MAX changelog

This page describes all the changes in each version of the MAX platform.

See how to update MAX with `magic`.

> **Switch to Magic**
>
> The `modular` command-line tool is deprecated (see how to uninstall it). We recommend that you now manage your packages with `magic`, but you can also use conda.

v24.5 (2024-09-13)

🔥 Legendary

- Mojo and MAX are magical! We've created a new package and virtual environment manager, `magic`, for MAX and Mojo. Check it out!
- New Llama3.1 pipeline built with the new MAX Graph Python API.
- We have not one, but two new Python APIs that we're introducing in this release:
  - MAX Graph Python API
  - MAX Driver Python API

⭐ New

- Added `repeat_interleave` graph op.
- Added caching for MAX graph models. This means that graph compilation is cached and the executable model is retrieved from cache on the 2nd and subsequent runs. Note that the model cache is architecture specific and isn't portable across different targets.
- Support for Python 3.12.

MAX Graph Python API

This Python API will ultimately provide the same low-level programming interface for high-performance inference graphs as the Mojo API. As with the Mojo API, it's an API for graph-building only, and it does not implement support

for training.

You can take a look at how the API works in the MAX Graph Python API reference.

MAX Driver Python API

The MAX Driver API allows you to interact with devices (such as CPUs and GPUs) and allocate memory directly onto them. With this API, you interact with this memory as tensors.

Note that this API is still under development, with support for non-host devices, such as GPUs, planned for a future release.

To learn more, check out the MAX Driver Python API reference.

MAX C API

New APIs for adding torch metadata libraries:

- `M_setTorchMetadataLibraryPath`
- `M_setTorchMetadataLibraryPtr`

🦋 Changed

MAX Engine performance

- Compared to v24.4, MAX Engine v24.5 generates tokens for Llama an average of 15%-48% faster.

MAX C API

Simplified the API for adding torch library paths, which now only takes one path per API call, but can be called multiple times to add paths to the config:

- `M_setTorchLibraries -> M_setTorchLibraryPath`

⚠️ Deprecated

- The `max` command line tool is no longer supported and will be removed in a future release.

❌ Removed

- Dropped support for Ubuntu 20.04. If you're using Ubuntu, we currently support Ubuntu 22.04 LTS only.
- Dropped support for Python 3.8.

- Removed built-in PyTorch libraries from the max package. See the FAQ for information on supported torch versions.

v24.4 (2024-06-07)

🔥 Legendary

- MAX is now available on macOS! Try it now.
- New quantization APIs for MAX Graph. You can now build high-performance graphs in Mojo that use the latest quantization techniques, enabling even faster performance and more system compatibility for large models.

  Learn more in the guide to quantize your graph weights.

⭐ New

MAX Mojo APIs

- Added AI pipeline examples in the `max` repo, with Mojo implementations for common transformer layers, including quantization support.
  - New Llama3 pipeline built with MAX Graph.
  - New Replit Code pipeline built with MAX Graph.
  - New TinyStories pipeline (based on TinyLlama) that offers a simple demo of the MAX Graph quantization API.
- Added Mojo API inference example with the TorchScript BERT model.
- Added `max.graph.checkpoint` package to save and load model weights.

  All weights are stored in a `TensorDict`. You can save and load a `TensorDict` to disk with `save()` and `load()` functions.
- Added MAX Graph quantization APIs:
  - Added quantization encodings `BFloat16Encoding`, `Q4_0Encoding`, `Q4_KEncoding`, and `Q6_KEncoding`.
  - Added the `QuantizationEncoding` trait so you can build custom quantization encodings.
  - Added `Graph.quantize()` to create a quantized tensor node.
  - Added `qmatmul()` to perform matrix-multiplication with a float32 and a quantized matrix.
- Added some MAX Graph ops:
  - `avg_pool()`
  - `max_pool()`
  - `conv2d()`

    - conv3d()
    - layer_norm()
    - tile()
    - select()
- Added a layer() context manager and current_layer() function to aid in debugging during graph construction. For example:

```
with graph.layer("foo"):
    with graph.layer("bar"):
        print(graph.current_layer())  # prints "foo.bar"
        x = graph.constant[DType.int64](1)
        graph.output(x)
```

  This adds a path `foo.bar` to the added nodes, which will be reported during errors.
- Added format_system_stack() function to format the stack trace, which we use to print better error messages from error().
- Added TensorMap.keys() to get all the tensor key names.

MAX C API

Miscellaneous new APIs:

- `M_cloneCompileConfig()`
- `M_copyAsyncTensorMap()`
- `M_tensorMapKeys()` and `M_deleteTensorMapKeys()`
- `M_setTorchLibraries()`

🦋 Changed

MAX Mojo API

- EngineNumpyView.data() and EngineTensorView.data() functions that return a type-erased pointer were renamed to `unsafe_ptr()`.
- TensorMap now conforms to `CollectionElement` trait to be copyable and movable.
- `custom_nv()` was removed, and its functionality moved into custom() as an function overload, so it can now output a list of tensor symbols.

v24.3 (2024-05-02)

🔥 Legendary

- You can now write custom ops for your models with Mojo!

  Learn more about MAX extensibility.

🦋 Changed

- Added support for named dynamic dimensions. This means you can specify when two or more dimensions in your model's input are dynamic but their sizes at run time must match each other. By specifying each of these dimension sizes with a name (instead of using `None` to indicate a dynamic size), the MAX Engine compiler can perform additional optimizations. See the notes below for the corresponding API changes that support named dimensions.
- Simplified all the APIs to load input specs for models, making them more consistent.

MAX Engine performance

- Compared to v24.2, MAX Engine v24.3 shows an average speedup of 10% on PyTorch models, and an average 20% speedup on dynamically quantized ONNX transformers.

MAX Graph API

The `max.graph` APIs are still changing rapidly, but starting to stabilize.

See the updated guide to build a graph with MAX Graph.

- `AnyMoType` renamed to `Type`, `MOTensor` renamed to `TensorType`, and `MOList` renamed to `ListType`.
- Removed `ElementType` in favor of using `DType`.
- Removed `TypeTuple` in favor of using `List[Type]`.
- Removed the `Module` type so you can now start building a graph by directly instantiating a `Graph`.
- Some new ops in `max.ops`, including support for custom ops.

  See how to create a custom op in MAX Graph.

MAX Engine Python API

- Redesigned `InferenceSession.load()` to replace the confusing `options` argument with a `custom_ops_path` argument for use when loading a custom op, and an `input_specs` argument for use when loading TorchScript models.

  As a result, `CommonLoadOptions`, `TorchLoadOptions`, and `TensorFlowLoadOptions` have all been removed.
- `TorchInputSpec` now supports named dynamic dimensions (previously, dynamic dimension sizes could be specified only as `None`). This lets you tell MAX which dynamic dimensions are required to have the same size,

which helps MAX better optimize your model.

MAX Engine Mojo API

- `InferenceSession.load_model()` was renamed to `load()`.
- Redesigned `InferenceSession.load()` to replace the confusing `config` argument with a `custom_ops_path` argument for use when loading a custom op, and an `input_specs` argument for use when loading TorchScript models.

  Doing so removed `LoadOptions` and introduced the new `InputSpec` type to define the input shape/type of a model (instead of `LoadOptions`).
- New `ShapeElement` type to allow for named dynamic dimensions (in `InputSpec`).
- `max.engine.engine` module was renamed to `max.engine.info`.

MAX Engine C API

- `M_newTorchInputSpec()` now supports named dynamic dimensions (via new `dimNames` argument).

❌ Removed

- Removed TensorFlow support in the MAX SDK, so you can no longer load a TensorFlow SavedModel for inference. However, TensorFlow is still available for enterprise customers.

  We removed TensorFlow because industry-wide TensorFlow usage has declined significantly, especially for the latest AI innovations. Removing TensorFlow also cuts our package size by over 50% and accelerates the development of other customer-requested features. If you have a production use-case for a TensorFlow model, please contact us.
- Removed the Python `CommonLoadOptions`, `TorchLoadOptions`, and `TensorFlowLoadOptions` classes. See note above about `InferenceSession.load()` changes.
- Removed the Mojo `LoadOptions` type. See the note above about `InferenceSession.load()` changes.

v24.2.1 (2024-04-11)

- You can now import more MAX Graph functions from `max.graph.ops` instead of using `max.graph.ops.elementwise`. For example:

```
from max.graph import ops

var relu = ops.relu(matmul)
```

v24.2 (2024-03-28)

- MAX Engine now supports TorchScript models with dynamic input shapes.

  No matter what the input shapes are, you still need to specify the input specs for all TorchScript models.

- The Mojo standard library is now open source!

  Read more about it in this blog post.

- And, of course, lots of Mojo updates, including implicit traits, support for keyword arguments in Python calls, a new `List` type (previously `DynamicVector`), some refactoring that might break your code, and much more.

  For details, see the Mojo changelog.

v24.1.1 (2024-03-18)

This is a minor release that improves error reports.

v24.1 (2024-02-29)

The first release of the MAX platform is here! 🚀

This is a **preview version** of the MAX platform. That means it is not ready for production deployment and designed only for local development and evaluation.

Because this is a preview, some API libraries are still in development and subject to change, and some features that we previously announced are not quite ready yet. But there is a lot that you can do in this release!

This release includes our flagship developer tools, currently for **Linux only**:

- **MAX Engine**: Our state-of-the-art graph compiler and runtime library that executes models from PyTorch and ONNX, with incredible inference speed on a wide range of hardware.
  - API libraries in Python, C, and Mojo to run inference with your existing models. See the API references.
  - The `max benchmark` tool, which runs MLPerf benchmarks on any compatible model without writing any code.
  - The `max visualize` tool, which allows you to visualize your model in Netron after partially lowering in MAX Engine.
  - An early look at the MAX Graph API, our low-level library for building high-performance inference graphs in Mojo.

- **MAX Serving**: A preview of our serving wrapper for MAX Engine that provides full interoperability with existing AI serving systems (such as Triton) and that seamlessly deploys within existing container infrastructure (such as Kubernetes).
  - A Docker image that runs MAX Engine as a backend for NVIDIA Triton Inference Server. Try it now.
- **Mojo**: The world's first programming language built from the ground-up for AI developers, with cutting-edge compiler technology that delivers unparalleled performance and programmability for any hardware.
  - The latest version of Mojo, the standard library, and the `mojo` command line tool. These are always included in MAX, so you don't need to download any separate packages.
  - The Mojo changes in each release are often quite long, so we're going to continue sharing those in the existing Mojo changelog.

Additionally, we've started a new GitHub repo for MAX, where we currently share a bunch of code examples for our API libraries, including some large model pipelines such as Stable Diffusion in Mojo and Llama2 built with MAX Graph. You can also use this repo to report issues with MAX.

To get a peek at what's coming soon, and learn about some of the bugs we're working on right now, see the MAX roadmap & known issues.