting disabling Cloud Billing, you can selectively control resources (/billing/docs/how-to/notify#selectively_control_usage) or cap (/billing/docs/how-to/notify#cap_disable_billing_to_stop_usage) the resources altogether, which stops resources at predetermined thresholds. You can also use billing roles (/billing/docs/how-to/billing-access) to manage costs by restricting who has permission to link resources to your billing account.

Take advantage of serverless

Serverless products let you pay only for what you use. BigQuery offers flexible pricing options (/bigquery/pricing), and Dataflow service usage is billed (/dataflow/pricing) in per-second increments, on a per-job basis.

Run your services in the same region

Ideally, your ML services (Dataflow, Cloud Storage, AI Platform, and BigQuery) should be in the same region to avoid incurring ingress and egress costs (/compute/network-pricing) for transferring data between regions.

Take advantage of discounts for compute resources

Google Cloud offers a number of discounts, such as sustained use discounts (/compute/docs/sustained-use-discounts) (including GPUs), which are automatically applied when you run resources for a significant portion of each month. You can also take advantage of committed use discounts (/compute/docs/instances/signing-up-committed-use-discounts#how_committed_use_discounts_work)
, which are additional discounts that apply when you commit to use a specific volume of resources (including vCPUs, RAM, and GPUs). You can request these discounts through the Cloud Console. Committed use discounts are ideal for predictable steady-state workloads.

Forecast costs to effectively plan and prioritize your cloud investment

Use the Cloud Billing reports page (/billing/docs/how-to/reports) to see forecasted costs (/billing/docs/how-to/reports#cost-forecast) for up to 12 months in the future. You can also use BigQuery to do your own forecasting. At a service level, you can use the Google Cloud pricing calculator (/products/calculator) to estimate what your costs might be.