API Collection

Machine-learning passes

Add machine-learning model inference to your Metal app's GPU workflow.

Overview

Metal 4 introduces the ability to run CoreML models efficiently from within the Metal workflow. This is useful for apps that need to apply the output from a model in a Metal context, such as rendering a scene or running a compute dispatch. Add machine-learning inference to your app's Metal workflow by converting a CoreML model into a Metal machine-learning (ML) package at development time, and then applying that package in a machine-learning encoder at runtime.

Your app can combine its render, compute, and machine-learning work within the same command buffer, without needing to synchronize or wait for the CPU. By running inference with Core ML models in the GPU timeline, your app can provide model inputs, such as from a compute pass, and immediately work with a model's outputs from a machine-learning pass.

Metal 4 introduces new types for *tensors*, which are multidimensional-data arrays that serve as inputs, outputs, and intermediate values for machine-learning models. Metal Shading Language (MSL) also adds tensor operators and other functionalities, such as cooperative tensors, which your app's shader code can use when working with tensors and their data in parallel during any GPU stage.

Convert a Core ML model into a Metal package

Convert a Core ML model by creating a Metal machine-learning (ML) package from it with the `metal-package-builder` tool — which is part of the tools bundled in Xcode 26 and later — and then add the Metal ML package to your Xcode project. When you build the project, Xcode compiles the model in the Metal ML package to a Metal library that your app can load at runtime.

Apply the model from your app's GPU workflow by encoding machine-learning commands

Metal 4 introduces a new encoder type, `MTL4MachineLearningCommandEncoder`, which encodes inference commands that run a Core ML model in a machine-learning pass that runs alongside your other Metal tasks, such as render and compute passes. To encode machine-learning inference commands for the GPU to run, you need to create a machine-learning pipeline state, provide an `MTLHeap` for temporary scratch memory, and `MTLTensor` instances for the model's inputs and output. You create a machine-learning pipeline-state from the library that Xcode creates from a Metal ML package, which you can then apply to a machine-learning encoder.

> Note
>
> Machine-learning encoders run Core ML models but they can't build new networks or modify layers and inputs of existing ones; for those tasks, see Core ML andMetal Performance Shaders Graph

The system automatically chooses an inference engine, such as a device's GPU or Apple Neural Engine (ANE) for each machine-learning model. The GPU can run additional, independent render or compute work with the GPU when the system chooses to run a model on the ANE.

Provide model inputs and retrieve outputs with tensors

Metal 4 introduces `MTLTensor` a resource type that stores multi-dimensional data arrays for machine-learning models. The tensor types works with the common weight and input data types, such as `int8` and `fp16`. You create input and output tensors to provide data to, and retrieve data from, a model, respectively, by passing those tensors to a machine-learning encoder when encoding a pass that invokes the model on the GPU timeline.

> Tip
>
> You can inspect a tensor's underlying data with the Metal debugger in Xcode 26.

Work with tensors on the GPU timeline

Metal 4 also adds tensor types and basic tensor operators to the Metal Shading Language (MSL), which include convolution, matrix multiplication, and reduction. You can use these operators in your MSL code that runs during the machine-learning GPU stage, and all other stages, such as blit, dispatch, vertex, fragment, and so on. This functionality gives you the option to work with tensor data in your app’s various GPU functions, such as modifying weights in an intermediate tensor between model inference invocations.

The MSL tensor types include:

`tensor_handle`
: Parameter type for the entry point of a machine-learning kernel

`tensor_offset`
: Provides a subset of another tensor type

`tensor_inline`
: Transient and intermediate tensor type for passing data between kernels

`cooperative_tensor`
: Distributes each part its data to the threads that operates on it

Cooperative tensors provide temporary memory for transient tensors by equally distributing their data among the threads that work with that tensor. This memory distribution reduces memory bandwidth by allocating the memory from thread-private or threadgroup-private address spaces, which is important for latency-critical, machine-learning algorithms.

MSL version 4 also introduces operation descriptors, with which you can define custom operations and run them directly in your shader code.

Synchronize a machine-learning pass with other passes

Your app can encode a machine-learning pass that works with other passes synchronizing the dependencies between its stage, `machineLearning`, and the relevant stages of one or more the other passes. To synchronize with stages in other passes, add barriers with the appropriate scope. For example, you can encode a machine-learning pass that:

- Depends on the output from a previous render pass
- Produces an output that a subsequent render pass consumes as its input
- Synchronizes with both the previous and subsequent render passes with a consumer and producer queue-barrier, respectively

For more information about stages and barriers, see `MTLStages` and Resource synchronization , respectively.

Topics

Encoding a machine-learning pass

`protocol MTL4MachineLearningCommandEncoder`

Encodes dispatch commands that run machine-learning model inference on Apple silicon.

`protocol MTL4MachineLearningPipelineState`

A pipeline state that you can use with machine-learning encoder instances.

Configuring a machine-learning pipeline

`class MTL4MachineLearningPipelineDescriptor`

Description for a machine learning pipeline state.

`class MTL4MachineLearningPipelineReflection`

Represents reflection information for a machine learning pipeline state.

See Also

Command encoders

Render passes

Encode a render pass to draw graphics into an image.

Compute passes

Encode a compute pass that runs computations in parallel on a thread grid, processing and manipulating Metal resource data on multiple cores of a GPU.

Blit passes

Encode a block information transfer pass to adjust and copy data to and from GPU resources, such as buffers and textures.

Indirect command encoding

Store draw commands in Metal buffers and run them at a later time on the GPU, either once or repeatedly.

Ray tracing with acceleration structures

Build a representation of your scene's geometry using triangles and bounding volumes to quickly trace rays through the scene.