RTNS 2025
5-7 November 2025, Pisa, Italy

Is Mojo Ready for Real-Time? A Language Evaluation for Time-Sensitive System Software

AARON BOGAERT ANTONIO PAOLILLO

Vrije Universiteit Brussel

So many programming languages

What is the **best** programming language?

Meaningless in a vacuum

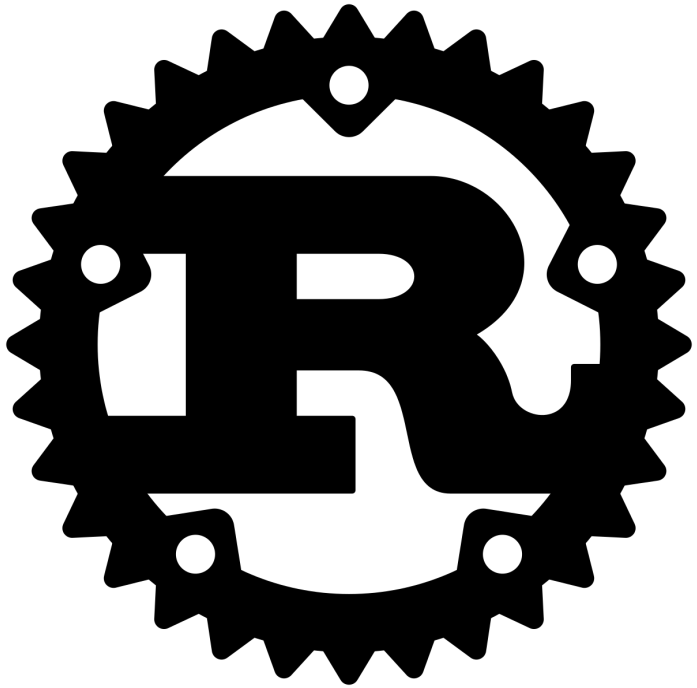
Framing the question

- **Speed**: C
- **Math**: Matlab
- **Fancy UI**: JavaScript

What is the **best** programming language **for real time systems**?

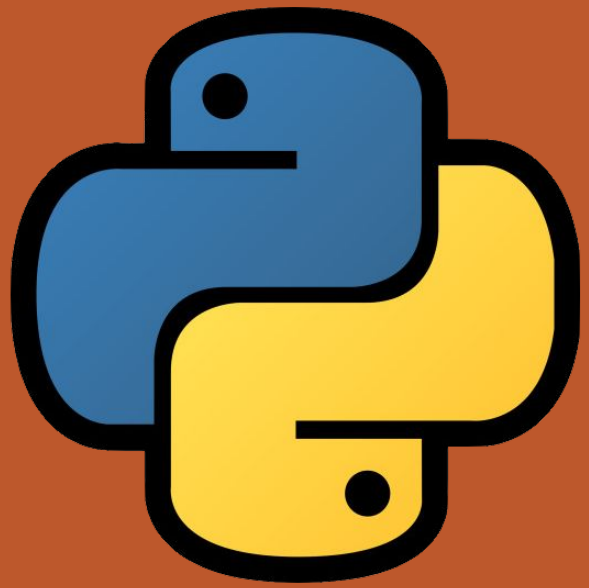
Let us explore our options

Let us explore our options



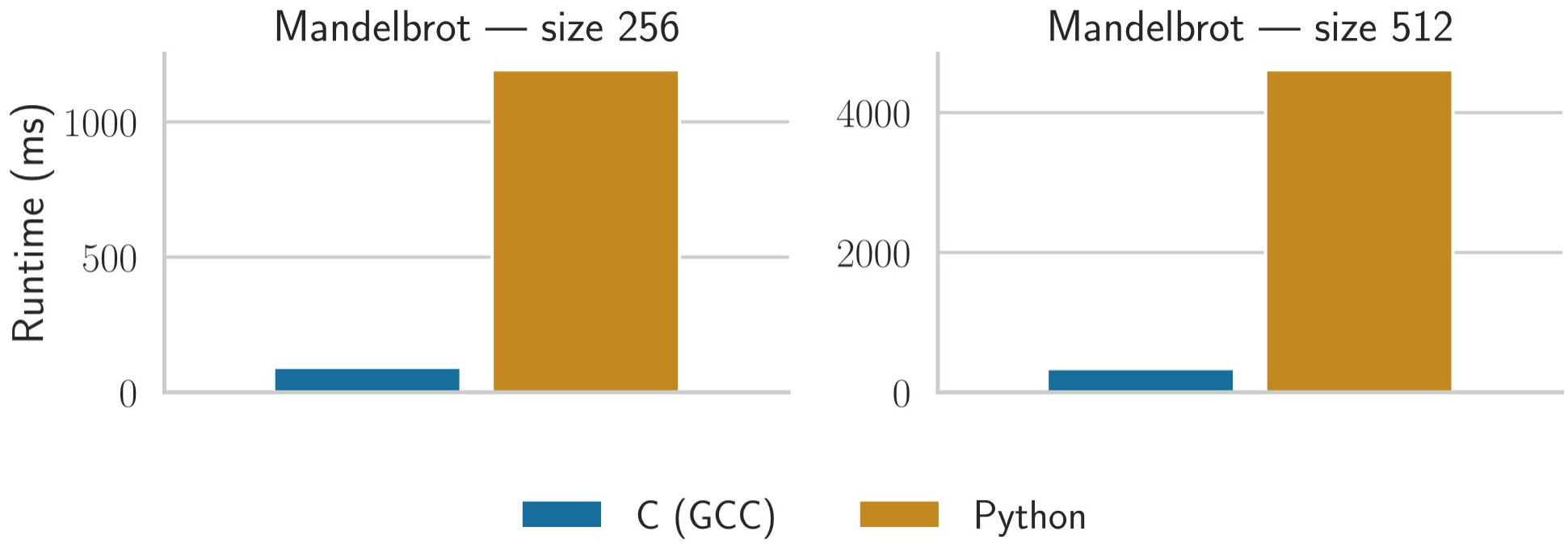
Go exploring **low-level** languages

Why not Python?



- Readable syntax
- High-level
- Fast development
- Mature

Why not!


Mandelbrot — size 256
Runtime (ms)
1000
500
0
Mandelbrot — size 512
4000
2000
0
C (GCC)
Python

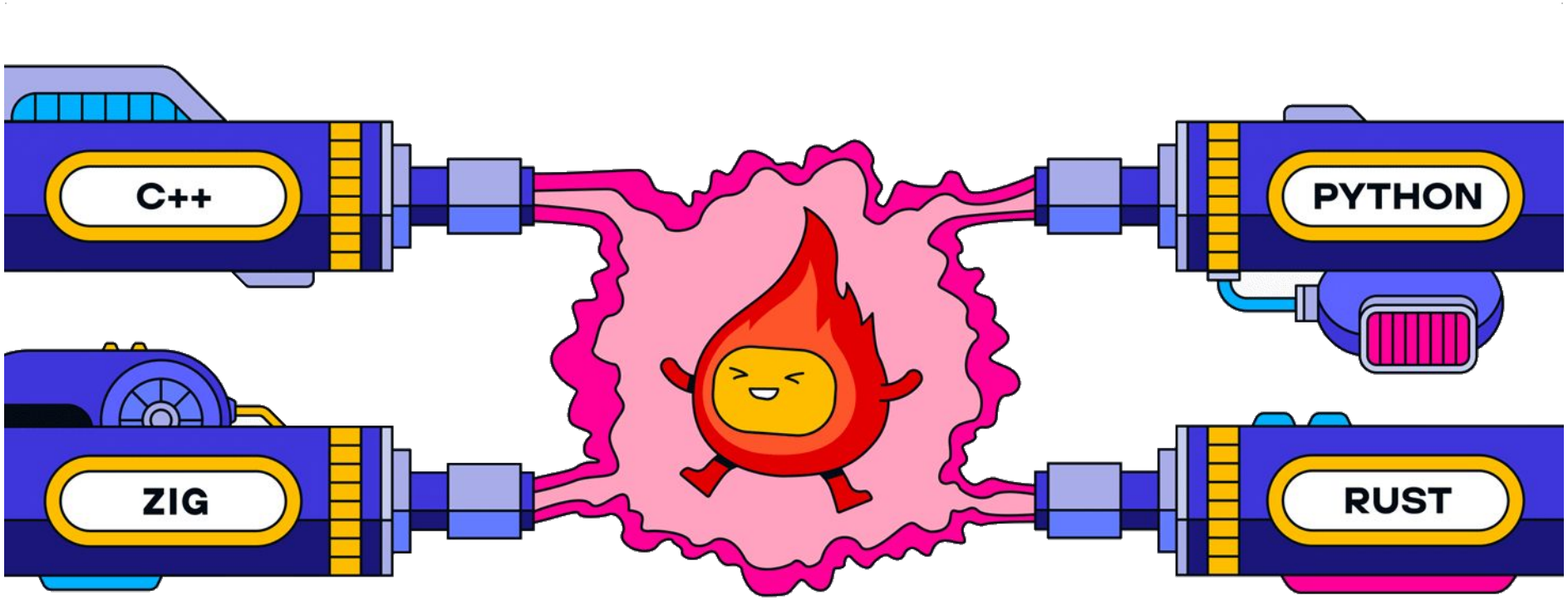
Many reasons

- Unpredictable execution times
  - **Interpreter**
  - Dynamic typing
  - **Varying** just-in-time optimizations
- Non deterministic **garbage collection**
- **Limited control** over scheduling

Keep the best parts

Keep the best parts

Why Mojo?



High level programming language

- **Metaprogramming** power
- Memory **safety**, borrow checker
- Easy-to-read **Pythonic** syntax

Pythonic syntax is easy

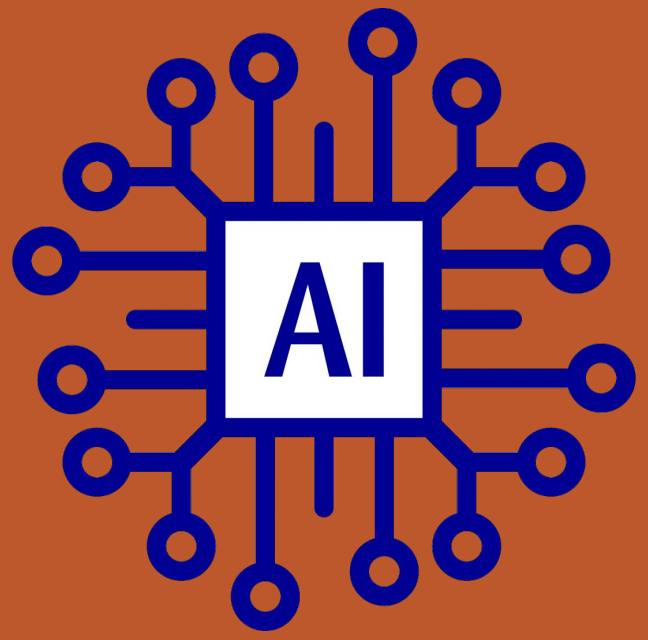
```
def computeLPSArray(pattern: String, m:Int) -> List[Int]:
    var LPS:List[Int] = List[Int](capacity=m)
    var length = 0
    var i = 1
    LPS[0] = 0
    while (i < m):
        if (pattern[i] == pattern[length]):
            length += 1
            LPS[i] = length
            i += 1
        else:
            if not (length == 0):
                length = LPS[length - 1]
            else:
                LPS[i] = 0
                i += 1
```

Pythonic syntax is easy

```
def computeLPSArray(pattern: String, m:Int) -> List[Int]:
    var LPS:List[Int] = List[Int](capacity=m)
    var length = 0
    var i = 1
    LPS[0] = 0
    while (i < m):
        if (pattern[i] == pattern[length]):
            length += 1
            LPS[i] = length
            i += 1
        else:
            if not (length == 0):
                length = LPS[length - 1]
            else:
                LPS[i] = 0
                i += 1
```

Promises of high performance

- **Compiled** language
- Zero cost abstractions
- Systems language performance
- Low memory usage



- Designed for AI
- **MAX:** inference framework for AI
- Heavy focus on **GPU compatibility**
- **Platform independent** GPU development

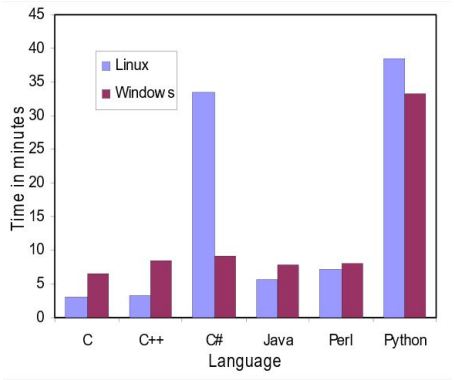
MLIR*

MULTI-LEVEL INTERMEDIATE REPRESENTATION

- **Modular** intermediate representation
- **Compiler agnostic** optimizations
- Extends LLVM with **abstraction levels**
- Active development
- Allows **compiler decoupling**:
  - Optimization passes
  - Certification passes

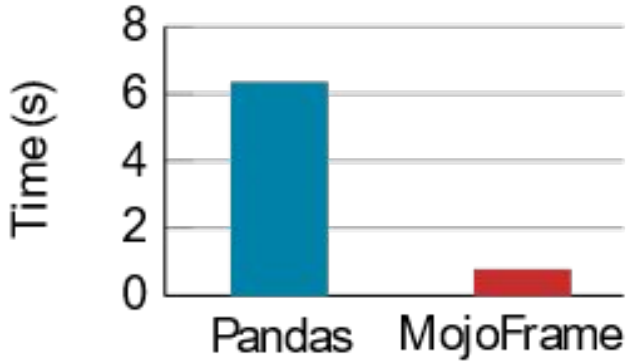
* Lattner, Chris, et al. "MLIR: Scaling compiler infrastructure for domain specific computation." 2021 IEEE/ACM International Symposium on Code Generation and Optimization (CGO). IEEE, 2021.

How to evaluate a programming language



Fourment, Mathieu, and Michael R. Gillings. "A comparison of common programming languages used in bioinformatics." *BMC bioinformatics* 9.1 (2008): 82.

?



Huang, Shengya, et al. "MojoFrame: Dataframe Library in Mojo Language." *arXiv preprint arXiv:2505.04080* (2025).

Wide Scope

Narrow Scope

Evaluating for real time

Qualitative evaluation

How usable is the language?

Does it support our needed feature set?

Quantitative evaluation

Measurable characteristics

- Execution time
- Mean & variance
- File size
- Etc.

Qualitative evaluation

- Documentation
- Interoperability with Python
- Systems level control (Threads, CPU control)
- Cross-compilation and linking

Documentation

- Rapidly developing
- **Breaking** changes to the interface
- Mostly **auto-generated** documentation
- **Outdated** examples

Interoperability with Python

- **Why**:
  - Piecewise conversion
  - Existing projects: ROS
- **How**:
  - Make use of Python modules
- **Impact:**
  - Enter Python runtime
  - Garbage collected
  - Memory overhead

Python robotics example

```mojo
from python import Python

fn main():
    try:
        robotmod = Python.import_module("robotmod")
        robotmod.robot_move()
    except:
        print("Exception occurred!")
```

Mojo 🔥

Importing Python

```python
import rtde_control
import rtde_receive

def robot_move() -> None:
    robot_ip = "172.17.0.2"

    rtde_c = rtde_control.RTDEControlInterface(hostname=robot_ip)
    rtde_c.moveL([-0.143, -0.435, 0.20, -0.001, 3.12, 0.04], 0.5, 0.3)
```

Cross-compilation and linking

- Ahead of time compilation to object files

```
# Compile Mojo to an AArch64 object file
mojo build sum.mojo --emit object \
    --target-triple aarch64-none-elf \
    --mcpu cortex-a72 -o sum.o
```

- C Application Binary interface

```
# sum.mojo
@export("sum", ABI="C")
def sum(a: Int32, b: Int32) -> Int32:
    return a + b
```

Systems level control

- **Not directly** available:
  - Setting thread priorities
  - Adjusting CPU affinity
  - Configuring system files such as `cpu_dma_latency`
- IO: **Open()** with write
  - **Error**: `Unable to remove <file>`
  - Closed source

Systems level control

- **Syscalls** with `sys.external.call()`
  - Through the **C library**
  - No errno
  - Hard to debug

```
def lock_pages():
    r = sys.external_call["mlockall",Int32](Int(3))
```

Thread control

No explicit control

- No **pthread**-style
- **OpenMP**-style of parallelism
- **Thread pool** of 4

```
parallelize[origins: OriginSet, //,
            func: fn(Int) capturing -> None]
           (num_work_items: Int)
```

Qualitative evaluation

Documentation

Python interoperability

Cross compilation and linking

Systems level control

More in paper

Quantitative evaluation

- Execution time and variability
- Binary size
- Latency and predictability under load

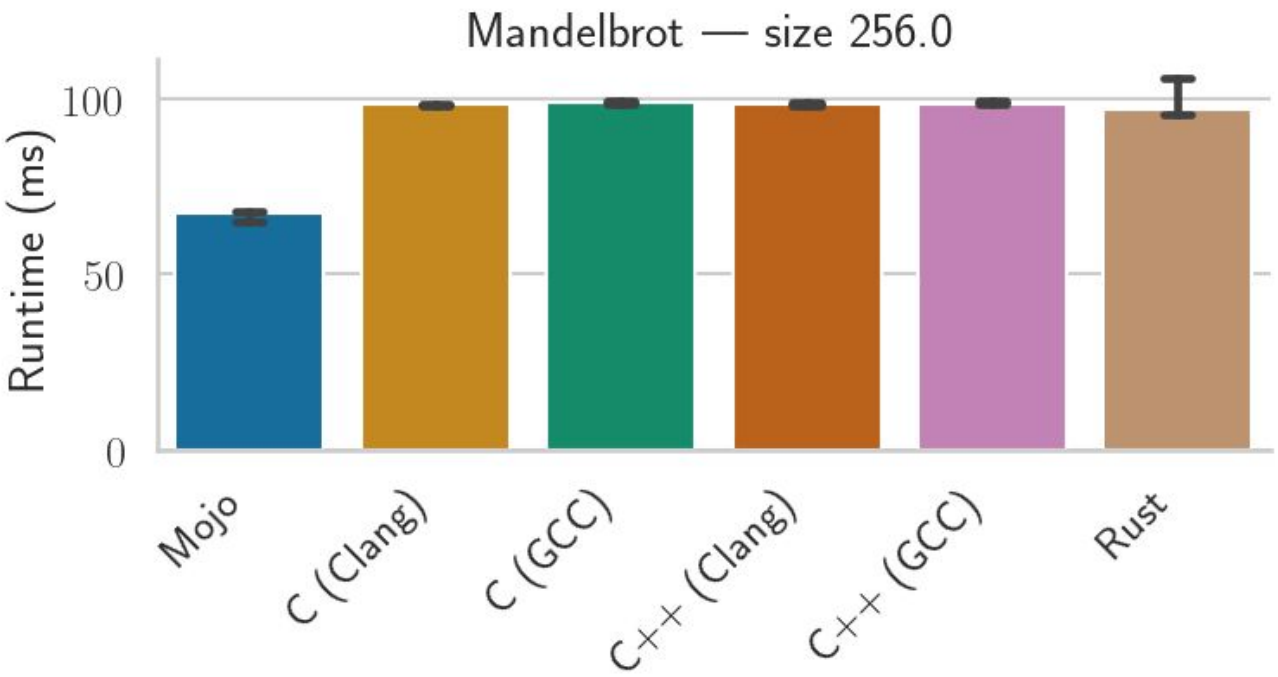
Microbenchmarks

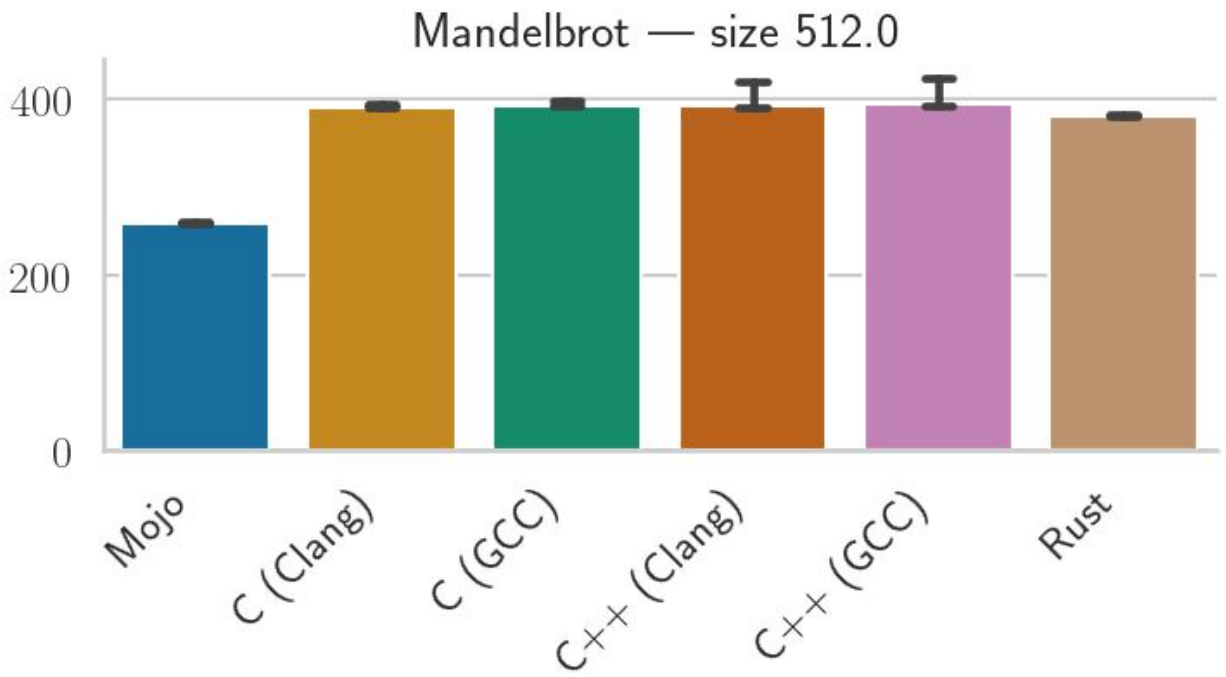
EXECUTION TIME
VARIABILITY
BINARY SIZE

- 3 workloads
  - Compute heavy ⟶ Mandelbrot
  - Exploiting SIMD ⟶ Matrix multiplication
  - Branch heavy ⟶ String search
- Configured environment
  - Core isolation through `isolcpus`
  - Multiple optimization configurations
    ⇒ `O0,O1,O2,O3`

Compute heavy workload

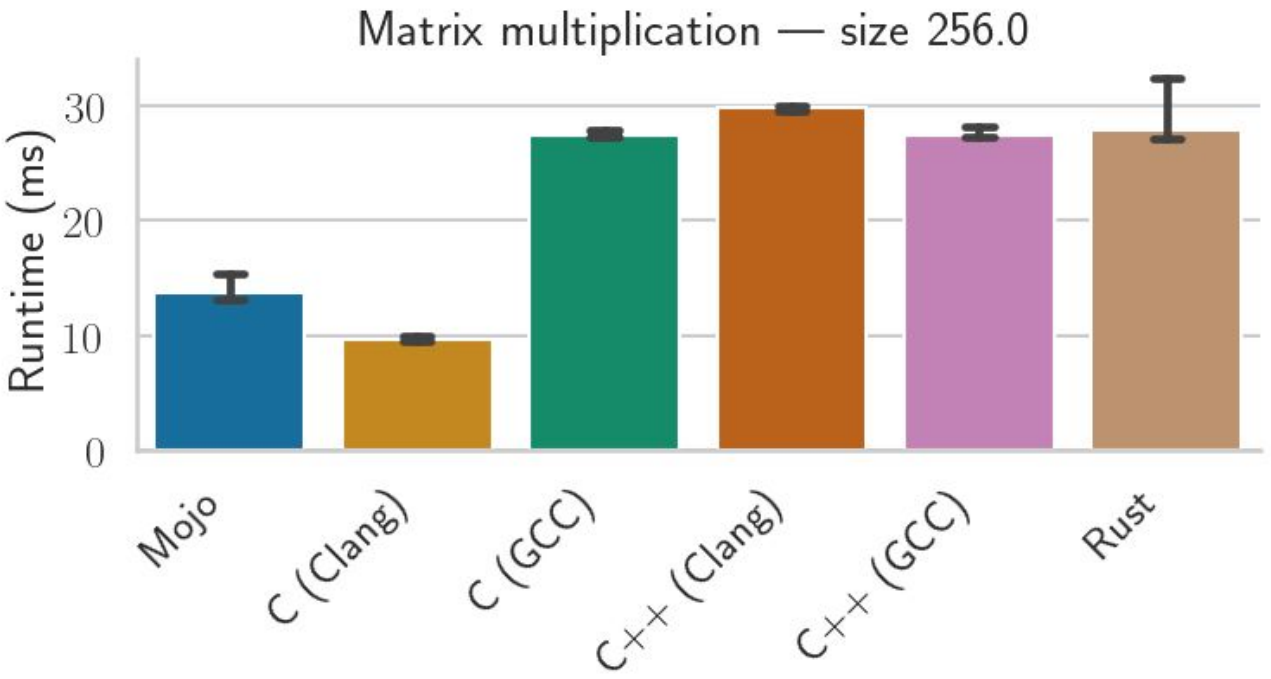
MANDELBROT

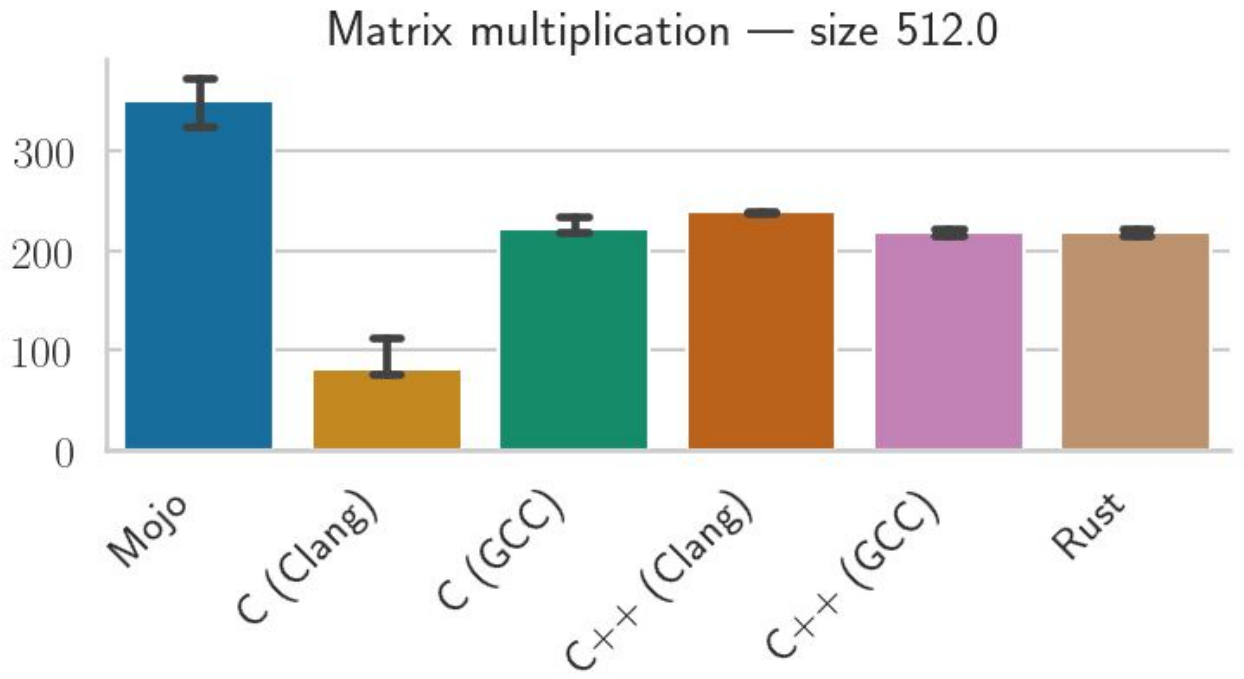




Exploiting SIMD

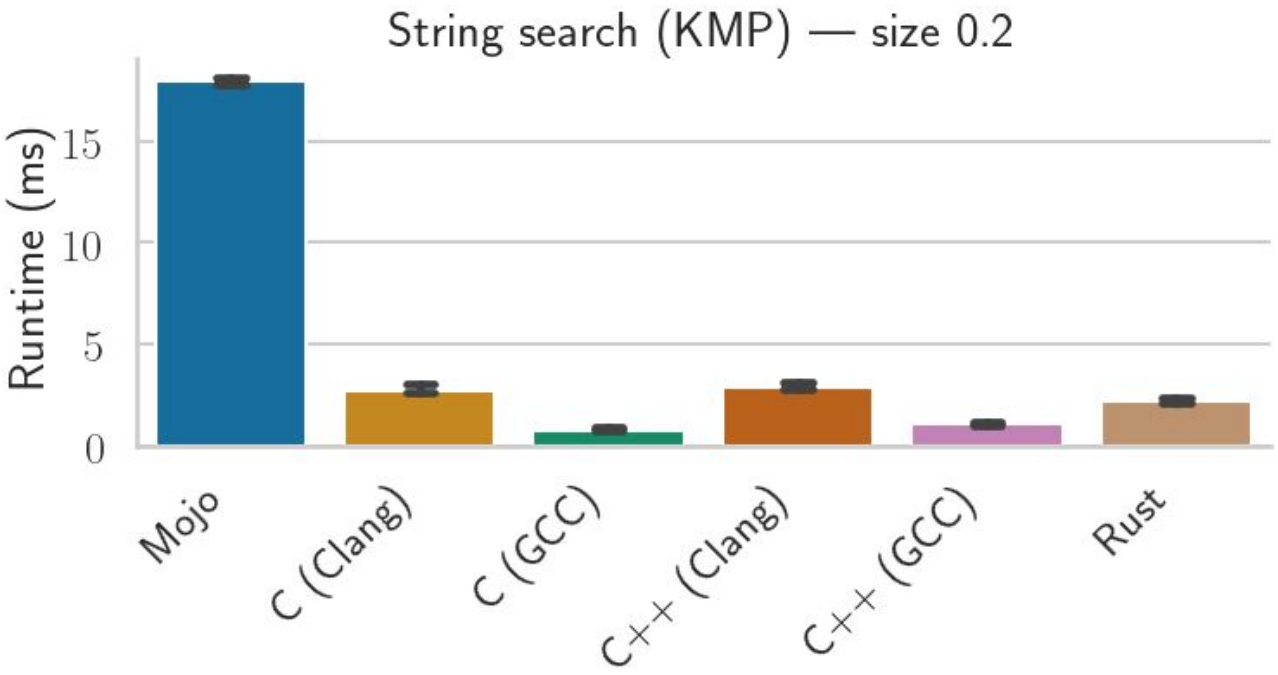
MATRIX MULTIPLICATION

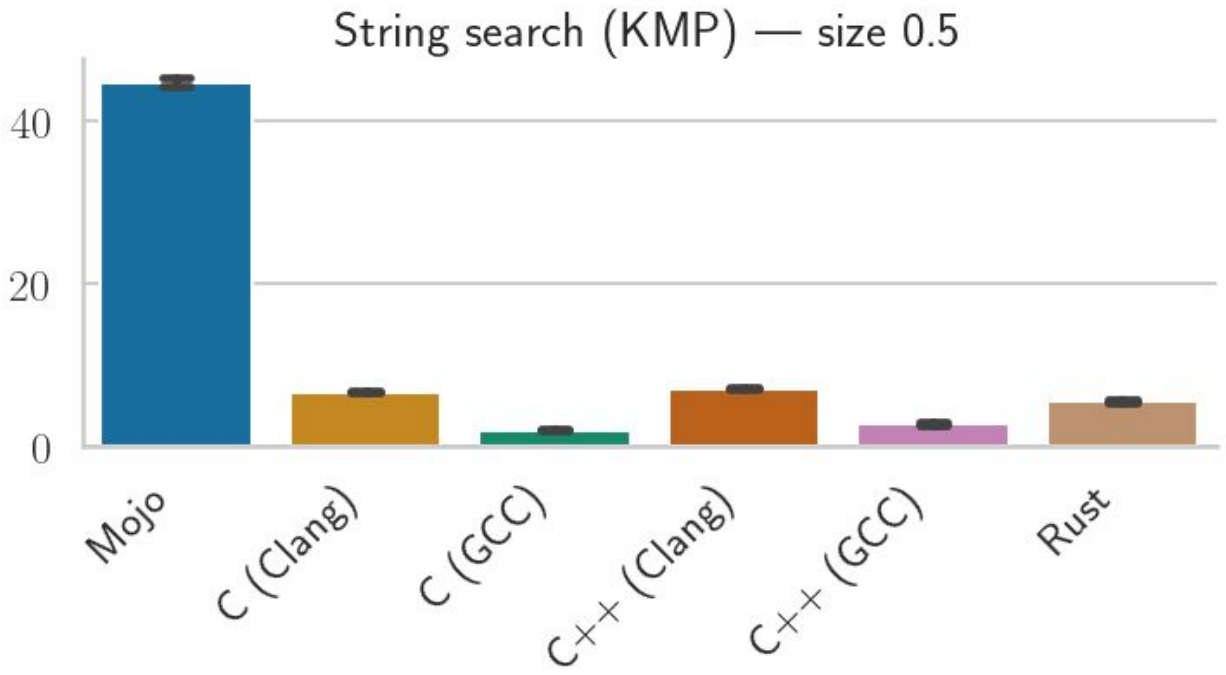




Branch heavy

KNUTH–MORRIS–PRATT ALGORITHM





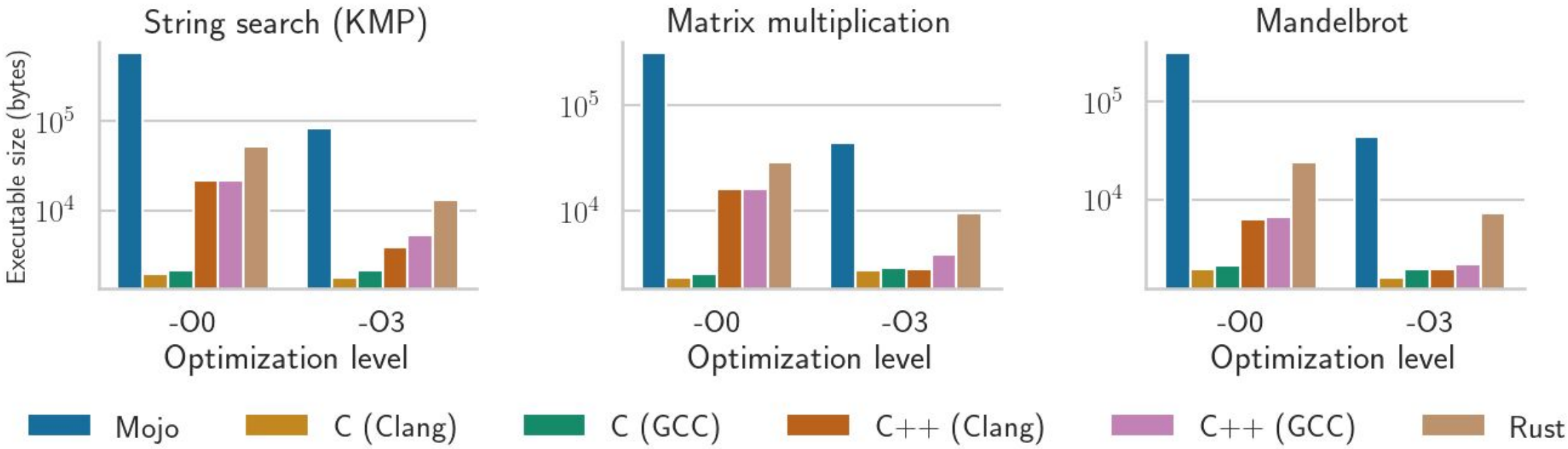
Execution time and variability

- Performs best in simple environments
  - Rivaling C performance
  - Less mature compiler optimizations
- Not ready for real time
  - Unexpected results
  - Less consistent

Binary file size

90% overhead

STRIPPED OBJECT FILES



- Inclusion of **standard libraries**
- Large **jump tables** for typing

Latency and predictability under load

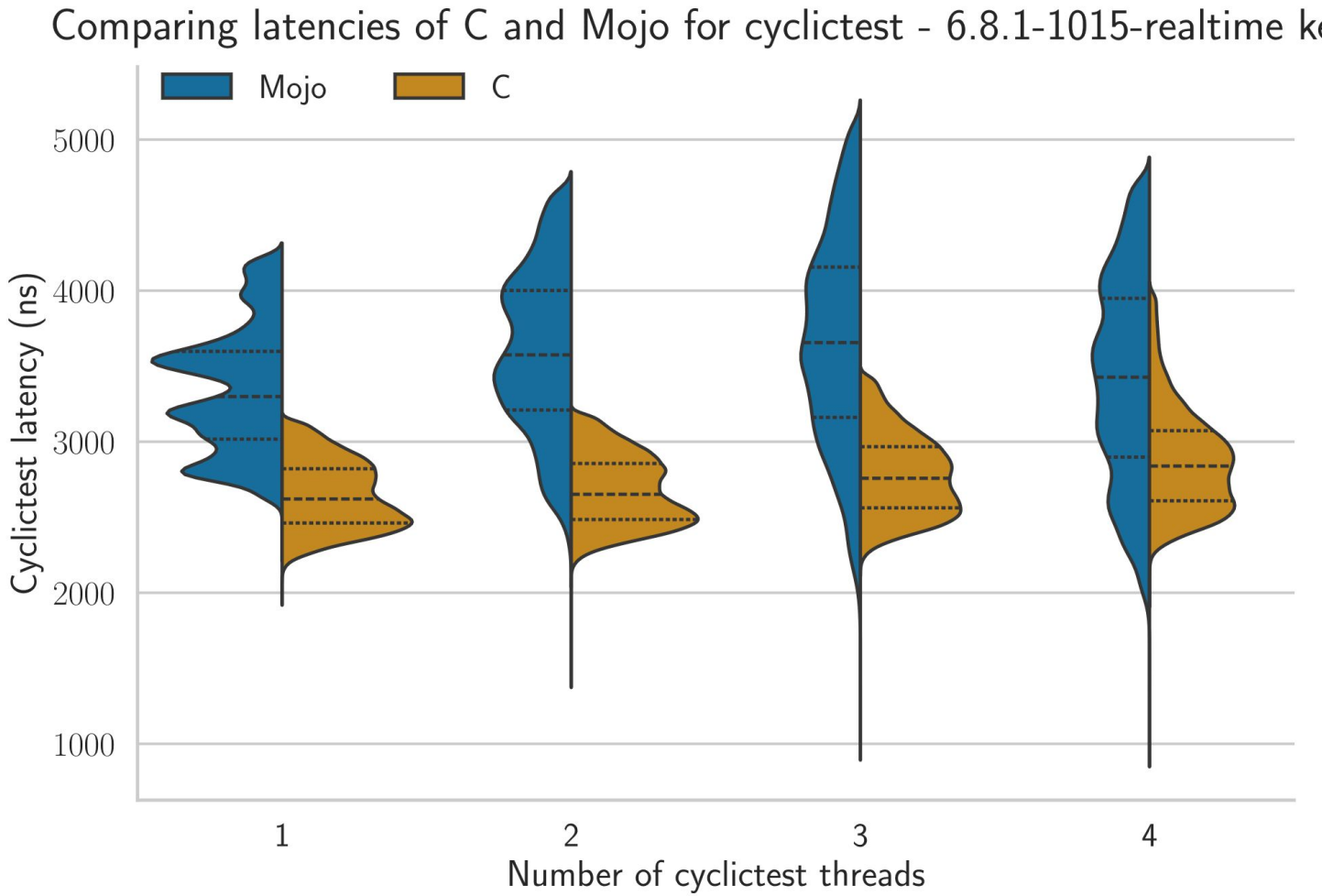
- Cyclictest
  - Evaluate the **wake-up time**
  - Spawning N **threads**
  - Setting system parameters
- **Stress-ng** to stress the system
  - Full system **CPU load**
  - Configurable

Methodology

- Reimplement in **Mojo**
  - *cyclictest –nsecs -vnm -i 100 -p 99 -t N –duration=1m –mlockall*
- Run under **PREEMPT_RT** Linux
- Compare to the original C version

⇒ Difference is produced by Mojo

Latency and predictability under load



- Higher average and mean
- Wider spread

Quantitative evaluation

- Execution time and variability
  - Inconsistent and unpredictable
  - Rivals C execution at times
- 90% overhead in binary size
- Latency and predictability under load
  - Mojo introduces latency and variability

Reproducibility

Building a framework

Pythainer

Benchkit

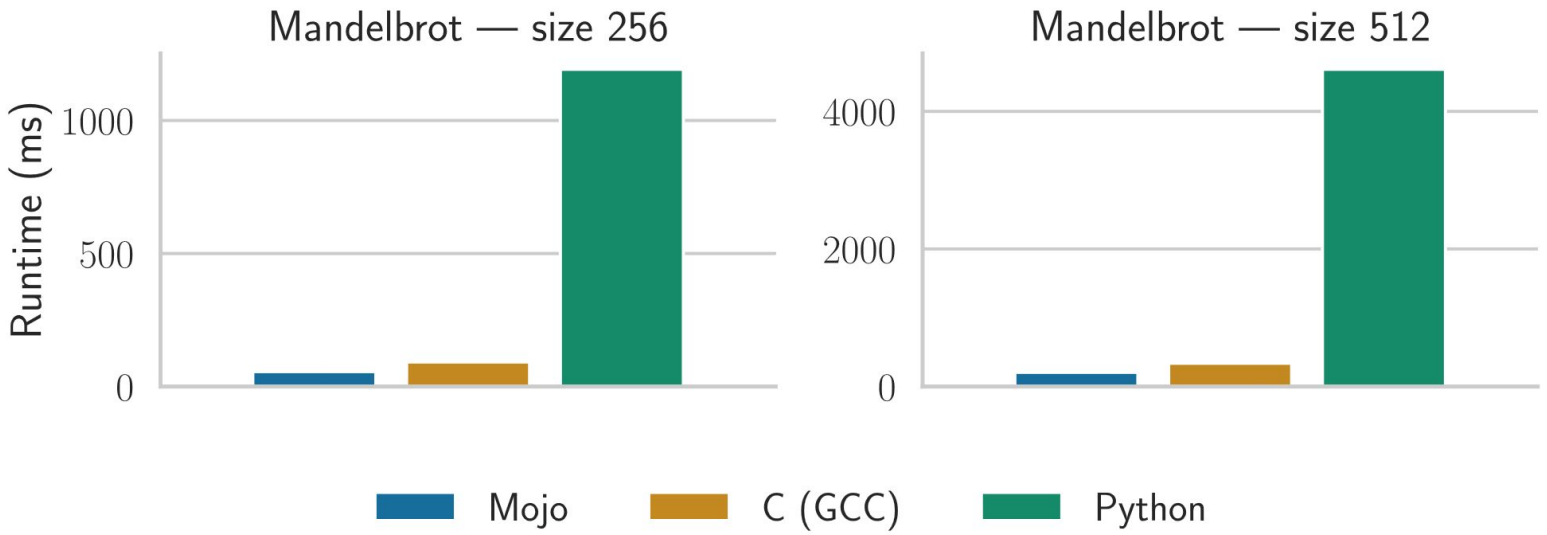
Developed in house

Free and open source

https://github.com/apaolillo/**mojort**
https://github.com/apaolillo/**pythainer**
https://github.com/open-s4c/**benchkit**

Recap

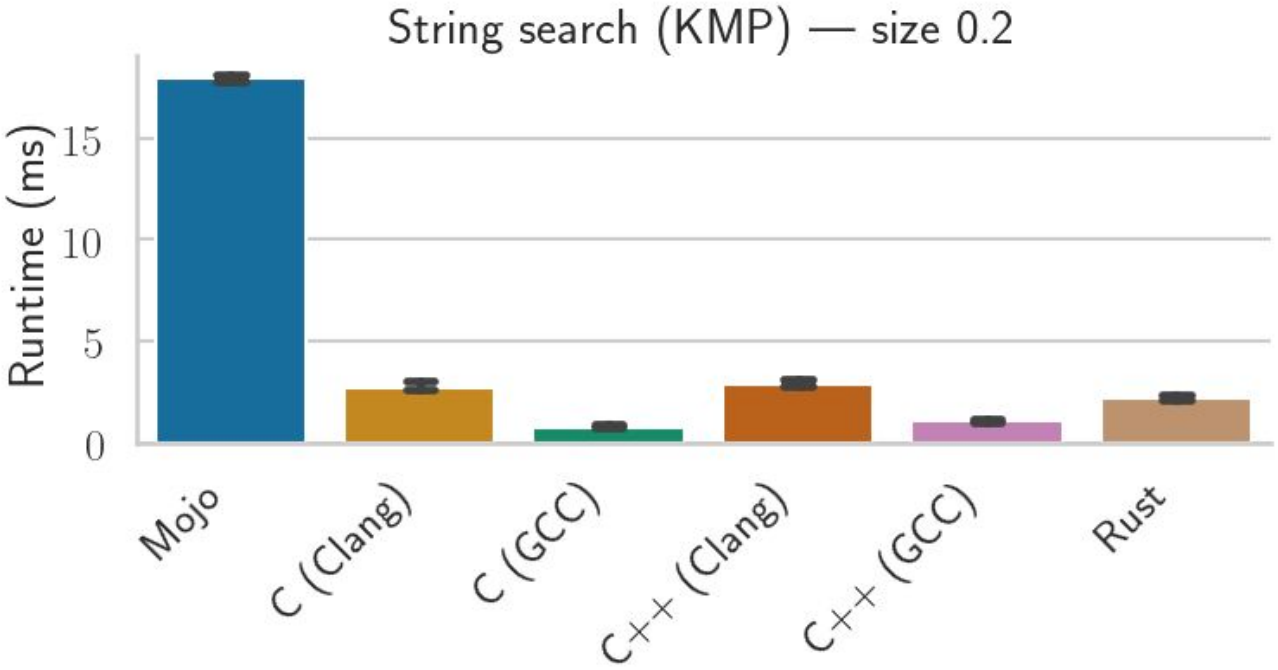
Luckily, Not Python

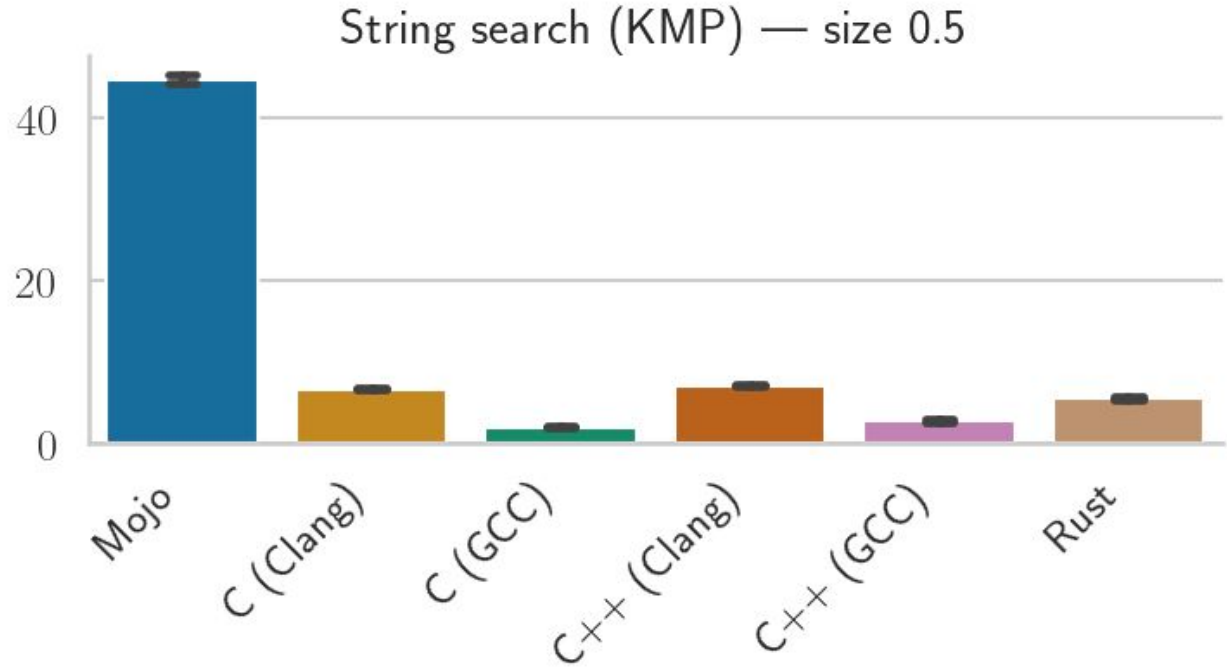


- Performant
- Semi-predictable latency
- Matches C and Rust

Unfortunately, Not C

- Inconsistent performance
- Introduces latency
- Lacks systems level control
- Still evolving





Constant development

Promises for the future

- **One** programming language
  to target **diverse** hardware
- Python's **intuitive** syntax
- **Systems** programming capabilities

Future work

Generalizing	Best for → speed, safety, databases
Expanding	Rosetta code Existing benchmark sets
Exploring	Zig Carbon

Actively working on expanding

Open to suggestions

Feel free to approach me

Any questions?