Mojo

Python syntax, systems-level performance.

Nick Bloor / Stephen Goddard

What Is Mojo?

The short version.

THE PITCH

Readability of Python that runs at the speed of C.

A superset of Python, designed from day one for AI workloads.

Created by	**Modular Inc. - Chris Lattner - former Apple Engineer**
Same person who built	**Swift, LLVM, Clang, MAX (Modular Action Engine)**
Announced	**May 2023**
Built on	**MLIR compiler stack**
Status	**Closed compiler / open stdlib**
Adoption	**175K+ devs, 50K+ orgs**

Why Build a New Language?

Python dominates AI development because of its readability and ecosystem but it is slow. Performance-critical code traditionally has to be rewritten in C, C++, or CUDA, producing a two-language split that slows research and deployment. Mojo collapses that gap.

01

Python is too slow.

- AI workloads need C-level speed
- Interpreted, GIL-locked, single-threaded

02

One language, full stack.

- Today: Python prototype - C++/CUDA in production
- Mojo: prototype and ship in one language

03

Hardware is plural.

- AI utilizes CPUs, GPUs, TPUs, custom chips
- Need a language that targets all of them

Design Goals

What Mojo is explicitly optimizing for.

THE CORE IDEA

Keep the ease of Python. Gain the speed of C.

A single language for the full AI stack - from notebook prototyping to production-grade kernels.

01 **Python compatibility**

Existing Python code and libraries just work.

02 **Systems-level performance**

Compete with C and Rust on raw speed.

03 **Hardware portability**

One codebase - CPU, GPU, accelerators.

04 **Unified AI toolchain**

Kill the Python-then-C++ handoff.

Features → Goals (1)

Typing, declarations, and Python interop.

WHAT'S NEW VS PYTHON

fn - typed, compiled functions

→ enables compile-time optimization

var / let - mutable / immutable

→ memory safety

struct - value-type record

→ predictable memory layout, fast

Python imports work as-is

→ keep NumPy, PyTorch, the whole ecosystem

mojo

```mojo
from python import Python

fn main() raises:
    # var = mutable, let = immutable
    var x: Int = 10
    let msg: String = "hi"

    # Call Python from Mojo
    let np = Python.import_module("numpy")
    print(np.array([1, 2, 3]).mean())
```

Features → Goals (2)

Where the speed actually comes from.

01 MLIR compiler

Multi-Level Intermediate Representation compiles one source file to CPU, GPU, or accelerators.

→ *Hardware portability*

02 SIMD vectorization

Single Instruction, Multiple Data: one CPU instruction processes 4, 8, or 16 numbers at once.

→ *100×+ on numeric loops*

03 True parallelism

No GIL. Mojo's parallelize() spreads work across CPU cores natively.

→ *Uses every core, not just one*

mojo

```mojo
# A loop the compiler can vectorize
alias nelts = simdwidthof[Float32]()

fn row(m: Int):
    @parameter
    fn dot[w: Int](n: Int):
        C[m, n] += A[m, k] * B[k, n]
    vectorize[nelts, dot](C.cols)  # SIMD

parallelize[row](C.rows)  # multi-core
```

How Mojo Differs from Python

Same shape, different engine.

Functions

Python def - dynamic, untyped

Mojo **fn - typed, compiled, type-checked**

Variables

Python x = 10 - always mutable

Mojo **var x: Int = 10 / let y = 5 (immutable)**

Records

Python class - heap-allocated

Mojo **struct - value type, fixed memory layout**

Concurrency

Python GIL - true threads not allowed

Mojo **No GIL - native multi-core parallelism**

The Speed Story

Same algorithm, layered Mojo optimizations. Matrix multiplication, 512×512.

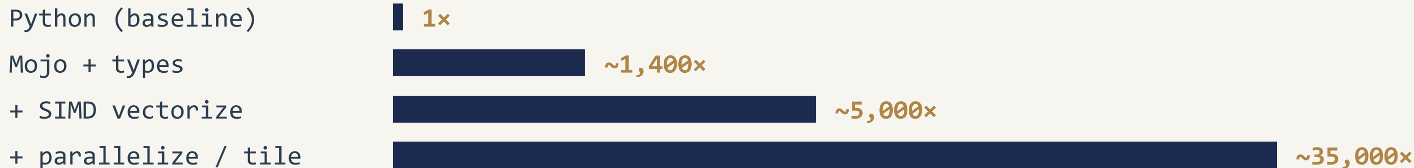
python

```python
# Python — pure loops
def matmul(C, A, B):
  for m in range(C.rows):
    for n in range(C.cols):
      for k in range(A.cols):
        C[m,n] += A[m,k]*B[k,n]
```

mojo

```mojo
# Mojo — typed + vectorized + parallel
fn matmul(C: Matrix, ...):
  @parameter
  fn row(m: Int):
    vectorize[nelts, dot](C.cols)
  parallelize[row](C.rows)
```

REPORTED SPEEDUP OVER PURE PYTHON



Source: Modular benchmarks, matmul example. Real-world speedups vary; pure-Python loops are the worst case for Python.

Why Mojo Is Faster

Three design choices that compound.

TYPES

Types are known at compile time (Assuming var or let keywords).

Python checks types every operation, at runtime. Mojo resolves them once, at compile time, and emits optimized machine code.

→ Goal: C-level performance

SIMD

One instruction processes many values.

Modern CPUs have wide registers (256–512 bits). Mojo's vectorize() uses them automatically; Python loops process one number at a time.

→ Goal: Use the hardware fully

CORES

No Global Interpreter Lock.

Python's GIL prevents true multi-threading. Mojo's parallelize() splits work across all CPU cores natively - a 16-core machine actually runs 16× faster.

→ Goal: Hardware portability + scale

Running Mojo

How to open, edit, and execute a .mojo file.

~ terminal

```
$ wsl.exe
$ cd ~/mojo_demo
$ nano hello.mojo
  (Ctrl+O to save, Ctrl+X to exit)
$ mojo hello.mojo
Hello, world!
```

STEPS

01 Launch WSL (required on Windows).

02 Navigate to the project directory.

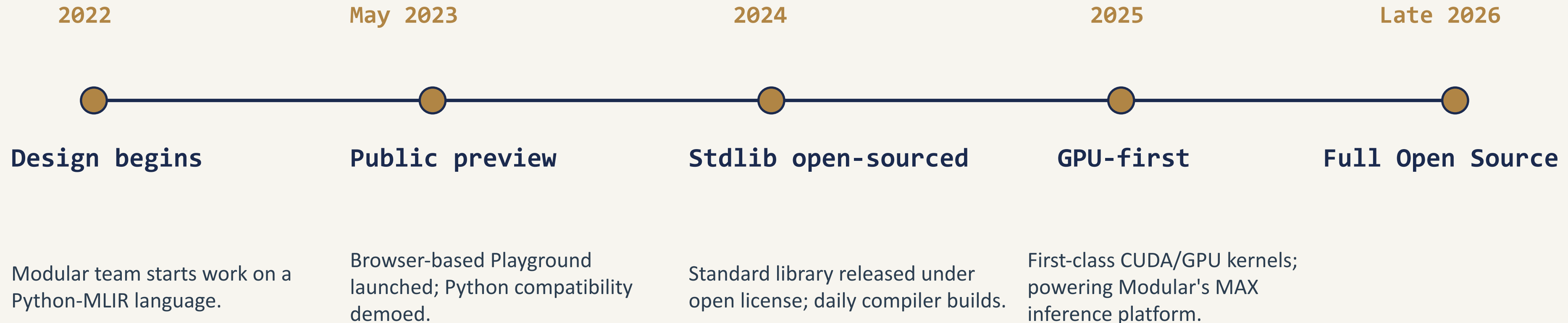
03 Edit a .mojo file in nano or VS Code.

04 Run with the mojo command.

05 Output prints directly to the shell.

Where Mojo Is Going

A young language, moving quickly.



Who Is Actually Using Mojo

Still early, but real production use is starting.

Modular / MAX Platform

Production AI inference engine

Mojo is the implementation language for Modular's commercial MAX platform running LLMs and serving AI models in production

Inworld AI

Real-time AI characters

Uses Mojo to write custom GPU kernels including a tailored silence-detection kernel that runs directly on the GPU for low-latency voice AI

Qwerky AI

Memory-efficient LLMs

Uses Mojo to compile custom GPU kernels accelerating Mamba's linear-time complexity for long conversation histories

Pvotal Technologies

ML infrastructure tooling

Adopted Mojo after benchmarking it against Python + NumPy for matrix and regression workloads in their data pipeline

Open-source community

175K+ developers, 50K+ orgs

Active community contributing to the standard library; pure-Mojo Llama2 implementation, geospatial and 3D-graphics projects

Caveat

Still early

Adoption is small compared to Python or C++. Most production AI in 2026 is still Python + CUDA

Mojo is a credible challenger, not the default

References

Sources used in this presentation.

01 **Modular — Official Mojo Documentation**

https://docs.modular.com/

Language reference, matmul tutorial, installation.

02 **Modular — Mojo Product Page (Inworld, Qwerky case studies)**

https://www.modular.com/mojo

Real-world production use cases.

03 **Pvotal Technologies — Mojo Performance Analysis (2025)**

https://pvotal.tech/mojo-performance-analysis/

Benchmarks: Mojo vs Python + NumPy.

04 **fast.ai — "Mojo may be the biggest programming language advance in decades"**

https://www.fast.ai/posts/2023-05-03-mojo-launch.html

Jeremy Howard's analysis of MLIR and Mojo's design.

05 **Wikipedia — Mojo (programming language)**

https://en.wikipedia.org/wiki/Mojo_(programming_language)

History, language status, technical details.