```
import CoreML.MLAllComputeDevices
import CoreML.MLArrayBatchProvider
import CoreML.MLBatchProvider
import CoreML.MLCPUComputeDevice
import CoreML.MLComputeDeviceProtocol
import CoreML.MLComputePlan
import CoreML.MLComputePlanCost
import CoreML.MLComputePlanDeviceUsage
import CoreML.MLCustomLayer
import CoreML.MLCustomModel
import CoreML.MLDictionaryConstraint
import CoreML.MLDictionaryFeatureProvider
import CoreML.MLExport
import CoreML.MLFeatureDescription
import CoreML.MLFeatureProvider
import CoreML.MLFeatureType
import CoreML.MLFeatureValue
import
CoreML.MLFeatureValue_MLImageConversion
import CoreML.MLGPUComputeDevice
import CoreML.MLImageConstraint
import CoreML.MLImageSize
import CoreML.MLImageSizeConstraint
import CoreML.MLImageSizeConstraintType
import CoreML.MLKey
import CoreML.MLMetricKey
import CoreML.MLModel
import CoreML.MLModelAsset
import CoreML.MLModelCollection
import CoreML.MLModelCollectionEntry
import CoreML.MLModelConfiguration
import CoreML.MLModelDescription
import CoreML.MLModelError
```

```
import CoreML.MLModelMetadataKeys
import CoreML.MLModelStructure
import
CoreML.MLModelStructureNeuralNetwork
import
CoreML.MLModelStructureNeuralNetworkLayer
import CoreML.MLModelStructurePipeline
import CoreML.MLModelStructureProgram
import
CoreML.MLModelStructureProgramArgument
import
CoreML.MLModelStructureProgramBinding
import
CoreML.MLModelStructureProgramBlock
import
CoreML.MLModelStructureProgramFunction
import
CoreML.MLModelStructureProgramNamedValueT
ype
import
CoreML.MLModelStructureProgramOperation
import
CoreML.MLModelStructureProgramValue
import
CoreML.MLModelStructureProgramValueType
import CoreML.MLModel_MLComputeDevice
import CoreML.MLModel_MLModelCompilation
import CoreML.MLModel_MLState
import CoreML.MLMultiArray
import CoreML.MLMultiArrayConstraint
import CoreML.MLMultiArrayShapeConstraint
import
CoreML.MLMultiArrayShapeConstraintType
```

```
import CoreML.MLNeuralEngineComputeDevice
import CoreML.MLNumericConstraint
import CoreML.MLOptimizationHints
import CoreML.MLParameterDescription
import CoreML.MLParameterKey
import CoreML.MLPredictionOptions
import CoreML.MLReshapeFrequencyHint
import CoreML.MLSequence
import CoreML.MLSequenceConstraint
import CoreML.MLSpecializationStrategy
import CoreML.MLState
import CoreML.MLStateConstraint
import CoreML.MLTask
import CoreML.MLUpdateContext
import CoreML.MLUpdateProgressEvent
import CoreML.MLUpdateProgressHandlers
import CoreML.MLUpdateTask
import CoreML.MLWritable
import CoreVideo
import Foundation
import _Concurrency
import _StringProcessing
import _SwiftConcurrencyShims

/// Represents a compute device capable
of running machine learning computations
/// and other tasks like analysis and
processing of images, sound, etc.
@available(macOS 14.0, iOS 17.0, watchOS
10.0, tvOS 17.0, *)
public enum MLComputeDevice : Equatable,
Hashable, CustomStringConvertible,
Sendable {
```

```
    /// Represents a CPU device. The
associated value has information
    /// about the represented device.
    case cpu(MLCPUComputeDevice)

    /// Represents a GPU device. The
associated value has information
    /// about the represented device.
    case gpu(MLGPUComputeDevice)

    /// Represents a NeuralEngine device.
The associated value has information
    /// about the represented device.
    case
neuralEngine(MLNeuralEngineComputeDevice)

    /// A textual representation of this
instance.
    ///
    /// Calling this property directly is
discouraged. Instead, convert an
    /// instance of any type to a string
by using the `String(describing:)`
    /// initializer. This initializer
works with any type, and uses the custom
    /// `description` property for types
that conform to
    /// `CustomStringConvertible`:
    ///
    ///     struct Point:
CustomStringConvertible {
    ///         let x: Int, y: Int
```

```
    ///
    ///         var description: String {
    ///             return "(\(x), \(y))"
    ///         }
    ///     }
    ///
    ///     let p = Point(x: 21, y: 30)
    ///     let s = String(describing: p)
    ///     print(s)
    ///     // Prints "(21, 30)"
    ///
    /// The conversion of `p` to a string
in the assignment to `s` uses the
    /// `Point` type's `description`
property.
    public var description: String {
get }

    /// An array of all compute devices.
    ///
    /// The array contains all the
compute devices that are accessible. If a
compute device becomes
    /// inaccessible for some reason (for
e.g. if an external GPU is unplugged)
then `allComputeDevices`
    /// will return an array without the
compute device.
    ///
    /// Some compute devices are
exclusive to the domain ML frameworks
such as Vision and not available
    /// to CoreML. Use
```

```
`MLModel.availableComputeDevices` to get
the subset of devices available to
CoreML.
    public static var allComputeDevices:
[MLComputeDevice] { get }

    /// Returns a Boolean value
indicating whether two values are equal.
    ///
    /// Equality is the inverse of
inequality. For any values `a` and `b`,
    /// `a == b` implies that `a != b` is
`false`.
    ///
    /// - Parameters:
    ///   - lhs: A value to compare.
    ///   - rhs: Another value to
compare.
    public static func == (a:
MLComputeDevice, b: MLComputeDevice) ->
Bool

    /// Hashes the essential components
of this value by feeding them into the
    /// given hasher.
    ///
    /// Implement this method to conform
to the `Hashable` protocol. The
    /// components used for hashing must
be the same as the components compared
    /// in your type's `==` operator
implementation. Call `hasher.combine(_:)`
    /// with each of these components.
```

```
    ///
    /// - Important: In your
implementation of `hash(into:)`,
    ///   don't call `finalize()` on the
`hasher` instance provided,
    ///   or replace it with a different
instance.
    ///   Doing so may become a compile-
time error in the future.
    ///
    /// - Parameter hasher: The hasher to
use when combining the components
    ///   of this instance.
    public func hash(into hasher: inout
Hasher)

    /// The hash value.
    ///
    /// Hash values are not guaranteed to
be equal across different executions of
    /// your program. Do not save hash
values to use during a future execution.
    ///
    /// - Important: `hashValue` is
deprecated as a `Hashable` requirement.
To
    ///   conform to `Hashable`,
implement the `hash(into:)` requirement
instead.
    ///   The compiler provides an
implementation for `hashValue` for you.
    public var hashValue: Int { get }
}
```

```
/// A class representing the compute plan
of a model.
///
/// The application can use the compute
plan to estimate the necessary cost
/// and resources of the model before
running the predictions.
///
/// ```
/// // Load the compute plan of an ML
Program model.
/// let computePlan = try await
MLComputePlan.load(contentsOf: modelURL,
configuration: configuration)
/// guard case let .program(program) =
computePlan.modelStructure else {
///     fatalError("Unexpected model
type.")
/// }
///  // Get the main function.
/// guard let mainFunction =
program.functions["main"] else {
///     fatalError("Missing main
function.")
/// }
///
/// let operations =
mainFunction.block.operations
/// for operation in operations {
///     // Get the compute device usage
for the operation.
///     let computeDeviceUsage =
```

```
computePlan.deviceUsage(for: operation)
///     // Get the estimated cost of
executing the operation.
///     let estimatedCost =
computePlan.estimatedCost(of: operation)
/// }
/// ```
@available(macOS 14.4, iOS 17.4, watchOS
10.4, tvOS 17.4, *)
public class MLComputePlan {

    /// The anticipated compute devices
that would be used for executing a
layer/operation.
    public struct DeviceUsage : Sendable
{

        /// The compute devices that can
execute the layer/operation.
        public let supported:
[MLComputeDevice]

        /// The compute device that the
framework prefers to execute the
layer/operation.
        public let preferred:
MLComputeDevice
    }

    /// A struct containing information
on the estimated cost of executing a
layer/operation.
    public struct Cost : Sendable {
```

```
        /// The estimated workload of
executing the operation over the total
model evaluation.
        /// The value is between [0.0,
1.0].
        public let weight: Double
    }

    /// The model structure.
    final public let modelStructure:
MLModelStructure

    /// Construct the compute plan of a
model asynchronously given the model
asset.
    ///
    /// - Parameters:
    ///   - asset: The model asset.
    ///   - configuration: The model
configuration.
    public static func load(asset:
MLModelAsset, configuration:
MLModelConfiguration) async throws ->
MLComputePlan

    /// Construct the compute plan of a
model asynchronously given the location
of its on-disk representation.
    ///
    /// - Parameters:
    ///   - url: The on-disk location of
the compiled model (.mlmodelc directory).
```

```
    ///   - configuration: The model
configuration.
    public static func load(contentsOf
url: URL, configuration:
MLModelConfiguration) async throws ->
MLComputePlan

    /// Returns the estimated cost of
executing a MLProgram operation.
    ///
    /// - Parameters:
    ///   - operation: A MLProgram
operation
    /// - Returns: The estimated cost of
executing the operation.
    public func estimatedCost(of
operation:
MLModelStructure.Program.Operation) ->
MLComputePlan.Cost?

    /// Returns the anticipated compute
devices that would be used for executing
a NeuralNetwork layer.
    ///
    /// - Parameters:
    ///   - layer: A NeuralNetwork layer
    /// - Returns: The anticipated
compute devices that would be used for
evaluating the layer or `nil` if the
usage couldn't be determined.
    public func deviceUsage(for layer:
MLModelStructure.NeuralNetwork.Layer) ->
MLComputePlan.DeviceUsage?
```

```
    /// Returns the anticipated compute
devices that would be used for executing
a MLProgram operation.
    ///
    /// - Parameters:
    ///   - operation: A MLProgram
operation
    /// - Returns: The anticipated
compute devices that would be used for
executing the layer or `nil` if the usage
couldn't be determined.
    public func deviceUsage(for
operation:
MLModelStructure.Program.Operation) ->
MLComputePlan.DeviceUsage?
}

/// The compute policy determining what
compute device, or compute devices, to
execute ML workloads on.
@available(macOS 15.0, iOS 18.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
public struct MLComputePolicy : Sendable,
Hashable, CustomStringConvertible {

    /// Hashes the essential components
of this value by feeding them into the
    /// given hasher.
    ///
    /// Implement this method to conform
to the `Hashable` protocol. The
    /// components used for hashing must
```

```
be the same as the components compared
    /// in your type's `==` operator
implementation. Call `hasher.combine(_:)`
    /// with each of these components.
    ///
    /// - Important: In your
implementation of `hash(into:)`,
    ///   don't call `finalize()` on the
`hasher` instance provided,
    ///   or replace it with a different
instance.
    ///   Doing so may become a compile-
time error in the future.
    ///
    /// - Parameter hasher: The hasher to
use when combining the components
    ///   of this instance.
    public func hash(into hasher: inout
Hasher)

    /// Returns a Boolean value
indicating whether two values are equal.
    ///
    /// Equality is the inverse of
inequality. For any values `a` and `b`,
    /// `a == b` implies that `a != b` is
`false`.
    ///
    /// - Parameters:
    ///   - lhs: A value to compare.
    ///   - rhs: Another value to
compare.
    public static func == (a:
```

```
MLComputePolicy, b: MLComputePolicy) ->
Bool

    /// The hash value.
    ///
    /// Hash values are not guaranteed to
be equal across different executions of
    /// your program. Do not save hash
values to use during a future execution.
    ///
    /// - Important: `hashValue` is
deprecated as a `Hashable` requirement.
To
    ///   conform to `Hashable`,
implement the `hash(into:)` requirement
instead.
    ///   The compiler provides an
implementation for `hashValue` for you.
    public var hashValue: Int { get }
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
extension MLComputePolicy {

    /// A textual representation of the
compute policy.
    public var description: String {
get }
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
```

```
extension MLComputePolicy :
CustomReflectable {

    /// The custom mirror for
`MLComputePolicy`.
    public var customMirror: Mirror { get
}
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
extension MLComputePolicy {

    /// Execute ML workloads using the
GPU if available, otherwise falling back
to the CPU.
    public static var cpuAndGPU:
MLComputePolicy { get }

    /// Execute ML workloads using the
CPU.
    public static var cpuOnly:
MLComputePolicy { get }

    /// Creates a new compute policy
using the given compute units.
    public init(_ computeUnits:
MLComputeUnits)
}

/// An enum representing the structure of
a model.
///
```

```
/// ```
/// // Load the model structure.
/// let modelStructure = await try
MLModelStructure.load(contentsOf:
modelURL)
/// switch modelStructure {
/// case .program(let program):
///     // Examine ML Program model.
/// case .neuralNetwork(let
neuralNetwork):
///     // Examine Neural network model
/// case .pipeline(let pipeline)
///     // Examine Pipeline model
/// default:
///     // The model type is something
else.
/// }
/// ```
@available(macOS 14.4, iOS 17.4, watchOS
10.4, tvOS 17.4, *)
public enum MLModelStructure : Sendable {

    /// A struct representing the
structure of an ML Program model.
    public struct Program : Sendable {

        /// A struct representing the
type of a variable in the Program.
        public struct ValueType :
Sendable {
        }

        /// A struct representing the
```

```
value of a variable in the Program.
        public struct Value : Sendable {
        }

        /// A struct representing a named
type in a Program.
        public struct NamedValueType :
Sendable {

            /// The name of the
parameter.
            public let name: String

            /// The type of the
parameter.
            public let type:
MLModelStructure.Program.ValueType
        }

        /// An enum representing a
binding.
        ///
        /// A Binding  is either a
previously defined name of a variable or
a constant value in the Program.
        public enum Binding : Sendable {

            /// A constant value in the
Program.
            case
value(MLModelStructure.Program.Value)

            /// The name of the
```

```
previously defined variable in the
Program.
            case name(String)
        }

        /// A struct representing an
argument in the Program.
        public struct Argument : Sendable
{

            /// The bindings.
            public let bindings:
[MLModelStructure.Program.Binding]
        }

        /// A struct representing a
function in the Program.
        public struct Function : Sendable
{

            /// The inputs to the
function.
            public let inputs:
[MLModelStructure.Program.NamedValueType]

            /// The active block in the
function.
            public let block:
MLModelStructure.Program.Block
        }

        /// A struct representing a block
in the Program.
```

```
        public struct Block : Sendable {

            /// The inputs to the block.
            public let inputs:
[MLModelStructure.Program.NamedValueType]

            /// The output names.
            public let outputNames:
[String]

            /// The operations in the
block.
            public let operations:
[MLModelStructure.Program.Operation]
        }

        /// A struct representing an
Operation in the Program.
        public struct Operation :
Sendable {

            /// The name of the operator,
e.g., "conv", "pool", "softmax", etc.
            public let operatorName:
String

            /// The arguments to the
Operation.
            public let inputs: [String :
MLModelStructure.Program.Argument]

            /// The outputs of the
Operation.
```

```
            public let outputs:
[MLModelStructure.Program.NamedValueType]

            /// Nested blocks for loops
and conditionals, e.g., a conditional
block will have two entries here.
            public let blocks:
[MLModelStructure.Program.Block]
        }

        /// The functions in the program.
        public let functions: [String :
MLModelStructure.Program.Function]
    }

    /// A struct representing the
structure of a NeuralNetwork model..
    public struct NeuralNetwork :
Sendable {

        /// A struct representing the
layer in a NeuralNetwork model.
        public struct Layer : Sendable {

            /// The layer name.
            public let name: String

            /// The type of the layer,
e,g, "elementwise", "pooling", etc.
            public let type: String

            /// The input names.
            public let inputNames:
```

```
[String]

            /// The output names.
            public let outputNames:
[String]
        }

        /// The topologically sorted
layers in the NeuralNetwork.
        public let layers:
[MLModelStructure.NeuralNetwork.Layer]
    }

    /// A struct representing the
structure of a Pipeline model..
    public struct Pipeline : Sendable {

        /// The names of the sub models
in the Pipeline.
        public let subModelNames:
[String]

        /// The structure of sub models
in the Pipeline.
        public let subModels:
[MLModelStructure]
    }

    /// Represents a NeuralNetwork model,
the associated value is the structure of
the NeuralNetwork.
    case
neuralNetwork(MLModelStructure.NeuralNetw
```

```
ork)

    /// Represents a MLProgram model. the
associated value is the structure of the
Program.
    case
program(MLModelStructure.Program)

    /// Represents a Pipeline model, the
associated value is the structure of the
Pipeline.
    case
pipeline(MLModelStructure.Pipeline)

    /// Represents an unsupported model.
    case unsupported

    /// Load the model structure
asynchronously given the location of its
on-disk representation.
    ///
    /// - Parameters:
    ///   - url: The location of its on-
disk representation (.mlmodelc
directory).
    public static func load(contentsOf
url: URL) async throws ->
MLModelStructure

    /// Load the model structure
asynchronously from the model asset.
    ///
    /// - Parameters:
```

```
    ///   - asset: The model asset.
    public static func load(asset:
MLModelAsset) async throws ->
MLModelStructure
}

@available(macOS 14.4, iOS 17.4, watchOS
10.4, tvOS 17.4, *)
public struct MLOptimizationHints :
Equatable, Sendable {

    /// The anticipated frequency of
changing input shapes.
    @available(macOS 14.4, iOS 17.4,
watchOS 10.4, tvOS 17.4, *)
    public enum ReshapeFrequency : Int,
Sendable {

        /// The input shape is expected
to change frequently on each prediction
sent to this loaded model instance. Core
ML will try to minimize the latency
associated with shape changes and avoid
expensive shape-specific optimizations
prior to prediction computation. While
prediction computation may be slower for
each specific shape, switching between
shapes should be faster.
        /// This is the default.
        case frequent

        /// The input shape is expected
to be stable and many/all predictions
```

```
sent to this loaded model instance would
use the same input shapes repeatedly. On
the shape change, Core ML re-optimizes
the internal engine for the new shape if
possible. The re-optimization takes some
time, but the subsequent predictions for
the shape should run faster.
        case infrequent

        /// Creates a new instance with
the specified raw value.
        ///
        /// If there is no value of the
type that corresponds with the specified
raw
        /// value, this initializer
returns `nil`. For example:
        ///
        ///     enum PaperSize: String {
        ///         case A4, A5, Letter,
Legal
        ///     }
        ///
        ///     print(PaperSize(rawValue:
"Legal"))
        ///     // Prints
"Optional("PaperSize.Legal")"
        ///
        ///     print(PaperSize(rawValue:
"Tabloid"))
        ///     // Prints "nil"
        ///
        /// - Parameter rawValue: The raw
```

```
value to use for the new instance.
        public init?(rawValue: Int)

        /// The raw type that can be used
to represent all values of the conforming
        /// type.
        ///
        /// Every distinct value of the
conforming type has a corresponding
unique
        /// value of the `RawValue` type,
but there may be values of the `RawValue`
        /// type that don't have a
corresponding value of the conforming
type.
        @available(iOS 17.4, tvOS 17.4,
watchOS 10.4, macOS 14.4, *)
        public typealias RawValue = Int

        /// The corresponding value of
the raw type.
        ///
        /// A new instance initialized
with `rawValue` will be equivalent to
this
        /// instance. For example:
        ///
        ///     enum PaperSize: String {
        ///         case A4, A5, Letter,
Legal
        ///     }
        ///
        ///     let selectedSize =
```

```
PaperSize.Letter
        ///
print(selectedSize.rawValue)
        ///     // Prints "Letter"
        ///
        ///     print(selectedSize ==
PaperSize(rawValue:
selectedSize.rawValue)!)
        ///     // Prints "true"
        public var rawValue: Int { get }
    }

    /// The optimization strategy for the
model specialization.
    @available(macOS 15.0, iOS 18.0,
watchOS 11.0, tvOS 18.0, visionOS 2.0, *)
    public enum SpecializationStrategy :
Int, Sendable {

        /// The strategy that should work
well for most applications.
        case `default`

        /// Prefer the prediction latency
at the potential cost of specialization
time, memory footprint, and the disk
space usage of specialized artifacts.
        case fastPrediction

        /// Creates a new instance with
the specified raw value.
        ///
        /// If there is no value of the
```

```
type that corresponds with the specified
raw
        /// value, this initializer
returns `nil`. For example:
        ///
        ///     enum PaperSize: String {
        ///         case A4, A5, Letter,
Legal
        ///     }
        ///
        ///     print(PaperSize(rawValue:
"Legal"))
        ///     // Prints
"Optional("PaperSize.Legal")"
        ///
        ///     print(PaperSize(rawValue:
"Tabloid"))
        ///     // Prints "nil"
        ///
        /// - Parameter rawValue: The raw
value to use for the new instance.
        public init?(rawValue: Int)

        /// The raw type that can be used
to represent all values of the conforming
        /// type.
        ///
        /// Every distinct value of the
conforming type has a corresponding
unique
        /// value of the `RawValue` type,
but there may be values of the `RawValue`
        /// type that don't have a
```

```
corresponding value of the conforming
type.
        @available(iOS 18.0, tvOS 18.0,
watchOS 11.0, visionOS 2.0, macOS 15.0,
*)
        public typealias RawValue = Int

        /// The corresponding value of
the raw type.
        ///
        /// A new instance initialized
with `rawValue` will be equivalent to
this
        /// instance. For example:
        ///
        ///     enum PaperSize: String {
        ///         case A4, A5, Letter,
Legal
        ///     }
        ///
        ///     let selectedSize =
PaperSize.Letter
        ///
print(selectedSize.rawValue)
        ///     // Prints "Letter"
        ///
        ///     print(selectedSize ==
PaperSize(rawValue:
selectedSize.rawValue)!)
        ///     // Prints "true"
        public var rawValue: Int { get }
    }
```

```
    /// The anticipated reshape frequency
    ///
    /// CoreML framework needs to reshape
the model with new shapes for models with
flexible input.
    /// Specify the anticipated reshape
frequency (frequent or infrequent), so
that the framework can optimize for
    ///  fast shape switching or fast
prediction on seen shapes.
    ///
    /// The default value is frequent,
which means CoreML tries to switch to new
shapes as fast as possible
    public var reshapeFrequency:
MLOptimizationHints.ReshapeFrequency

    /// Optimization strategy for the
model specialization.
    ///
    /// Core ML segments the model's
compute graph and specializes each
segment for the
    /// target compute device. This
process can affect the model loading time
and the prediction latency.
    ///
    /// Use this option to tailor the
specialization strategy for your model.
    @available(macOS 15.0, iOS 18.0,
watchOS 11.0, tvOS 18.0, visionOS 2.0, *)
    public var specializationStrategy:
MLOptimizationHints.SpecializationStrateg
```

```
y

    /// Construct an MLOptimizationHints
    public init()

    /// Returns a Boolean value
indicating whether two values are equal.
    ///
    /// Equality is the inverse of
inequality. For any values `a` and `b`,
    /// `a == b` implies that `a != b` is
`false`.
    ///
    /// - Parameters:
    ///   - lhs: A value to compare.
    ///   - rhs: Another value to
compare.
    public static func == (a:
MLOptimizationHints, b:
MLOptimizationHints) -> Bool
}

@available(macOS 14.4, iOS 17.4, watchOS
10.4, tvOS 17.4, *)
extension
MLOptimizationHints.ReshapeFrequency :
Equatable {
}

@available(macOS 14.4, iOS 17.4, watchOS
10.4, tvOS 17.4, *)
extension
MLOptimizationHints.ReshapeFrequency :
```

```
Hashable {
}

@available(macOS 14.4, iOS 17.4, watchOS
10.4, tvOS 17.4, *)
extension
MLOptimizationHints.ReshapeFrequency :
RawRepresentable {
}

@available(macOS 15.0, iOS 18.0, watchOS
11.0, tvOS 18.0, visionOS 2.0, *)
extension
MLOptimizationHints.SpecializationStrateg
y : Equatable {
}

@available(macOS 15.0, iOS 18.0, watchOS
11.0, tvOS 18.0, visionOS 2.0, *)
extension
MLOptimizationHints.SpecializationStrateg
y : Hashable {
}

@available(macOS 15.0, iOS 18.0, watchOS
11.0, tvOS 18.0, visionOS 2.0, *)
extension
MLOptimizationHints.SpecializationStrateg
y : RawRepresentable {
}

/// A sendable feature value.
///
```

```
/// This version of feature value is
similar to ``MLFeatureValue`` but it can
be passed across concurrency domains.
/// Once in the target concurrency
domain, you can then convert it to a
``MLFeatureValue``.
@available(macOS 15.0, iOS 18.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
public struct MLSendableFeatureValue :
Equatable, Sendable {

    /// The type of value.
    public var type: MLFeatureType {
get }

    /// A Boolean value indicating
whether the value is missing or
undefined.
    public var isUndefined: Bool { get }

    /// A Boolean value indicating
whether the value is a single number.
    public var isScalar: Bool { get }

    /// A Boolean value indicating
whether the value is a shaped array.
    public var isShapedArray: Bool {
get }

    /// The integer value, if the
contained value is an integer.
    public var integerValue: Int? { get }
```

```
    /// The 16-bit floating-point value,
if the contained value is a 16-bit float.
    @available(macOS 15.0, iOS 18.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
    public var float16Value: Float16? {
get }

    /// The single-precision floating-
point value, if the contained value is a
float.
    public var floatValue: Float? { get }

    /// The double-precision floating-
point value, if the contained value is a
double.
    public var doubleValue: Double? { get
}

    /// The string value, if the
contained value is a string.
    public var stringValue: String? { get
}

    /// The string array value, if the
contained value is an array of string.
    public var stringArrayValue:
[String]? { get }

    /// The string dictionary value, if
the contained value is a dictionary of
strings to numbers.
    public var stringDictionaryValue:
[String : Double]? { get }
```

```
    /// The integer dictionary value, if
the contained value is a dictionary of
integers to numbers.
    public var integerDictionaryValue:
[Int : Double]? { get }

    /// Returns the shaped array value,
if the contained value is a shaped array
of the specified type.
    public func
shapedArrayValue<Scalar>(of type:
Scalar.Type) -> MLShapedArray<Scalar>?
where Scalar : MLShapedArrayScalar

    /// Creates an undefined feature
value of a specific type.
    public init(undefined type:
MLFeatureType)

    /// Creates a feature value
containing an integer.
    public init(_ value: Int)

    /// Creates a feature value
containing an integer.
    public init(_ value: Int32)

    /// Creates a feature value
containing a 16-bit floating-point value.
    @available(macOS 15.0, iOS 18.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
    public init(_ value: Float16)
```

```
    /// Creates a feature value
containing a single-precision floating-
point value.
    public init(_ value: Float)

    /// Creates a feature value
containing a double-precision floating-
point value.
    public init(_ value: Double)

    /// Creates a feature value
containing a string value.
    public init(_ value: String)

    /// Creates a feature value
containing an array of string values.
    public init(_ value: [String])

    /// Creates a feature value
containing a dictionary of strings to
doubles.
    public init(_ value: [String :
Double])

    /// Creates a feature value
containing a dictionary of strings to
integers.
    public init(_ value: [String : Int])

    /// Creates a feature value
containing a dictionary of integers to
doubles.
```

```
    public init(_ value: [Int : Double])

    /// Creates a feature value
containing a dictionary of integers to
integers.
    public init(_ value: [Int : Int])

    /// Creates a feature value
containing a shaped array.
    public init<Scalar>(_ value:
MLShapedArray<Scalar>) where Scalar :
MLShapedArrayScalar

    /// Creates a sendable feature value
from a feature value.
    ///
    /// Some feature values are not
representable as a sendable values. In
those cases this initializer will return
    /// `nil`.
    public init?(_ value: MLFeatureValue)

    /// Returns a Boolean value
indicating whether two values are equal.
    ///
    /// Equality is the inverse of
inequality. For any values `a` and `b`,
    /// `a == b` implies that `a != b` is
`false`.
    ///
    /// - Parameters:
    ///   - lhs: A value to compare.
    ///   - rhs: Another value to
```

```
compare.
    public static func == (a:
MLSendableFeatureValue, b:
MLSendableFeatureValue) -> Bool
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
extension MLSendableFeatureValue :
CustomDebugStringConvertible {

    /// A textual representation of this
instance, suitable for debugging.
    ///
    /// Calling this property directly is
discouraged. Instead, convert an
    /// instance of any type to a string
by using the `String(reflecting:)`
    /// initializer. This initializer
works with any type, and uses the custom
    /// `debugDescription` property for
types that conform to
    /// `CustomDebugStringConvertible`:
    ///
    ///     struct Point:
CustomDebugStringConvertible {
    ///         let x: Int, y: Int
    ///
    ///         var debugDescription:
String {
    ///             return "(\(x), \(y))"
    ///         }
    ///     }
```

```
    ///
    ///     let p = Point(x: 21, y: 30)
    ///     let s = String(reflecting: p)
    ///     print(s)
    ///     // Prints "(21, 30)"
    ///
    /// The conversion of `p` to a string
in the assignment to `s` uses the
    /// `Point` type's `debugDescription`
property.
    public var debugDescription: String {
get }
}

/// MLShapedArray is a N-dimensional
array with non-contiguous strides.
@available(macOS 12.0, iOS 15.0, watchOS
8.0, tvOS 15.0, *)
public struct MLShapedArray<Scalar> :
MLShapedArrayProtocol, @unchecked
Sendable where Scalar :
MLShapedArrayScalar {

    /// The type of the elements of an
array literal.
    public typealias ArrayLiteralElement
= Scalar

    /// A collection representing a
contiguous subrange of this collection's
    /// elements. The subsequence shares
indices with the original collection.
    ///
```

```
    /// The default subsequence type for
collections that don't define their own
    /// is `Slice`.
    public typealias SubSequence =
MLShapedArraySlice<Scalar>

    /// Creates a shaped array with
memory content without copying the bytes.
    ///
    /// - Parameter bytes:       The
buffer pointer
    /// - Parameter shape:       The
shape of the buffer
    /// - Parameter strides:     The
strides of the buffer
    /// - Parameter deallocator:
Specifies the mechanism to free the
indicated buffer, or `.none`.
    public init(bytesNoCopy bytes:
UnsafeRawPointer, shape: [Int], strides:
[Int], deallocator: Data.Deallocator)

    /// Creates a shaped array from Data.
    ///
    /// - Parameter data: The storage of
the scalars.
    /// - Parameter shape: The shape
    /// - Parameter strides: The strides
of the buffer.
    public init(data: Data, shape: [Int],
strides: [Int])

    /// Initialize with a closure.
```

```
    ///
    /// - Parameter shape: The shape
    /// - Parameter initializer: The
initializing closure.
    /// - Parameter ptr: The buffer
pointer to be initialized by the closure.
    /// - Parameter strides: The strides
of the buffer.
    public init(unsafeUninitializedShape
shape: [Int], initializingWith
initializer: (_ ptr: inout
UnsafeMutableBufferPointer<Scalar>, _
strides: [Int]) throws -> Void) rethrows

    /// Creates a scalar.
    ///
    /// A scalar is a MLShapedArray with
rank zero. The `.scalar` property vends
the value.
    ///
    /// - Parameter scalar: The value of
the scalar.
    public init(scalar: Scalar)

    /// Initialize with a sequence and
the shape.
    ///
    /// The length of the sequence must
not be less than the number of scalars in
the shaped array.
    ///
    /// - Parameter scalars: The
initializing sequence.
```

```
    /// - Parameter shape: The shape
    @available(macOS 15.0, iOS 18.0,
watchOS 11.0, tvOS 18.0, *)
    public init<S>(scalars: S, shape:
[Int]) where Scalar == S.Element, S :
Sequence

    /// Creates a new MLShapedArray by
concatenating one or more MLShapedArrays
or MLShapedArraySlices.
    ///
    /// All the source
MLShapedArray(Slice)s must have a same
shape except the specified axis. The
    /// resultant MLShapedArray has the
same shape as inputs except this axis,
which dimension will
    /// be the sum of all the input
dimensions of the axis.
    ///
    /// For example,
    ///
    /// ```swift
    /// //  0 1 2 | 3 4
    /// //  5 6 7 | 8 9
    /// let shapedArray1 =
MLShapedArray<Int32>(scalars: [0, 1, 2,
5, 6, 7], shape: [2, 3])
    /// let shapedArray2 =
MLShapedArray<Int32>(scalars: [3, 4, 8,
9], shape: [2, 2])
    /// let shapedArray3 =
MLShapedArray(concatenating:
```

```
[shapedArray1, shapedArray2], alongAxis:
1)
    /// ```
    ///
    /// The method raises a runtime error
if the shapes of input shaped arrays are
not compatible
    /// for concatenation.
    ///
    /// - Parameter shapedArrays:
Sequence of MLShapedArrays to be
concatenated.
    /// - Parameter alongAxis: Axis index
with which the concatenation will
performed.
    ///                          The value
is wrapped by the dimension of the axis.
For example, -1 is the last axis.
    public init<S>(concatenating
shapedArrays: S, alongAxis: Int) where
Scalar == S.Element.Scalar, S : Sequence,
S.Element : MLShapedArrayProtocol

    /// Creates a new `MLShapedArray`
using a pixel buffer as the backing
storage.
    ///
    /// Use this initializer to create an
`IOSurface` backed `MLShapedArray`, which
can reduce the inference latency by
avoiding the buffer copy.
    ///
    /// The pixel buffer's pixel format
```

```
type must be `OneComponent16Half`. As
such, the scalar type must be `Float16`.
    ///
    /// ```swift
    /// var pixelBuffer: CVPixelBuffer?
    /// let pixelBufferAttributes = [
    ///
kCVPixelBufferIOSurfacePropertiesKey :
[:]
    /// ]
    /// // Pixel buffer's width is the
last dimension of `shape`, which is 4.
    /// // The height is the product of
the rest of the dimensions, which is
    /// // 2 * 3 = 6.
    ///
CVPixelBufferCreate(kCFAllocatorDefault,
    ///                     4, 6,
    ///
kCVPixelFormatType_OneComponent16Half,
    ///
pixelBufferAttributes as CFDictionary,
    ///                     &pixelBuffer)
    ///
    /// let shapedArray =
MLShapedArray<Float16>(mutating:
pixelBuffer!,
    ///
shape: [2, 3, 4])
    /// ```
    ///
    /// When there is one and only one
owner of the shaped array, mutating
```

```
operations modifies the underlying pixel
buffer.
    ///
    /// ```swift
    /// var shapedArray =
MLShapedArray<Float16>(mutating:
pixelBuffer, shape: [1])
    /// shapedArray[scalarAt: 0] = 42
    /// // The pixel buffer now has 42 in
its frame buffer.
    /// ```
    ///
    /// It follows the value semantics.
The mutation doesn't affect the copy.
    ///
    /// ```swift
    /// var array1 =
MLShapedArray<Float16>(mutating:
pixelBuffer, shape: [1])
    /// array1[scalarAt: 0] = 0 //
pixelBuffer is mutated.
    /// let array2 = array1
    /// array1[scalarAt: 0] = 42 // Copy-
on-Write
    ///
    /// assert(array1[scalarAt: 0] == 42)
    /// assert(array2[scalarAt: 0] == 0)
    /// ```
    ///
    /// It is undefined behavior to
mutate the pixel buffer directly without
using
`.withMutablePixelBufferIfAvailable`.
```

```
    ///
    /// - Parameters:
    ///   - pixelBuffer: The backing
pixel buffer. It must be backed by
`IOSurface`.
    ///   - shape: The shape of the
MLShapedArray. The last dimension of
`shape` must match the pixel buffer's
width. The product of the rest of the
dimensions must match the height.
    @available(macOS 15.0, iOS 18.0,
watchOS 11.0, tvOS 18.0, *)
    public init(mutating pixelBuffer:
CVPixelBuffer, shape: [Int])

    @available(macOS 15.0, iOS 18.0,
watchOS 11.0, tvOS 18.0, *)
    public init(_ multiArray:
MLMultiArray)

    /// The shape of the array.
    ///
    /// For example, 2 x 3 matrix may be
represented as a shaped array with the
shape of `[2, 3]`.
    public var shape: [Int] { get }

    /// The strides of the underlying
buffer.
    ///
    /// The strides defines where each
scalar is placed in the memory. With
strides `[s1, s2, .. sn]`,
```

```
    /// the element at `[x1, x2, .., xn]`
is located in `s1 * x1 + s2 * x2 + ... +
sn * xn`-th
    /// position in the memory.
    ///
    /// Use the property to access the
buffer provided through
Unsafe(Mutable)BufferPointer.
    /// There is no need to use the
property in index, subscript, or scalars
array access.
    public var strides: [Int] { get }

    /// Calls a closure with a pointer to
the shaped array's storage.
    ///
    /// The storage contains the scalars
according to the strides in the closure
parameters. For example, the scalar at
    /// `[x1, x2, ... xn]` with strides
`[s1, s2, ... sn]` is stored at
    /// `ptr[x1 * s1 + x2 * s2 + ... + xn
* sn]`
    ///
    /// - Parameter body: A closure with
an `UnsafeBufferPointer` parameter that
points to the
    ///   storage for the shaped array.
If body has a return value, that value is
also used as the
    ///   return value for the
`withUnsafeShapedBufferPointer(_:)`
method. The pointer argument is
```

```
    ///   valid only for the duration of
the method's execution.
    /// - Parameter ptr: The pointer to
the memory buffer of the shaped array.
    /// - Parameter shape: The shape of
the array.
    /// - Parameter strides: The strides
of the array.
    public func
withUnsafeShapedBufferPointer<R>(_ body:
(_ ptr: UnsafeBufferPointer<Scalar>, _
shape: [Int], _ strides: [Int]) throws ->
R) rethrows -> R

    /// Calls the given closure with a
pointer to the array’s mutable storage.
    ///
    /// The storage contains the scalars
according to the strides in the closure
parameters. For example, the scalar at
    /// `[x1, x2, ... xn]` with strides
`[s1, s2, ... sn]` is stored at
    /// `ptr[x1 * s1 + x2 * s2 + ... + xn
* sn]`
    ///
    /// The function may change the
underlying buffer layout. Always use the
`strides` passed in the
    /// closure parameters to access the
buffer because it can be different from
`self.strides`
    /// before invoking this function.
    ///
```

```
    /// - Parameter body: A closure with
an `UnsafeMutableBufferPointer` parameter
that points to
    ///   the storage for the array. If
no such storage exists, it is created. If
body has a return
    ///   value, that value is also used
as the return value for the
    ///
`withUnsafeMutableShapedBufferPointer(_:)
` method. The pointer argument is valid
only for the
    ///   duration of the method's
execution.
    /// - Parameter ptr: The pointer to
the memory buffer of the shaped array.
    /// - Parameter shape: The shape of
the array.
    /// - Parameter strides: The strides
of the array.
    public mutating func
withUnsafeMutableShapedBufferPointer<R>(_
body: (inout
UnsafeMutableBufferPointer<Scalar>, _
shape: [Int], _ strides: [Int]) throws ->
R) rethrows -> R

    /// Reads the underlying pixel
buffer.
    ///
    /// Use this method to read the
contents of the underlying pixel buffer.
The pixel buffer is read only. Do not
```

```
write to it.
    ///
    /// ```swift
    /// let array =
MLShapedArray<Float16>(mutating:
pixelBuffer, shape: [2, 3])
    /// array.withPixelBuffer
{ backingPixelBuffer in
    ///       // read backingPixelBuffer
here.
    /// }
    /// ```
    /// - Parameters:
    ///   - body: The closure to run with
the pixel buffer.
    ///   - pixelBuffer: The backing
pixel buffer.
    ///
    /// - Returns:
    ///   The value returned from body,
unless the shaped array doesn't use a
pixel buffer backing, in which case the
method ignores body and returns nil.
    @available(macOS 15.0, iOS 18.0,
watchOS 11.0, tvOS 18.0, *)
    public func
withPixelBufferIfAvailable<R>(_ body: (_
pixelBuffer: CVPixelBuffer) throws -> R)
rethrows -> R?

    /// Writes to the underlying pixel
buffer.
    ///
```

```
    /// Use this method to writes the
contents of the underlying pixel buffer.
    ///
    /// ```swift
    /// let array =
MLShapedArray<Float16>(mutating:
pixelBuffer, shape: [2, 3])
    /// array.withMutablePixelBuffer
{ backingPixelBuffer in
    ///       // write backingPixelBuffer
here.
    /// }
    /// ```
    /// - Parameters:
    ///   - body: The closure to run with
the pixel buffer.
    ///   - pixelBuffer: The backing
pixel buffer. `nil` if the shaped array
doesn't use pixel buffer backing.
    @available(macOS 15.0, iOS 18.0,
watchOS 11.0, tvOS 18.0, *)
    public mutating func
withMutablePixelBufferIfAvailable<R>(_
body: (_ pixelBuffer: CVPixelBuffer)
throws -> R) rethrows -> R?

    /// A type representing the
sequence's elements.
    @available(iOS 15.0, tvOS 15.0,
watchOS 8.0, macOS 12.0, *)
    public typealias Element =
MLShapedArraySlice<Scalar>
```

```
    /// A type that represents a position
in the collection.
    ///
    /// Valid indices consist of the
position of every element and a
    /// "past the end" position that's
not valid for use as a subscript
    /// argument.
    @available(iOS 15.0, tvOS 15.0,
watchOS 8.0, macOS 12.0, *)
    public typealias Index = Int

    /// A type that represents the
indices that are valid for subscripting
the
    /// collection, in ascending order.
    @available(iOS 15.0, tvOS 15.0,
watchOS 8.0, macOS 12.0, *)
    public typealias Indices = Range<Int>

    /// A type that provides the
collection's iteration interface and
    /// encapsulates its iteration state.
    ///
    /// By default, a collection conforms
to the `Sequence` protocol by
    /// supplying `IndexingIterator` as
its associated `Iterator`
    /// type.
    @available(iOS 15.0, tvOS 15.0,
watchOS 8.0, macOS 12.0, *)
    public typealias Iterator =
IndexingIterator<MLShapedArray<Scalar>>
```

```
}

/// Encodable support
@available(macOS 13.0, iOS 16.0, watchOS
9.0, tvOS 16.0, *)
extension MLShapedArray : Encodable where
Scalar : Encodable {

    /// Encodes the shaped array with the
encoder.
    ///
    ///  - Parameter encoder: The
encoder.
    public func encode(to encoder: any
Encoder) throws
}

/// Decodable support
@available(macOS 13.0, iOS 16.0, watchOS
9.0, tvOS 16.0, *)
extension MLShapedArray : Decodable where
Scalar : Decodable {

    /// Decodes a shaped array from the
decoder.
    ///
    /// The framework may choose to use
different strides from the coded ones.
    ///
    ///  - Parameter decoder: The
decoder.
    public init(from decoder: any
Decoder) throws
```

```
}

@available(macOS 12.0, iOS 15.0, watchOS
8.0, tvOS 15.0, *)
extension MLShapedArray :
CustomStringConvertible {

    /// A textual representation of this
`ShapedArray`.
    public var description: String {
get }
}

/// Implement MutableCollection
@available(macOS 12.0, iOS 15.0, watchOS
8.0, tvOS 15.0, *)
extension MLShapedArray {

    /// The indices that are valid for
subscripting the collection, in ascending
    /// order.
    ///
    /// A collection's `indices` property
can hold a strong reference to the
    /// collection itself, causing the
collection to be nonuniquely referenced.
    /// If you mutate the collection
while iterating over its indices, a
strong
    /// reference can result in an
unexpected copy of the collection. To
avoid
    /// the unexpected copy, use the
```

```
`index(after:)` method starting with
    /// `startIndex` to produce indices
instead.
    ///
    ///     var c =
MyFancyCollection([10, 20, 30, 40, 50])
    ///     var i = c.startIndex
    ///     while i != c.endIndex {
    ///         c[i] /= 5
    ///         i = c.index(after: i)
    ///     }
    ///     // c == MyFancyCollection([2,
4, 6, 8, 10])
    public var indices: Range<Int> {
get }

    /// The position of the first element
in a nonempty collection.
    ///
    /// If the collection is empty,
`startIndex` is equal to `endIndex`.
    public var startIndex: Int { get }

    /// The collection's "past the end"
position---that is, the position one
    /// greater than the last valid
subscript argument.
    ///
    /// When you need a range that
includes the last element of a
collection, use
    /// the half-open range operator
(`..<`) with `endIndex`. The `..<`
```

```
operator
    /// creates a range that doesn't
include the upper bound, so it's always
    /// safe to use with `endIndex`. For
example:
    ///
    ///     let numbers = [10, 20, 30,
40, 50]
    ///     if let index =
numbers.firstIndex(of: 30) {
    ///         print(numbers[index ..<
numbers.endIndex])
    ///     }
    ///     // Prints "[30, 40, 50]"
    ///
    /// If the collection is empty,
`endIndex` is equal to `startIndex`.
    public var endIndex: Int { get }

    /// Accesses the element at the
specified position.
    ///
    /// For example, you can replace an
element of an array by using its
    /// subscript.
    ///
    ///     var streets = ["Adams",
"Bryant", "Channing", "Douglas",
"Evarts"]
    ///     streets[1] = "Butler"
    ///     print(streets[1])
    ///     // Prints "Butler"
    ///
```

```
    /// You can subscript a collection
with any valid index other than the
    /// collection's end index. The end
index refers to the position one
    /// past the last element of a
collection, so it doesn't correspond with
an
    /// element.
    ///
    /// - Parameter position: The
position of the element to access.
`position`
    ///   must be a valid index of the
collection that is not equal to the
    ///   `endIndex` property.
    ///
    /// - Complexity: O(1)
    public subscript(index: Int) ->
MLShapedArraySlice<Scalar>

    /// A slice of the shaped array for
the selected leading axes.
    ///
    /// The slice has a rank of
`self.rank - indices.count`. For example,
given a shaped array `m`
    /// with the shape being `3 x 3`,
`m[[1]]` returns a slice of shape `[3]`
with the contents
    /// labeld as `x` below.
    ///
    /// ```
    ///  0  0  0
```

```
    ///  x  x  x
    ///  0  0  0
    /// ```
    ///
    /// - Parameter indices: The indices to slice the array.
    public subscript<C>(indices: C) -> MLShapedArraySlice<Scalar> where C : Collection, C.Element == Int

    /// The scalar value at the indices.
    ///
    /// The subscript is the scalar value of the slice pointed by the specified indices. It raises
    /// a runtime error if the specified indices is a slice with `.isScalar == false`.
    ///
    /// - Parameter indices: The indices to a scalar
    public subscript<C>(scalarAt indices: C) -> Scalar where C : Collection, C.Element == Int

    /// A slice of the shaped array for the specified ranges.
    ///
    /// The slice has the same rank as the source. For example, given a shaped array `m` with the
    /// shape being `[3, 3]`, `m[1...2, 0...1]` returns a slice of shape `[2, 2]`
```

```
with the contents
    /// labeld as `x` below.
    ///
    /// ```
    ///  0  0  0
    ///  x  x  0
    ///  x  x  0
    /// ```
    public subscript<C>(sliceRanges: C)
-> MLShapedArraySlice<Scalar> where C :
Collection, C.Element == Range<Int>
}

/// Convenient initializers
@available(macOS 12.0, iOS 15.0, watchOS
8.0, tvOS 15.0, *)
extension MLShapedArray {

    /// Creates a shaped array from Data
using default strides
    ///
    /// - Parameter data: The storage of
the scalars.
    /// - Parameter shape: The shape
    public init(data: Data, shape: [Int])
}

/// Reshaping methods
@available(macOS 15.0, iOS 18.0, watchOS
11.0, tvOS 18.0, *)
extension MLShapedArray {

    /// Returns a new reshaped shaped
```

```
array.
    ///
    /// The reshaped array gets scalars
of the original array in first-major
    /// order. Therefore, the initializer
is semantically equivalent to:
    ///
    /// ```swift
    /// let reshaped =
MLShapedArray(scalars: original.scalars,
shape: newShape)
    /// ```
    ///
    /// Usage example:
    ///
    /// ```swift
    /// let original =
MLShapedArray<Int32>(scalars: 0...,
shape: [4])
    /// let reshaped =
original.reshaping(to: [1, 2, 2])
    /// ```
    ///
    /// A scalar can be reshaped to and
from a shape where the product of
    /// dimensions is one.
    ///
    /// The method raises a runtime error
if the product of dimensions in the new
    /// shape is different from the
current one.
    ///
    /// - Parameter newShape: The new
```

```
shape after reshaping.
    public func reshaped(to newShape:
[Int]) -> MLShapedArray<Scalar>

    /// Returns a new squeezed shaped
array.
    ///
    /// The new shape removes 1s in the
original shape.
    ///
    /// ```swift
    /// let original =
MLShapedArray<Int32>(scalars: 0...,
shape: [1, 2, 1, 2])
    /// let squeezed =
original.squeezingShape()
    /// squeezed.shape // [2, 2]
    /// ```
    ///
    /// When all the dimensions of the
original shape is one, the resultant
    /// shaped array is a scalar.
    ///
    /// ```swift
    /// let original =
MLShapedArray<Int32>(scalars: 42, shape:
[1, 1])
    /// let squeezed =
original.squeezingShape()
    /// squeezed.scalar // 42
    /// ```
    ///
    public func squeezingShape() ->
```

```
MLShapedArray<Scalar>

    /// Returns a new shaped array with
expanded dimensions.
    ///
    /// The shape of the new
`MLShapedArray` gets a new dimension of 1
at the specified axis index.
    ///
    /// ```swift
    /// let original =
MLShapedArray<Int32>(scalars: 0...,
shape: [2, 3])
    /// let expanded =
original.expandedingShape(at: 0)
    /// expanded.shape // [1, 2, 3]
    /// ```
    ///
    /// - Parameter dimension: The index
into the current shape where dimension of
1 will be inserted.
    public func expandingShape(at axis:
Int) -> MLShapedArray<Scalar>

    /// Returns a new transposed shaped
array.
    ///
    /// This is equivalent to
`transposed(permutation:)` where
`permutation:` parameter is
    /// `[shape.count-1,
shape.count-2, ..., 0]`, which reverses
the shape.
```

```
    ///
    /// ```swift
    /// let original =
MLShapedArray<Int32>(scalars: 0...,
shape: [1, 2, 3])
    /// let transposed =
original.transposed()
    /// transposed.shape // [3, 2, 1]
    /// ```
    public func transposed() ->
MLShapedArray<Scalar>

    /// Returns a transposed shaped array
using a custom permutation.
    ///
    /// Use this method to convert, for
example, the image data layout from `[C,
H, W]` to `[C, W, H]`, where `C` is
channel, `W` is width, and `H` is height.
    ///
    /// ```swift
    /// // The source tensor has 3
channels, 128 x 64 image in [C, H, W]
layout.
    /// let imageCHW =
MLShapedArray<Int32>(scalars:
pixelValues,
    ///
shape: [3, 64, 128])
    /// let imageCWH =
imageCHW.transposed(permutation: [0, 2,
1])
    /// imageCHW.shape // [3, 64, 128]
```

```
    /// imageCWH.shape // [3, 128, 64]
    /// ```
    ///
    /// The shape (`shape_out`) is
transposed from the input shape
(`shape_in`) as follows.
    ///
    /// ```swift
    /// shape_out[i]
    ///   == permutation.map
{ shape_in[$0] }
    /// ```
    ///
    /// The scalar value of the output
shaped array (`array_out`) is related to
the input shaped array (`array_in`) as
follows.
    ///
    /// ```swift
    /// array_out(scalarAt:
permutation.map { indices[$0] })
    ///   == array_in[scalarAt: indices]]
    /// ```
    /// - Parameter permutation: The
permutation of source axis index.
    public func transposed(permutation
axes: [Int]) -> MLShapedArray<Scalar>
}

/// Buffer Layout methods
@available(macOS 15.0, iOS 18.0, watchOS
11.0, tvOS 18.0, visionOS 2.0, *)
extension MLShapedArray {
```

```
    /// Returns a copy with the specified
buffer layout.
    ///
    /// The returned shaped array will
have `.strides` property according to the
requested
    /// layout.
    ///
    /// The function may return a heap-
memory backed shaped array even if `self`
is backed by a
    /// pixel buffer.
    ///
    /// ```swift
    /// let source =
MLShapedArray<Int32>(scalars: 0...,
shape: [2, 2])
    ///
    /// // Returns a new MLShapedArray
with the specified strides.
    /// _ =
source.changingLayout(to: .custom(strides
: [4, 1]))
    ///
    /// // Returns a new MLShapedArray
with the first-major contiguous layout.
    /// _ =
source.changingLayout(to: .firstMajorCont
iguous)
    ///
    /// // Returns a new MLShapedArray
with the last-major contiguous layout.
```

```
    /// _ =
source.changingLayout(to: .lastMajorConti
guous)
    /// ```
    ///
    /// The
`withUnsafeShapedBufferPointer` function
provides read-only access to the
underlying
    /// buffer of the layout.
    ///
    /// The
`withUnsafeMutableShapedBufferPointer(bod
y:)` function may provide a buffer of
    /// different layout due to copy-on-
write. Use
    ///
`withUnsafeMutableShapedBufferPointer(buf
ferLayout:body:)` if you need a specific
buffer
    /// layout.
    ///
    /// It raises a precondition error if
the custom strides and the shape have
different ranks.
    ///
    /// - Parameters:
    ///   - bufferLayout: The desired
buffer layout.
    public func changingLayout(to
bufferLayout: MLShapedArrayBufferLayout)
-> MLShapedArray<Scalar>
```

```
    /// Calls the given closure with a
pointer to the array’s mutable storage
that has a
    /// specified buffer layout.
    ///
    /// The storage contains the scalars
according to buffer layout parameter. For
example, the
    /// scalar at `[x1, x2, ... xn]` with
strides `[s1, s2, ... sn]` is stored at
`ptr[x1 * s1 +
    /// x2 * s2 + ... + xn * sn]`
    ///
    /// Unlike
`withUnsafeMutableShapedBufferPointer(:)`
, this function allows the caller to
    /// specify the buffer layout.
    ///
    /// ```swift
    /// // Initialize 2x3 shaped array
using an external data that lays out
scalars in
    /// // last-major order.
    /// let data: [Int32] = [0, 3,
    ///                      1, 4,
    ///                      2, 5]
    /// var array =
MLShapedArray<Int32>(repeating: 0, shape:
[2, 3])
    ///
array.withUnsafeMutableShapedBufferPointe
r(using: .lastMajorContiguous)
{ destinationPtr, _, _ in
```

```
    ///
data.withUnsafeBufferPointer()
{ sourcePtr in
    ///
destinationPtr.update(from: sourcePtr)
    ///     }
    /// }
    ///
    /// print(array.strides) //  [1, 2]
    /// print(array.scalars) // [0, 1, 2,
3, 4, 5])
    /// ```
    public mutating func
withUnsafeMutableShapedBufferPointer<R>(u
sing bufferLayout:
MLShapedArrayBufferLayout, _ body: (_
ptr: inout
UnsafeMutableBufferPointer<Scalar>, _
shape: [Int], _ strides: [Int]) throws ->
R) rethrows -> R
}

@available(macOS 14.0, iOS 17.0, watchOS
10.0, tvOS 17.0, *)
extension MLShapedArray : Equatable where
Scalar : Equatable {

    /// Returns a Boolean value
indicating whether two values are equal.
    ///
    /// Equality is the inverse of
inequality. For any values `a` and `b`,
    /// `a == b` implies that `a != b` is
```

```
`false`.
    ///
    /// - Parameters:
    ///   - lhs: A value to compare.
    ///   - rhs: Another value to
compare.
    public static func == (lhs:
MLShapedArray<Scalar>, rhs:
MLShapedArray<Scalar>) -> Bool
}

/// Buffer layout enum
@available(macOS 15.0, iOS 18.0, watchOS
11.0, tvOS 18.0, visionOS 2.0, *)
public enum MLShapedArrayBufferLayout :
Sendable {

    /// Layout scalars contiguously in
first-major order.
    case firstMajorContiguous

    /// Layout scalars contiguously in
last-major order.
    case lastMajorContiguous

    /// Layout scalars according to the
strides.
    case strides([Int])
}

/// Protocol for the shaped array and the
slice of shaped array.
@available(macOS 12.0, iOS 15.0, watchOS
```

```
8.0, tvOS 15.0, *)
public protocol
MLShapedArrayProtocol<Scalar> :
ExpressibleByArrayLiteral,
MutableCollection, RandomAccessCollection
where Self.Index == Int {

    /// Scalar data type such as Int32
and Float32.
    associatedtype Scalar :
MLShapedArrayScalar

    /// Creates a shaped array with
memory content without copying the bytes.
    ///
    /// - Parameter bytes:       The
buffer pointer
    /// - Parameter shape:       The
shape of the buffer
    /// - Parameter strides:     The
strides of the buffer
    /// - Parameter deallocator:
Specifies the mechanism to free the
indicated buffer, or `.none`.
    init(bytesNoCopy bytes:
UnsafeRawPointer, shape: [Int], strides:
[Int], deallocator: Data.Deallocator)

    /// Creates a shaped array with the
specified shape, then calls the given
closure with a buffer covering the
array's uninitialized memory.
    ///
```

```
    /// - Parameter shape: The shape of
the array to allocate.
    /// - Parameter initializer: The
initializing closure.
    /// - Parameter ptr: The buffer
pointer to be initialized by the closure.
    /// - Parameter strides: The strides
of the buffer.
    init(unsafeUninitializedShape shape:
[Int], initializingWith initializer: (_
ptr: inout
UnsafeMutableBufferPointer<Self.Scalar>,
_ strides: [Int]) throws -> Void)
rethrows

    /// The shape of the array.
    ///
    /// For example, 2 x 3 matrix may be
represented as a shaped array with the
shape of `[2, 3]`.
    var shape: [Int] { get }

    /// The strides of the underlying
buffer.
    ///
    /// The strides defines where each
scalar is placed in the memory. With
strides `[s1, s2, .. sn]`,
    /// the element at `[x1, x2, .., xn]`
is located in `s1 * x1 + s2 * x2 + ... +
sn * xn`-th
    /// position in the memory.
    ///
```

```
    /// Use the property to access the
buffer provided through
Unsafe(Mutable)BufferPointer.
    /// There is no need to use the
property in index, subscript, or scalars
array access.
    var strides: [Int] { get }

    /// A slice of the shaped array for
the specified ranges.
    ///
    /// The slice has the same rank as
the source. For example, given a shaped
array `m` with the
    /// shape being `[3, 3]`, `m[1...2,
0...1]` returns a slice of shape `[2, 2]`
with the contents
    /// labeld as `x` below.
    ///
    /// ```
    ///  0  0  0
    ///  x  x  0
    ///  x  x  0
    /// ```
    subscript<C>(sliceRanges: C) ->
MLShapedArraySlice<Self.Scalar> where C :
Collection, C.Element == Range<Int> { get
set }

    /// A slice of the shaped array for
the selected leading axes.
    ///
    /// The slice has a rank of
```

```
`self.rank - indices.count`. For example,
given a shaped array `m`
    /// with the shape being `3 x 3`,
`m[[1]]` returns a slice of shape `[3]`
with the contents
    /// labeld as `x` below.
    ///
    /// ```
    ///  0  0  0
    ///  x  x  x
    ///  0  0  0
    /// ```
    ///
    /// - Parameter indices: The indices
to slice the array.
    subscript<C>(indices: C) ->
MLShapedArraySlice<Self.Scalar> where C :
Collection, C.Element == Int { get set }

    /// The scalar value at the indices.
    ///
    /// The subscript is the scalar value
of the slice pointed by the specified
indices. It raises
    /// a runtime error if the specified
indices is a slice with `.isScalar ==
false`.
    ///
    /// - Parameter indices: The indices
to a scalar
    subscript<C>(scalarAt indices: C) ->
Self.Scalar where C : Collection,
C.Element == Int { get set }
```

```
    /// Calls a closure with a pointer to
the shaped array's storage.
    ///
    /// The storage contains the scalars
according to the strides in the closure
parameters. For example, the scalar at
    /// `[x1, x2, ... xn]` with strides
`[s1, s2, ... sn]` is stored at
    /// `ptr[x1 * s1 + x2 * s2 + ... + xn
* sn]`
    ///
    /// - Parameter body: A closure with
an `UnsafeBufferPointer` parameter that
points to the
    ///   storage for the shaped array.
If body has a return value, that value is
also used as the
    ///   return value for the
`withUnsafeShapedBufferPointer(_:)`
method. The pointer argument is
    ///   valid only for the duration of
the method's execution.
    /// - Parameter ptr: The pointer to
the memory buffer of the shaped array.
    /// - Parameter shape: The shape of
the array.
    /// - Parameter strides: The strides
of the array.
    func
withUnsafeShapedBufferPointer<R>(_ body:
(_ ptr: UnsafeBufferPointer<Self.Scalar>,
_ shape: [Int], _ strides: [Int]) throws
```

```
-> R) rethrows -> R

    /// Calls the given closure with a
pointer to the array's mutable storage.
    ///
    /// The storage contains the scalars
according to the strides in the closure
parameters. For example, the scalar at
    /// `[x1, x2, ... xn]` with strides
`[s1, s2, ... sn]` is stored at
    /// `ptr[x1 * s1 + x2 * s2 + ... + xn
* sn]`
    ///
    /// The function may change the
underlying buffer layout. Always use the
`strides` passed in the
    /// closure parameters to access the
buffer because it can be different from
`self.strides`
    /// before invoking this function.
    ///
    /// - Parameter body: A closure with
an `UnsafeMutableBufferPointer` parameter
that points to
    ///   the storage for the array. If
no such storage exists, it is created. If
body has a return
    ///   value, that value is also used
as the return value for the
    ///
`withUnsafeMutableShapedBufferPointer(_:)
` method. The pointer argument is valid
only for the
```

```
    ///   duration of the method's
execution.
    /// - Parameter ptr: The pointer to
the memory buffer of the shaped array.
    /// - Parameter shape: The shape of
the array.
    /// - Parameter strides: The strides
of the array.
    mutating func
withUnsafeMutableShapedBufferPointer<R>(_
body: (_ ptr: inout
UnsafeMutableBufferPointer<Self.Scalar>,
_ shape: [Int], _ strides: [Int]) throws
-> R) rethrows -> R
}

@available(macOS 12.0, iOS 15.0, watchOS
8.0, tvOS 15.0, *)
extension MLShapedArrayProtocol {

    /// A slice of the shaped array for
the specified ranges.
    ///
    /// When the range expression array
is shorter than the rank, the remaining
axes are fully
    /// selected. For example, given a
shaped array `m` of shape `[2, 3]`,
`m[[0..<1]]`
    /// results in a slice with shape
`[1, 3]`.
    public
subscript<C>(partialSliceRanges: C) ->
```

```
MLShapedArraySlice<Self.Scalar> where C :
Collection, C.Element == any
MLShapedArrayRangeExpression

    /// A slice of the shaped array for
the specified ranges.
    ///
    /// This overrides the method from
Collection.
    public subscript(sliceRange:
Range<Int>) ->
MLShapedArraySlice<Self.Scalar>

    /// A slice of the shaped array for
the specified ranges.
    ///
    /// When the range expression array
is shorter than the rank, the remaining
axes are fully
    /// selected. For example, given a
shaped array `m` of shape `[2, 3]`,
`m[0..<1]`
    /// results in a slice with shape
`[1, 3]`.
    public subscript(sliceRanges: any
MLShapedArrayRangeExpression...) ->
MLShapedArraySlice<Self.Scalar>

    /// A slice of the shaped array for
the specified ranges.
    public subscript(sliceRange: any
MLShapedArrayRangeExpression) ->
MLShapedArraySlice<Self.Scalar>
```

```
}

/// Index based slicing extensions
@available(macOS 12.0, iOS 15.0, watchOS
8.0, tvOS 15.0, *)
extension MLShapedArrayProtocol {

    /// A slice of the shaped array for
the selected leading axes.
    ///
    /// The slice has a rank of
`self.rank - indices.count`. For example,
given a shaped array `m`
    /// with the shape being `[3, 3]`,
`m[1]` returns a slice of shape `[3]`
with the contents
    /// labeld as `x` below.
    ///
    /// ```
    ///  0  0  0
    ///  x  x  x
    ///  0  0  0
    /// ```
    public subscript(indices: Int...) ->
MLShapedArraySlice<Self.Scalar>

    /// The scalar value at the indices.
    ///
    /// The subscript returns scalar
value of the slice pointed by the
specified indices. It raises
    /// a runtime error if the specified
indices is not a scalar.
```

```
    ///
    /// - Parameter indices: The indices
to a scalar
    public subscript(scalarAt indices:
Int...) -> Self.Scalar
}

/// UnboundedRange slicing extensions
@available(macOS 12.0, iOS 15.0, watchOS
8.0, tvOS 15.0, *)
extension MLShapedArrayProtocol {

    public subscript(_: (UnboundedRange_)
-> Void) ->
MLShapedArraySlice<Self.Scalar>
}

/// Scalar accessor extensions.
@available(macOS 12.0, iOS 15.0, watchOS
8.0, tvOS 15.0, *)
extension MLShapedArrayProtocol {

    /// True if the shaped array is a
rank-0 primitive value.
    ///
    /// For example, given a shaped array
`m` of shape `[2, 2]`, `m[0, 1]`
represents a scalar
    /// located at row 0 and column 1.
    public var isScalar: Bool { get }

    /// The number of elements in the
first axis
```

```
    public var count: Int { get }

    /// The number of scalars in the
shaped array.
    public var scalarCount: Int { get }

    /// Flatten representation of scalars
in first-major order.
    public var scalars: [Self.Scalar]

    /// Scalar value
    ///
    /// The getter returns `nil` if `!
isScalar`.
    public var scalar: Self.Scalar?
}

/// Filling extension
@available(macOS 12.0, iOS 15.0, watchOS
8.0, tvOS 15.0, *)
extension MLShapedArrayProtocol {

    /// Fills the array with a value.
    ///
    /// - Parameter value: The filling
value
    public mutating func fill(with value:
Self.Scalar)

    /// Fills the array with values in
the collection.
    ///
    /// The values are filled in first-
```

```
major order. When the shaped array's
scalar count is greater
    /// than the collection size, it
reads the collection from the start
repeatedly.
    ///
    /// ```
    /// var m =
MLShapedArray<Int32>(shape: [2, 2])
    /// m.fill(with: [1, 2])  //
m.scalars becomes  [1, 2, 1, 2]
    /// ```
    ///
    /// - Parameter collection: The
collection of values with which the
shaped array will be filled.
    public mutating func fill<C>(with
collection: C) where C : Collection,
Self.Scalar == C.Element
}

/// RandomAccessCollection extensions
@available(macOS 12.0, iOS 15.0, watchOS
8.0, tvOS 15.0, *)
extension MLShapedArrayProtocol {

    public func index(after: Self.Index)
-> Self.Index

    public func index(_ index:
Self.Index, offsetBy distance: Int) ->
Self.Index
```

```
    public subscript(index: Int) ->
MLShapedArraySlice<Self.Scalar>
}

@available(macOS 12.0, iOS 15.0, watchOS
8.0, tvOS 15.0, *)
extension MLShapedArrayProtocol where
Self.Scalar : BinaryFloatingPoint,
Self.Scalar.RawSignificand :
FixedWidthInteger {

    /// Initialize as an identity matrix.
    ///
    /// The initializer creates a shaped
array of shape size x size where the
contents are zeros
    /// except array[scalarAt: x, x],
which are ones.
    ///
    /// - Parameter size: The size
(order) of the matrix
    public init(identityMatrixOfSize
size: Int)

    /// Initialize the shaped array with
random scalar values.
    ///
    /// - Parameter range: The value
range of each scalar
    /// - Parameter shape: The shape of
the shaped array
    public init(randomScalarsIn range:
Range<Self.Scalar>, shape: [Int])
```

```
}

@available(macOS 12.0, iOS 15.0, watchOS
8.0, tvOS 15.0, *)
extension MLShapedArrayProtocol where
Self.Scalar : FixedWidthInteger {

    /// Initialize as an identity matrix.
    ///
    /// The initializer creates a shaped
array of shape size x size where the
contents are zeros
    /// except array[scalarAt: x, x],
which are ones.
    ///
    /// - Parameter size: The size
(order) of the matrix
    public init(identityMatrixOfSize
size: Int)

    /// Initialize the shaped array with
random scalar values.
    ///
    /// - Parameter range: The value
range of each scalar
    /// - Parameter shape: The shape of
the shaped array
    public init(randomScalarsIn range:
Range<Self.Scalar>, shape: [Int])
}

@available(macOS 12.0, iOS 15.0, watchOS
8.0, tvOS 15.0, *)
```

```
extension MLShapedArrayProtocol {

    /// Construct a rank-1 shaped array
with an array literal.
    public init(arrayLiteral elements:
Self.Scalar...)

    /// Initialize with shape.
    ///
    /// The contents are initialized by
repeating the `value`.
    ///
    /// - Parameter value:  The
initializing value.
    /// - Parameter shape: The shape.
    public init(repeating value:
Self.Scalar, shape: [Int])

    /// Initialize with a sequence and
the shape.
    ///
    /// The length of the sequence must
not be less than the number of scalars in
the shaped array.
    ///
    /// - Parameter scalars: The
initializing sequence.
    /// - Parameter shape: The shape
    public init<S>(scalars: S, shape:
[Int]) where S : Sequence, Self.Scalar ==
S.Element

    /// Creates a shaped array with
```

```
memory content without copying the bytes.
    ///
    /// This initializer assumes a flat
buffer in first-major order without
padding, so there is no strides
parameter.
    ///
    /// - Parameter bytes:       The
buffer pointer
    /// - Parameter shape:       The
shape
    /// - Parameter deallocator:
Specifies the mechanism to free the
indicated buffer, or `.none`.
    public init(bytesNoCopy bytes:
UnsafeRawPointer, shape: [Int],
deallocator: Data.Deallocator)
}

@available(macOS 12.0, iOS 15.0, watchOS
8.0, tvOS 15.0, *)
extension MLShapedArrayProtocol {

    /// Initialize by converting a shaped
array of different scalar type.
    ///
    /// Converting a floating number to
an integer uses rounding-towards-zero
method.
    ///
    /// When necessary, the source values
are truncated to fit the destination
type, but the behavior is undefined if
```

```
the
    /// source value is too large, too
small, or otherwise not representable in
the destination type.
    ///
    /// - Parameter source: The source
shaped array.
    public init<T>(converting source: T)
where T : MLShapedArrayProtocol

    /// Creates a new MLShapedArray using
a `MLMultiArray` as a backing storage.
    ///
    /// Use this initializer to access
`MLMultiArray` through `MLShapedArray`
interface.
    ///
    /// Mutating operations trigger copy-
on-write. Non-mutating operations access
the `MLMultiArray`'s backing storage
including the pixel buffer.
    ///
    /// - Parameters:
    ///   - multiArray: The
`MLMultiArray` object.
    ///
    public init(_ multiArray:
MLMultiArray)

    /// Initialize by converting a
MLMultiArray of different scalar type.
    ///
    /// Converting a floating number to
```

```
an integer uses rounding-towards-zero
method.
    ///
    /// When necessary, the source values
are truncated to fit the destination
type, but the behavior is undefined if
the
    /// source value is too large, too
small, or otherwise not representable in
the destination type.
    ///
    /// - Parameter multiArray:
MLMultiArray object
    public init(converting multiArray:
MLMultiArray)
}

/// Declares a set of operations a range
expression (e.g. 0 ..< 10) must implement
in order to
/// specify the slice range of
MLShapedArray.
@available(macOS 12.0, iOS 15.0, watchOS
8.0, tvOS 15.0, *)
public protocol
MLShapedArrayRangeExpression {

    /// Returns Range<Int> for the
dimension.
    ///
    /// For example, when the range
expression specifies `1...` on the axis
with dimension 3, the
```

```
    /// resultant Range<Int> is `1 ..<
3`.
    ///
    /// - Parameter dimension: The
dimension of the axis on which the range
expression is used.
    /// - Returns: The range of the
selected dimension.
    func relative(toShapedArrayAxis
range: Range<Int>) -> Range<Int>
}

/// Declares a set of operations all the
data types of MLShapedArray's element
must implement.
@available(macOS 12.0, iOS 15.0, watchOS
8.0, tvOS 15.0, *)
public protocol MLShapedArrayScalar {

    /// Returns MLMultiArrayDataType enum
corresponding to the data type.
    static var multiArrayDataType:
MLMultiArrayDataType { get }
}

@available(macOS 12.0, iOS 15.0, watchOS
8.0, tvOS 15.0, *)
public struct
MLShapedArraySlice<Scalar> :
MLShapedArrayProtocol, @unchecked
Sendable where Scalar :
MLShapedArrayScalar {
```

```
    /// The type of the elements of an
array literal.
    public typealias ArrayLiteralElement
= Scalar

    /// A collection representing a
contiguous subrange of this collection's
    /// elements. The subsequence shares
indices with the original collection.
    ///
    /// The default subsequence type for
collections that don't define their own
    /// is `Slice`.
    public typealias SubSequence =
MLShapedArraySlice<Scalar>

    /// The shape of the array.
    ///
    /// For example, 2 x 3 matrix may be
represented as a shaped array with the
shape of `[2, 3]`.
    public let shape: [Int]

    /// Creates a shaped array with
memory content without copying the bytes.
    ///
    /// - Parameter bytes:       The
buffer pointer
    /// - Parameter shape:       The
shape of the buffer
    /// - Parameter strides:     The
strides of the buffer
    /// - Parameter deallocator:
```

```
Specifies the mechanism to free the
indicated buffer, or `.none`.
    public init(bytesNoCopy bytes:
UnsafeRawPointer, shape: [Int], strides:
[Int], deallocator: Data.Deallocator)

    /// Initialize with a closure.
    ///
    /// - Parameter shape: The shape
    /// - Parameter initializer: The
initializing closure.
    /// - Parameter ptr: The pointer to
the buffer to be initialized by the
closure.
    /// - Parameter strides: The strides
of the buffer.
    public init(unsafeUninitializedShape
shape: [Int], initializingWith
initializer: (_ ptr: inout
UnsafeMutableBufferPointer<Scalar>, _
strides: [Int]) throws -> Void) rethrows

    /// Creates a scalar.
    ///
    /// A scalar is a MLShapedArray with
rank zero. The `.scalar` property vends
the value.
    ///
    /// - Parameter scalar: The value of
the scalar.
    public init(scalar: Scalar)

    /// Creates a shaped array slice from
```

```
Data.
    ///
    /// - Parameter data: The storage of
the scalars.
    /// - Parameter shape: The shape
    /// - Parameter strides: The strides
of the buffer.
    public init(data: Data, shape: [Int],
strides: [Int])

    /// Initialize with a sequence and
the shape.
    ///
    /// The length of the sequence must
not be less than the number of scalars in
the shaped array.
    ///
    /// - Parameter scalars: The
initializing sequence.
    /// - Parameter shape: The shape
    @available(macOS 15.0, iOS 18.0,
watchOS 11.0, tvOS 18.0, *)
    public init<S>(scalars: S, shape:
[Int]) where Scalar == S.Element, S :
Sequence

    /// Creates a new shaped array by
concatenating one or more shaped arrays.
    ///
    /// All the source
MLShapedArray(Slice)s must have a same
shape except the specified axis. The
    /// resultant MLShapedArray has the
```

```
same shape as inputs except this axis,
which dimension will
    /// be the sum of all the input
dimensions of the axis.
    ///
    /// For example,
    ///
    /// ```swift
    /// //  0 1 2 | 3 4
    /// //  5 6 7 | 8 9
    /// let shapedArray1 =
MLShapedArray<Int32>(scalars: [0, 1, 2,
5, 6, 7], shape: [2, 3])
    /// let shapedArray2 =
MLShapedArray<Int32>(scalars: [3, 4, 8,
9], shape: [2, 2])
    /// let shapedArraySlice =
MLShapedArraySlice(concatenating:
[shapedArray1, shapedArray2], alongAxis:
1)
    /// ```
    ///
    /// The method raises a runtime error
if the shapes of input shaped arrays are
not compatible
    /// for concatenation.
    ///
    /// - Parameter shapedArrays:
Sequence of MLShapedArray(Slice)s to be
concatenated.
    /// - Parameter alongAxis: Axis index
with which the concatenation will
performed.
```

```
    ///                             The value
is wrapped by the dimension of the axis.
For example, -1 is the last axis.
    public init<S>(concatenating
shapedArrays: S, alongAxis: Int) where
Scalar == S.Element.Scalar, S : Sequence,
S.Element : MLShapedArrayProtocol

    /// Creates a new
`MLShapedArraySlice` using a pixel buffer
as the backing storage.
    ///
    /// Use this initializer to create an
IOSurface backed `MLShapedArraySlice`,
which can reduce the inference latency by
avoiding the buffer copy.
    ///
    /// The pixel buffer's pixel format
type must be `OneComponent16Half`. As
such, the scalar type must be `Float16`.
    ///
    /// ```swift
    /// var pixelBuffer: CVPixelBuffer?
    /// let pixelBufferAttributes = [
    ///
kCVPixelBufferIOSurfacePropertiesKey :
[:]
    /// ]
    /// // Pixel buffer's width is the
last dimension of `shape`, which is 4.
    /// // The height is the product of
the rest of the dimensions, which is
    /// // 2 * 3 = 6.
```

```
    ///
CVPixelBufferCreate(kCFAllocatorDefault,
    ///                         4, 6,
    ///
kCVPixelFormatType_OneComponent16Half,
    ///
pixelBufferAttributes as CFDictionary,
    ///                         &pixelBuffer)
    ///
    /// let shapedArray =
MLShapedArraySlice<Float16>(mutating:
pixelBuffer!,
    ///
shape: [2, 3, 4])
    /// ```
    ///
    /// When there is one and only one
owner of the shaped array, mutating
operations modifies the underlying pixel
buffer.
    ///
    /// ```swift
    /// var slice =
MLShapedArraySlice<Float16>(mutating:
pixelBuffer, shape: [1])
    /// slice[scalarAt: 0] = 42
    /// // The pixel buffer now has 42 in
its frame buffer.
    /// ```
    ///
    /// It follows the value semantics.
The mutation doesn't affect the copy.
    ///
```

```
    /// ```swift
    /// var slice1 =
MLShapedArraySlice<Float16>(mutating:
pixelBuffer, shape: [1])
    /// slice1[scalarAt: 0] = 0 //
pixelBuffer is mutated.
    /// let slice2 = slice1
    /// slice1[scalarAt: 0] = 42 // Copy-
on-Write
    ///
    /// assert(slice1[scalarAt: 0] == 42)
    /// assert(slice2[scalarAt: 0] == 0)
    /// ```
    ///
    /// - Parameters:
    ///   - pixelBuffer: The pixel buffer
to be owned by the instance.
    ///   - shape: The shape of the
MLShapedArray. The last dimension of
`shape` must match the pixel buffer's
width. The product of the rest of the
dimensions must match the height.
    @available(macOS 15.0, iOS 18.0,
watchOS 11.0, tvOS 18.0, *)
    public init(mutating pixelBuffer:
CVPixelBuffer, shape: [Int])

    /// Creates a new MLShapedArraySlice
using a `MLMultiArray` as a backing
storage.
    ///
    /// Use this initializer to access
`MLMultiArray` through `MLShapedArray`
```

```
interface.
    ///
    /// Mutating operations trigger copy-
on-write. Non-mutating operations access
the `MLMultiArray`'s backing storage
including the pixel buffer.
    ///
    /// - Parameters:
    ///   - multiArray: The
`MLMultiArray` object.
    ///
    @available(macOS 15.0, iOS 18.0,
watchOS 11.0, tvOS 18.0, *)
    public init(_ multiArray:
MLMultiArray)

    /// The strides of the underlying
buffer.
    ///
    /// The strides defines where each
scalar is placed in the memory. With
strides `[s1, s2, .. sn]`,
    /// the element at `[x1, x2, .., xn]`
is located in `s1 * x1 + s2 * x2 + ... +
sn * xn`-th
    /// position in the memory.
    ///
    /// Use the property to access the
buffer provided through
Unsafe(Mutable)BufferPointer.
    /// There is no need to use the
property in index, subscript, or scalars
array access.
```

```
    public var strides: [Int] { get }

    /// A slice of the shaped array for
the specified ranges.
    ///
    /// The slice has the same rank as
the source. For example, given a shaped
array `m` with the
    /// shape being `[3, 3]`, `m[1...2,
0...1]` returns a slice of shape `[2, 2]`
with the contents
    /// labeld as `x` below.
    ///
    /// ```
    ///  0  0  0
    ///  x  x  0
    ///  x  x  0
    /// ```
    public subscript<C>(sliceRanges: C)
-> MLShapedArraySlice<Scalar> where C :
Collection, C.Element == Range<Int>

    /// Calls a closure with a pointer to
the shaped array's storage.
    ///
    /// The storage contains the scalars
according to the strides in the closure
parameters. For example, the scalar at
    /// `[x1, x2, ... xn]` with strides
`[s1, s2, ... sn]` is stored at
    /// `ptr[x1 * s1 + x2 * s2 + ... + xn
* sn]`
    ///
```

```
    /// - Parameter body: A closure with
an `UnsafeBufferPointer` parameter that
points to the
    ///   storage for the shaped array.
If body has a return value, that value is
also used as the
    ///   return value for the
`withUnsafeShapedBufferPointer(_:)`
method. The pointer argument is
    ///   valid only for the duration of
the method's execution.
    /// - Parameter ptr: The pointer to
the memory buffer of the shaped array.
    /// - Parameter shape: The shape of
the array.
    /// - Parameter strides: The strides
of the array.
    public func
withUnsafeShapedBufferPointer<R>(_ body:
(_ ptr: UnsafeBufferPointer<Scalar>, _
shape: [Int], _ strides: [Int]) throws ->
R) rethrows -> R

    /// Calls the given closure with a
pointer to the array’s mutable storage.
    ///
    /// The storage contains the scalars
according to the strides in the closure
parameters. For example, the scalar at
    /// `[x1, x2, ... xn]` with strides
`[s1, s2, ... sn]` is stored at
    /// `ptr[x1 * s1 + x2 * s2 + ... + xn
* sn]`
```

```
    ///
    /// The function may change the
underlying buffer layout. Always use the
`strides` passed in the
    /// closure parameters to access the
buffer because it can be different from
`self.strides`
    /// before invoking this function.
    ///
    /// - Parameter body: A closure with
an `UnsafeMutableBufferPointer` parameter
that points to
    ///   the storage for the array. If
no such storage exists, it is created. If
body has a return
    ///   value, that value is also used
as the return value for the
    ///
`withUnsafeMutableShapedBufferPointer(_:)
` method. The pointer argument is valid
only for the
    ///   duration of the method's
execution.
    /// - Parameter ptr: The pointer to
the memory buffer of the shaped array.
    /// - Parameter shape: The shape of
the array.
    /// - Parameter strides: The strides
of the array.
    public mutating func
withUnsafeMutableShapedBufferPointer<R>(_
body: (_ ptr: inout
UnsafeMutableBufferPointer<Scalar>, _
```

```
shape: [Int], _ strides: [Int]) throws ->
R) rethrows -> R

    /// A type that represents a position
in the collection.
    ///
    /// Valid indices consist of the
position of every element and a
    /// "past the end" position that's
not valid for use as a subscript
    /// argument.
    @available(iOS 15.0, tvOS 15.0,
watchOS 8.0, macOS 12.0, *)
    public typealias Index = Int

    /// A type that represents the
indices that are valid for subscripting
the
    /// collection, in ascending order.
    @available(iOS 15.0, tvOS 15.0,
watchOS 8.0, macOS 12.0, *)
    public typealias Indices = Range<Int>

    /// A type that provides the
collection's iteration interface and
    /// encapsulates its iteration state.
    ///
    /// By default, a collection conforms
to the `Sequence` protocol by
    /// supplying `IndexingIterator` as
its associated `Iterator`
    /// type.
    @available(iOS 15.0, tvOS 15.0,
```

```
watchOS 8.0, macOS 12.0, *)
    public typealias Iterator =
IndexingIterator<MLShapedArraySlice<Scala
r>>
}

@available(macOS 12.0, iOS 15.0, watchOS
8.0, tvOS 15.0, *)
extension MLShapedArraySlice {

    /// A type representing the
sequence's elements.
    public typealias Element =
MLShapedArraySlice<Scalar>

    /// The position of the first element
in a nonempty collection.
    ///
    /// If the collection is empty,
`startIndex` is equal to `endIndex`.
    public var startIndex: Int { get }

    /// The collection's "past the end"
position---that is, the position one
    /// greater than the last valid
subscript argument.
    ///
    /// When you need a range that
includes the last element of a
collection, use
    /// the half-open range operator
(`..<`) with `endIndex`. The `..<`
operator
```

```
    /// creates a range that doesn't
include the upper bound, so it's always
    /// safe to use with `endIndex`. For
example:
    ///
    ///     let numbers = [10, 20, 30,
40, 50]
    ///     if let index =
numbers.firstIndex(of: 30) {
    ///         print(numbers[index ..<
numbers.endIndex])
    ///     }
    ///     // Prints "[30, 40, 50]"
    ///
    /// If the collection is empty,
`endIndex` is equal to `startIndex`.
    public var endIndex: Int { get }

    /// A slice of the shaped array for
the selected leading axes.
    ///
    /// The slice has a rank of
`self.rank - indices.count`. For example,
given a shaped array `m`
    /// with the shape being `3 x 3`,
`m[[1]]` returns a slice of shape `[3]`
with the contents
    /// labeld as `x` below.
    ///
    /// ```
    ///  0  0  0
    ///  x  x  x
    ///  0  0  0
```

```
    /// ```
    ///
    /// - Parameter indices: The indices
to slice the array.
    public subscript<C>(indices: C) ->
MLShapedArraySlice<Scalar> where C :
Collection, C.Element == Int

    /// The scalar value at the indices.
    ///
    /// The subscript is the scalar value
of the slice pointed by the specified
indices. It raises
    /// a runtime error if the specified
indices is a slice with `.isScalar ==
false`.
    ///
    /// - Parameter indices: The indices
to a scalar
    public subscript<C>(scalarAt indices:
C) -> Scalar where C : Collection,
C.Element == Int
}

/// Convenient initializers
@available(macOS 12.0, iOS 15.0, watchOS
8.0, tvOS 15.0, *)
extension MLShapedArraySlice {

    /// Creates a shaped array slice from
Data using default strides.
    ///
    /// - Parameter data: The storage of
```

```
the scalars.
    /// - Parameter shape: The shape
    public init(data: Data, shape: [Int])
}

/// Reshaping methods
@available(macOS 15.0, iOS 18.0, watchOS
11.0, tvOS 18.0, *)
extension MLShapedArraySlice {

    /// Returns a new reshaped shaped
array.
    ///
    /// The reshaped array gets scalars
of the original array in first-major
    /// order. Therefore, the initializer
is semantically equivalent to:
    ///
    /// ```swift
    /// let reshaped =
MLShapedArray(scalars: original.scalars,
shape: newShape)
    /// ```
    ///
    /// Usage example:
    ///
    /// ```swift
    /// let original =
MLShapedArraySlice<Int32>(scalars: 0...,
shape: [4])
    /// let reshaped =
original.reshaped(to: [1, 2, 2])
    /// ```
```

```
    ///
    /// A scalar can be reshaped to and
from a shape where the product of
    /// dimensions is one.
    ///
    /// The method raises a runtime error
if the product of dimensions in the new
    /// shape is different from the
current one.
    ///
    /// - Parameter newShape: The new
shape after reshaping.
    public func reshaped(to newShape:
[Int]) -> MLShapedArraySlice<Scalar>

    /// Returns a new squeezed shaped
array.
    ///
    /// The new shape removes 1s in the
original shape.
    ///
    /// ```swift
    /// let original =
MLShapedArraySlice<Int32>(scalars: 0...,
shape: [1, 2, 1, 2])
    /// let squeezed =
original.squeezed()
    /// squeezed.shape // [2, 2]
    /// ```
    ///
    /// When all the dimensions of the
original shape is one, the resultant
    /// shaped array is a scalar.
```

```
    ///
    /// ```swift
    /// let original =
MLShapedArray<Int32>(scalars: 42, shape:
[1, 1])
    /// let squeezed =
original.squeezed()
    /// squeezed.scalar // 42
    /// ```
    ///
    public func squeezingShape() ->
MLShapedArraySlice<Scalar>

    /// Returns a new shaped array with
expanded dimensions
    ///
    /// The shape of the new
`MLShapedArraySlice` gets a new dimension
of 1 at the specified axis index.
    ///
    /// ```swift
    /// let original =
MLShapedArraySlice<Int32>(scalars: 0...,
shape: [2, 3])
    /// let expanded =
original.expanded(alongAxis: 0)
    /// expanded.shape // [1, 2, 3]
    /// ```
    ///
    /// - Parameter alongAxis: The index
into the current shape where dimension of
1 will be inserted.
    public func expandingShape(at axis:
```

```
Int) -> MLShapedArraySlice<Scalar>

    /// Returns a new transposed shaped
array
    ///
    /// This is equivalent to
`transposed(permutation:)` where
`permutation:` parameter is
    /// `[shape.count-1,
shape.count-2, ..., 0]`, which reverses
the shape.
    ///
    /// ```swift
    /// let original =
MLShapedArraySlice<Int32>(scalars: 0...,
shape: [1, 2, 3])
    /// let transposed =
original.transposed()
    /// transposed.shape // [3, 2, 1]
    /// ```
    public func transposed() ->
MLShapedArraySlice<Scalar>

    /// Returns a new transposed shaped
array using a custom permutation.
    ///
    /// Use this method to convert, for
example, the image data layout from `[C,
H, W]` to `[C, W, H]`, where `C` is
channel, `W` is width, and `H` is height.
    ///
    /// ```swift
    /// // The source tensor has 3
```

```
channels, 128 x 64 image in [C, H, W]
layout.
    /// let imageCHW =
MLShapedArray<Int32>(scalars:
pixelValues,
    ///
shape: [3, 64, 128])
    /// // Slice for the first two
channels.
    /// let imageSliceCHW =
imageCHW[0..<2]
    ///
    /// // Transpose.
    /// let imageSliceCWH =
imageSliceCHW.transposed(permutation: [0,
2, 1])
    /// imageSliceCHW.shape // [2, 64,
128]
    /// imageSliceCWH.shape // [2, 128,
64]
    /// ```
    ///
    /// The shape (`shape_out`) is
transposed from the input shape
(`shape_in`) as follows.
    ///
    /// ```swift
    /// shape_out[i]
    ///   == permutation.map
{ shape_in[$0] }
    /// ```
    ///
    /// The scalar value of the output
```

```
shaped array (`array_out`) is related to
the input shaped array (`array_in`) as
follows.
    ///
    /// ```swift
    /// array_out(scalarAt:
permutation.map { indices[$0] })
    ///   == array_in[scalarAt: indices]]
    /// ```
    /// - Parameter permutation: The
permutation of source axis index.
    public func transposed(permutation
axes: [Int]) ->
MLShapedArraySlice<Scalar>
}

/// Buffer Layout
@available(macOS 15.0, iOS 18.0, watchOS
11.0, tvOS 18.0, *)
extension MLShapedArraySlice {

    /// Returns a copy with the specified
buffer layout.
    ///
    /// The returned shaped array slice
will have `.strides` property according
to the requested
    /// layout.
    ///
    /// The function may return a heap-
memory backed shaped array even if `self`
is backed by a
    /// pixel buffer.
```

```
    ///
    /// ```swift
    /// let source =
MLShapedArray<Int32>(scalars: 0...,
shape: [4, 4])
    /// let slice = source[1...2,
1...2] // slice.shape == [2, 2]
    ///
    /// // Returns a new
MLShapedArraySlice with the specified
strides.
    /// _ =
slice.changingLayout(to: .custom(strides:
[3, 1]))
    ///
    /// // Returns a new
MLShapedArraySlice with the first-major
contiguous layout.
    /// _ =
source.changingLayout(to: .firstMajorCont
iguous) // strides = [2, 1]
    ///
    /// // Returns a new
MLShapedArraySlice with the last-major
contiguous layout.
    /// _ =
source.changingLayout(to: .lastMajorConti
guous) // strides = [1, 2]
    /// ```
    ///
    /// The
`withUnsafeShapedBufferPointer` function
provides read-only access to the
```

```
underlying
    /// buffer of the layout.
    ///
    /// The
`withUnsafeMutableShapedBufferPointer(bod
y:)` function may provide a buffer of
    /// different layout due to copy-on-
write. Use
    ///
`withUnsafeMutableShapedBufferPointer(buf
ferLayout:body:)` if you need a specific
buffer
    /// layout.
    ///
    /// It raises a precondition error if
the custom strides and the shape have
different ranks.
    ///
    /// - Parameters:
    ///   - bufferLayout: The desired
buffer layout.
    public func changingLayout(to
bufferLayout: MLShapedArrayBufferLayout)
-> MLShapedArraySlice<Scalar>

    /// Calls the given closure with a
pointer to the array's mutable storage
that has a
    /// specified buffer layout.
    ///
    /// The storage contains the scalars
according to buffer layout parameter. For
example, the
```

```
    /// scalar at `[x1, x2, ... xn]` with
strides `[s1, s2, ... sn]` is stored at
`ptr[x1 * s1 +
    /// x2 * s2 + ... + xn * sn]`
    ///
    /// Unlike
`withUnsafeMutableShapedBufferPointer(:)`
, this function allows the caller to
    /// specify the buffer layout.
    ///
    /// ```swift
    /// // Initialize the middle 2x2 of
4x4 shaped array using an external data
that lays out
    /// // scalars in last-major order.
    /// let data: [Int32] = [1, 3,
    ///                      2, 4]
    /// var array =
MLShapedArray<Int32>(repeating: 0, shape:
[4, 4])
    /// array[1...2,
1...2].withUnsafeMutableShapedBufferPoint
er(using: .lastMajorContiguous)
{ destinationPtr, _, _ in
    ///
data.withUnsafeBufferPointer()
{ sourcePtr in
    ///
destinationPtr.update(from: sourcePtr)
    ///     }
    /// }
    ///
    /// // array.scalars == [0, 0, 0, 0,
```

```
    /// //                        0, 1, 2, 0,
    /// //                        0, 3, 4, 0,
    /// //                        0, 0, 0,
0,])
    /// ```
    public mutating func
withUnsafeMutableShapedBufferPointer<R>(u
sing bufferLayout:
MLShapedArrayBufferLayout, _ body: (_
ptr: inout
UnsafeMutableBufferPointer<Scalar>, _
shape: [Int], _ strides: [Int]) throws ->
R) rethrows -> R
}

/// Encodable support
@available(macOS 13.0, iOS 16.0, watchOS
9.0, tvOS 16.0, *)
extension MLShapedArraySlice : Encodable
where Scalar : Encodable {

    /// Encodes the shaped array slice
with the encoder.
    ///
    ///  - Parameter encoder: The
encoder.
    public func encode(to encoder: any
Encoder) throws
}

/// Decodable support
@available(macOS 13.0, iOS 16.0, watchOS
9.0, tvOS 16.0, *)
```

```
extension MLShapedArraySlice : Decodable
where Scalar : Decodable {

    /// Decodes a shaped array slice from
the decoder.
    ///
    /// The framework may choose to use
different strides from the coded ones.
    ///
    ///  - Parameter decoder: The
decoder.
    public init(from decoder: any
Decoder) throws
}

@available(macOS 14.0, iOS 17.0, watchOS
10.0, tvOS 17.0, *)
extension MLShapedArraySlice : Equatable
where Scalar : Equatable {

    /// Returns a Boolean value
indicating whether two values are equal.
    ///
    /// Equality is the inverse of
inequality. For any values `a` and `b`,
    /// `a == b` implies that `a != b` is
`false`.
    ///
    /// - Parameters:
    ///   - lhs: A value to compare.
    ///   - rhs: Another value to
compare.
    public static func == (lhs:
```

```
MLShapedArraySlice<Scalar>, rhs:
MLShapedArraySlice<Scalar>) -> Bool
}

/// A multi-dimensional array of
numerical or Boolean scalars tailored to
ML use cases, containing methods to
perform
/// transformations and mathematical
operations efficiently using a ML compute
device.
@available(macOS 15.0, iOS 18.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
public struct MLTensor : Sendable {
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
extension MLTensor {

    /// Computes logical not on the
tensor's elements.
    prefix public static func .! (rhs:
MLTensor) -> MLTensor
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
extension MLTensor {

    /// Computes element-wise logical AND
where both operands are expected contain
Boolean scalar elements.
```

```
    ///
    /// For example:
    /// ```swift
    /// let x = MLTensor([true, true,
true])
    /// let y = MLTensor([true, false,
true])
    /// x .& y // is [true, false, true]
    /// ```
    public static func .& (lhs: MLTensor,
rhs: MLTensor) -> MLTensor

    /// Computes element-wise logical OR
where both operands are expected contain
Boolean scalar elements.
    ///
    /// For example:
    /// ```swift
    /// let x = MLTensor([true, true,
true])
    /// let y = MLTensor([true, false,
true])
    /// x .| y // is [true, true, true]
    /// ```
    public static func .| (lhs: MLTensor,
rhs: MLTensor) -> MLTensor

    /// Computes element-wise logical XOR
where both operands are expected contain
Boolean scalar elements.
    ///
    /// For example:
    /// ```swift
```

```
    /// let x = MLTensor([true, true,
true])
    /// let y = MLTensor([true, false,
true])
    /// x .^ y // is [false, true, false]
    /// ```
    public static func .^ (lhs: MLTensor,
rhs: MLTensor) -> MLTensor
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
extension MLTensor {

    /// Computes logical AND on elements
across all axes of a tensor where the
scalar type of the tensor is expected to
be Boolean.
    ///
    /// - Parameter keepRank: A Boolean
indicating whether to keep the reduced
axes or not. The default value is
`false`.
    /// - Returns: The reduced Boolean
tensor.
    public func all(keepRank: Bool =
false) -> MLTensor

    /// Computes logical AND on elements
across the specified axes of a tensor
where the scalar type of the tensor is
expected
    /// to be Boolean.
```

```
    ///
    /// - Parameters:
    ///   - axes: The axes to reduce.
    ///   - keepRank: A Boolean
indicating whether to keep the reduced
axes or not. The default value is
`false`.
    /// - Returns: The reduced Boolean
tensor.
    public func all(alongAxes axes:
Int..., keepRank: Bool = false) ->
MLTensor

    /// Computes logical AND on elements
across the specified axes of a tensor
where the scalar type of the tensor is
expected
    /// to be Boolean.
    ///
    /// - Parameters:
    ///   - axes: The axes to reduce.
    ///   - keepRank: A Boolean
indicating whether to keep the reduced
axes or not. The default value is
`false`.
    /// - Returns: The reduced Boolean
tensor.
    public func all(alongAxes axes:
[Int], keepRank: Bool = false) ->
MLTensor

    /// Computes logical OR on elements
across all dimensions of a tensor where
```

```
the scalar type of the tensor is expected
to be Boolean.
    ///
    /// - Parameter keepRank: A Boolean
indicating whether to keep the reduced
axes or not. The default value is
`false`.
    /// - Returns: The reduced tensor.
    public func any(keepRank: Bool =
false) -> MLTensor

    /// Computes logical OR on elements
across the specified axes of a tensor
where the scalar type of the tensor is
expected
    /// to be Boolean.
    ///
    /// - Parameters:
    ///   - axes: The axes to reduce.
    ///   - keepRank: A Boolean
indicating whether to keep the reduced
axes or not. The default value is
`false`.
    /// - Returns: The reduced Boolean
tensor.
    public func any(alongAxes axes:
Int..., keepRank: Bool = false) ->
MLTensor

    /// Computes logical OR on elements
across the specified axes of a tensor
where the scalar type of the tensor is
expected
```

```
    /// to be Boolean.
    ///
    /// - Parameters:
    ///   - axes: The axes to reduce.
    ///   - keepRank: A Boolean
indicating whether to keep the reduced
axes or not. The default value is
`false`.
    /// - Returns: The reduced Boolean
tensor.
    public func any(alongAxes axes:
[Int], keepRank: Bool = false) ->
MLTensor
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
extension MLTensor {

    /// Casts the elements of the tensor
to the scalar type of the given array.
    ///
    /// For example:
    /// ```swift
    /// let x = MLTensor([1, 2, 3],
scalarType: Float.self)
    /// let y = MLTensor([1, 2, 3],
scalarType: Int32.self)
    /// let z = y.cast(like: x)
    /// z.scalarType // is Float
    /// ```
    ///
    /// - Parameter other: The other
```

```
tensor whose scalar type is used for the
cast.
    /// - Returns: A new tensor with its
contents cast to the scalar type of
`other`.
    public func cast(like other:
MLTensor) -> MLTensor

    /// Casts the elements of the tensor
to the given scalar type.
    ///
    /// For example:
    /// ```swift
    /// let x = MLTensor([1, 2, 3],
scalarType: Int32.self)
    /// let y = x.cast(to: Float.self)
    /// y.scalarType // is Float
    /// ```
    ///
    /// - Parameter scalarType: The
destination scalar type.
    /// - Returns: A new tensor with its
contents cast to the given scalar type.
    public func cast<Scalar>(to
scalarType: Scalar.Type) -> MLTensor
where Scalar : MLTensorScalar
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
extension MLTensor {

    /// Removes all dimensions of size 1
```

```
from the shape of the tensor.
    ///
    /// For example:
    /// ```swift
    /// let x = MLTensor(shape: [1, 2,
1], scalars: [1, 2], scalarType:
Float.self)
    /// let y = x.squeezingShape()
    /// y.shape // is [2]
    /// ```
    public func squeezingShape() ->
MLTensor

    /// Removes the specified dimensions
of size 1 from the shape of the tensor.
    ///
    /// For example:
    /// ```swift
    /// let x = MLTensor(shape: [1, 2,
1], scalars: [1, 2], scalarType:
Float.self)
    /// let y = x.squeezingShape(at: 0)
    /// y.shape // is [2, 1]
    /// ```
    ///
    /// - Parameter axes: The axes to
remove if the size is `1`.
    public func squeezingShape(at axes:
Int...) -> MLTensor

    /// Removes the specified dimensions
of size 1 from the shape of the tensor.
    ///
```

```
    /// For example:
    /// ```swift
    /// let x = MLTensor(shape: [1, 2,
1], scalars: [1, 2], scalarType:
Float.self)
    /// let y = x.squeezingShape(at: [0])
    /// y.shape // is [2, 1]
    /// ```
    ///
    /// - Parameter axes: The axes to
remove if the size is `1`.
    public func squeezingShape(at axes:
[Int]) -> MLTensor
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
extension MLTensor {

    /// Returns a shape-expanded tensor
with a dimension of 1 inserted at the
specified shape indices.
    ///
    /// For example:
    /// ```swift
    /// let x = MLTensor(shape: [2],
scalars: [1, 2], scalarType: Float.self)
    /// let y = x.expandingShape(at: 0)
    /// y.shape // is [1, 2]
    /// ```
    ///
    /// - Parameter axes: The axes at
which to expand the shape.
```

```swift
    public func expandingShape(at axes:
Int...) -> MLTensor

    /// Returns a shape-expanded tensor
with a dimension of 1 inserted at the
specified shape indices.
    ///
    /// For example:
    /// ```swift
    /// let x = MLTensor(shape: [2],
scalars: [1, 2], scalarType: Float.self)
    /// let y = x.expandingShape(at: [0])
    /// y.shape // is [1, 2]
    /// ```
    ///
    /// - Parameter axes: The axes at
which to expand the shape.
    public func expandingShape(at axes:
[Int]) -> MLTensor
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
extension MLTensor {

    /// Reshape to the specified shape.
    ///
    /// For example:
    /// ```swift
    /// let x = MLTensor(shape: [2],
scalars: [1, 2], scalarType: Float.self)
    /// let y = x.reshaped(at: [1, 2, 1])
    /// y.shape // is [1, 2, 1]
```

```
    /// ```
    ///
    /// - Parameter newShape: The new
shape of the array. The number of scalars
matches the new shape.
    public func reshaped(to newShape:
[Int]) -> MLTensor

    /// Reshape to a one-dimensional
tensor.
    ///
    /// - Note: Flattening a zero-
dimensional tensor will return a one-
dimensional tensor.
    ///
    /// For example:
    /// ```swift
    /// let x = MLTensor(shape: [2, 2],
scalars: [1, 2, 3, 4], scalarType:
Float.self)
    /// let y = x.flattened()
    /// y.shape // is [4]
    /// ```
    public func flattened() -> MLTensor
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
extension MLTensor {

    /// Permutes the tensor with
dimensions permuted in reverse order.
    ///
```

```
    /// For example:
    /// ```swift
    /// let x = MLTensor(shape: [1, 2,
3], scalars: [1, 2, 3, 4, 5, 6],
scalarType: Float.self)
    /// let y = x.transposed()
    /// y.shape // is [3, 2, 1]
    /// ```
    ///
    /// - Returns: A permuted tensor.
    public func transposed() -> MLTensor

    /// Permutes the dimensions of the
tensor in the specified order.
    ///
    /// For example:
    /// ```swift
    /// let x = MLTensor(shape: [1, 2,
3], scalars: [1, 2, 3, 4, 5, 6],
scalarType: Float.self)
    /// let y = x.transposed(1, 0, 2)
    /// y.shape // is [2, 1, 3]
    /// ```
    ///
    /// - Parameter permutation: An array
of integers defining the permutation,
must be of length `rank` and define
    ///   a valid permutation.
    /// - Returns: A permuted tensor.
    public func transposed(permutation:
Int...) -> MLTensor

    /// Permutes the dimensions of the
```

```
tensor in the specified order.
    ///
    /// For example:
    /// ```swift
    /// let x = MLTensor(shape: [1, 2,
3], scalars: [1, 2, 3, 4, 5, 6],
scalarType: Float.self)
    /// let y = x.transposed(permutation:
[1, 0, 2])
    /// y.shape // is [2, 1, 3]
    /// ```
    ///
    /// - Parameter permutation: An array
of integers defining the permutation,
must be of length `rank` and define
    ///   a valid permutation.
    /// - Returns: A permuted tensor.
    public func transposed(permutation:
[Int]) -> MLTensor
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
extension MLTensor {

    /// Returns a new tensor replacing
values from `other` with the
corresponding element in `self` where the
    /// associated element in `mask` is
`true`.
    ///
    /// For example:
    /// ```swift
```

```
    /// let x = MLTensor([1, 2, 3],
scalarType: Float.self)
    /// let y = MLTensor([4, 5, 6],
scalarType: Float.self)
    /// let mask = MLTensor([false, true,
false])
    /// let z = x.replacing(with: y,
where: mask)
    /// await z.shapedArray(of:
Float.self) // is [1, 5, 3]
    /// ```
    ///
    /// - Parameters:
    ///   - replacement: The replacement
values where `mask` is `true`.
    ///   - mask: The Boolean mask that
determines whether the corresponding
element / row should be taken from `self`
    ///     (if the element in `mask` is
`false`) or `other` (if `true`).
    /// - Returns: A new tensor of the
same shape and type as `self`.
    public func replacing(with
replacement: MLTensor, where mask:
MLTensor) -> MLTensor

    /// Returns a new tensor replacing
the value of `other` with the
corresponding element in `self` where the
    /// associated element in `mask` is
`true`.
    ///
    /// For example:
```

```
    /// ```swift
    /// let x = MLTensor([1, 2, 3],
scalarType: Float.self)
    /// let mask = MLTensor([false, true,
false])
    /// let z = x.replacing(with: 5,
where: mask)
    /// await z.shapedArray(of:
Float.self) // is [1, 5, 3]
    /// ```
    ///
    /// - Parameters:
    ///   - replacement: The replacement
value where `mask` is `true`.
    ///   - mask: The Boolean mask that
determines whether the corresponding
element / row should be taken from `self`
    ///     (if the element in `mask` is
`false`) or `other` (if `true`).
    /// - Returns: A new tensor of the
same shape and type as `self`.
    public func replacing(with
replacement: some MLTensorScalar, where
mask: MLTensor) -> MLTensor
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
extension MLTensor {

    /// Concatenates `tensors` along the
`axis` dimension.
    ///
```

```
    /// For example:
    /// ```swift
    /// // t1 is [[1, 2, 3], [4, 5, 6]]
    /// // t2 is [[7, 8, 9], [10, 11,
12]]
    /// MLTensor(concatenating: [t1, t2])
// is [[1, 2, 3], [4, 5, 6], [7, 8, 9],
[10, 11, 12]]
    /// MLTensor(concatenating: [t1, t2],
alongAxis: 1) // is [[1, 2, 3, 7, 8, 9],
[4, 5, 6, 10, 11, 12]]
    ///
    /// // t3 has shape [2, 3]
    /// // t4 has shape [2, 3]
    /// MLTensor(concatenating: [t3, t4])
// has shape [4, 3]
    /// MLTensor(concatenating: [t3, t4],
alongAxis: 1) // has shape [2, 6]
    /// ```
    ///
    /// - Note: If you are concatenating
along a new axis consider using
    ///
`MLTensor(stacking:alongAxis:)`.
    ///
    /// - Parameters:
    ///   - tensors: The tensors to
concatenate. All tensors must have the
same rank and all dimensions except
`axis` must be equal.
    ///   - axis: The axis along which to
concatenate. Negative values wrap around
but must be in the range `[-rank, rank)`,
```

```
where
    ///   `rank` is the rank of the
provided tensors.
    public init(concatenating tensors:
some Collection<MLTensor>, alongAxis
axis: Int = 0)

    /// Stacks the given tensors along
the `axis` dimension into a new tensor
with rank one higher than the current
tensor and
    /// each tensor.
    ///
    /// Given that `tensors` all have
shape `[A, B, C]`, and `tensors.count =
N`, then:
    /// - if `axis == 0` then the
resulting tensor will have the shape `[N,
A, B, C]`.
    /// - if `axis == 1` then the
resulting tensor will have the shape `[A,
N, B, C]`.
    /// - etc.
    ///
    /// For example:
    /// ```swift
    /// // 'x' is [1, 4]
    /// // 'y' is [2, 5]
    /// // 'z' is [3, 6]
    /// MLTensor(stacking: [x, y, z]) //
is [[1, 4], [2, 5], [3, 6]]
    /// MLTensor(stacking: [x, y, z],
alongAxis: 1) // is [[1, 2, 3], [4, 5,
```

```
6]]
    /// ```
    ///
    /// - Parameters:
    ///   - tensors: The tensors to
stack. All tensors must have the same
shape and scalar type.
    ///   - axis: The axis along which to
stack. Negative values wrap around but
must be in the range `[-rank, rank]`,
where `rank`
    ///   is the rank of the provided
tensors.
    public init(stacking tensors: some
Collection<MLTensor>, alongAxis axis: Int
= 0)

    /// Returns a concatenated tensor
along the specified axis.
    ///
    /// - Parameters:
    ///   - other: The tensor to
concatenate. The tensors must have the
same dimensions, except for the specified
axis.
    ///   - axis: The axis along which to
concatenate. Negative values wrap around
but must be in the range `[-rank, rank)`.
    public func concatenated(with other:
MLTensor, alongAxis axis: Int = 0) ->
MLTensor
}
```

```
@available(macOS 15.0, iOS 18.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
extension MLTensor :
ExpressibleByArrayLiteral {

    /// Creates an instance initialized
with the given elements.
    public init(arrayLiteral elements:
MLTensor...)

    /// The type of the elements of an
array literal.
    @available(iOS 18.0, tvOS 18.0,
watchOS 11.0, visionOS 2.0, macOS 15.0,
*)
    public typealias ArrayLiteralElement
= MLTensor
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
extension MLTensor {

    /// Creates a tensor by stacking the
given tensors along the specified axis.
    ///
    /// - Parameter axis: The axis along
which to stack. Negative values wrap
around.
    public init(_ elements: some
Collection<MLTensor>, alongAxis axis: Int
= 0)
}
```

```
@available(macOS 15.0, iOS 18.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
extension MLTensor {

    /// Unpacks the given dimension of a
rank-`R` tensor into multiple rank-
`(R-1)` tensors.
    ///
    /// Unpacks `N` tensors from this
tensor by chipping it along the `axis`
dimension, where `N` is inferred from
this tensor's
    /// shape. For example, given a
tensor with shape
    /// `[A, B, C, D]`:
    ///   - If `axis == 0` then the `i`-
th tensor in the returned array is the
slice `self[i, nil, nil, nil]`
    ///     and each tensor in that array
will have shape `[B, C, D]`.
    ///   - If `axis == 1` then the `i`-
th tensor in the returned array is the
slice `value[:, i, :, :]` and each
    ///     tensor in that array will
have shape `[A, C, D]`.
    ///   - Etc.
    ///
    /// This is the opposite of
`MLTensor.init(stacking:alongAxis:)`.
    ///
    /// - Parameter axis: The dimension
along which to unstack. The `axis` must
```

```
be in the range `[-rank, rank)`.
    /// - Returns: An array containing
the unstacked tensors.
    public func unstacked(alongAxis axis:
Int = 0) -> [MLTensor]

    /// Splits a tensor into multiple
tensors. The tensor is split along
dimension `axis` into `count` smaller
tensors.
    ///
    /// For example:
    /// ```
    /// // 'value' is a tensor with shape
[5, 30]
    /// // Split 'value' into 3 tensors
along dimension 1:
    /// let parts = value.split(count: 3,
alongAxis: 1)
    /// parts[0] // has shape [5, 10]
    /// parts[1] // has shape [5, 10]
    /// parts[2] // has shape [5, 10]
    /// ```
    ///
    /// - Parameters:
    ///   - count: The number of splits
to create, must divide the size of
dimension `axis` evenly.
    ///   - axis: The dimension along
which to split this tensor. The `axis`
must be in the range `[-rank, rank)`.
    /// - Returns: An array containing
the tensor parts.
```

```
    public func split(count: Int,
alongAxis axis: Int = 0) -> [MLTensor]

    /// Splits a tensor into multiple
tensors. The tensor is split  into
`sizes.shape[0]` parts.
    ///
    /// For example:
    /// ```
    /// // 'value' is a tensor with shape
[5, 30]
    /// // Split 'value' into 3 tensors
with sizes [4, 15, 11] along dimension 1:
    /// let parts = value.split(sizes:
[4, 15, 11], alongAxis: 1)
    /// parts[0] // has shape [5, 4]
    /// parts[1] // has shape [5, 15]
    /// parts[2] // has shape [5, 11]
    /// ```
    ///
    /// - Parameters:
    ///   - sizes: A one-dimensional
tensor containing the size of each split,
must add up to the size of dimension
`axis`.
    ///   - axis: The dimension along
which to split this tensor. Must be in
the range `[-rank, rank)`.
    /// - Returns: Array containing the
tensors parts.
    public func split(sizes: [Int],
alongAxis axis: Int = 0) -> [MLTensor]
}
```

```
@available(macOS 15.0, iOS 18.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
extension MLTensor {

    /// Returns a new tensor with the
specified dimensions reversed.
    ///
    /// For example:
    /// ```swift
    /// let x = MLTensor(shape: [4],
scalars: [0,  1,  2,  3], scalarType:
Float.self)
    /// let y = x.reversed(alongAxes: 0)
    /// // [3, 2, 1, 0]
    /// ```
    ///
    /// - Parameter axes: The indices of
the dimensions to reverse. Must be in the
range `[-rank, rank)`.
    /// - Returns: A new tensor with the
same shape and scalar type with the
specified dimensions reversed.
    public func reversed(alongAxes axes:
Int...) -> MLTensor

    /// Returns a new tensor with the
specified dimensions reversed.
    ///
    /// For example:
    /// ```swift
    /// let x = MLTensor(shape: [4],
scalars: [0,  1,  2,  3], scalarType:
```

```
Float.self)
    /// let y = x.reversed(alongAxes:
[0])
    /// // [3, 2, 1, 0]
    /// ```
    ///
    /// - Parameter axes: Axis or axes to
reverse the elements of the tensor. Must
be in the range `[-rank, rank)`.
    ///   If no axis is specified, the
tensor will be reversed along all axes.
    /// - Returns: A new tensor with the
same shape and scalar type with the
specified dimensions reversed.
    public func reversed(alongAxes axes:
[Int]) -> MLTensor
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
extension MLTensor {

    /// A resize algorithm.
    public enum ResizeMethod : Sendable,
Hashable {

        /// The bilinear interpolation
mode where values are computed using
bilinear interpolation of 4 neighboring
pixels.
        ///
        /// `alignCorners` is a Boolean
indicating whether to align the corners
```

```
of the upscaling grid to the centre of
the
        /// scaling dimensions rather
than the edges.
        case bilinear(alignCorners: Bool
= false)

        /// The nearest interpolation
mode where values are interpolated using
the closest neighbor pixel.
        case nearestNeighbor

        /// Hashes the essential
components of this value by feeding them
into the
        /// given hasher.
        ///
        /// Implement this method to
conform to the `Hashable` protocol. The
        /// components used for hashing
must be the same as the components
compared
        /// in your type's `==` operator
implementation. Call `hasher.combine(_:)`
        /// with each of these
components.
        ///
        /// - Important: In your
implementation of `hash(into:)`,
        ///   don't call `finalize()` on
the `hasher` instance provided,
        ///   or replace it with a
different instance.
```

```
        ///   Doing so may become a
compile-time error in the future.
        ///
        /// - Parameter hasher: The
hasher to use when combining the
components
        ///   of this instance.
        public func hash(into hasher:
inout Hasher)

        /// Returns a Boolean value
indicating whether two values are equal.
        ///
        /// Equality is the inverse of
inequality. For any values `a` and `b`,
        /// `a == b` implies that `a !=
b` is `false`.
        ///
        /// - Parameters:
        ///   - lhs: A value to compare.
        ///   - rhs: Another value to
compare.
        public static func == (a:
MLTensor.ResizeMethod, b:
MLTensor.ResizeMethod) -> Bool

        /// The hash value.
        ///
        /// Hash values are not
guaranteed to be equal across different
executions of
        /// your program. Do not save
hash values to use during a future
```

```
execution.
        ///
        /// - Important: `hashValue` is
deprecated as a `Hashable` requirement.
To
        ///   conform to `Hashable`,
implement the `hash(into:)` requirement
instead.
        ///   The compiler provides an
implementation for `hashValue` for you.
        public var hashValue: Int { get }
    }
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
extension MLTensor {

    /// Resize the tensor's spatial
dimensions to size using the specified
method.
    ///
    /// For example:
    /// ```swift
    /// let image = MLTensor(shape: [1,
1, 2, 2], scalars: [
    ///     1, 0,
    ///     0, 1
    /// ], scalarType: Float.self)
    /// let resizedImage =
image.resized(to: (4, 4),
method: .nearest)
    /// // [[[[1, 1, 0, 0],
```

```
    /// //    [1, 1, 0, 0],
    /// //    [0, 0, 1, 1],
    /// //    [0, 0, 1, 1]]]]
    /// ```
    ///
    /// The tensor must be either a 4-dimensional float tensor of shape `[batch, channels, height, width]` or 3-dimensional float
    /// tensor of shape `[channel, height, width]`.
    ///
    /// - Parameters:
    ///   - size: The new size for the spatial dimensions of the tensor. The size must be positive.
    ///   - method: The resize method. The default value is `.nearest`.
    public func resized(to size: (newHeight: Int, newWidth: Int), method: MLTensor.ResizeMethod = .nearestNeighbor) -> MLTensor
}

@available(macOS 15.0, iOS 18.0, tvOS 18.0, watchOS 11.0, visionOS 2.0, *)
extension MLTensor {

    /// Clamps all elements to the given lower and upper bounds, inclusively.
    ///
    /// For example:
    /// ```swift
```

```
    /// let x = MLTensor([-1.0, 1.0,
2.0])
    /// let y = x.clamped(to: 0...1)
    /// await y.shapedArray(of:
Float.self) // is [0.0, 1.0, 1.0]
    /// ```
    public func clamped(to bounds:
ClosedRange<Float>) -> MLTensor

    /// Clamps all elements to the given
lower bound, inclusively.
    ///
    /// For example:
    /// ```swift
    /// let x = MLTensor([-1.0, 1.0,
2.0])
    /// let y = x.clamped(to: 0...)
    /// await y.shapedArray(of:
Float.self) // is [0.0, 1.0, 2.0]
    /// ```
    public func clamped(to bounds:
PartialRangeFrom<Float>) -> MLTensor

    /// Clamps all elements to the given
upper bound, inclusively.
    ///
    /// For example:
    /// ```swift
    /// let x = MLTensor([-1.0, 1.0,
2.0])
    /// let y = x.clamped(to: ...1)
    /// await y.shapedArray(of:
Float.self) // is [-1.0, 1.0, 1.0]
```

```
    /// ```
    public func clamped(to bounds:
PartialRangeThrough<Float>) -> MLTensor
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
extension MLTensor {

    /// Computes the softmax of the
specified tensor along the specified
axis.
    ///
    /// - Parameter axis: The axis along
which softmax will be computed.
    /// - Returns: A new tensor with the
same shape and scalar type.
    public func softmax(alongAxis axis:
Int = -1) -> MLTensor
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
extension MLTensor {

    /// Computes element-wise addition.
    public static func + (lhs: MLTensor,
rhs: some MLTensorScalar & Numeric) ->
MLTensor

    /// Computes element-wise addition.
    public static func + (lhs: some
MLTensorScalar & Numeric, rhs: MLTensor)
```

```
-> MLTensor

    /// Computes element-wise addition.
    ///
    /// Shapes must be broadcastable,
where the broadcasted shape becomes the
shape of the output.
    public static func + (lhs: MLTensor,
rhs: MLTensor) -> MLTensor

    /// Computes element-wise addition.
    ///
    /// Shapes must be broadcastable,
where the broadcasted shape becomes the
shape of the output.
    public static func += (lhs: inout
MLTensor, rhs: MLTensor)
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
extension MLTensor {

    /// Computes element-wise
subtraction.
    public static func - (lhs: MLTensor,
rhs: some MLTensorScalar & Numeric) ->
MLTensor

    /// Computes element-wise
subtraction.
    public static func - (lhs: some
MLTensorScalar & Numeric, rhs: MLTensor)
```

```
-> MLTensor

    /// Computes element-wise
subtraction.
    ///
    /// Shapes must be broadcastable,
where the broadcasted shape becomes the
shape of the output.
    public static func - (lhs: MLTensor,
rhs: MLTensor) -> MLTensor

    /// Computes element-wise
subtraction.
    ///
    /// Shapes must be broadcastable,
where the broadcasted shape becomes the
shape of the output.
    public static func -= (lhs: inout
MLTensor, rhs: MLTensor)
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
extension MLTensor {

    /// Computes element-wise
multiplication.
    public static func * (lhs: MLTensor,
rhs: some MLTensorScalar & Numeric) ->
MLTensor

    /// Computes element-wise
multiplication.
```

```
    public static func * (lhs: some
MLTensorScalar & Numeric, rhs: MLTensor)
-> MLTensor

    /// Computes element-wise
multiplication.
    ///
    /// Shapes must be broadcastable,
where the broadcasted shape becomes the
shape of the output.
    public static func * (lhs: MLTensor,
rhs: MLTensor) -> MLTensor

    /// Computes element-wise
multiplication.
    ///
    /// Shapes must be broadcastable,
where the broadcasted shape becomes the
shape of the output.
    public static func *= (lhs: inout
MLTensor, rhs: MLTensor)
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
extension MLTensor {

    /// Computes element-wise division.
    public static func / (lhs: MLTensor,
rhs: some MLTensorScalar & Numeric) ->
MLTensor

    /// Computes element-wise division.
```

```
    public static func / (lhs: some
MLTensorScalar & Numeric, rhs: MLTensor)
-> MLTensor

    /// Computes element-wise division.
    ///
    /// Shapes must be broadcastable,
where the broadcasted shape becomes the
shape of the output.
    public static func / (lhs: MLTensor,
rhs: MLTensor) -> MLTensor

    /// Computes element-wise
multiplication.
    ///
    /// Shapes must be broadcastable,
where the broadcasted shape becomes the
shape of the output.
    public static func /= (lhs: inout
MLTensor, rhs: MLTensor)
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
extension MLTensor {

    /// Computes element-wise remainder
of division.
    public static func % (lhs: MLTensor,
rhs: some MLTensorScalar & Numeric) ->
MLTensor

    /// Computes element-wise remainder
```

```
of division.
    public static func % (lhs: some
MLTensorScalar & Numeric, rhs: MLTensor)
-> MLTensor

    /// Computes element-wise remainder
of division.
    public static func % (lhs: MLTensor,
rhs: MLTensor) -> MLTensor

    /// Computes element-wise remainder
of division.
    public static func %= (lhs: inout
MLTensor, rhs: MLTensor)
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
extension MLTensor {

    /// Computes element-wise power of
each element with `exponent`.
    ///
    /// - Parameter exponent: The
exponent.
    public func pow(_ exponent: some
MLTensorScalar & Numeric) -> MLTensor

    /// Computes element-wise power of
each element with `exponent`.
    ///
    /// - Parameter exponent: The
exponent.
```

```swift
    public func pow(_ exponent: MLTensor)
-> MLTensor
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
extension MLTensor {

    /// Returns a tensor by replicating
its elements multiple times.
    ///
    /// The `multiples` argument
specifies the multiplier across each
dimensions,
    /// i.e., the output dimension at
index `i` is equal to `shape[i] *
multiples[i]`.
    ///
    /// For example:
    /// ```swift
    /// let x = MLTensor(shape: [2, 2],
scalars: [1, 2, 3, 4], scalarType:
Float.self)
    /// let y = x.tiled(multiples: [1,
2])
    /// y.shape // is [2, 4]
    /// ```
    ///
    /// If `multiples` specifies fewer
dimensions than the tensor, then ones are
prepended to `multiples` until all
    /// the dimensions are specified.
    ///
```

```
    /// If tensor has fewer dimensions
than `multiples`, then the tensor is
reshaped by prepending ones to the
dimensions
    /// until all the dimensions are
specified.
    ///
    /// - Parameter multiples: The
multiplier across each dimension. All
values must be greater than zero.
    /// - Returns: The replicated tensor.
    public func tiled(multiples: [Int])
-> MLTensor
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
extension MLTensor {

    /// Returns a new tensor with the
same shape where everything outside a
central band in each innermost matrix is
set to zero.
    ///
    /// The framework copies a diagonal
band of values from the tensor to the
result tensor of the same size.
    /// A coordinate `[..., i, j]` is
considered in band if
    /// ```md
    /// (lowerBandCount < 0 || (i-j) <=
lowerBandCount) && (upperBandCount < 0 ||
(j-i) <= upperBandCount)
```

```
    /// ```
    /// Values outside of the band are
set to `0`.
    ///
    /// For example:
    /// ```swift
    /// let t = Tensor([
    ///     [ 5,  1,  2, 3],
    ///     [-1,  5,  1, 2],
    ///     [-2, -1,  5, 1],
    ///     [-3, -2, -1, 5]
    /// ], scalarType: Float.self)
    ///
    /// t.bandPart(lowerBandCount: 0,
upperBandCount: 0)
    /// // [[ 5,  0,  0, 0]
    /// //  [ 0,  5,  0, 0]
    /// //  [ 0,  0,  5, 0]
    /// //  [ 0,  0,  0, 5]]
    ///
    /// t.bandPart(lowerBandCount: 0,
upperBandCount: -1)
    /// // [[ 5,  1,  2, 3]
    /// //  [ 0,  5,  1, 2]
    /// //  [ 0,  0,  5, 1]
    /// //  [ 0,  0,  0, 5]]
    ///
    /// t.bandPart(lowerBandCount: -1,
upperBandCount: 0)
    /// // [[ 5,  0,  0, 0]
    /// //  [-1,  5,  0, 0]
    /// //  [-2, -1,  5, 0]
    /// //  [-3, -2, -1, 5]]
```

```
    /// ```
    ///
    /// - Parameters:
    ///   - lowerBandCount: The number of
diagonals in the lower triangle to keep.
If `-1`, keep the entire lower triangle.
    ///   - upperBandCount: The number of
diagonals in the upper triangle to keep.
If `-1`, keep the entire upper triangle.
    /// - Returns: The band part of the
tensor.
    public func bandPart(lowerBandCount:
Int, upperBandCount: Int) -> MLTensor
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
extension MLTensor {

    /// Computes the cumulative product
along the specified axis.
    ///
    /// The scalar type of the tensor
must be numeric.
    ///
    /// For example:
    /// ```swift
    /// MLTensor([1, 2,
3]).cumulativeProduct() = [1, 1 * 2, 1 *
2 * 3]
    /// ```
    ///
    /// - Parameter axis: The axis along
```

```
which to perform the cumulative product.
The default value is `0`. Must be in the
range
    ///   `[-rank, rank)` and have a rank
greater than zero.
    /// - Returns: The result of the
cumulative product operation.
    public func
cumulativeProduct(alongAxis axis: Int =
0) -> MLTensor

    /// Computes the cumulative sum along
the specified axis.
    ///
    /// The scalar type of the tensor
must be numeric.
    ///
    /// For example:
    /// ```swift
    /// MLTensor([1, 2,
3]).cumulativeSum() = [1, 1 + 2, 1 + 2 +
3]
    /// ```
    ///
    /// - Parameter axis: The axis along
which to perform the cumulative sum. The
default value is `0`. Must be in the
range
    ///   `[-rank, rank)` and have a rank
greater than zero.
    /// - Returns: The result of the
cumulative sum operation.
    public func cumulativeSum(alongAxis
```

```
axis: Int = 0) -> MLTensor
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
extension MLTensor {

    /// Returns the *k* largest values
along the last axis of the tensor.
    ///
    /// - Note: The tensor must have at
least k elements along its last axis and
order returned by largest values which
are
    ///   equal is not deterministic.
    ///
    /// For example:
    /// ```swift
    /// let x = MLTensor(shape: [1, 5],
scalars: [1.0, 2.0, 3.0, 4.0, 5.0])
    /// let (values, indices) = x.topK(3)
    /// // values = [[5.0, 4.0, 3.0]]
    /// // indices = [[4, 3, 2]]
    /// ```
    ///
    /// - Parameter k: The number of
largest values to return.
    /// - Returns: A tuple of (values,
indices) with values containing the top k
values along the last axis and indices, a
`Int32` tensor,
    ///   containing the indices to the
corresponding values.
```

```
    public func topK(_ k: Int) ->
(values: MLTensor, indices: MLTensor)
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
extension MLTensor {

    /// A mode that dictates how a tensor
is padded.
    public enum PaddingMode : Hashable,
Sendable {

        /// Pads the input tensor
boundaries with a constant value.
        case constant(Float)

        /// Pads the input tensor using
the reflection of the input boundary.
        case reflection

        /// Pads the input tensor using
the reflection of the input, including
the edge value.
        case symmetric

        /// Hashes the essential
components of this value by feeding them
into the
        /// given hasher.
        ///
        /// Implement this method to
conform to the `Hashable` protocol. The
```

```
        /// components used for hashing
must be the same as the components
compared
        /// in your type's `==` operator
implementation. Call `hasher.combine(_:)`
        /// with each of these
components.
        ///
        /// - Important: In your
implementation of `hash(into:)`,
        ///   don't call `finalize()` on
the `hasher` instance provided,
        ///   or replace it with a
different instance.
        ///   Doing so may become a
compile-time error in the future.
        ///
        /// - Parameter hasher: The
hasher to use when combining the
components
        ///   of this instance.
        public func hash(into hasher:
inout Hasher)

        /// Returns a Boolean value
indicating whether two values are equal.
        ///
        /// Equality is the inverse of
inequality. For any values `a` and `b`,
        /// `a == b` implies that `a !=
b` is `false`.
        ///
        /// - Parameters:
```

```
        ///   - lhs: A value to compare.
        ///   - rhs: Another value to
compare.
        public static func == (a:
MLTensor.PaddingMode, b:
MLTensor.PaddingMode) -> Bool

        /// The hash value.
        ///
        /// Hash values are not
guaranteed to be equal across different
executions of
        /// your program. Do not save
hash values to use during a future
execution.
        ///
        /// - Important: `hashValue` is
deprecated as a `Hashable` requirement.
To
        ///   conform to `Hashable`,
implement the `hash(into:)` requirement
instead.
        ///   The compiler provides an
implementation for `hashValue` for you.
        public var hashValue: Int { get }
    }
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
extension MLTensor {

    /// Returns a tensor padded with the
```

```
given constant according to the specified
padding sizes.
    ///
    /// For example:
    /// ```swift
    /// let x = MLTensor(shape: [2, 3],
scalars: [
    ///     1, 2, 3,
    ///     4, 5, 6
    /// ], scalarType: Float32.self)
    /// let y = x.padded(forSizes: [(0,
0), (2, 2)], with: 0.0)
    /// // [[0, 0, 1, 2, 3, 0, 0],
    /// //  [0, 0, 4, 5, 6, 0, 0]]
    /// ```
    ///
    /// - Parameters:
    ///   - sizes: An array of tuples
describing the size to be inserted before
and after each dimension.
    ///   - value: The constant value
used for padding.
    /// - Returns: The padded tensor.
    public func padded(forSizes sizes:
[(before: Int, after: Int)], with value:
Float) -> MLTensor

    /// Returns a padded tensor according
to the specified padding sizes and mode.
    ///
    /// For example:
    /// ```swift
    /// let x = MLTensor(shape: [2, 3],
```

```
scalars: [
    ///     1, 2, 3,
    ///     4, 5, 6
    /// ], scalarType: Float32.self)
    ///
    /// let constantPadding =
x.padded(forSizes: [(0, 0), (2, 2)],
mode: .constant(Float(0)))
    /// // [[0, 0, 1, 2, 3, 0, 0],
    /// //  [0, 0, 4, 5, 6, 0, 0]]
    ///
    /// let reflectionPadding =
x.padded(forSizes: [(0, 0), (2, 2)],
mode: .reflection)
    /// // [[3, 2, 1, 2, 3, 2, 1],
    /// //  [6, 5, 4, 5, 6, 5, 4]]
    ///
    /// let symmetricPadding =
x.padded(forSizes: [(0, 0), (2, 2)],
mode: .symmetric)
    /// // [[2, 1, 1, 2, 3, 3, 2],
    /// //  [5, 4, 4, 5, 6, 6, 5]]
    /// ```
    ///
    /// - Parameters:
    ///   - sizes: An array of tuples
describing the size to be inserted before
and after each dimension.
    ///   - mode: The mode of padding,
etiher constant, reflection, or
symmetric.
    /// - Returns: The padded tensor.
    public func padded(forSizes sizes:
```

```
[(before: Int, after: Int)], mode:
MLTensor.PaddingMode) -> MLTensor
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
extension MLTensor {

    /// Returns the indices (or
arguments) of a tensor that give its
sorted order along the specified axis.
    ///
    /// For example:
    /// ```swift
    /// let x = MLTensor([1.0, 3.0, 2.0])
    /// let y = x.argSort()
    /// await y.shapedArray(of:
Int32.self) // is [0, 2, 1]
    /// ```
    ///
    /// - Parameters:
    ///   - axis: The axis along which to
sort. The default is `-1`, which sorts
the last axis.
    ///   - descendingOrder: A Boolean
value that determines the sort order. The
default is `false` which
    ///     sorts from largest to least.
    /// - Returns: A `Int32` tensor of
sorted indices.
    public func argsort(alongAxis axis:
Int = -1, descendingOrder: Bool = false)
-> MLTensor
```

```
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
extension MLTensor {

    /// Computes element-wise equality
between two tensors.
    ///
    /// - Returns: A Boolean tensor where
each element is true if `self` is equal
to than the corresponding
    ///   element of `other`, and false
otherwise.
    public static func .== (lhs:
MLTensor, rhs: some MLTensorScalar &
Numeric) -> MLTensor

    /// Computes element-wise equality
between two tensors.
    ///
    /// Shapes must be broadcastable,
where the broadcasted shape becomes the
shape of the output.
    ///
    /// - Returns: A Boolean tensor where
each element is true if `self` is equal
to than the corresponding
    ///   element of `other`, and false
otherwise.
    public static func .== (lhs:
MLTensor, rhs: MLTensor) -> MLTensor
}
```

```
@available(macOS 15.0, iOS 18.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
extension MLTensor {

    /// Computes element-wise inequality
between two tensors.
    ///
    /// - Returns: A Boolean tensor where
each element is true if `self` is not
equal to than the corresponding
    ///   element of `other`, and false
otherwise.
    public static func .!= (lhs:
MLTensor, rhs: some MLTensorScalar &
Numeric) -> MLTensor

    /// Computes element-wise inequality
between two tensors.
    ///
    /// Shapes must be broadcastable,
where the broadcasted shape becomes the
shape of the output.
    ///
    /// - Returns: A Boolean tensor where
each element is true if `self` is not
equal to than the corresponding
    ///   element of `other`, and false
otherwise.
    public static func .!= (lhs:
MLTensor, rhs: MLTensor) -> MLTensor
}
```

```
@available(macOS 15.0, iOS 18.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
extension MLTensor {

    /// Computes element-wise greater
comparison between two tensors.
    ///
    /// - Returns: A Boolean tensor where
each element is true if the left hand
side tensor is greater than the
    ///   corresponding element of the
right hand side, and false otherwise.
    public static func .> (lhs: MLTensor,
rhs: some MLTensorScalar & Numeric) ->
MLTensor

    /// Computes element-wise greater
comparison between two tensors.
    ///
    /// Shapes must be broadcastable,
where the broadcasted shape becomes the
shape of the output.
    ///
    /// - Returns: A Boolean tensor where
each element is true if the left hand
side tensor is greater than the
    ///   corresponding element of the
right hand side, and false otherwise.
    public static func .> (lhs: MLTensor,
rhs: MLTensor) -> MLTensor
}

@available(macOS 15.0, iOS 18.0, tvOS
```

```
18.0, watchOS 11.0, visionOS 2.0, *)
extension MLTensor {

    /// Computes element-wise greater
than or equal to comparison between two
tensors.
    ///
    /// - Returns: A Boolean tensor where
each element is true if the left hand
side tensor is greater than, or equal,
    ///   to the corresponding element of
the right hand side, and false otherwise.
    public static func .>= (lhs:
MLTensor, rhs: some MLTensorScalar &
Numeric) -> MLTensor

    /// Computes element-wise greater
than or equal to comparison between two
tensors.
    ///
    /// Shapes must be broadcastable,
where the broadcasted shape becomes the
shape of the output.
    ///
    /// - Returns: A Boolean tensor where
each element is true if the left hand
side tensor is greater than, or equal,
    ///   to the corresponding element of
the right hand side, and false otherwise.
    public static func .>= (lhs:
MLTensor, rhs: MLTensor) -> MLTensor
}
```

```
@available(macOS 15.0, iOS 18.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
extension MLTensor {

    /// Computes element-wise less
comparison between two tensors.
    ///
    /// - Returns: A Boolean tensor where
each element is true if the left hand
side tensor is less than the
    ///   corresponding element of the
right hand side, and false otherwise.
    public static func .< (lhs: MLTensor,
rhs: some MLTensorScalar & Numeric) ->
MLTensor

    /// Computes element-wise less
comparison between two tensors.
    ///
    /// Shapes must be broadcastable,
where the broadcasted shape becomes the
shape of the output.
    ///
    /// - Returns: A Boolean tensor where
each element is true if the left hand
side tensor is less than the
    ///   corresponding element of the
right hand side, and false otherwise.
    public static func .< (lhs: MLTensor,
rhs: MLTensor) -> MLTensor
}

@available(macOS 15.0, iOS 18.0, tvOS
```

```
18.0, watchOS 11.0, visionOS 2.0, *)
extension MLTensor {

    /// Computes element-wise less than
or equal to comparison between two
tensors.
    ///
    /// - Returns: A Boolean tensor where
each element is true if the left hand
side tensor is less than, or equal,
    ///   to the corresponding element of
the right hand side, and false otherwise.
    public static func .<= (lhs:
MLTensor, rhs: some MLTensorScalar &
Numeric) -> MLTensor

    /// Computes element-wise less than
or equal to comparison between two
tensors.
    ///
    /// Shapes must be broadcastable,
where the broadcasted shape becomes the
shape of the output.
    ///
    /// - Returns: A Boolean tensor where
each element is true if the left hand
side tensor is less than, or equal,
    ///   to the corresponding element of
the right hand side, and false otherwise.
    public static func .<= (lhs:
MLTensor, rhs: MLTensor) -> MLTensor
}
```

```swift
@available(macOS 15.0, iOS 18.0, tvOS 18.0, watchOS 11.0, visionOS 2.0, *)
extension MLTensor {

    /// Returns a tensor by gathering slices along the given axis at the specified indices.
    ///
    /// For example:
    /// ```swift
    /// let x = MLTensor(shape: [3, 3], scalars: [
    ///       0,  1,  2,
    ///      10, 11, 12,
    ///      20, 21, 22
    /// ], scalarType: Float.self)
    /// let i = MLTensor([2, 1], scalarType: Int32.self)
    /// let y0 = x.gathering(atIndices: i)
    /// // [[20, 21, 22],
    /// //  [10, 11, 12]]
    ///
    /// let y1 = x.gathering(atIndices: i, alongAxis: 1)
    /// // [[ 2,  1],
    /// //  [12, 11],
    /// //  [22, 21]]
    /// ```
    ///
    /// - Parameters:
    ///   - indices: A 32-bit integer tensor containing indices to gather at.
```

```
    ///   - axis: The dimension to gather
along. Must be in the range `[-rank,
rank)`.
    ///
    /// - Returns: The gathered tensor.
    public func gathering(atIndices
indices: MLTensor, alongAxis axis: Int)
-> MLTensor
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
extension MLTensor {

    /// Returns a tensor by gathering
slices at the specified indices.
    ///
    /// The `indices` tensor is
interpreted as a `(rank-1)` dimensional
set of one-dimensional lookup vectors,
for example:
    /// ```swift
    /// let x = MLTensor(shape: [2, 2],
scalars: [
    ///     0,  1,
    ///     10, 11
    /// ], scalarType: Float.self)
    /// let i = MLTensor(shape: [2, 2],
scalars: [
    ///     0, 0,
    ///     1, 1
    /// ], scalarType: Int32.self)
    /// let y = x.gathering(atIndices: i)
```

```
    /// // [ 0, 11]
    /// ```
    ///
    /// If the one-dimensional lookup
vectors do not give a full set of
indices, the remaining indices are
treated as a slice, for example:
    /// ```swift
    /// let x = MLTensor(shape: [3, 3],
scalars: [
    ///     0,  1,  2,
    ///     10, 11, 12,
    ///     20, 21, 22
    /// ], scalarType: Float.self)
    /// let i = MLTensor(shape: [3, 1],
scalars: [
    ///     2,
    ///     1
    /// ], scalarType: Int32.self)
    /// let y = x.gathering(atIndices: i)
    /// // [[20 21 22]
    /// //  [10 11 12]]
    /// ```
    ///
    /// - Parameter indices: A 32-bit
integer tensor containing indices to
gather at.
    /// - Returns: The gathered tensor.
    public func gathering(atIndices
indices: MLTensor) -> MLTensor
}

@available(macOS 15.0, iOS 18.0, tvOS
```

```
18.0, watchOS 11.0, visionOS 2.0, *)
extension MLTensor {

    /// Replaces slices along the
specified indices with the given
replacement values.
    ///
    /// For example:
    /// ```swift
    /// let x = MLTensor(shape: [2, 3],
scalars: [
    ///     10, 30, 20,
    ///     60, 40, 50
    /// ], scalarType: Float.self)
    /// let i = MLTensor(shape: [2, 1],
scalars: [
    ///     1,
    ///     0
    /// ], scalarType: Int32.self)
    /// let y = x.replacing(with: 99,
atIndices: i, alongAxis: 1)
    /// // [[10, 99, 20],
    /// //  [99, 40, 50]]
    /// ```
    ///
    /// - Parameters:
    ///   - indices: A 32-bit integer
tensor containing indices to scatter
values from `replacement`. `indices` must
have the same
    ///     shape as `self` except at
`axis`. Must be in the range `[-rank,
rank)`.
```

```
    ///   - replacement: The replacement
value.
    ///   - axis: The axis to scatter to.
    /// - Returns: The updated tensor.
    public func replacing(atIndices
indices: MLTensor, with replacement: some
MLTensorScalar, alongAxis axis: Int) ->
MLTensor

    /// Replaces slices along the
specified indices with the given
replacement values.
    ///
    /// For example:
    /// ```swift
    /// let x = MLTensor(shape: [2, 3],
scalars: [
    ///     10, 30, 20,
    ///     60, 40, 50
    /// ], scalarType: Float.self)
    /// let i = MLTensor(shape: [2, 1],
scalars: [
    ///     1,
    ///     0
    /// ], scalarType: Int32.self)
    /// let updates = MLTensor(shape: [2,
1], scalars: [
    ///     99,
    ///     99
    /// ], scalarType: Float.self)
    /// let y = x.replacing(atIndices: i,
with: updates, alongAxis: 1)
    /// // [[10, 99, 20],
```

```
    /// //  [99, 40, 50]]
    /// ```
    ///
    /// - Parameters:
    ///   - indices: A 32-bit integer
tensor containing indices to scatter
values from `replacement`. Must have the
same
    ///     shape as `replacement`. Must
have the same shape as `self` except at
`axis`.
    ///   - replacement: The replacement
values.
    ///   - axis: The axis to scatter to.
Must be in the range `[-rank, rank)`.
    /// - Returns: The updated tensor.
    public func replacing(with
replacement: MLTensor, atIndices indices:
MLTensor, alongAxis axis: Int) ->
MLTensor
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
extension MLTensor {

    public subscript(firstRange:
(UnboundedRange_) -> (), trailingRanges:
(any MLTensorRangeExpression)?...) ->
MLTensor { get }

    public subscript(firstRange: (any
MLTensorRangeExpression)?, secondRange:
```

```
(UnboundedRange_) -> (), trailingRanges:
(any MLTensorRangeExpression)?...) ->
MLTensor { get }

    public subscript(firstRange: (any
MLTensorRangeExpression)?, secondRange:
(any MLTensorRangeExpression)?,
thirdRange: (UnboundedRange_) -> (),
trailingRanges: (any
MLTensorRangeExpression)?...) -> MLTensor
{ get }

    public subscript(firstRange: (any
MLTensorRangeExpression)?, secondRange:
(any MLTensorRangeExpression)?,
thirdRange: (any
MLTensorRangeExpression)?, fourthRange:
(UnboundedRange_) -> (), trailingRanges:
(any MLTensorRangeExpression)?...) ->
MLTensor { get }

    public subscript(ranges: (any
MLTensorRangeExpression)?...) -> MLTensor
{ get }
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
extension MLTensor {

    /// Returns the sum along all axes.
    ///
    /// For example:
```

```
    /// ```swift
    /// let x = MLTensor(shape: [3, 2],
scalars: [2, 3, 4, 5, 6, 7], scalarType:
Float.self)
    /// let y = x.sum()
    /// y.shape // is []
    /// await y.shapedArray(of:
Float.self) // is [27]
    /// ```
    ///
    /// - Parameter keepRank: A Boolean
indicating whether to keep the reduced
axes or not. The default value is
`false`.
    /// - Returns: The reduced tensor.
    public func sum(keepRank: Bool =
false) -> MLTensor

    /// Returns the sum along the
specified axes.
    ///
    /// For example:
    /// ```swift
    /// let x = MLTensor(shape: [3, 2],
scalars: [2, 3, 4, 5, 6, 7], scalarType:
Float.self)
    /// let y = x.sum(alongAxes: 0,
keepRank: true)
    /// y.shape // is [1, 2]
    /// await y.shapedArray(of:
Float.self) // is [[12, 15]]
    /// ```
    ///
```

```
    /// - Parameters:
    ///   - axes: The axes to reduce.
    ///   - keepRank: A Boolean
indicating whether to keep the reduced
axes or not. The default value is
`false`.
    /// - Returns: The reduced tensor.
    public func sum(alongAxes axes:
Int..., keepRank: Bool = false) ->
MLTensor

    /// Returns the sum along the
specified axes.
    ///
    /// For example:
    /// ```swift
    /// let x = MLTensor(shape: [3, 2],
scalars: [2, 3, 4, 5, 6, 7], scalarType:
Float.self)
    /// let y = x.sum(alongAxes: [0],
keepRank: true)
    /// y.shape // is [1, 2]
    /// await y.shapedArray(of:
Float.self) // is [[12, 15]]
    /// ```
    ///
    /// - Parameters:
    ///   - axes: The axes to reduce.
    ///   - keepRank: A Boolean
indicating whether to keep the reduced
axes or not. The default value is
`false`.
    /// - Returns: The reduced tensor.
```

```
    public func sum(alongAxes axes:
[Int], keepRank: Bool = false) ->
MLTensor
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
extension MLTensor {

    /// Returns the mean along all axes.
    ///
    /// For example:
    /// ```swift
    /// let x = MLTensor(shape: [3, 2],
scalars: [2, 3, 4, 5, 6, 7], scalarType:
Float.self)
    /// let y = x.mean()
    /// y.shape // is []
    /// await y.shapedArray(of:
Float.self) // is [4.5]
    /// ```
    ///
    /// - Parameter keepRank: A Boolean
indicating whether to keep the reduced
axes or not. The default value is
`false`.
    /// - Returns: The reduced tensor.
    public func mean(keepRank: Bool =
false) -> MLTensor

    /// Returns the mean along the
specified axes.
    ///
```

```
    /// For example:
    /// ```swift
    /// let x = MLTensor(shape: [3, 2],
scalars: [2, 3, 4, 5, 6, 7], scalarType:
Float.self)
    /// let y = x.mean(alongAxes: 0,
keepRank: true)
    /// y.shape // is [1, 2]
    /// await y.shapedArray(of:
Float.self) // is [4.0 5.0]
    /// ```
    ///
    /// - Parameters:
    ///   - axes: The axes to reduce.
    ///   - keepRank: A Boolean
indicating whether to keep the reduced
axes or not. The default value is
`false`.
    /// - Returns: The reduced tensor.
    public func mean(alongAxes axes:
Int..., keepRank: Bool = false) ->
MLTensor

    /// Returns the mean along the
specified axes.
    ///
    /// For example:
    /// ```swift
    /// let x = MLTensor(shape: [3, 2],
scalars: [2, 3, 4, 5, 6, 7], scalarType:
Float.self)
    /// let y = x.mean(alongAxes: [0],
keepRank: true)
```

```
    /// y.shape // is [1, 2]
    /// await y.shapedArray(of:
Float.self) // is [4.0 5.0]
    /// ```
    ///
    /// - Parameters:
    ///   - axes: The axes to reduce.
    ///   - keepRank: A Boolean
indicating whether to keep the reduced
axes or not. The default value is
`false`.
    /// - Returns: The reduced tensor.
    public func mean(alongAxes axes:
[Int], keepRank: Bool = false) ->
MLTensor
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
extension MLTensor {

    /// Returns the minimum value in the
array.
    ///
    /// ```swift
    /// let x = MLTensor(shape: [3, 2],
scalars: [2, 3, 4, 5, 6, 7], scalarType:
Float.self)
    /// let y = x.min()
    /// y.shape // is []
    /// await y.shapedArray(of:
Float.self) // is [2.0]
    /// ```
```

```
    ///
    /// - Parameter keepRank: A Boolean
indicating whether to keep the reduced
axes or not. The default value is
`false`.
    /// - Returns: The reduced tensor.
    public func min(keepRank: Bool =
false) -> MLTensor

    /// Returns the minimum values along
the specified axes.
    ///
    /// ```swift
    /// let x = MLTensor(shape: [3, 2],
scalars: [2, 3, 4, 5, 6, 7], scalarType:
Float.self)
    /// let y = x.min(alongAxes: 0)
    /// y.shape // is [1, 2]
    /// await y.shapedArray(of:
Float.self) // is [2.0 3.0]
    /// ```
    ///
    /// - Parameters:
    ///   - axes: The axes to reduce.
    ///   - keepRank: A Boolean
indicating whether to keep the reduced
axes or not. The default value is
`false`.
    /// - Returns: The reduced tensor.
    public func min(alongAxes axes:
Int..., keepRank: Bool = false) ->
MLTensor
```

```
    /// Returns the minimum values along
the specified axes.
    ///
    /// ```swift
    /// let x = MLTensor(shape: [3, 2],
scalars: [2, 3, 4, 5, 6, 7], scalarType:
Float.self)
    /// let y = x.min(alongAxes: [0])
    /// y.shape // is [1, 2]
    /// await y.shapedArray(of:
Float.self) // is [2.0 3.0]
    /// ```
    ///
    /// - Parameters:
    ///   - axes: The axes to reduce.
    ///   - keepRank: A Boolean
indicating whether to keep the reduced
axes or not. The default value is
`false`.
    /// - Returns: The reduced tensor.
    public func min(alongAxes axes:
[Int], keepRank: Bool = false) ->
MLTensor
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
extension MLTensor {

    /// Returns the maximum value in the
array.
    ///
    /// ```swift
```

```
    /// let x = MLTensor(shape: [3, 2],
scalars: [2, 3, 4, 5, 6, 7], scalarType:
Float.self)
    /// let y = x.max()
    /// y.shape // is []
    /// await y.shapedArray(of:
Float.self) // is [7.0]
    /// ```
    ///
    /// - Parameter keepRank: A Boolean
indicating whether to keep the reduced
axes or not. The default value is
`false`.
    /// - Returns: The reduced tensor.
    public func max(keepRank: Bool =
false) -> MLTensor

    /// Returns the maximum values along
the specified axes.
    ///
    /// ```swift
    /// let x = MLTensor(shape: [3, 2],
scalars: [2, 3, 4, 5, 6, 7], scalarType:
Float.self)
    /// let y = x.max(alongAxes: 0,
keepRank: true)
    /// y.shape // is [1, 2]
    /// await y.shapedArray(of:
Float.self) // is [6.0 7.0]
    /// ```
    ///
    /// - Parameters:
    ///   - axes: The axes to reduce.
```

```
    ///   - keepRank: A Boolean
indicating whether to keep the reduced
axes or not. The default value is
`false`.
    /// - Returns: The reduced tensor.
    public func max(alongAxes axes:
Int..., keepRank: Bool = false) ->
MLTensor

    /// Returns the maximum values along
the specified axes.
    ///
    /// ```swift
    /// let x = MLTensor(shape: [3, 2],
scalars: [2, 3, 4, 5, 6, 7], scalarType:
Float.self)
    /// let y = x.max(alongAxes: [0],
keepRank: true)
    /// y.shape // is [1, 2]
    /// await y.shapedArray(of:
Float.self) // is [6.0 7.0]
    /// ```
    ///
    /// - Parameters:
    ///   - axes: The axes to reduce.
    ///   - keepRank: A Boolean
indicating whether to keep the reduced
axes or not. The default value is
`false`.
    /// - Returns: The reduced tensor.
    public func max(alongAxes axes:
[Int], keepRank: Bool = false) ->
MLTensor
```

```
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
extension MLTensor {

    /// Returns the product along all
axes.
    ///
    /// ```swift
    /// let x = MLTensor(shape: [3, 2],
scalars: [2, 3, 4, 5, 6, 7], scalarType:
Float.self)
    /// let y = x.product()
    /// y.shape // is []
    /// await y.shapedArray(of:
Float.self) // is [5040.0]
    /// ```
    ///
    /// - Parameter keepRank: A Boolean
indicating whether to keep the reduced
axes or not. The default value is
`false`.
    /// - Returns: The reduced tensor.
    public func product(keepRank: Bool =
false) -> MLTensor

    /// Returns the product along the
specified axes.
    ///
    /// ```swift
    /// let x = MLTensor(shape: [3, 2],
scalars: [2, 3, 4, 5, 6, 7], scalarType:
```

```
Float.self)
    /// let y = x.product(alongAxes: 0)
    /// y.shape // is [1, 2]
    /// await y.shapedArray(of:
Float.self) // is [48.0 105.0]
    /// ```
    ///
    /// - Parameters:
    ///   - axes: The axes to reduce.
    ///   - keepRank: A Boolean
indicating whether to keep the reduced
axes or not. The default value is
`false`.
    /// - Returns: The reduced tensor.
    public func product(alongAxes axes:
Int..., keepRank: Bool = false) ->
MLTensor

    /// Returns the product along the
specified axes.
    ///
    /// ```swift
    /// let x = MLTensor(shape: [3, 2],
scalars: [2, 3, 4, 5, 6, 7], scalarType:
Float.self)
    /// let y = x.product(alongAxes: [0],
keepRank: true)
    /// y.shape // is [1, 2]
    /// await y.shapedArray(of:
Float.self) // is [48.0 105.0]
    /// ```
    ///
    /// - Parameters:
```

```
    ///   - axes: The axes to reduce.
    ///   - keepRank: A Boolean
indicating whether to keep the reduced
axes or not. The default value is
`false`.
    /// - Returns: The reduced tensor.
    public func product(alongAxes axes:
[Int], keepRank: Bool = false) ->
MLTensor
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
extension MLTensor {

    /// Returns the index of the maximum
value of the flattened scalars.
    ///
    /// ```swift
    /// let x = MLTensor(shape: [3, 2],
scalars: [2, 3, 4, 5, 6, 7], scalarType:
Float.self)
    /// let y = x.argmax()
    /// y.shape // is []
    /// y.scalarType // is Int32
    /// await y.shapedArray(of:
Int32.self) // is 5
    /// ```
    public func argmax() -> MLTensor

    /// Returns the indices of the
maximum values along the specified axis.
    ///
```

```
    /// ```swift
    /// let x = MLTensor(shape: [3, 2],
scalars: [2, 3, 4, 5, 6, 7], scalarType:
Float.self)
    /// let y = x.argmax(alongAxis: 0)
    /// y.shape // is [2]
    /// y.scalarType // is Int32
    /// await y.shapedArray(of:
Int32.self) // is 2 2
    /// ```
    ///
    /// - Parameters:
    ///   - axis: The axis to reduce.
    ///   - keepRank: A Boolean
indicating whether to keep the reduced
axis or not. The default value is
`false`.
    /// - Returns: The reduced tensor.
    public func argmax(alongAxis axis:
Int, keepRank: Bool = false) -> MLTensor
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
extension MLTensor {

    /// Returns the index of the minimum
value of the flattened scalars.
    ///
    /// ```swift
    /// let x = MLTensor(shape: [3, 2],
scalars: [2, 3, 4, 5, 6, 7], scalarType:
Float.self)
```

```
    /// let y = x.argmin()
    /// y.shape // is []
    /// y.scalarType // is Int32
    /// await y.shapedArray(of:
Int32.self) // is 0
    /// ```
    public func argmin() -> MLTensor

    /// Returns the indices of the
minimum values along the specified axis.
    ///
    /// ```swift
    /// let x = MLTensor(shape: [3, 2],
scalars: [2, 3, 4, 5, 6, 7], scalarType:
Float.self)
    /// let y = x.argmin(alongAxis: 0)
    /// y.shape // is [2]
    /// y.scalarType // is Int32
    /// await y.shapedArray(of:
Int32.self) // is 0 0
    /// ```
    ///
    /// - Parameters:
    ///   - axis: The axis to reduce.
    ///   - keepRank: A Boolean
indicating whether to keep the reduced
axis or not. The default value is
`false`.
    /// - Returns: The reduced tensor.
    public func argmin(alongAxis axis:
Int, keepRank: Bool = false) -> MLTensor
}
```

```
@available(macOS 15.0, iOS 18.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
extension MLTensor {

    /// Returns the negation of the
tensor.
    prefix public static func - (rhs:
MLTensor) -> MLTensor
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
extension MLTensor {

    /// Computes the absolute of the
tensor's elements.
    public func abs() -> MLTensor

    /// Computes the inverse cosine of
the tensor's elements.
    public func acos() -> MLTensor

    /// Computes the inverse hyperbolic
cosine of the tensor's elements.
    public func acosh() -> MLTensor

    /// Computes the inverse sine of the
tensor's elements.
    public func asin() -> MLTensor

    /// Computes the inverse hyperbolic
sine of the tensor's elements.
    public func asinh() -> MLTensor
```

```
    /// Computes the inverse tangent of
the tensor's elements.
    public func atan() -> MLTensor

    /// Computes the inverse hyperbolic
tangent of the tensor's elements.
    public func atanh() -> MLTensor

    /// Computes the ceiling of the
tensor's elements.
    public func ceil() -> MLTensor

    /// Computes the cosine of the
tensor's elements.
    public func cos() -> MLTensor

    /// Computes the hyperbolic cosine of
the tensor's elements.
    public func cosh() -> MLTensor

    /// Computes the natural exponent of
the tensor's elements.
    public func exp() -> MLTensor

    /// Computes the exponent with base
two of the tensor's elements.
    public func exp2() -> MLTensor

    /// Computes the floor of the
tensor's elements.
    public func floor() -> MLTensor
```

```
    /// Computes the natural logarithm of
the tensor's elements.
    public func log() -> MLTensor

    /// Computes the reciprocal of the
tensor's elements.
    public func reciprocal() -> MLTensor

    /// Rounds the tensor's elements.
    public func round() -> MLTensor

    /// Computes reverse square root of
the tensor's elements.
    ///
    /// The reverse square root operation
is the reciprocal of the square root.
    public func rsqrt() -> MLTensor

    /// Returns the sign of the tensor's
elements.
    ///
    /// This operation returns `1.0` if
the correspnding element is greater than
`0`, `-1.0` if it is less than `0`,
    /// `-0.0` if it is equal to `-0.0`,
and `+0.0` if it is equal to `+0.0`.
    public func sign() -> MLTensor

    /// Computes sine of the tensor's
elements.
    public func sin() -> MLTensor

    /// Computes hyperbolic sine of the
```

```
tensor's elements.
    public func sinh() -> MLTensor

    /// Computes square root of the
tensor's elements.
    public func squareRoot() -> MLTensor

    /// Computes square of the tensor's
elements.
    public func squared() -> MLTensor

    /// Computes tangent of the tensor's
elements.
    public func tan() -> MLTensor

    /// Computes hyperbolic tangent of
the tensor's elements.
    public func tanh() -> MLTensor
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
extension MLTensor {

    /// Creates a tensor with the
specified shape, randomly sampling scalar
values from a normal distribution.
    ///
    /// - Note: Using the same seed
produces the same result only when
dispatched to the same compute device.
    ///
    /// - Parameters:
```

```
    ///   - shape: The shape of the
tensor.
    ///   - mean: The mean of the
distribution.
    ///   - standardDeviation: The
standard deviation of the distribution.
    ///   - seed: The random seed.
    ///   - scalarType: The scalar type.
    public init<Scalar>(randomNormal
shape: [Int], mean: Scalar =
Scalar.init(0.0), standardDeviation:
Scalar = Scalar.init(1.0), seed: UInt64?
= nil, scalarType: Scalar.Type =
Scalar.self) where Scalar :
MLTensorScalar, Scalar :
BinaryFloatingPoint
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
extension MLTensor {

    /// Creates a tensor with the
specified shape, randomly sampling scalar
values from a uniform distribution in
`bounds`.
    ///
    /// - Note: Using the same seed
produces the same result only when
dispatched to the same compute device.
    ///
    /// - Parameters:
    ///   - shape: The shape of the
```

```
tensor.
    ///   - bounds: The bounds of the
distribution. The default value is
`0..<1`.
    ///   - seed: The random seed.
    ///   - scalarType: The scalar type.
    public init<Scalar>(randomUniform
shape: [Int], in bounds: Range<Scalar> =
0..<1, seed: UInt64? = nil, scalarType:
Scalar.Type = Scalar.self) where Scalar :
MLTensorScalar, Scalar :
BinaryFloatingPoint
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
extension MLTensor {

    /// Creates a tensor with the
specified shape, randomly sampling scalar
values from a uniform distribution in
`bounds`.
    ///
    /// - Note: Using the same seed
produces the same result only when
dispatched to the same compute device.
    ///
    /// - Parameters:
    ///   - shape: The shape of the
tensor.
    ///   - bounds: The bounds of the
distribution. The default value is
`0...1`.
```

```
    ///   - seed: The random seed.
    ///   - scalarType: The scalar type.
    public init<Scalar>(randomUniform
shape: [Int], in bounds:
ClosedRange<Scalar> = 0...1, seed:
UInt64? = nil, scalarType: Scalar.Type =
Scalar.self) where Scalar :
MLTensorScalar, Scalar : BinaryInteger
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
extension MLTensor {

    /// Multiplies two tensors together
using matrix multiplication.
    ///
    /// The specific behaviour is
determined by the dimensionality of the
inputs, as described below:
    /// - If both tensors are 1-
dimensional, the dot-product is returned.
    /// - If the first is 2-dimensional
and other 1-dimensional, the matrix-
vector product is returned.
    /// - If both have a rank of 2, the
matrix-matrix product is returned and
when the rank is greater than 2, the
batched matrix
    ///   multiply is returned
    ///
    /// For exmaple:
    /// ```swift
```

```
    /// let v1 = MLTensor([1.0, 2.0, 3.0,
4.0])
    /// let v2 = MLTensor([5.0, 6.0, 7.0,
8.0])
    /// let v3 = v1.matmul(v2)
    /// v3.shape // is []
    /// await v3.shapedArray(of:
Float.self) // is 70.0
    ///
    /// let m1 = MLTensor(shape: [2, 3],
scalars: [
    ///     1, 2, 3,
    ///     4, 5, 6
    /// ], scalarType: Float.self)
    /// let m2 = MLTensor(shape: [3, 2],
scalars: [
    ///      7,  8,
    ///      9, 10,
    ///     11, 12
    /// ], scalarType: Float.self)
    /// let m3 = t1.matmul(r2)
    /// m3.shape // is [2, 2]
    /// await m3.shapedArray(of:
Float.self) // is [[58, 64], [139, 154]]
    ///
    /// // Supports broadcasting
    /// let m4 = MLTensor(randomNormal:
[3, 1, 1, 4], scalarType: Float.self)
    /// let m5 = MLTensor(randomNormal:
[4, 2], scalarType: Float.self)
    /// let m6 = t4.matmul(t5)
    /// m6.shape // is [3, 1, 1, 2]
    /// ```
```

```
    ///
    /// - Parameter other: The right-hand
side tensor to be multiplied.
    /// - Returns: The result of the
matrix multiplication.
    public func matmul(_ other: MLTensor)
-> MLTensor
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
extension MLTensor {

    /// Creates a tensor with the
specified shape, then calls the given
closure with a buffer covering the
tensor's uninitialized memory.
    ///
    /// - Parameters:
    ///   - shape: The dimensions of the
tensor.
    ///   - scalarType: The scalar type.
    ///   - initializer: A closure which
will be called to initialize the
underlying memory of the tensor. Scalars
expected to be
    ///     stored contiguously in first-
major order.
    public init(unsafeUninitializedShape
shape: [Int], scalarType: any
MLTensorScalar.Type, initializingWith
initializer:
(UnsafeMutableRawBufferPointer) throws ->
```

```
Void) rethrows
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
extension MLTensor {

    /// Creates a tensor with the
specified shape and contiguous scalars in
first-major order.
    ///
    /// - Parameters:
    ///   - shape: The dimensions of the
tensor. The product of the dimensions of
the shape must equal the number of
scalars.
    ///   - scalars: The scalar contents
of the tensor.
    public init(shape: [Int], scalars:
some Collection<Float>)

    /// Creates a tensor with the
specified shape and contiguous scalars in
row-major order.
    ///
    /// - Parameters:
    ///   - shape: The dimensions of the
tensor.
    ///   - scalars: The scalar contents
of the tensor. The product of the
dimensions of the shape must equal the
number of scalars.
    ///   - scalarType: The scalar type.
```

```
    public init<Scalar>(shape: [Int],
scalars: some Collection, scalarType:
Scalar.Type = Scalar.self) where Scalar :
MLTensorScalar

    /// Creates a zero-dimensional tensor
from a scalar value.
    ///
    /// - Parameters:
    ///   - value: The value.
    ///   - scalarType: The scalar type.
    public init<Scalar>(_ value: Scalar,
scalarType: Scalar.Type = Scalar.self)
where Scalar : MLTensorScalar

    /// Creates a one-dimensional tensor
from scalars.
    ///
    /// - Parameters:
    ///   - scalars: A collection of
scalars.
    ///   - scalarType: The scalar type.
    public init<Scalar>(_ scalars: some
Collection, scalarType: Scalar.Type =
Scalar.self) where Scalar :
MLTensorScalar

    /// Creates a one-dimensional tensor
from scalars.
    ///
    /// - Parameter scalars: A collection
of scalars.
    public init(_ scalars: some
```

```
Collection<Float>)

    /// Creates a one-dimensional tensor
from scalars.
    ///
    /// - Parameter scalars: A collection
of scalars.
    public init(_ scalars: some
Collection<Int32>)
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
extension MLTensor {

    /// Creates a tensor with the
specified shape and a single, repeated
scalar value.
    ///
    /// - Parameters:
    ///   - repeatedValue: The float
value to repeat.
    ///   - shape: The shape of the
tensor.
    public init(repeating repeatedValue:
Float, shape: [Int])

    /// Creates a tensor with the
specified shape and a single, repeated
scalar value.
    ///
    /// - Parameters:
    ///   - repeatedValue: The scalar
```

```
value to repeat.
    ///   - shape: The shape of the
tensor.
    ///   - scalarType: The scalar type.
    public init<Scalar>(repeating
repeatedValue: Scalar, shape: [Int],
scalarType: Scalar.Type = Scalar.self)
where Scalar : MLTensorScalar

    /// Creates a tensor with all scalars
set to zero.
    ///
    /// - Parameters:
    ///   - shape: The shape of the
tensor.
    ///   - scalarType: The scalar type.
    public init<Scalar>(zeros shape:
[Int], scalarType: Scalar.Type =
Scalar.self) where Scalar :
MLTensorScalar, Scalar :
FixedWidthInteger

    /// Creates a tensor with all scalars
set to zero.
    ///
    /// - Parameters:
    ///   - shape: The shape of the
tensor.
    ///   - scalarType: The scalar type.
    public init<Scalar>(zeros shape:
[Int], scalarType: Scalar.Type =
Scalar.self) where Scalar :
MLTensorScalar, Scalar :
```

```
BinaryFloatingPoint,
Scalar.RawSignificand : FixedWidthInteger

    /// Creates a tensor with all scalars
set to one.
    ///
    /// - Parameters:
    ///   - shape: The shape of the
tensor.
    ///   - scalarType: The scalar type.
    public init<Scalar>(ones shape:
[Int], scalarType: Scalar.Type =
Scalar.self) where Scalar :
MLTensorScalar, Scalar :
FixedWidthInteger

    /// Creates a tensor with all scalars
set to ones.
    ///
    /// - Parameters:
    ///   - shape: The shape of the
tensor.
    ///   - scalarType: The scalar type.
    public init<Scalar>(ones shape:
[Int], scalarType: Scalar.Type =
Scalar.self) where Scalar :
MLTensorScalar, Scalar :
BinaryFloatingPoint,
Scalar.RawSignificand : FixedWidthInteger
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
```

```
extension MLTensor {

    /// Creates a one-dimensional tensor representing a sequence from a starting value to, but not including, an end value, stepping by the
    /// specified amount.
    ///
    /// - Parameters:
    ///   - start: The starting value to use for the sequence. If the sequence contains any values, the first one is `start`.
    ///   - end: An end value to limit the sequence. `end` is never an element of the resulting sequence.
    ///   - stride: The amount to step by with each iteration. `stride` must be positive.
    ///   - scalarType: The scalar type.
    @inlinable public init(rangeFrom start: Float, to end: Float, by stride: Float.Stride)

    /// Creates a one-dimensional tensor representing a sequence from a starting value to, but not including, an end value, stepping by the
    /// specified amount.
    ///
    /// - Parameters:
    ///   - start: The starting value to use for the sequence. If the sequence
```

```
contains any values, the first one is
`start`.
    ///   - end: An end value to limit
the sequence. `end` is never an element
of the resulting sequence.
    ///   - stride: The amount to step by
with each iteration. `stride` must be
positive.
    ///   - scalarType: The scalar type.
    @inlinable public
init<Scalar>(rangeFrom start: Scalar, to
end: Scalar, by stride: Scalar.Stride,
scalarType: Scalar.Type = Scalar.self)
where Scalar : MLTensorScalar, Scalar :
Strideable

    /// Creates a one-dimensional tensor
representing a sequence from a starting
value, up to and including an end value,
spaced evenly to
    /// generate the number of values
specified.
    ///
    /// - Parameters:
    ///   - start: The starting value to
use for the sequence. If the sequence
contains any values, the first one is
`start`.
    ///   - end: An end value to limit
the sequence. `end` is the last element
of the resulting sequence.
    ///   - count: The number of values
in the resulting sequence. `count` must
```

```
be positive and greater than `1`.
    public init(linearSpaceFrom start:
Float, through end: Float, count: Int)

    /// Creates a one-dimensional tensor
representing a sequence from a starting
value, up to and including an end value,
spaced evenly to
    /// generate the number of values
specified.
    ///
    /// - Parameters:
    ///   - start: The starting value to
use for the sequence. If the sequence
contains any values, the first one is
`start`.
    ///   - end: An end value to limit
the sequence. `end` is the last element
of the resulting sequence.
    ///   - count: The number of values
in the resulting sequence. `count` must
be greater than `1`.
    ///   - scalarType: The scalar type.
    public init<Scalar>(linearSpaceFrom
start: Scalar, through end: Scalar,
count: Int, scalarType: Scalar.Type =
Scalar.self) where Scalar :
MLTensorScalar, Scalar :
BinaryFloatingPoint
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
```

```
extension MLTensor :
ExpressibleByFloatLiteral {

    /// Creates an instance initialized
to the specified floating-point value.
    ///
    /// Do not call this initializer
directly. Instead, initialize a variable
or
    /// constant using a floating-point
literal. For example:
    ///
    ///     let x = 21.5
    ///
    /// In this example, the assignment
to the `x` constant calls this
    /// floating-point literal
initializer behind the scenes.
    ///
    /// - Parameter value: The value to
create.
    public init(floatLiteral: Float)

    /// A type that represents a
floating-point literal.
    ///
    /// Valid types for
`FloatLiteralType` are `Float`, `Double`,
and `Float80`
    /// where available.
    @available(iOS 18.0, tvOS 18.0,
watchOS 11.0, visionOS 2.0, macOS 15.0,
*)
```

```
    public typealias FloatLiteralType =
Float
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
extension MLTensor :
ExpressibleByIntegerLiteral {

    /// Creates an instance initialized
to the specified integer value.
    ///
    /// Do not call this initializer
directly. Instead, initialize a variable
or
    /// constant using an integer
literal. For example:
    ///
    ///     let x = 23
    ///
    /// In this example, the assignment
to the `x` constant calls this integer
    /// literal initializer behind the
scenes.
    ///
    /// - Parameter value: The value to
create.
    public init(integerLiteral: Int32)

    /// A type that represents an integer
literal.
    ///
    /// The standard library integer and
```

```
floating-point types are all valid types
    /// for `IntegerLiteralType`.
    @available(iOS 18.0, tvOS 18.0,
watchOS 11.0, visionOS 2.0, macOS 15.0,
*)
    public typealias IntegerLiteralType =
Int32
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
extension MLTensor :
ExpressibleByBooleanLiteral {

    /// Creates an instance initialized
to the given Boolean value.
    ///
    /// Do not call this initializer
directly. Instead, initialize a variable
or
    /// constant using one of the Boolean
literals `true` and `false`. For
    /// example:
    ///
    ///     let twasBrillig = true
    ///
    /// In this example, the assignment
to the `twasBrillig` constant calls this
    /// Boolean literal initializer
behind the scenes.
    ///
    /// - Parameter value: The value of
the new instance.
```

```
    public init(booleanLiteral: Bool)

    /// A type that represents a Boolean
literal, such as `Bool`.
    @available(iOS 18.0, tvOS 18.0,
watchOS 11.0, visionOS 2.0, macOS 15.0,
*)
    public typealias BooleanLiteralType =
Bool
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
extension MLTensor {

    /// Creates a tensor from a ML shaped
array.
    ///
    /// - Parameter shapedArray: The
shaped array.
    public init<ShapedArray>(_
shapedArray: ShapedArray) where
ShapedArray : MLShapedArrayProtocol,
ShapedArray.Scalar : MLTensorScalar
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
extension MLTensor {

    /// Creates a tensor by copying the
given block of data.
    ///
```

```
    /// - Parameters:
    ///   - shape: The shape of the
tensor.
    ///   - data: The block of data that
holds the contents of the tensor whose
data is expected to zero-offset, stored
in a C-contiguous
    ///     (aka row major) way, and
alignment to match the alignment of the
scalar type.
    ///   - scalarType: The scalar type.
    public init(shape: [Int], data: Data,
scalarType: any MLTensorScalar.Type)

    /// Creates a tensor with memory
content without copying the bytes.
    ///
    /// - Parameters:
    ///   - bytes: A pointer to the C-
contiguous memory address for the tensor.
Expecting the data to be zero-offset,
alignment to match
    ///     the alignment of the scalar
type, and byte count to be equal or
greater than the given shape's contiguous
size.
    ///   - shape: The shape of the
tensor.
    ///   - scalarType: The scalar type.
    ///   - deallocator: Specifies the
mechanism to free the indicated buffer.
    public init(bytesNoCopy bytes:
UnsafeRawBufferPointer, shape: [Int],
```

```
scalarType: any MLTensorScalar.Type,
deallocator: Data.Deallocator)
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
extension MLTensor {

    /// The number of dimensions of the
tensor.
    ///
    /// Rank is equal to the number of
dimensions of the shape, i.e.,
`tensor.rank == tensor.shape.count`.
    public var rank: Int { get }

    /// The shape of the tensor.
    ///
    /// For example, 2 x 3 matrix may be
represented as a tensor with the shape of
`[2, 3]`.
    public var shape: [Int] { get }

    /// The number of scalar elements in
the tensor.
    public var scalarCount: Int { get }

    /// A Boolean value indicating
whether the tensor is a scalar (when the
`rank` is equal to `0`) or not.
    public var isScalar: Bool { get }

    /// The type of scalars in the
```

```
tensor.
    public var scalarType: any
MLTensorScalar.Type { get }
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
extension MLTensor :
CustomStringConvertible {

    /// A textual representation of this
instance.
    ///
    /// Calling this property directly is
discouraged. Instead, convert an
    /// instance of any type to a string
by using the `String(describing:)`
    /// initializer. This initializer
works with any type, and uses the custom
    /// `description` property for types
that conform to
    /// `CustomStringConvertible`:
    ///
    ///     struct Point:
CustomStringConvertible {
    ///         let x: Int, y: Int
    ///
    ///         var description: String {
    ///             return "(\(x), \(y))"
    ///         }
    ///     }
    ///
    ///     let p = Point(x: 21, y: 30)
```

```
    ///     let s = String(describing: p)
    ///     print(s)
    ///     // Prints "(21, 30)"
    ///
    /// The conversion of `p` to a string
in the assignment to `s` uses the
    /// `Point` type's `description`
property.
    public var description: String {
get }
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
extension MLTensor : CustomReflectable {

    /// The custom mirror for `MLTensor`.
    public var customMirror: Mirror { get
}
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
extension MLTensor {

    /// Returns a materialized
representation of the tensor.
    ///
    /// - Returns: A `MLShapedArray` with
the contents of the tensor.
    public func shapedArray<Scalar>(of
scalarType: Scalar.Type) async ->
MLShapedArray<Scalar> where Scalar :
```

```
MLShapedArrayScalar, Scalar :
MLTensorScalar
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
extension MLTensor.ResizeMethod :
CustomStringConvertible {

    /// A textual representation of this
instance.
    ///
    /// Calling this property directly is
discouraged. Instead, convert an
    /// instance of any type to a string
by using the `String(describing:)`
    /// initializer. This initializer
works with any type, and uses the custom
    /// `description` property for types
that conform to
    /// `CustomStringConvertible`:
    ///
    ///     struct Point:
CustomStringConvertible {
    ///         let x: Int, y: Int
    ///
    ///         var description: String {
    ///             return "(\(x), \(y))"
    ///         }
    ///     }
    ///
    ///     let p = Point(x: 21, y: 30)
    ///     let s = String(describing: p)
```

```
    ///     print(s)
    ///     // Prints "(21, 30)"
    ///
    /// The conversion of `p` to a string
in the assignment to `s` uses the
    /// `Point` type's `description`
property.
    public var description: String {
get }
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
extension MLTensor.PaddingMode :
CustomStringConvertible {

    /// A textual representation of this
instance.
    ///
    /// Calling this property directly is
discouraged. Instead, convert an
    /// instance of any type to a string
by using the `String(describing:)`
    /// initializer. This initializer
works with any type, and uses the custom
    /// `description` property for types
that conform to
    /// `CustomStringConvertible`:
    ///
    ///     struct Point:
CustomStringConvertible {
    ///         let x: Int, y: Int
    ///
```

```
    ///         var description: String {
    ///             return "(\(x), \(y))"
    ///         }
    ///     }
    ///
    ///     let p = Point(x: 21, y: 30)
    ///     let s = String(describing: p)
    ///     print(s)
    ///     // Prints "(21, 30)"
    ///
    /// The conversion of `p` to a string
in the assignment to `s` uses the
    /// `Point` type's `description`
property.
    public var description: String {
get }
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
extension MLTensor.IndexPath : Sendable {
}

/// A type that can be used to slice a
dimension of a tensor. Don't use this
type directly.
@available(macOS 15.0, iOS 18.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
public protocol MLTensorRangeExpression :
Sendable {
}

@available(macOS 15.0, iOS 18.0, tvOS
```

```
18.0, watchOS 11.0, visionOS 2.0, *)
extension MLTensorRangeExpression where
Self == _MLTensorRange {

    /// The same as the ellipsis literal
`...` used to indicate unspecified
dimensions of the tensor.
    ///
    /// For example:
    /// ```swift
    /// let x = MLTensor(randomNormal:
[1, 3, 28, 28], scalarType: Float.self)
    /// let y = x[..., 0] // or
x[.fillAll, 0]
    /// y.shape // is [1, 3, 28]
    /// ```
    public static var fillAll: any
MLTensorRangeExpression { get }

    /// Expand the tensor at the
specified dimension.
    ///
    /// For example:
    /// ```swift
    /// let x = MLTensor(randomNormal:
[1, 3, 28, 28], scalarType: Float.self)
    /// let y = x[.newAxis, ...]
    /// y.shape // is [1, 1, 3, 28, 28]
    /// ```
    public static var newAxis: any
MLTensorRangeExpression { get }

    /// Squeeze the tensor at the
```

```
specified dimension.
    ///
    /// For example:
    /// ```swift
    /// let x = MLTensor(randomNormal:
[1, 3, 28, 28], scalarType: Float.self)
    /// let y = x[.squeezeAxis, ...]
    /// y.shape // is [3, 28, 28]
    /// ```
    public static var squeezeAxis: any
MLTensorRangeExpression { get }

    /// Slice the tensor at the specified
dimension.
    ///
    /// For example:
    /// ```swift
    /// let x = MLTensor(randomNormal:
[1, 3, 28, 28], scalarType: Float.self)
    /// let y = x[..., 0] // or
x[..., .index(0)]
    /// y.shape // is [1, 3, 28]
    /// ```
    public static func index(_ index:
Int) -> any MLTensorRangeExpression

    /// Slice the tensor at the specified
dimension.
    ///
    /// ```swift
    /// let x = MLTensor(randomNormal:
[1, 3, 28, 28], scalarType: Float.self)
    /// let y = x[..., 0..<2] // or
```

```
x[..., .range(0..<2, stride: 1)]
    /// y.shape // is [1, 3, 28, 2]
    /// ```
    public static func range(_ range:
Range<Int>, stride: Int = 1) -> any
MLTensorRangeExpression

    /// Slice the tensor at the specified
dimension.
    ///
    /// For example:
    /// ```swift
    /// let x = MLTensor(randomNormal:
[1, 3, 28, 28], scalarType: Float.self)
    /// let y = x[..., 0...2] // or
x[..., .closedRange(0...2, stride: 1)]
    /// y.shape // is [1, 3, 28, 3]
    /// ```
    public static func closedRange(_
range: ClosedRange<Int>, stride: Int = 1)
-> any MLTensorRangeExpression

    /// Slice the tensor at the specified
dimension.
    ///
    /// For example:
    /// ```swift
    /// let x = MLTensor(randomNormal:
[1, 3, 28, 28], scalarType: Float.self)
    /// let y = x[..., 1...] // or x[...,
.partialRangeFrom(1..., stride: 1)]
    /// y.shape // is [1, 3, 28, 27]
    /// ```
```

```
    public static func partialRangeFrom(_
range: PartialRangeFrom<Int>, stride: Int
= 1) -> any MLTensorRangeExpression

    /// Slice the tensor at the specified
dimension.
    ///
    /// For example:
    /// ```swift
    /// let x = MLTensor(randomNormal:
[1, 3, 28, 28], scalarType: Float.self)
    /// let y = x[..., ..<3] // or x[...,
.partialRangeUpTo(..<3, stride: 1)]
    /// y.shape // is [1, 3, 28, 3]
    /// ```
    public static func partialRangeUpTo(_
range: PartialRangeUpTo<Int>, stride: Int
= 1) -> any MLTensorRangeExpression

    /// Slice the tensor at the specified
dimension.
    ///
    /// For example:
    /// ```swift
    /// let x = MLTensor(randomNormal:
[1, 3, 28, 28], scalarType: Float.self)
    /// let y = x[..., ...3] // or x[...,
.partialRangeThrough(...3, stride: 1)]
    /// y.shape // is [1, 3, 28, 4]
    /// ```
    public static func partialRangeUpTo(_
range: PartialRangeThrough<Int>, stride:
Int = 1) -> any MLTensorRangeExpression
```

```
}

/// A type that represents the tensor
scalar types supported by the framework.
Don't use this type directly.
@available(macOS 15.0, iOS 18.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
public protocol MLTensorScalar : Sendable
{
}

/// Computes the element-wise maximum of
two tensors.
///
/// For example:
/// ```swift
/// let x = MLTensor([1.0, 3.0, 6.0])
/// let y = MLTensor([6.0, 3.0, 1.0])
/// let z = pointwiseMax(x, y)
/// await z.shapedArray(of:
Float.self) // is [6.0, 3.0, 6.0]
/// ```
///
/// Shapes must be broadcastable, where
the broadcasted shape becomes the shape
of the output.
@available(macOS 15.0, iOS 18.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
public func pointwiseMax(_ lhs: MLTensor,
_ rhs: MLTensor) -> MLTensor

/// Computes the element-wise minimum of
two tensors.
```

```
///
/// For example:
/// ```swift
/// let x = MLTensor([1.0, 3.0, 6.0])
/// let y = MLTensor([6.0, 3.0, 1.0])
/// let z = pointwiseMin(x, y)
/// await z.shapedArray(of:
Float.self) // is [1.0, 3.0, 1.0]
/// ```
///
/// Shapes must be broadcastable, where
the broadcasted shape becomes the shape
of the output.
@available(macOS 15.0, iOS 18.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
public func pointwiseMin(_ lhs: MLTensor,
_ rhs: MLTensor) -> MLTensor

/// Calls the given closure within a
task-local context using the specified
compute policy to influence what compute
device tensor
/// operations are executed on.
///
/// - Parameters:
///   - computePolicy: A compute policy
that will be set before the closure gets
called and restored after the closure
returns.
///   - body: A nullary closure. If the
closure has a return value, that value is
also used as the return value of the
///     `withMLTensorComputePolicy(_:_:)`
```

```
function.
/// - Returns: The return value, if any,
of the `body` closure.
@available(macOS 15.0, iOS 18.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
public func
withMLTensorComputePolicy<Result>(_
computePolicy: MLComputePolicy, _ body:
() throws -> Result) rethrows -> Result

/// Calls the given closure within a
task-local context using the specified
compute policy to influence what compute
device tensor
/// operations are executed on.
///
/// - Parameters:
///   - computePolicy: A compute policy
that will be set before the closure gets
called and restored after the closure
returns.
///   - body: A nullary closure. If the
closure has a return value, that value is
also used as the return value of the
///     `withMLTensorComputePolicy(_:_:)`
function.
/// - Returns: The return value, if any,
of the `body` closure.
@available(macOS 15.0, iOS 18.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
public func
withMLTensorComputePolicy<R>(_
computePolicy: MLComputePolicy, _ body:
```

```
() async throws -> R) async rethrows -> R

@available(macOS 12.0, iOS 15.0, watchOS
8.0, tvOS 15.0, *)
extension MLFeatureValue {

    /// Constructs from MLShapedArray.
    ///
    /// The value will be semantically
copied (i.e. Copy-on-Write) to
MLFeatureValue object. A
    /// mutation to MLFeatureValue won't
affect the source MLShapedArray and vise
versa.
    ///
    /// - Parameter shapedArray The
MLShapedArray object.
    public convenience
init<Scalar>(shapedArray:
MLShapedArray<Scalar>) where Scalar :
MLShapedArrayScalar

    /// Returns MLShapedArray of the
specified scalar type.
    ///
    /// The function returns non-nil
value when the feature value has
MLMultiArray with the
    /// specified type.
    ///
    /// - Parameter type: The scalar type
    public func
shapedArrayValue<Scalar>(of type:
```

```
Scalar.Type) -> MLShapedArray<Scalar>?
where Scalar : MLShapedArrayScalar
}

extension MLFeatureValue {

    /// Creates a feature value from a
sendable feature value.
    @available(macOS 15.0, iOS 18.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
    public convenience init(_ value:
MLSendableFeatureValue)
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
extension Int : MLTensorRangeExpression {
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
extension Range : MLTensorRangeExpression
where Bound == Int {
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
extension ClosedRange :
MLTensorRangeExpression where Bound ==
Int {
}

@available(macOS 15.0, iOS 18.0, tvOS
```

```
18.0, watchOS 11.0, visionOS 2.0, *)
extension PartialRangeFrom :
MLTensorRangeExpression where Bound ==
Int {
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
extension PartialRangeUpTo :
MLTensorRangeExpression where Bound ==
Int {
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
extension PartialRangeThrough :
MLTensorRangeExpression where Bound ==
Int {
}

@available(macOS 10.15, iOS 13.0, watchOS
6.0, tvOS 13.0, *)
extension MLMultiArray {

    /// An MLMultiArray constructed with
the FixedWidthInteger elements of the
collection
    /// converted to Int32.
    /// ~~~~
    /// let v:[Int32] = [3, 2, 1]
    /// let m = try MLMultiArray(v)
    /// print(m)
    /// Int32 3 vector
```

```
    /// [3,2,1]
    /// ~~~~
    ///
    /// This initializer will trap if
called with data containing
FixedWidthInteger elements
    /// that cannot be safely converted
to Int32, but it is safe to use with
    /// wider types so long as the actual
data is within range.
    /// ~~~~
    /// let a = try
MLMultiArray([Int.max]) // trap!
    /// let b = try
MLMultiArray([Int(Int32.max),
Int(Int32.min)]) // This is fine.
    /// ~~~~
    public convenience init<C>(_ data: C)
throws where C : Collection, C.Element :
FixedWidthInteger

    /// An MLMultiArray constructed with
the elements of a collection
    /// whose elements are type Float.
    /// ~~~~
    /// let v:[Float] = [3.14159,
2.71828, 1.61803]
    /// let m = try MLMultiArray(v)
    /// print(m)
    /// Float32 3 vector
    /// [3.14159,2.71828,1.61803]
    /// ~~~~
    public convenience init<C>(_ data: C)
```

```
throws where C : Collection, C.Element ==
Float

    /// An MLMultiArray constructed with
the elements of a collection
    /// whose elements are type Double.
    /// ~~~~
    /// let v:[Double] = [3.14159,
2.71828, 1.61803]
    /// let m = try MLMultiArray(v)
    /// print(m)
    /// Double 3 vector
    /// [3.14159,2.71828,1.61803]
    /// ~~~~
    public convenience init<C>(_ data: C)
throws where C : Collection, C.Element ==
Double

    /// Creates the object with specified
strides.
    ///
    /// The contents of the object are
left uninitialized; the client must
initialize it.
    ///
    /// - Parameters:
    ///   - shape: The shape
    ///   - dataType: The data type
    ///   - strides: The strides.
    @available(macOS 15.0, iOS 18.0,
watchOS 11.0, tvOS 18.0, visionOS 2.0, *)
    public convenience init(shape: [Int],
dataType: MLMultiArrayDataType, strides:
```

```
[Int])

    /// An MLMultiArray constructed from
MLShapedArray or its slice.
    /// ~~~~
    /// let s =
MLShapedArray<Int32>(scalars: 0 ..< 6,
shape: [3, 2])
    /// let m = try MLMultiArray(s)
    /// print(m)
    /// Int32 3 x 2 matrix
    /// [[0, 1]
    ///  [2, 3]
    ///  [4, 5]]
    /// ~~~~
    @available(macOS 12.0, iOS 15.0,
watchOS 8.0, tvOS 15.0, *)
    public convenience
init<ShapedArray>(_ shapedArray:
ShapedArray) where ShapedArray :
MLShapedArrayProtocol

    /// Calls a given closure with a raw
pointer to the multi array's storage.
    ///
    /// ```
    /// let A = try MLMultiArray(shape:
[3, 2], dataType: .int32)
    /// A[[1, 2]] = 42
    /// A.withUnsafeBytes { bytes in
    ///     let scalarBuffer =
bytes.bindMemory(to: Int32.self)
    ///     let strideY =
```

```
A.strides[0].intValue
    ///     // Print 42
    ///     print("Scalar at (1, 2): \
(scalarBuffer[1 * strideY + 2])")
    /// }
    /// ```
    ///
    /// - Parameter body: A closure with
an `UnsafeRawBufferPointer` parameter
that points to the
    ///   storage for the multi array. If
body has a return value, that value is
also used as the
    ///   return value for the
`withUnsafeBytes(_:)` method. The pointer
argument is
    ///   valid only for the duration of
the method's execution.
    /// - Parameter ptr: The pointer to
the memory buffer of the multi array.
    @available(macOS 12.3, iOS 15.4,
watchOS 8.5, tvOS 15.4, *)
    public func withUnsafeBytes<R>(_
body: (_ ptr: UnsafeRawBufferPointer)
throws -> R) rethrows -> R

    /// Calls a given closure with a raw
pointer to the multi array’s mutable
storage.
    ///
    /// Use strides passed to the closure
to access the buffer. This is a mutating
function, which
```

```
    /// may change the underlying buffer
layoout.
    ///
    /// ```
    /// let A = try MLMultiArray(shape:
[3, 2], dataType: .int32)
    /// A.withUnsafeMutableBytes { bytes,
strides in
    ///     let scalarBuffer =
bytes.bindMemory(to: Int32.self)
    ///     let strideY = strides[0]
    ///     scalarBuffer[1 * strideY + 2]
= 42  // Set 42 at A[1, 2]
    /// }
    /// ```
    ///
    /// - Parameter body: A closure with
an `UnsafeRawBufferPointer` parameter
that points to the
    ///   storage for the multi array. If
body has a return value, that value is
also used as the
    ///   return value for the
`withUnsafeMutableBytes(_:)` method. The
pointer argument is
    ///   valid only for the duration of
the method's execution.
    /// - Parameter ptr: The pointer to
the memory buffer of the multi array.
    /// - Parameter strides: The strides
of the buffer in number of scalars, not
bytes.
    @available(macOS 12.3, iOS 15.4,
```

```
watchOS 8.5, tvOS 15.4, *)
    public func
withUnsafeMutableBytes<R>(_ body: (_ ptr:
UnsafeMutableRawBufferPointer, _ strides:
[Int]) throws -> R) rethrows -> R

    /// Calls a given closure with a raw
pointer to the multi array's storage.
    ///
    /// ```
    /// let A = try MLMultiArray(shape:
[3, 2], dataType: .int32)
    /// A[[1, 2]] = 42
    /// A.withUnsafeBufferPointer(ofType:
Int32.self) { scalarBuffer in
    ///     let strideY =
A.strides[0].intValue
    ///     // Print 42
    ///     print("Scalar at (1, 2): \
(scalarBuffer[1 * strideY + 2])")
    /// }
    /// ```
    ///
    /// - Parameter body: A closure with
an `UnsafeBufferPointer` parameter that
points to the
    ///   storage for the multi array. If
body has a return value, that value is
also used as the
    ///   return value for the
`withUnsafeBufferPointer(_:)` method. The
pointer argument is
    ///   valid only for the duration of
```

```
the method's execution.
    /// - Parameter ptr: The pointer to
the memory buffer of the multi array.
    @available(macOS 12.3, iOS 15.4,
watchOS 8.5, tvOS 15.4, *)
    public func
withUnsafeBufferPointer<S, R>(ofType
type: S.Type, _ body: (_ ptr:
UnsafeBufferPointer<S>) throws -> R)
rethrows -> R where S :
MLShapedArrayScalar

    /// Calls a given closure with a
buffer pointer to the multi array's
mutable storage.
    ///
    /// Use strides passed to the closure
to access the buffer. This is a mutating
function, which
    /// may change the underlying buffer
layoout.
    ///
    /// ```
    /// let A = try MLMultiArray(shape:
[3, 2], dataType: .int32)
    ///
A.withUnsafeMutableBufferPointer(ofType:
Int32.self) { scalarBuffer, strides in
    ///     let strideY = strides[0]
    ///     scalarBuffer[1 * strideY + 2]
= 42  // Set 42 at A[1, 2]
    /// }
    /// ```
```

```
    ///
    /// - Parameter body: A closure with
an `UnsafeRawBufferPointer` parameter
that points to the
    ///   storage for the multi array. If
body has a return value, that value is
also used as the
    ///   return value for the
`withUnsafeMutableBufferPointer(_:)`
method. The pointer argument is
    ///   valid only for the duration of
the method's execution.
    /// - Parameter ptr: The pointer to
the memory buffer of the multi array.
    /// - Parameter strides: The strides
of the buffer in number of scalars, not
bytes.
    @available(macOS 12.3, iOS 15.4,
watchOS 8.5, tvOS 15.4, *)
    public func
withUnsafeMutableBufferPointer<S,
R>(ofType type: S.Type, _ body: (_ ptr:
UnsafeMutableBufferPointer<S>, _ strides:
[Int]) throws -> R) rethrows -> R where S
: MLShapedArrayScalar
}

@available(macOS 10.15, iOS 13.0, watchOS
6.0, tvOS 13.0, *)
@available(macOS, deprecated: 100000.0,
message: "Use .dataPointer and
withExtendedLifetime instead")
@available(iOS, deprecated: 100000.0,
```

```
message: "Use .dataPointer and
withExtendedLifetime instead")
@available(watchOS, deprecated: 100000.0,
message: "Use .dataPointer and
withExtendedLifetime instead")
@available(tvOS, deprecated: 100000.0,
message: "Use .dataPointer and
withExtendedLifetime instead")
extension UnsafeBufferPointer {

    public init(_ multiArray:
MLMultiArray) throws
}

@available(macOS 10.15, iOS 13.0, watchOS
6.0, tvOS 13.0, *)
@available(macOS, deprecated: 100000.0,
message: "Use .dataPointer and
withExtendedLifetime instead")
@available(iOS, deprecated: 100000.0,
message: "Use .dataPointer and
withExtendedLifetime instead")
@available(watchOS, deprecated: 100000.0,
message: "Use .dataPointer and
withExtendedLifetime instead")
@available(tvOS, deprecated: 100000.0,
message: "Use .dataPointer and
withExtendedLifetime instead")
extension UnsafeMutableBufferPointer {

    public init(_ multiArray:
MLMultiArray) throws
}
```

```
extension MLModelConfiguration {

    /// A group of hints for CoreML to
optimize
    @available(macOS 14.4, iOS 17.4,
watchOS 10.4, tvOS 17.4, *)
    public var optimizationHints:
MLOptimizationHints
}

@available(macOS 15.0, iOS 18.0, watchOS
11.0, tvOS 18.0, visionOS 2.0, *)
extension MLStateConstraint {

    /// The shape of the state buffer.
    @available(macOS 15.0, iOS 18.0,
watchOS 11.0, tvOS 18.0, visionOS 2.0, *)
    public var bufferShape: [Int] { get }
}

/// Extension to Int32 to implement
MLShapedArrayScalar
@available(macOS 12.0, iOS 15.0, watchOS
8.0, tvOS 15.0, *)
extension Int32 : MLShapedArrayScalar {

    /// Returns MLMultiArrayDataType enum
corresponding to the data type.
    public static var multiArrayDataType:
MLMultiArrayDataType { get }
}
```

```
/// Extension to Float64 to implement
MLShapedArrayScalar
@available(macOS 12.0, iOS 15.0, watchOS
8.0, tvOS 15.0, *)
extension Double : MLShapedArrayScalar {

    /// Returns MLMultiArrayDataType enum
corresponding to the data type.
    public static var multiArrayDataType:
MLMultiArrayDataType { get }
}

/// Extension to Float32 to implement
MLShapedArrayScalar
@available(macOS 12.0, iOS 15.0, watchOS
8.0, tvOS 15.0, *)
extension Float : MLShapedArrayScalar {

    /// Returns MLMultiArrayDataType enum
corresponding to the data type.
    public static var multiArrayDataType:
MLMultiArrayDataType { get }
}

/// Extension to Float16 to implement
MLShapedArrayScalar
@available(macOS 15.0, iOS 16.0, watchOS
9.0, tvOS 16.0, *)
extension Float16 : MLShapedArrayScalar {

    /// Returns MLMultiArrayDataType enum
corresponding to the data type.
    public static var multiArrayDataType:
```

```
MLMultiArrayDataType { get }
}

extension MLModel {

    /// Run a prediction on a model.
    ///
    /// This method requires all inputs
and outputs to be multidimensional
arrays. If your model doesn't satisfy
    /// this requirement, materialize the
tensor inputs to `MLShapedArray` values
to create feature
    /// values for each, for example:
    /// ```swift
    /// let shapedArray = await
tensor.shapedArray(of: Float.self)
    /// let inputFeatures = try
MLDictionaryFeatureProvider(dictionary: [
    ///     "x":
MLFeatureValue(shapedArray: shapedArray),
    ///     // Other non-multidimensional
array inputs
    /// ])
    /// let prediction = try await
model.prediction(from: inputFeatures)
    /// ```
    ///
    /// - Parameter inputs: The named
input, or inputs, to make a prediction
from.
    /// - Returns: The output, or
outputs, from the prediction.
```

```
    @available(macOS 15.0, iOS 18.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
    public func prediction(from inputs:
[String : MLTensor]) async throws ->
[String : MLTensor]

    /// Run a stateful prediction on a
model.
    ///
    /// This method requires all inputs
and outputs to be multidimensional
arrays. If your model doesn't satisfy
    /// this requirement, materialize the
tensor inputs to `MLShapedArray` values
to create feature
    /// values for each, for example:
    /// ```swift
    /// let shapedArray = await
tensor.shapedArray(of: Float.self)
    /// let inputFeatures = try
MLDictionaryFeatureProvider(dictionary: [
    ///     "x":
MLFeatureValue(shapedArray: shapedArray),
    ///     // Other non-multidimensional
array inputs
    /// ])
    /// let state = model.newState()
    /// let prediction = try await
model.prediction(from: inputFeatures,
using: state)
    /// ```
    ///
    /// - Parameters:
```

```
    ///   - inputs: The named input, or
inputs, to make a prediction from.
    ///   - state: The state.
    /// - Returns: The output, or
outputs, from the prediction.
    @available(macOS 15.0, iOS 18.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
    public func prediction(from inputs:
[String : MLTensor], using state:
MLState) async throws -> [String :
MLTensor]
}

/// Slice range specified with syntax: `a
..< b`
@available(macOS 12.0, iOS 15.0, watchOS
8.0, tvOS 15.0, *)
extension Range :
MLShapedArrayRangeExpression where Bound
== Int {

    /// Returns Range<Int> for the
dimension.
    ///
    /// For example, when the range
expression specifies `1...` on the axis
with dimension 3, the
    /// resultant Range<Int> is `1 ..<
3`.
    ///
    /// - Parameter dimension: The
dimension of the axis on which the range
expression is used.
```

```
    /// - Returns: The range of the
selected dimension.
    public func
relative(toShapedArrayAxis range:
Range<Int>) -> Range<Int>
}

/// Slice range specified with syntax: `a
...`
@available(macOS 12.0, iOS 15.0, watchOS
8.0, tvOS 15.0, *)
extension PartialRangeFrom :
MLShapedArrayRangeExpression where Bound
== Int {

    /// Returns Range<Int> for the
dimension.
    ///
    /// For example, when the range
expression specifies `1...` on the axis
with dimension 3, the
    /// resultant Range<Int> is `1 ..<
3`.
    ///
    /// - Parameter dimension: The
dimension of the axis on which the range
expression is used.
    /// - Returns: The range of the
selected dimension.
    public func
relative(toShapedArrayAxis range:
Range<Int>) -> Range<Int>
}
```

```
/// Slice range specified with syntax:
`..< a`
@available(macOS 12.0, iOS 15.0, watchOS
8.0, tvOS 15.0, *)
extension PartialRangeUpTo :
MLShapedArrayRangeExpression where Bound
== Int {

    /// Returns Range<Int> for the
dimension.
    ///
    /// For example, when the range
expression specifies `1...` on the axis
with dimension 3, the
    /// resultant Range<Int> is `1 ..<
3`.
    ///
    /// - Parameter dimension: The
dimension of the axis on which the range
expression is used.
    /// - Returns: The range of the
selected dimension.
    public func
relative(toShapedArrayAxis range:
Range<Int>) -> Range<Int>
}

/// Slice range specified with syntax:
`a...b`
@available(macOS 12.0, iOS 15.0, watchOS
8.0, tvOS 15.0, *)
extension ClosedRange :
```

```
MLShapedArrayRangeExpression where Bound
== Int {

    /// Returns Range<Int> for the
dimension.
    ///
    /// For example, when the range
expression specifies `1...` on the axis
with dimension 3, the
    /// resultant Range<Int> is `1 ..<
3`.
    ///
    /// - Parameter dimension: The
dimension of the axis on which the range
expression is used.
    /// - Returns: The range of the
selected dimension.
    public func
relative(toShapedArrayAxis range:
Range<Int>) -> Range<Int>
}

/// Slice range specified with syntax:
`...a`
@available(macOS 12.0, iOS 15.0, watchOS
8.0, tvOS 15.0, *)
extension PartialRangeThrough :
MLShapedArrayRangeExpression where Bound
== Int {

    /// Returns Range<Int> for the
dimension.
    ///
```

```
    /// For example, when the range
expression specifies `1...` on the axis
with dimension 3, the
    /// resultant Range<Int> is `1 ..<
3`.
    ///
    /// - Parameter dimension: The
dimension of the axis on which the range
expression is used.
    /// - Returns: The range of the
selected dimension.
    public func
relative(toShapedArrayAxis range:
Range<Int>) -> Range<Int>
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
extension Int8 : MLTensorScalar {
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
extension Int16 : MLTensorScalar {
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
extension Int32 : MLTensorScalar {
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
```

```
extension UInt8 : MLTensorScalar {
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
extension UInt16 : MLTensorScalar {
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
extension UInt32 : MLTensorScalar {
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
extension Float16 : MLTensorScalar {
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
extension Float : MLTensorScalar {
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
extension Bool : MLTensorScalar {
}

@available(macOS 11.0, iOS 14.0, watchOS
7.0, tvOS 14.0, *)
extension MLModel {

    /// Construct a model asynchronously
```

```
given the location of its on-disk
representation and configuration.
    ///
    /// Model loading may take time when
the model content is not immediately
available (e.g. encrypted model).
    /// Use this factory method
especially when the caller is on the main
thread.
    ///
    /// - Parameter url:
Location of its on-disk representation
(.mlmodelc directory).
    /// - Parameter configuration:
Configuration The model configuration
    /// - Parameter handler:       When
the model load completes successfully or
unsuccessfully,
    ///                            the
completion handler is invoked with a
valid MLModel instance or NSError object.
    public class func load(contentsOf
url: URL, configuration:
MLModelConfiguration =
MLModelConfiguration(), completionHandler
handler: @escaping (Result<MLModel, any
Error>) -> Void)

    /// Construct a model asynchronously
given the location of its on-disk
representation and configuration.
    ///
    /// Model loading may take time when
```

```
the model content is not immediately
available (e.g. encrypted model).
    /// Use this factory method
especially when the caller is on the main
thread.
    ///
    /// - Parameter url:
Location of its on-disk representation
(.mlmodelc directory).
    /// - Parameter configuration:
Configuration The model configuration
    /// - Returns: A constructed MLModel
object.
    @available(macOS 12.0, iOS 15.0,
watchOS 8.0, tvOS 15.0, *)
    public class func load(contentsOf
url: URL, configuration:
MLModelConfiguration =
MLModelConfiguration()) async throws ->
MLModel

    /// Compile a .mlmodel asynchronously
given its on-disk location.
    ///
    /// Model compilation may take a
considerable amount of time.
    /// Use this factory method
especially when the caller is on the main
thread.
    ///
    /// - Parameter url:
Location of the .mlmodel file.
    /// - Parameter handler:       When
```

```
the model compilation completes
successfully the completion handler is
invoked
    ///                                 with a
valid URL to the compiled .mlmodelc
directory.
    ///                                 On
failure the completion handler is invoked
with an NSError object.
    ///
    /// The model is compiled to a
temporary location on disk. You must move
the compiled model to a permanent
location if you wish to keep it.
    @available(macOS 13.0, iOS 16.0, tvOS
16.0, *)
    @available(watchOS, unavailable)
    public class func compileModel(at
url: URL, completionHandler handler:
@escaping (Result<URL, any Error>) ->
Void)

    /// Using Swift 'async' idiom.
    @available(macOS 13.0, iOS 16.0, tvOS
16.0, *)
    @available(watchOS, unavailable)
    public class func compileModel(at
url: URL) async throws -> URL

    /// Creates a new state object.
    ///
    /// Core ML framework will allocate
the state buffers declared in the model.
```

```
    ///
    /// The allocated state buffers are
initialized to zeros. To initialize with
different values, use
`.withMultiArray(for:)` to get the
mutable `MLMultiArray`-view to the state
buffer.
    ///
    /// ```swift
    /// // Create state that contains two
state buffers: s1 and s2.
    /// // Then, initialize s1 to 1.0 and
s2 to 2.0.
    /// let state = model.makeState()
    /// state.withMultiArray(for: "s1") {
stateMultiArray in
    ///     stateMultiArray[0] = 1.0
    /// }
    /// state.withMultiArray(for: "s2") {
stateMultiArray in
    ///     stateMultiArray[0] = 2.0
    /// }
    /// ```
    @available(macOS 15.0, iOS 18.0,
watchOS 11.0, tvOS 18.0, visionOS 2.0, *)
    public func makeState() -> MLState

    /// Run a prediction on a model
asynchronously.
    ///
    /// - Parameter input:   The input
features to make a prediction from.
    /// - Parameter options: Optional
```

```
prediction options to modify how the
prediction is run
    /// - Returns: The output from the
prediction.
    @available(macOS 14.0, iOS 17.0,
watchOS 10.0, tvOS 17.0, *)
    public func prediction(from input:
any MLFeatureProvider, options:
MLPredictionOptions =
MLPredictionOptions()) async throws ->
any MLFeatureProvider

    /// Run a stateful prediction on a
model asynchronously.
    ///
    /// ```swift
    /// let state = model.newState()
    /// let prediction = try await
model.prediction(from: inputFeatures,
using: state)
    /// ```
    ///
    /// - Parameters:
    ///   - input:   The input features
to make a prediction from.
    ///   - state:   The state.
    ///   - options: Optional prediction
options to modify how the prediction is
run
    /// - Returns: The output from the
prediction.
    @available(macOS 15.0, iOS 18.0,
watchOS 11.0, tvOS 18.0, visionOS 2.0, *)
```

```
    public func prediction(from input:
any MLFeatureProvider, using state:
MLState, options: MLPredictionOptions =
MLPredictionOptions()) async throws ->
any MLFeatureProvider

    /// The list of available compute
devices for CoreML.
    ///
    /// Use the method to get the list of
compute devices that MLModel's prediction
can use.
    ///
    /// The hardware may have other
compute devices that are not available to
CoreML. Use
    ///
`MLComputeDevice.allComputeDevices` to
get the complete list.
    @available(macOS 14.0, iOS 17.0,
watchOS 10.0, tvOS 17.0, *)
    public static var
availableComputeDevices:
[MLComputeDevice] { get }
}

@available(macOS 15.0, iOS 18.0, watchOS
11.0, tvOS 18.0, *)
extension MLState {

    @available(*, deprecated, message:
"Use withMultiArray(for:) instead.")
    public func withMultiArray<R>(_ body:
```

```
(MLMultiArray) -> R) throws -> R

    @available(macOS 15.0, iOS 18.0,
watchOS 11.0, tvOS 18.0, *)
    public func withMultiArray<R>(for
stateName: String, _ body: (MLMultiArray)
throws -> R) rethrows -> R
}
```