Mojo🔥 FAQ

We tried to anticipate your questions about Mojo on this page. If this page doesn't answer all your questions, also check out our community channels.

Motivation

Why did you build Mojo?

We built Mojo to solve an internal challenge at Modular, and we are using it extensively in our systems such as our AI Engine. As a result, we are extremely committed to its long term success and are investing heavily in it. Our overall mission is to unify AI software and we can’t do that without a unified language that can scale across the AI infrastructure stack. That said, we don’t plan to stop at AI—the north star is for Mojo to support the whole gamut of general-purpose programming over time. For a longer answer, read Why Mojo.

Why is it called Mojo?

Mojo means “a magical charm” or “magical powers.” We thought this was a fitting name for a language that brings magical powers to Python, including unlocking an innovative programming model for accelerators and other heterogeneous systems pervasive in AI today.

Why does mojo have the 🔥 file extension?

We paired Mojo with fire emoji 🔥 as a fun visual way to impart onto users that Mojo empowers them to get their Mojo on—to develop faster and more efficiently than ever before. We also believe that the world can handle a unicode extension at this point, but you can also just use the `.mojo` extension. :)

What problems does Mojo solve that no other language can?

Mojo combines the usability of Python with the systems programming features it’s missing. We are guided more by pragmatism than novelty, but Mojo’s use of MLIR allows it to scale to new exotic hardware types and domains in a way that other languages haven’t demonstrated (for an example of Mojo talking directly to MLIR, see our low-level IR in Mojo notebook). It also has caching and distributed compilation built into its core. We also believe Mojo has a good chance of unifying hybrid packages in the broader Python community.

What kind of developers will benefit the most from Mojo?

Mojo's initial focus is to bring programmability back to AI, enabling AI developers to customize and get the most out of their hardware. As such, Mojo will primarily benefit researchers and other engineers looking to write high-performance AI operations. Over time, Mojo will become much more interesting to the general Python community as it grows to be a superset of Python. We hope this will help lift the vast Python library ecosystem and empower more traditional systems developers that use C, C++, Rust, etc.

Why build upon Python?

Effectively, all AI research and model development happens in Python today, and there's a good reason for this! Python is a powerful high-level language with clean, simple syntax and a massive ecosystem of libraries. It's also one of the world's most popular programming languages, and we want to help it become even better. At Modular, one of our core principles is meeting customers where they are—our goal is not to further fragment the AI landscape but to unify and simplify AI development workflows.

Why not enhance CPython (the major Python implementation) instead?

We're thrilled to see a big push to improve CPython by the existing community, but our goals for Mojo (such as to deploy onto GPUs and other accelerators) need a fundamentally different architecture and compiler approach underlying it. CPython is a significant part of our compatibility approach and powers our Python interoperability.

Why not enhance another Python implementation (like Codon, PyPy, etc)?

Codon and PyPy aim to improve performance compared to CPython, but Mojo's goals are much deeper than this. Our objective isn't just to create "a faster Python," but to enable a whole new layer of systems programming that includes direct access to accelerated hardware, as outlined in Why Mojo. Our technical implementation approach is also very different, for example, we are not relying on heroic compiler and JIT technologies to "devirtualize" Python.

Furthermore, solving big challenges for the computing industry is hard and requires a fundamental rethinking of the compiler and runtime infrastructure. This drove us to build an entirely new approach and we're willing to put in the time required to do it properly (see our blog post about building a next-generation AI platform), rather than tweaking an existing system that would only solve a small part of the problem.

Why not make Julia better?

We think Julia is a great language and it has a wonderful community, but Mojo is completely different. While Julia and Mojo might share some goals and look similar as an easy-to-use and high-performance alternative to Python, we're taking a completely different approach to building Mojo. Notably, Mojo is Python-first and doesn't require existing Python developers to learn a new syntax.

Mojo also has a bunch of technical advancements compared to Julia, simply because Mojo is newer and we've been able to learn from Julia (and from Swift, Rust, C++ and many others that came before us). For example, Mojo takes a different approach to memory ownership and memory management, it scales down to smaller envelopes, and is designed with AI and MLIR-first principles (though Mojo is not only for AI).

That said, we also believe there's plenty of room for many languages and this isn't an OR proposition. If you use and love Julia, that's great! We'd love for you to try Mojo and if you find it useful, then that's great too.

Functionality

Where can I learn more about Mojo's features?

The best place to start is the Mojo Manual. And if you want to see what features are coming in the future, take a look at the roadmap.

What does it mean that Mojo is designed for MLIR?

MLIR provides a flexible infrastructure for building compilers. It's based upon layers of intermediate representations (IRs) that allow for progressive lowering of any code for any hardware, and it has been widely adopted by the hardware accelerator industry since its first release. Although you can use MLIR to create a flexible and powerful compiler for any programming language, Mojo is the world's first language to be built from the ground up with MLIR design principles. This means that Mojo not only offers high-performance compilation for heterogeneous hardware, but it also provides direct programming support for the MLIR intermediate representations. For a simple example of Mojo talking directly to MLIR, see our low-level IR in Mojo notebook.

Is Mojo only for AI or can it be used for other stuff?

Mojo is a general purpose programming language. We use Mojo at Modular to develop AI algorithms, but as we grow Mojo into a superset of Python, you can use it for other things like HPC, data transformations, writing pre/post processing operations, and much more. For examples of how Mojo can be used for other general programming tasks, see our Mojo examples.

Is Mojo interpreted or compiled?

Mojo supports both just-in-time (JIT) and ahead-of-time (AOT) compilation. In either a REPL environment or Jupyter notebook, Mojo is JIT'd. However, for AI deployment, it's important that Mojo also supports AOT compilation instead of having to JIT compile everything. You can compile your Mojo programs using the `mojo` CLI.

How does Mojo compare to Triton Lang?

Triton Lang is a specialized programming model for one type of accelerator, whereas Mojo is a more general language that will support more architectures over time and includes a debugger, a full tool suite, etc. For more about embedded domain-specific languages (EDSLs) like Triton, read the "Embedded DSLs in Python" section of Why Mojo.

How does Mojo help with PyTorch and TensorFlow acceleration?

Mojo is a general purpose programming language, so it has no specific implementations for ML training or serving, although we use Mojo as part of the overall Modular AI stack. The Modular AI Engine, for example, supports deployment of PyTorch and TensorFlow models, while Mojo is the language we use to write the engine's in-house kernels.

Does Mojo support distributed execution?

Not alone. You will need to leverage the Modular AI Engine for that. Mojo is one component of the Modular stack that makes it easier for you to author highly performant, portable kernels, but you'll also need a runtime (or "OS") that supports graph level transformations and heterogeneous compute.

Will Mojo support web deployment (such as Wasm or WebGPU)?

We haven't prioritized this functionality yet, but there's no reason Mojo can't support it.

How do I convert Python programs or libraries to Mojo?

Mojo is still early and not yet a Python superset, so only simple programs can be brought over as-is with no code changes. We will continue investing in this and build migration tools as the language matures.

What about interoperability with other languages like C/C++?

Yes, we want to enable developers to port code from languages other than Python to Mojo as well. We expect that due to Mojo's similarity to the C/C++ type systems, migrating code from C/C++ should work well and it's in our roadmap.

How does Mojo support hardware lowering?

Mojo leverages LLVM-level dialects for the hardware targets it supports, and it uses other MLIR-based code-generation backends where applicable. This also means that Mojo is easily extensible to any hardware backend. For more information, read about our vision for pluggable hardware.

Who writes the software to add more hardware support for Mojo?

Mojo provides all the language functionality necessary for anyone to extend hardware support. As such, we expect hardware vendors and community members will contribute additional hardware support in the future. We'll share more details about opening access to Mojo in the future, but in the meantime, you can read more about our hardware extensibility vision.

How does Mojo provide a 35,000x speed-up over Python?

Modern CPUs are surprisingly complex and diverse, but Mojo enables systems-level optimizations and flexibility that unlock the features of any device in a way that Python cannot. So the hardware matters for this sort of benchmark, and for the Mandelbrot benchmarks we show in our launch keynote, we ran them on an AWS r7iz.metal-16xl machine.

For lots more information, check out our 3-part blog post series about how Mojo gets a 35,000x speedup over Python.

By the way, all the kernels that power the Modular AI Engine are written in Mojo. We also compared our matrix multiplication implementation to other state-of-the-art implementations that are usually written in assembly. To see the results, see our blog post about unified matrix multiplication.

Performance

Mojo's matmul performance in the notebook doesn't seem that great. What's going on?

The Mojo Matmul notebook uses matrix multiplication to show off some Mojo features in a scenario that you would never attempt in pure Python. So that implementation is like a "toy" matmul implementation and it doesn't measure up to the state of the art.

Modular has a separate matmul implementation written in Mojo and used by MAX Engine, which you can read about it in this blog post.

Are there any AI related performance benchmarks for Mojo?

It’s important to remember that Mojo is a general-purpose programming language, and any AI-related benchmarks will rely heavily upon other framework components. For example, our in-house kernels for the Modular AI Engine are all written in Mojo and you can learn more about our kernel performance in our matrix multiplication blog post. For details about our end-to-end model performance relative to the latest releases of TensorFlow and PyTorch, check out our performance dashboard.

Mojo SDK

How can I get access to the SDK?

Mojo is included with the MAX SDK, which you can download and use for free.

Read more about why Mojo is bundled with MAX.

Is the Mojo Playground still available?

Yes, but it's different. When we first announced Mojo, it was available only through login, in a JupyterLab environment. Now that Mojo is available for local development, we've shut down that service (you can instead run Mojo notebooks locally).

The new Mojo Playground is built into the docs website and does not require login.

- It provides access to Mojo and the Mojo standard library. It does not have network access, so you can't install additional Mojo or Python packages.
- It doesn't include any Python packages by default. In the future, we intend to make some common Python packages available to import in the Playground.
- You can download your code or share it as a gist, but there's no mechanism for saving code in the Playground itself. Any changes will be lost when you switch code examples (as well as in the event of a server refresh or update). If you come up with something you want to save, download it or share it using buttons in the Playground toolbar.
- There might be some bugs. Please report issues and feedback on GitHub.

What are the license terms for the SDK?

Please read the Mojo SDK License Terms.

What operating systems are supported?

Currently, we support Ubuntu Linux 20.04/22.04 (64-bit x86) and macOS (Apple silicon). Support for Windows will follow. Until then, you have several options:

- Windows users can use Windows Subsystem for Linux version 2 (WSL 2) running a supported Linux distribution.
- Intel Mac users can use a Docker container running a supported Linux distribution.
- Users on any system can install the SDK on a remote machine running a supported Linux distribution.

Is there IDE Integration?

Yes, we've published an official Mojo language extension for VS Code.

The extension supports various features including syntax highlighting, code completion, formatting, hover, etc. It works seamlessly with remote-ssh and dev containers to enable remote development in Mojo.

Does the Mojo SDK collect telemetry?

Yes, the Mojo SDK collects some basic system information, basic compiler/runtime events, and crash reports that enable us to identify, analyze, and prioritize Mojo issues.

This telemetry is crucial to help us quickly identify problems and improve our products. Without this telemetry, we would have to rely on user-submitted bug reports, and in our decades of experience building developer products, we know that most people don't do that. The telemetry provides us the insights we need to build better products for you.

You can opt-out of the crash report and compiler/runtime telemetry, but package install/update/uninstall events cannot be disabled (see the MAX SDK terms).

To disable crash reports, use this command:

```
$ modular config-set crash_reporting.enabled=false
```

To reduce other telemetry to only the required telemetry events, use this command:

```
$ modular config-set telemetry.level=0
```

There are 3 telemetry levels: `0` currently records nothing (unless you're also using MAX, which records hardware information and session durations); `1` records high-level events such as when the compiler is invoked; and `2` records more detail such as the time spend compiling.

Versioning & compatibility

What's the Mojo versioning strategy?

Mojo is still in early development and not at a 1.0 version yet. It's still missing many foundational features, but please take a look at our roadmap to understand where things are headed. As such, the language is evolving rapidly and source stability is not guaranteed.

How often will you be releasing new versions of Mojo?

Mojo development is moving fast and we are regularly releasing updates. Please join the Mojo Discord channel for notifications and sign up for our newsletter for more coarse-grain updates.

Open Source

Will Mojo be open-sourced?

We expect to open-source Mojo progressively over time as it continues to mature. Mojo is still young, so we will continue to incubate it within Modular until more of its internal architecture is fleshed out.

Why not develop Mojo in the open from the beginning?

Mojo is a big project and has several architectural differences from previous languages. We believe a tight-knit group of engineers with a common vision can move faster than a community effort. This development approach is also well-established from other projects that are now open source (such as LLVM, Clang, Swift, MLIR, etc.).

Community

Where can I ask more questions or share feedback?

If you have questions about upcoming features or have suggestions for the language, be sure you first read the Mojo roadmap, which provides important information about our current priorities and links to our GitHub channels where you can report issues and discuss new features.

To get in touch with the Mojo team and developer community, use the resources on our community page.

Was this page helpful?

Edit this page