Modular

Mojo 🔥

MAX

Generativ AI - infrastruktur fra utvikling til implementering.

Modular

Chris Lattner
Medgrunner
& Daglig Leder

Tim Davis
Medgrunner &
President

... og et verdensklasse team av ingeniører med tidligere erfaring fra Google, Meta, Apple, Microsoft, NVIDIA, Tesla, Intel, AMD, Graphcore, SambaNova, Tenstorrent, Tesla, som har utviklet MLIR, Clang, LLVM, PyTorch, ONNX, TensorFlow, osv.

Fabian Tschopp
AI rammeverk ingeniør

- Opprinnelig fra Sveits
- Studerte ved **ETH Zürich** (BSc & MSc)
- 10 års erfaring med utvikling av AI-rammeverk
- Har bodd i Norge i 7 år
- Første ansatt i Modular i Norge!
- På en misjon for å gjøre AI-teknologi tilgjengelig for alle

Generativ AI - Status Quo

Vi ser potensial for forbedring over hele AI-stakken:

1. Utvikling
    - Fragmentert, kompleks, faste grenser
    - Flere programmeringsspråk/modeller

Modeller

System

Maskinvare

CUDA,
OpenMP

Generativ AI - Status Quo

Vi ser potensial for forbedring over hele AI-stakken:

1. Utvikling
   - Fragmentert, kompleks, faste grenser
   - Flere programmeringsspråk/modeller
2. Ytelse
   - Effektivitetsavveininger
   - Fleksibilitet på tvers av maskinvare

Modeller

System

Maskinvare

CUDA, OpenMP

Generativ AI - Status Quo

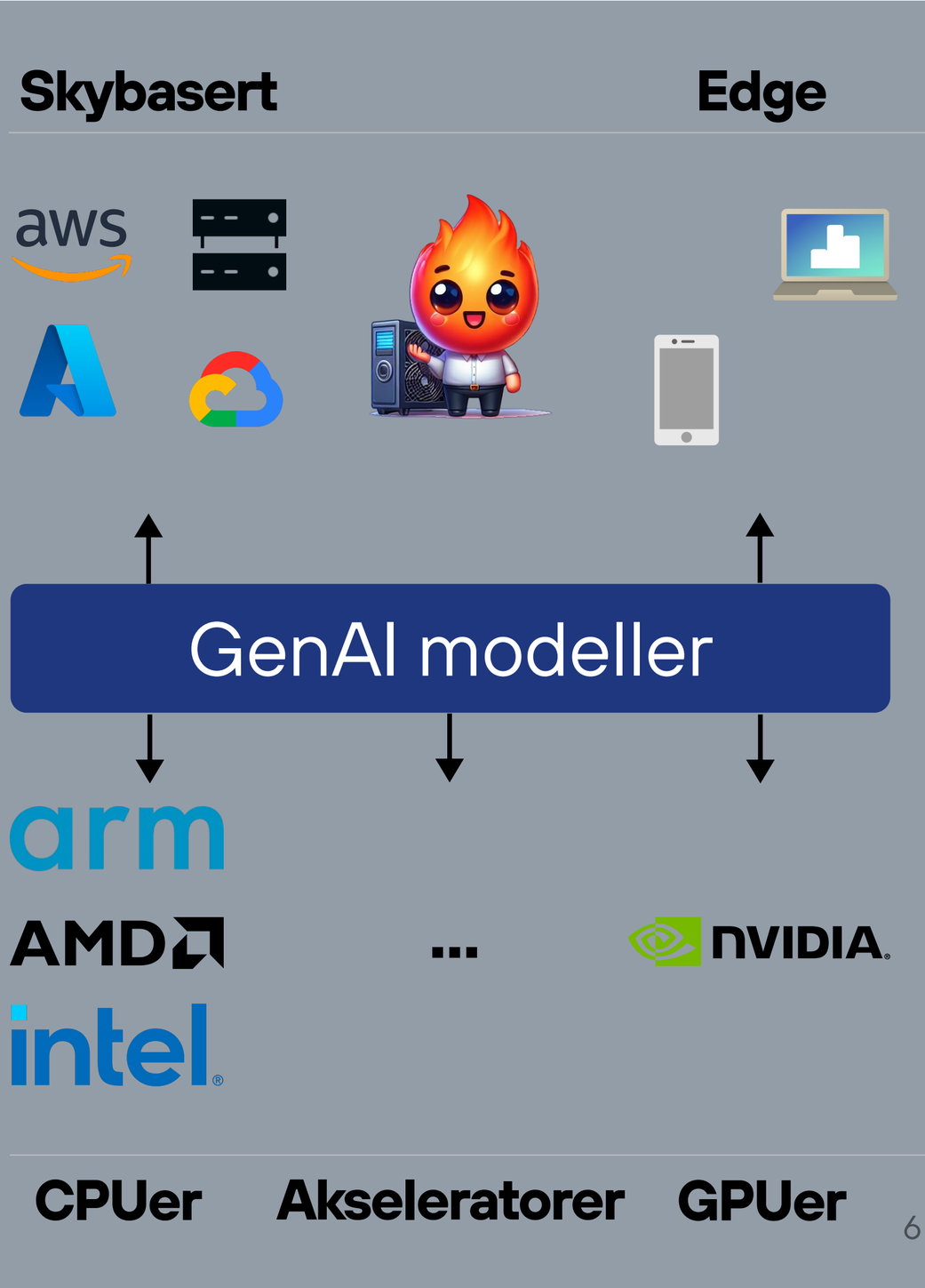
Vi ser potensial for forbedring over hele AI-stakken:

1. Utvikling
   - Fragmentert, kompleks, faste grenser
   - Flere programmeringsspråk/modeller
2. Ytelse
   - Effektivitetsavveininger
   - Fleksibilitet på tvers av maskinvare
3. Implementering
   - Kostnader
   - Python og andre avhengigheter
   - Pålitelighet med verktøy av forskingskvalitet
   - Kompleks programvare

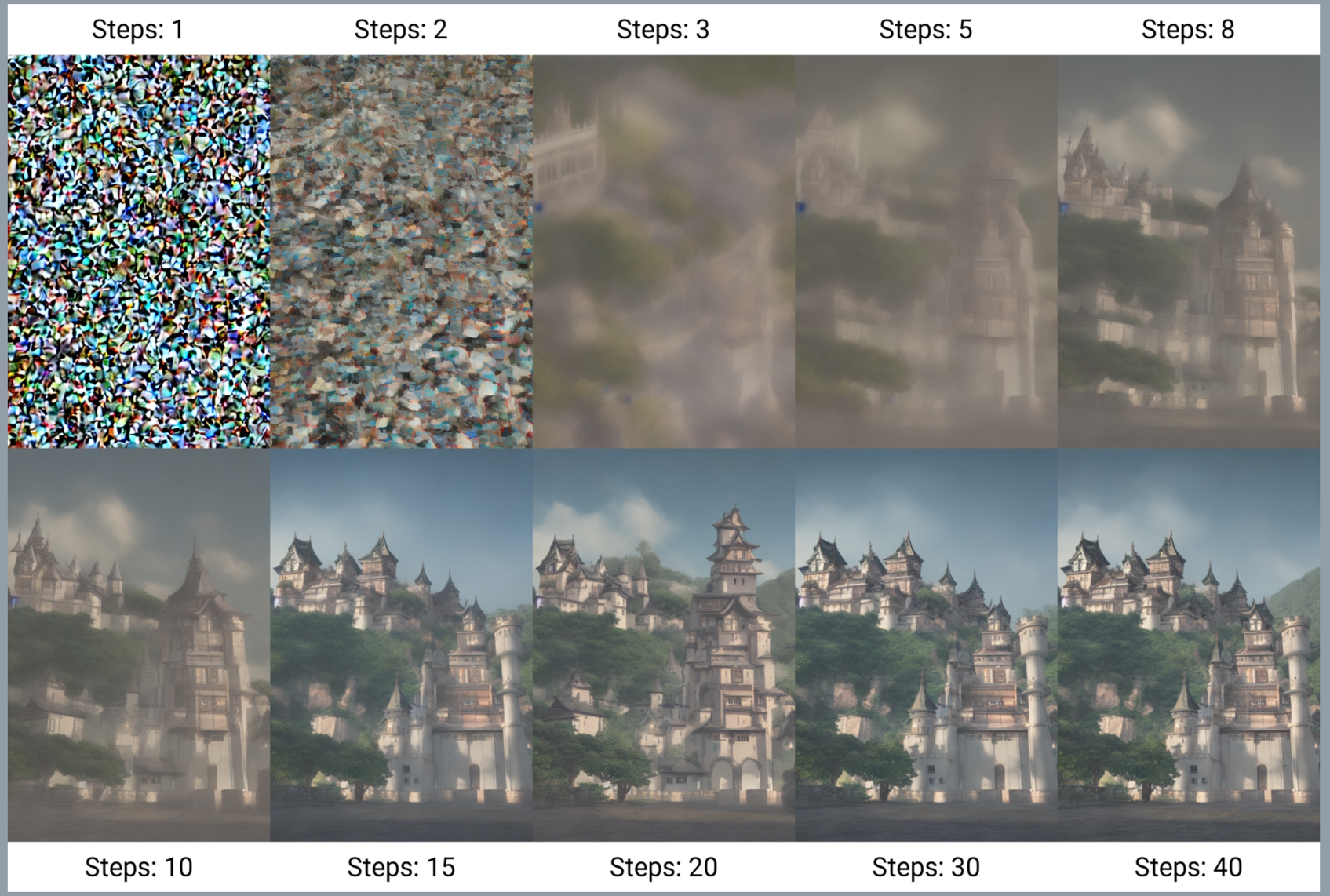
Generativ AI - Status Quo

Vi ser potensial for forbedring over hele AI-stakken:

1. Utvikling
   - Fragmentert, kompleks, faste grenser
   - Flere programmeringsspråk/modeller
2. Ytelse
   - Effektivitetsavveininger
   - Fleksibilitet på tvers av maskinvare
3. Implementering
   - Kostnader
   - Python og andre avhengigheter
   - Pålitelighet med verktøy av forskingskvalitet
   - Kompleks programvare

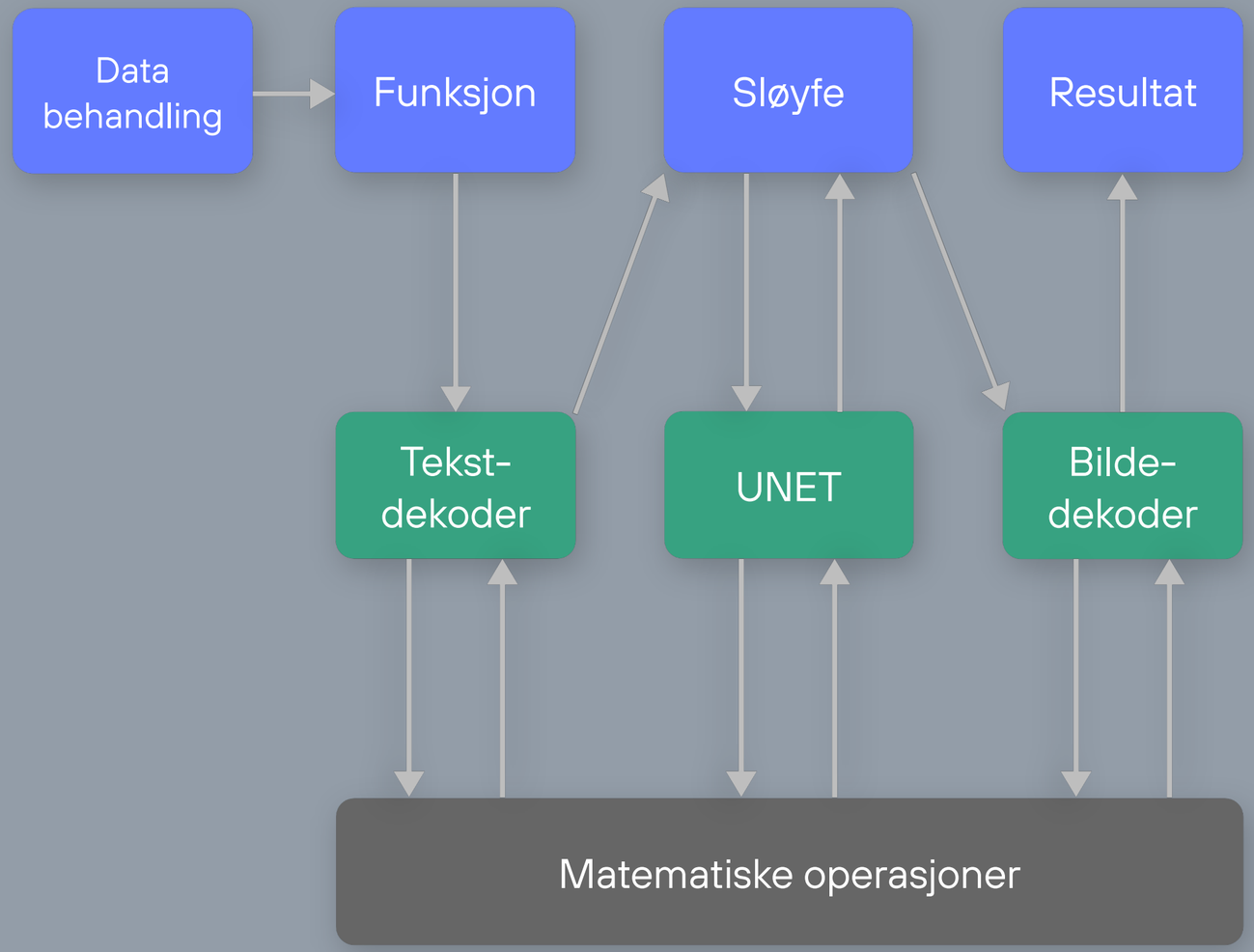


GenAI utviklings eksempel: Stable diffusion



https://en.wikipedia.org/wiki/Stable_Diffusion#/media/File:X-Y_plot_of_algorithmically-generated_AI_art_of_European-style_castle_in_Japan_demonstrating_DDIM_diffusion_steps.png

GenAI utviklings eksempel: Stable diffusion



Python

Dataanalytiker

C++

Ingeniør innen maskinlæring

CUDA

Ingeniør innen maskinvareytelse

Full-stakk AI utvikling

Det er vanskelig å ansette forskere & ingeniører...

- ... som har erfaring med AI-modellering
- ... som kjenner til eksotiske numeriske metoder
- ... som har kunnskap om maskinvaredetaljer

Forskning innen AI modellering kan ikke støtte seg på en: "kompilator-ingeniør i utviklingsprosessen"!

Tykke lag med "magi" **mellom** det du skriver og det som blir kjørt

for eksempel grafopptak, torch script, torch inductor, Triton-Lang, baksystemer, ONNX eksport, graf-kjøreomgivelser...

Utvikling

PyTorch

grafopptak, skripting, sporing, eksport

ONNX

graf-kjøreomgivelser, kompilatorer, tjenesteplattformer

ONNX RUNTIME

TensorRT

/serving

Implementering

Modular's tro

Vi må stoppe portvoktingen av AI teknologi!

Ikke lås teknologien bak en API!

Ikke standardiser og dermed begrens alle til en modellarkitektur!

Ikke la overdrevent rigide programvareløsninger stagnere forskningen!

AI er for nytt og viktig til å gi opp her!

Mojo🔥

Et nytt programmerins-språk som gir ***deg*** superkrefter

Bygge et nytt språk?

Dette er den eneste måten å levere det beste kvalitetsresultatet på!

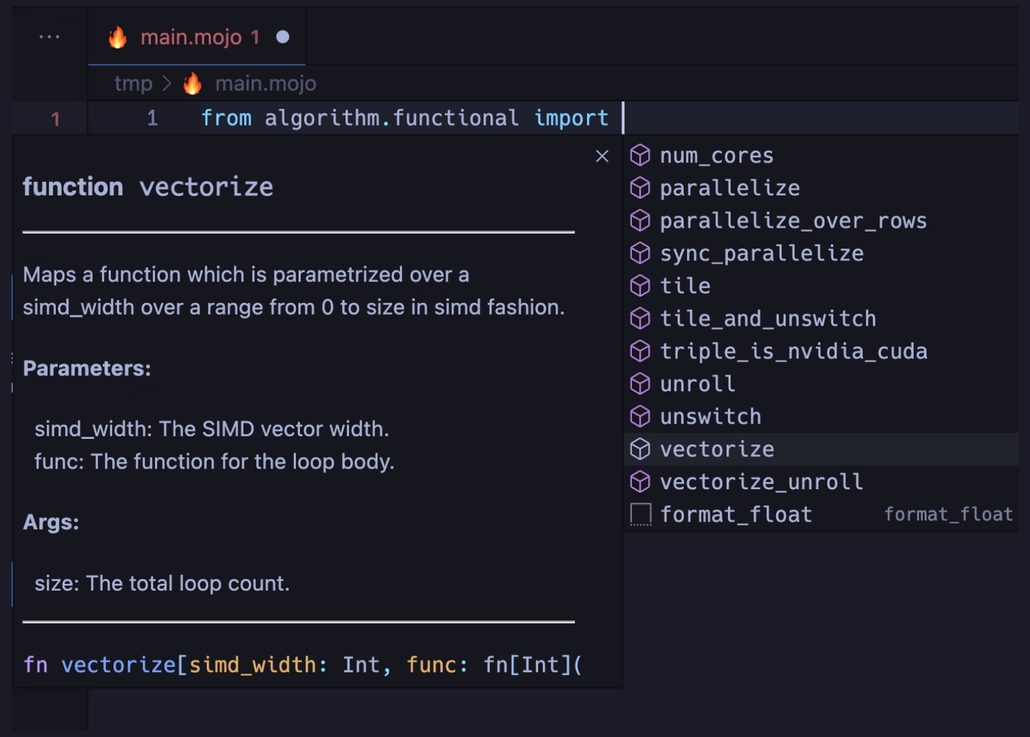
- Naturlig verktøyopplevelse med debugger osv.
- Python gir ikke generell støtte på tvers av CPUer.
- Punktløsninger, verktøy av forskningskvalitet osv. generaliserer ikke.

Men dette krever:

- En konsistent visjon
- Langsiktig planlegging og forpliktelser
- Tilstrekkelig finansering for utviklingen
- Evnen til å tiltrekke spesialisert talent
- Stort målmarked av utviklere

Og vi har gjørt dette før:

OpenCL™ Clang Swift



Mojo🔥 tilbyr full støtte for Jupyter, VSCode, LSP, REPL og debugger

Foren AI fra bunnen av

Ombygging av hele AI-stakken fra bunnen av

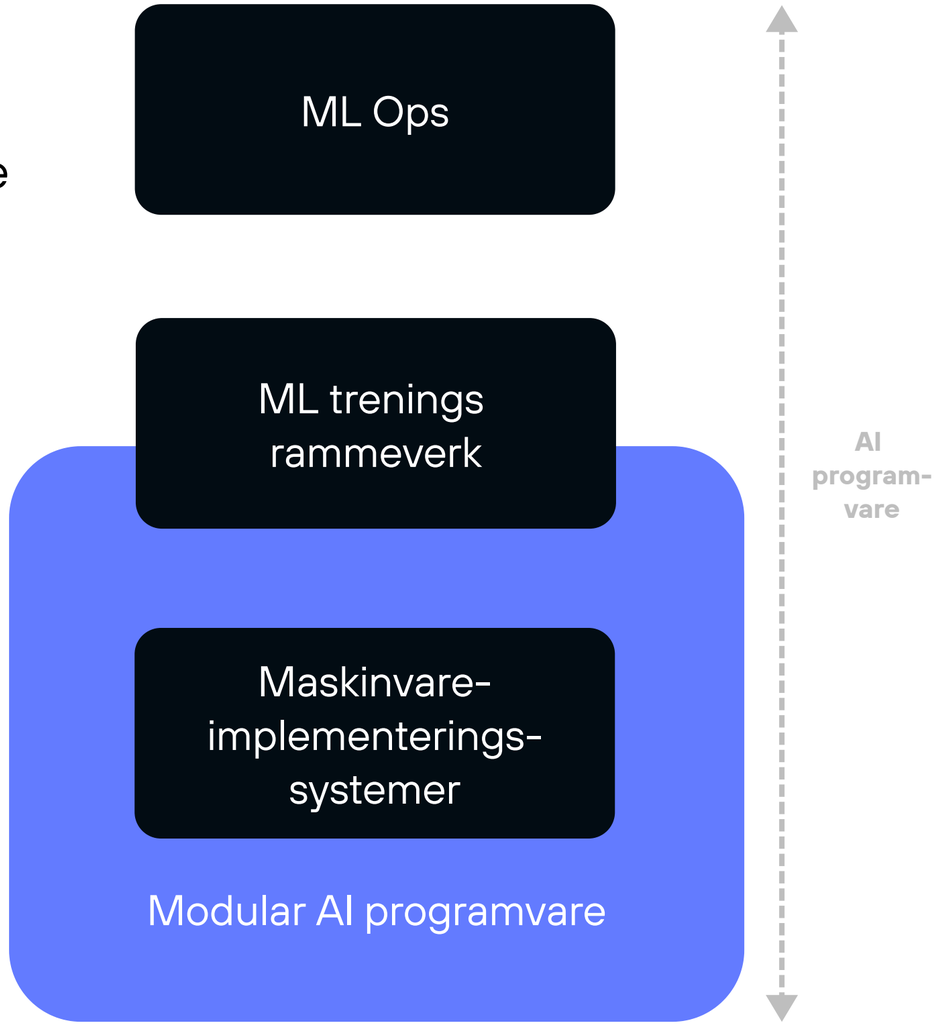
- Start med det mest utfordrende: Kompilatorer, språk, kjøreomgivelse
- Optimaliser for best multig utvikleropplevelse, ikke raske resultater
- Vi har arbeidet med dette i mange år!

Møt AI-utviklere der de er, og løft dem opp

- Kompatibel PyTorch, JAX og TensorFlow erstattning
- Få bedrifts-AI-utviklere ønsker å skrive om modellene sine

Katalyser forskning, men vær ikke et forskningsprosjekt!

- Vi har lært mye de siste ~10 årene innen AI-infrastruktur
- Bring beste praksis-teknikker inn i ett system
- Design programvare fra første prinsipper



Mojo🔥

Superkrefter

01

Programmerbarhet

Overmengde av Python

Moderne programmeringsspråkfunksjoner

Metaprogrammering

Høykvalitets utviklingsverktøy

Mojo🔥

Superkrefter

01

Programmerbarhet

Overmengde av Python

Moderne programmeringsspråkfunksjoner

Metaprogrammering

Høykvalitets utviklingsverktøy

02

Ytelse

Systemnivåkntroll av implementering

Abstraksjoner **uten kostand**

Automatisk justering

Eksponentiell hastighetsøkning over Python

Mojo🔥

Superkrefter

01

Programmerbarhet

Overmengde av Python

Moderne programmeringsspråkfunksjoner

Metaprogrammering

Høykvalitets utviklingsverktøy

02

Ytelse

Systemnivåkntroll av implementering

Abstraksjoner **uten kostand**

Automatisk justering

Eksponentiell hastighetsøkning over Python

03

Fleksibilitet

Verts- / akselerator-kode i samme fil

Samme kode fungerer på CPU og GPU

Integreres med **enhver akselerator** som støtter LLVM/MLIR

Mojo🔥

I et nøtteskal

Designet for AI-utviklere

- men ikke begrenset til AI

Kombinerer det beste fra Python og C

- Python syntaks og økosystem
- Systemnivåkontroll og ytelse

Best i klassen

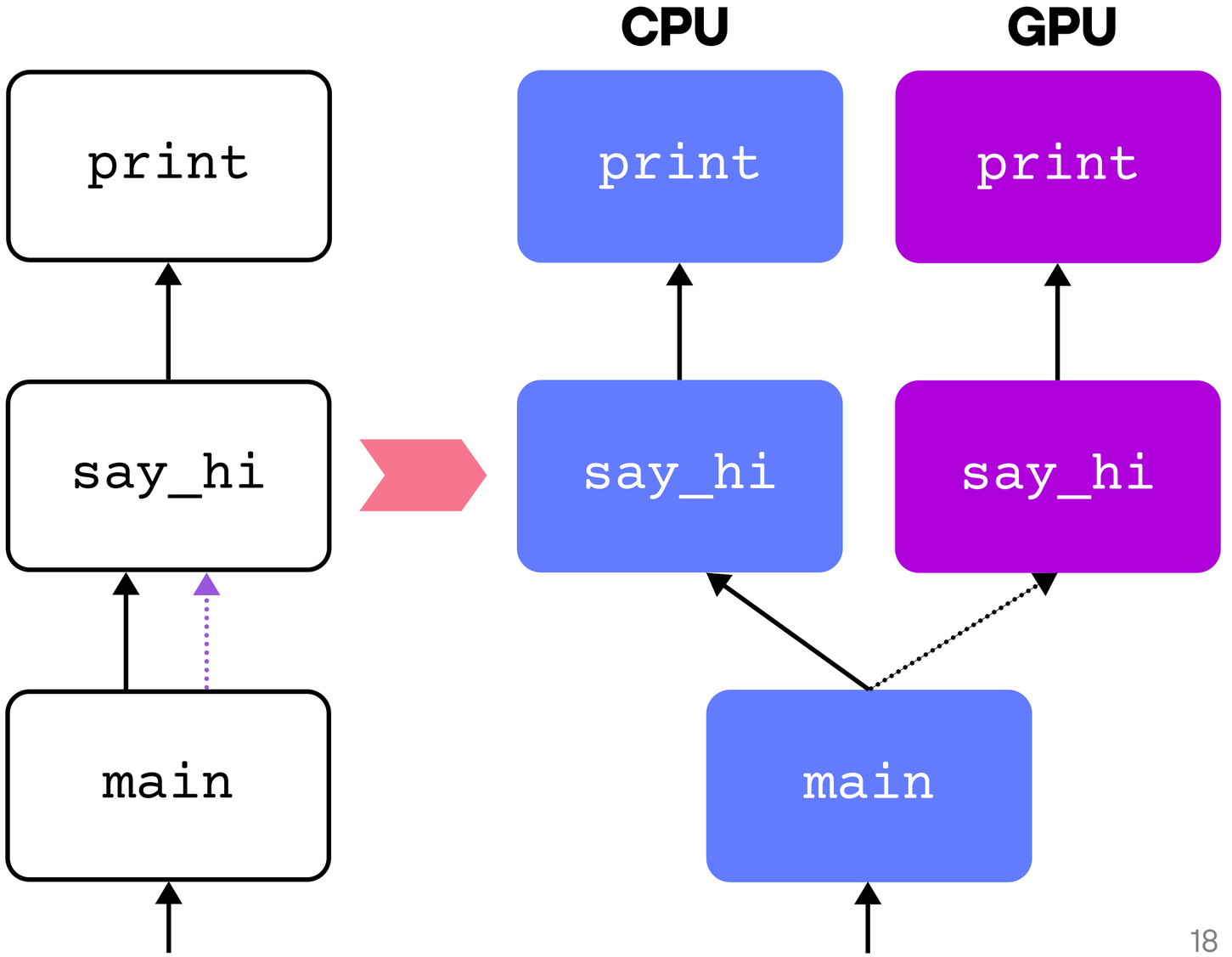
- Tar i bruk de beste kjente teknikkene
- Bygget fra bunnen av med MLIR

```
fn bpe_encode(inout tokens: DynamicVector[Int],
              text: String, inout tok: Tokenizer):
    for c in text:
        tokens.append(tok.find(c))
    while True:
        ...
        for i in range(len(tokens) - 1):
            let str = (tok.vocab[tokens[i]] +
                       tok.vocab[tokens[i + 1]])
            let id = tok.find(str)
            if id != -1 and tok.scores[id] > best_score:
                best_score = tok.scores[id]
                best_id, best_idx = (id, i)

        if best_idx == -1:
            break

        tokens[best_idx] = best_id
        del tokens[best_idx + 1]
```

from https://github.com/tairov/llama2.mojo/blob/master/llama2.mojo

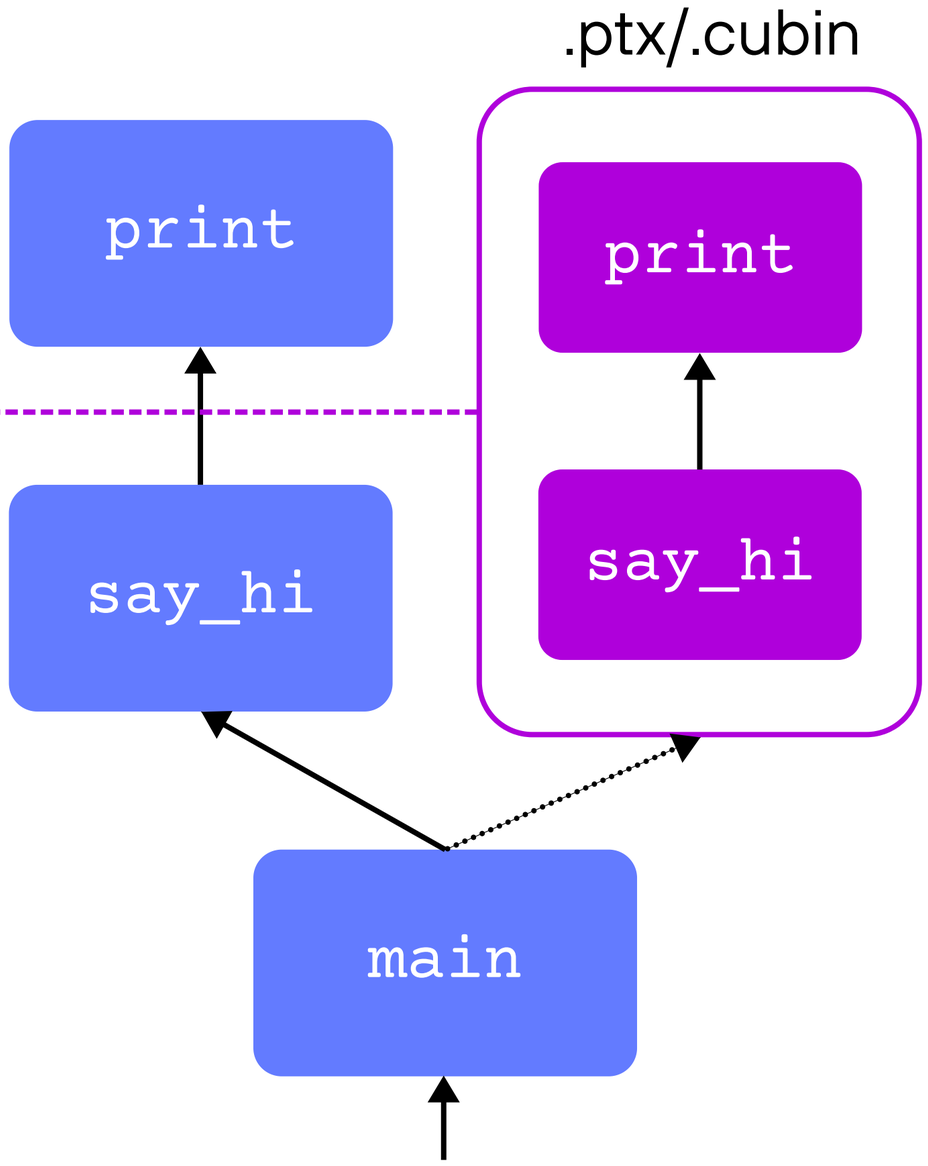
Mojo🔥 Forent programmering for CPU og GPU

```
fn say_hi(value: Int):
    print("hello", value)

def main():
    say_hi(42)

    let f = gpu.Function[say_hi]()
    f(11, grid_dim=1, block_dim=1)
```



Mojo🔥 Delt kompilering

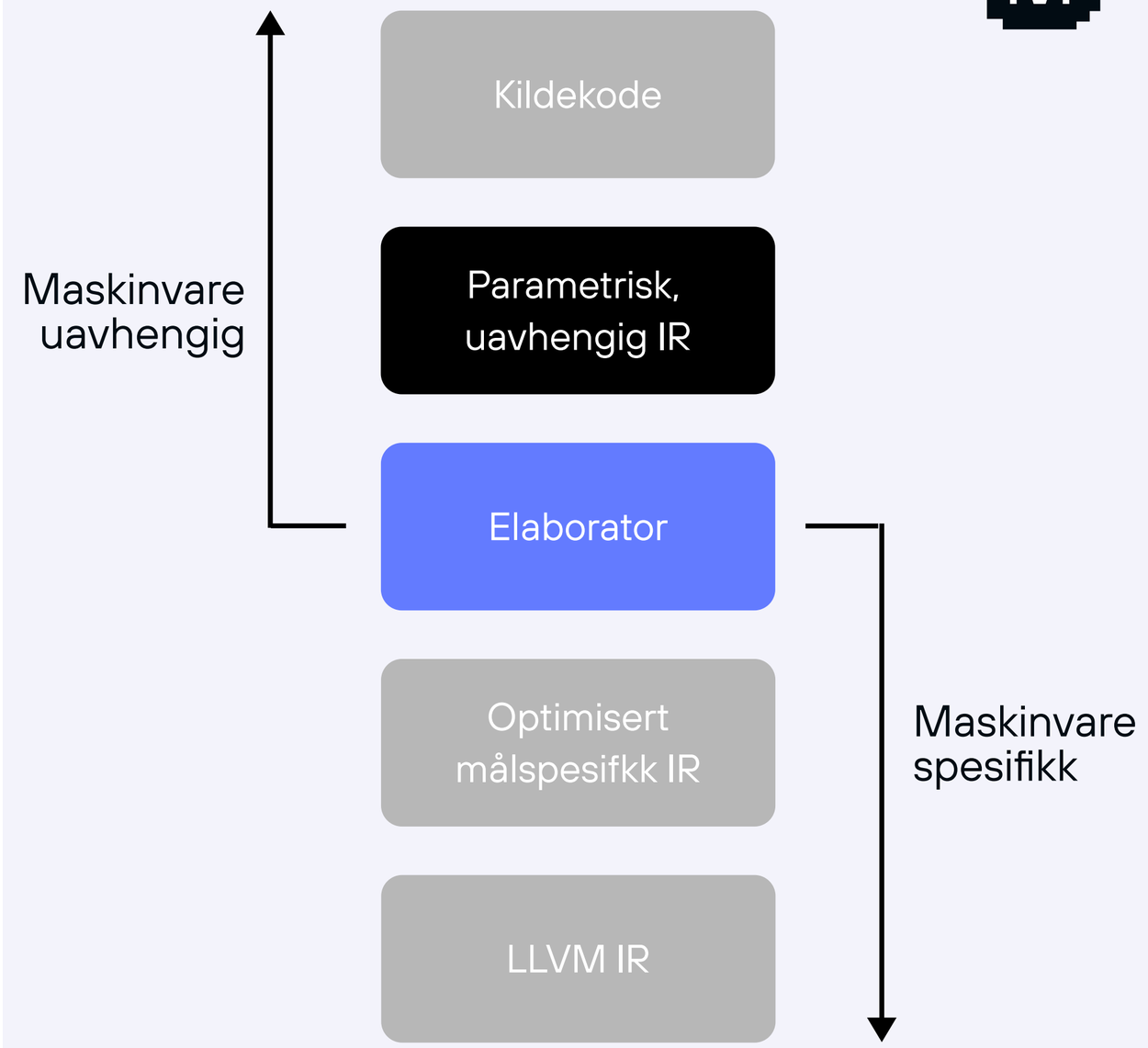
```
fn say_hi(value: Int):
    print("hello", value)

def main():
    say_hi(42)

    alias say_hi_ptx = (
        ".visible .entry say_hi() { ..."
    )
    let f = gpu.Function[say_hi_ptx]()
    f(11, grid_dim=1, block_dim=1)
```



Mojo🔥

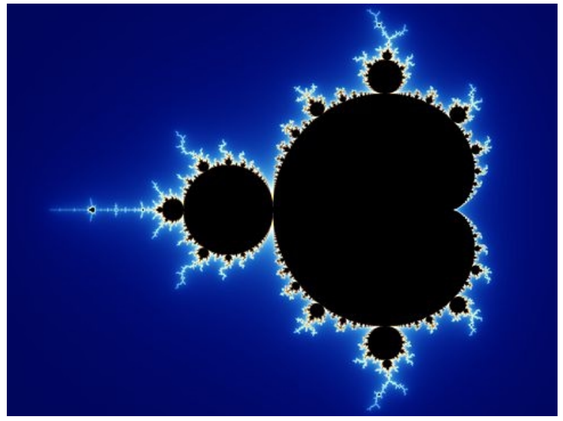
Kompilatorteknologi

- **Uavhengig IR (mellomrepresentasjon)**
  - Maskinvareuavhengig representatsjon
- **JIT kompiler (just-in-time)**
  - Automatisk tilpasning
- **Elaborator**
  - Parametrisk funksjonsinstansering



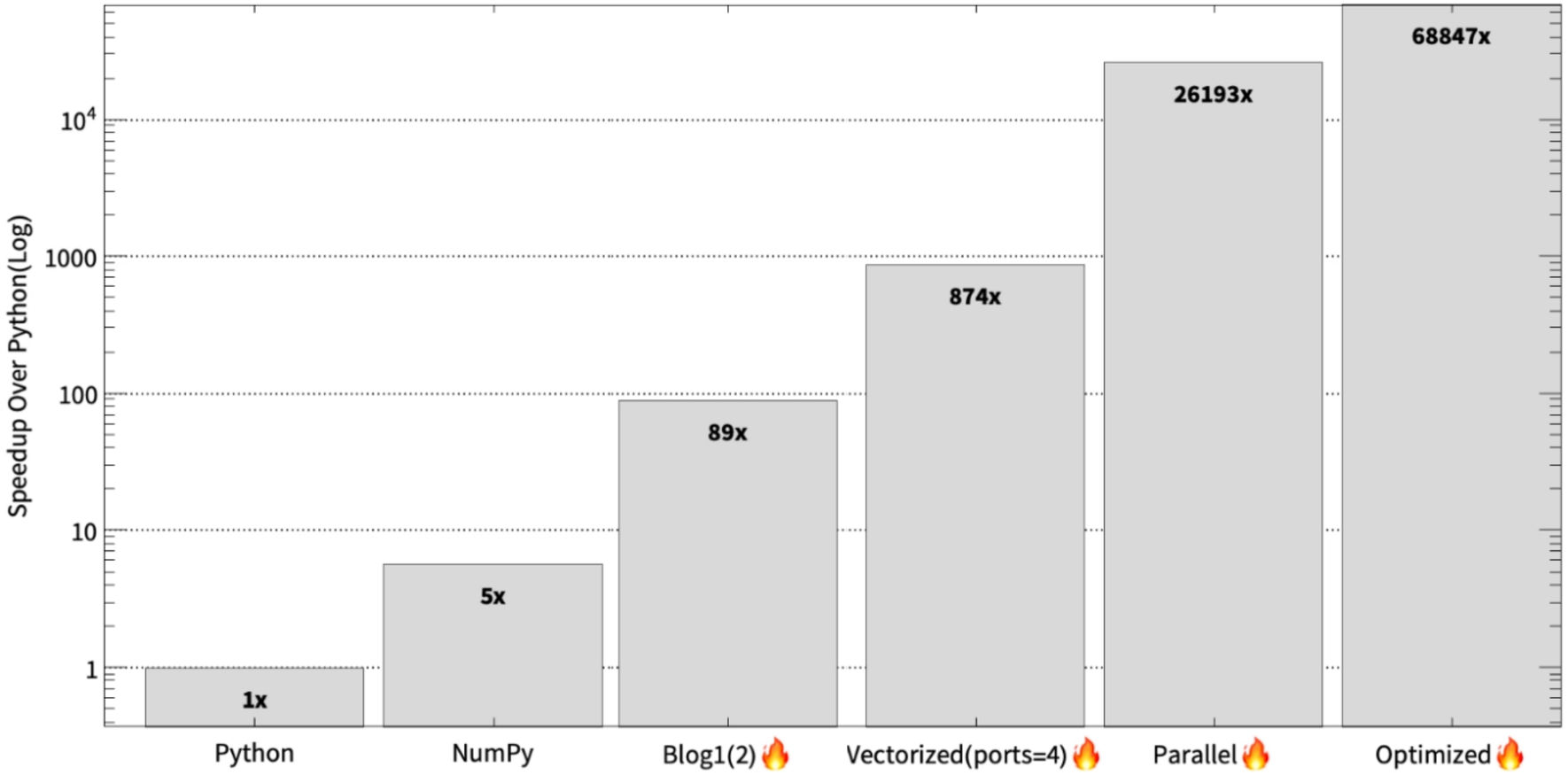
Mandelbrot

Mojo 🔥 er 68 000 ganger raskere enn Python 🐍



```
var in_set_mask: SIMD[DType.bool, simd_width] = True
for i in range(MAX_ITERS):
    if not in_set_mask.reduce_or():
        break
    in_set_mask = z.squared_norm() <= 4
    iters = in_set_mask.select(iters + 1, iters)
    z = z.squared_add(c)
return iters
```

Mandelbrot ytelse



MAX

powered by Mojo 🔥

AI motor APIer

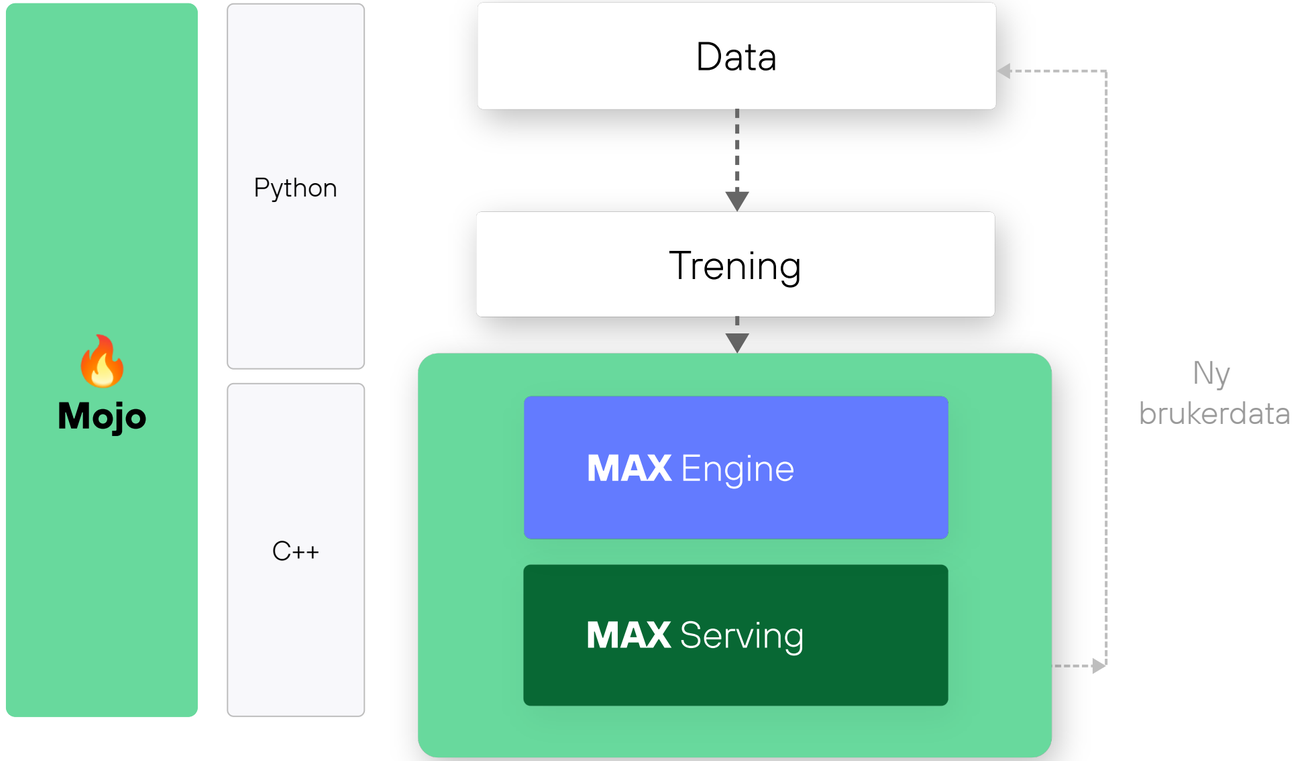
Graf APIer

CPU kjerner

GPU kjerner

Brukerdefinerte operasjoner

Registering av operasjoner



MAX

Hva er fordelene?

01

Drop-in kompatibilitet for PyTorch, JAX, TensorFlow osv. for bedrifter

Full kompatibilitet med alle operasjoner.

Det "beste PyTorch du kan få!"

Best ytelse og lav ventetid rett ut av esken.

MAX

Hva er fordelene?

01

Drop-in kompatibilitet for PyTorch, JAX, TensorFlow osv. for bedrifter

Full kompatibilitet med alle operasjoner.

Det "beste PyTorch du kan få!"

Best ytelse og lav ventetid rett ut av esken.

02

Best GPU- og CPU-ytelse på et bredt spekter av ML-grafer

XLA-stil automatisk sammensmelting, minneplanlegging, støtte for heterogen beregning.

Full generell støtte for dynamiske tensor dimensjoner, kontrollflyt, osv. osv.

MAX

Hva er fordelene?

01

Drop-in kompatibilitet for PyTorch, JAX, TensorFlow osv. for bedrifter

Full kompatibilitet med alle operasjoner.

Det "beste PyTorch du kan få!"

Best ytelse og lav ventetid rett ut av esken.

02

Best GPU- og CPU-ytelse på et bredt spekter av ML-grafer

XLA-stil automatisk sammensmelting, minneplanlegging, støtte for heterogen beregning.

Full generell støtte for dynamiske tensor dimensjoner, kontrollflyt, osv. osv.

03

Utvidbarhet og enkelt å tilpasse som ingen andre løsninger tilbyr

Fra lavnivå AI-motor styring og hele veien opp til å skrive dine egne API-er.

MAX
Graf APIer

Målgruppe
- Avanserte brukere, entusiaster, forskere
- Programvarebiblioteksutviklere

Funksjoner
- Lettvekt og enkel, men kraftig
- Skriv AI modeller direkte i Mojo
- Direkte, fleksibel tilgang
- Familiaritet: Grafer og matematiske operasjoner
- Full kontroll over modellen
- Ingen avhengighet av Python eller rammeverk
- Enkel å kompilere og distribuere

```
m = Module()
g = m.graph(
    name="example",
    inputs=m.types(m.f32(1, 28, 28, 1)),
    outputs=m.types(m.f32(1, 28, 28, 1)),
)

w = g.constant(w_data)
b = g.constant(b_data)

x = g.arg(0)
h = w @ x + b
z = g.relu(h)
g.output(z)
```

MAX

Arbeide med grafer

Når grafen er bygd, kan du

- skrive den ut
- lagre den
- utføre den

```
var g = m.graph(...)

# ... add ops ...

# Print the graph
print(str(g))

# Save
g.save("path/to/my/model")

# Or execute via Engine APIs!
print(session.execute(g, inputs))
```

MAX Graf APIer

```mojo
@value
struct RMSNorm:
    var eps: Float32
    var weight: Symbol

    def __call__(self, input: Symbol) -> Symbol:
        scale = G.rsqrt(G.mean(input**2.0, dim=-1) + self.eps)
        return input * scale * self.weight
```

Det er veldig enkelt å bygge kjente AI-nettverksabstraksjoner

MAX Graf APIer

```
@value
struct TransformerBlock(CollectionElement):
    var attention: Attention
    var feed_forward: FeedForward
    var attention_norm: RMSNorm
    var ffn_norm: RMSNorm

    def __call__(
        self, x: Symbol, pos: Symbol, freqs_cis: Symbol, inout cache: KVCache
    ) -> Symbol:
        normed = self.attention_norm(x)
        attention_out = self.attention(normed, pos, freqs_cis, cache)
        h = x + attention_out
        return h + self.feed_forward(self.ffn_norm(h))
```

Full kontroll over hvordan kildekoden kjøres...
med automatisk sammensmelting og andre optimaliseringer

MAX + Mojo 🔥

```
struct BPETokenizer:
    var sub_tokens: _SubTokenToIdMap
    var reverse_id_map: _SubTokenToIdMap
    var _sub_token_infos: DynamicVector[_SubTokenInfo]
    var _sub_token_id: Int
    var vocab: DynamicVector[StringRef]
    var vocab_size: Int

    def encode(self, str: StringRef) -> EncodedSentence:
        output = EncodedSentence()
```

Selvfølgelig kan du srikve høy-ytelses tokenisering direkte i Mojo og laste eksisterende modell-vekter

MAX

Utvidbarhet

```
@register_max_op("torch.aten.cumsum")
fn cumsum(
  x: Tensor,
  dim: Int
) -> Tensor[x.type]:
    var out = empty_like(x)

    var indices = x.get_nd_indices()
    var val: x.ElemType = 0
    for i in range(x.shape[dim]):
        indices[dim] = i
        val += x.load(indices)
        out.store(indices, val)

    return out
```

PyTorch brukerdefinerte operasjoner - byggeprosess

C++

```
#include <torch/script.h>

torch::Tensor cumsum(torch::Tensor x, int64_t dim) {
  auto out = torch::empty_like(x);

  float* inPtr = x.data_ptr<float>();
  float* outPtr = out.data_ptr<float>();

  float val = 0.0f;
  for (int64_t i = 0; i < x.size(dim); ++i) {
    val += inPtr[i];
    outPtr[i] = val;
  }

  return out;
}

TORCH_LIBRARY(my_ops, m) {
  m.def("cumsum", &cumsum);
}
```

CMake

```
cmake_minimum_required(VERSION 3.1 FATAL_ERROR
project(cumsum)

find_package(Torch REQUIRED)
set(CMAKE_CXX_FLAGS "-O3 -Wall -Wextra")

add_library(cumsum SHARED "src/cumsum.cpp")
target_compile_features(cumsum PRIVATE cxx_std
target_link_libraries(cumsum "${TORCH_LIBRARIE
```

Python

```
import torch
torch.ops.load_library("build/libcu
print(torch.ops.my_ops.cumsum(torch.
```

CMake skript for å finne og bygge mot PyTorch

```
cmake -DCMAKE_PREFIX_PATH="$(python3 -c 'import
torch.utils;
print(torch.utils.cmake_prefix_path)')" .
```

Avanserte **MAX** optimaliseringer

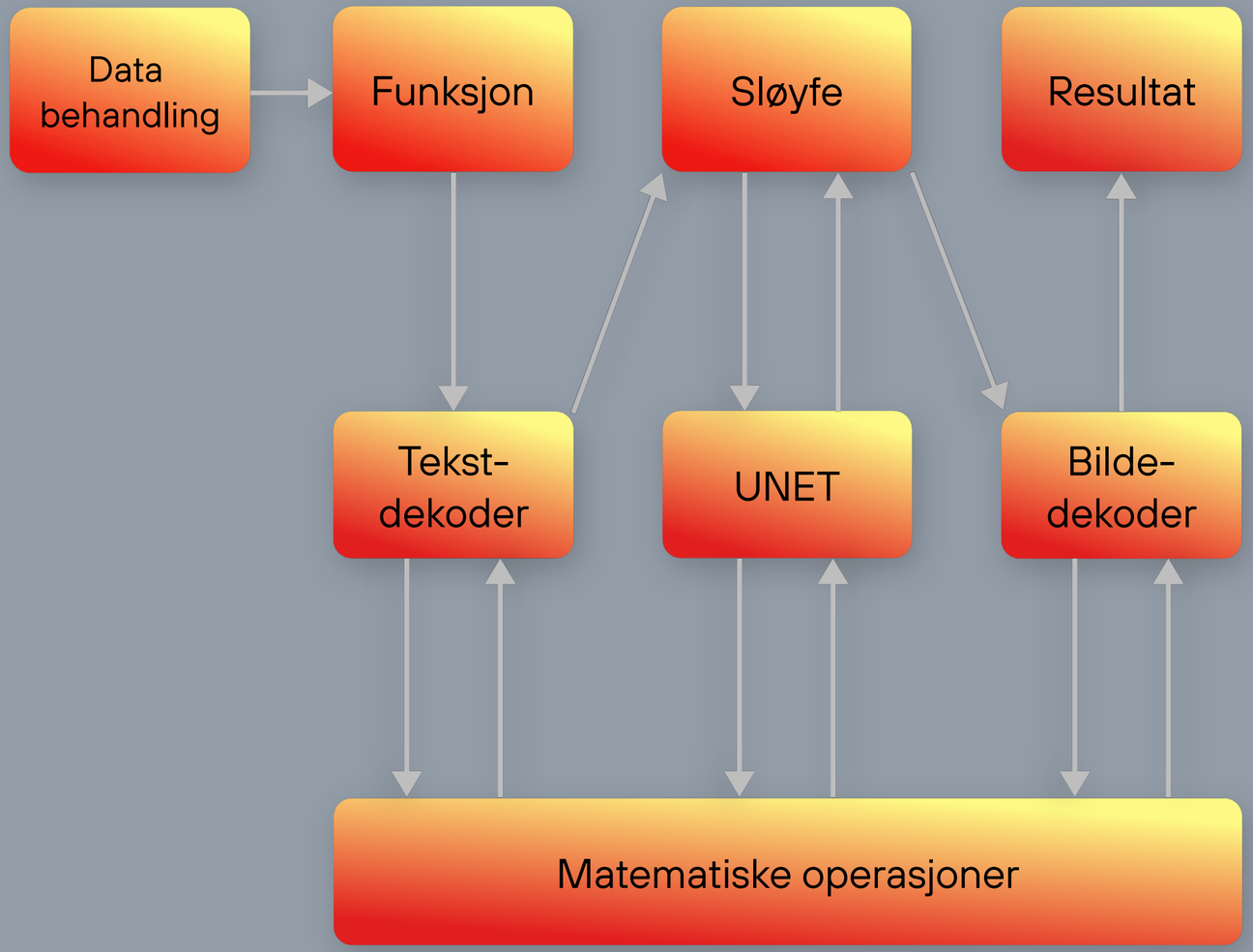
MAX kan "se inn i" dine operasjoner, som muliggjør:

- Grafoptimaliseringer
- Minneplanlegging
- Utsatt kompilering
- Spesialisering for maskinvare og modell
- Sammensmelting og omskrivinger

Koden er fullstendig dynamisk, men modellen kan optimeres for statiske tensor-dimensjoner!

```
@register_max_op("torch.aten.cumsum")
fn cumsum(
  x: Tensor,
  dim: Int
) -> Tensor[x.type]:
   var out = empty_like(x)

   var indices = x.get_nd_indices()
   var val: x.ElemType = 0
   for i in range(x.shape[dim]):
       indices[dim] = i
       val += x.load(indices)
       out.store(indices, val)

   return out
```

GenAI utviklings eksempel: Stable diffusion



Mojo
+
MAX API-er for inferens-motor
+
MAX graf APIer
+
Mojo operasjons-kjerner

Dataanalytiker

Ingeniør innen maskinlæring

Ingeniør innen maskinvareytelse

MAX
GenAI utvikling

Sjekk ut llama.🔥

- 30x mindre kildekode enn llama.cpp
  - llama.cpp: ~ 12 000 linjer
  - llama.🔥: ~ 400 linjer

- Full kontroll over inferensmodellene dine
- Ingen rammeverksavhengigheter
- Full kraft av Mojo🔥

Designet for AI-forskere!

```
model = Llama2(MODEL_PATH)
m = Module()
model.build_into(m, "llama_model")
compiled_model = engine.compile(m)
```

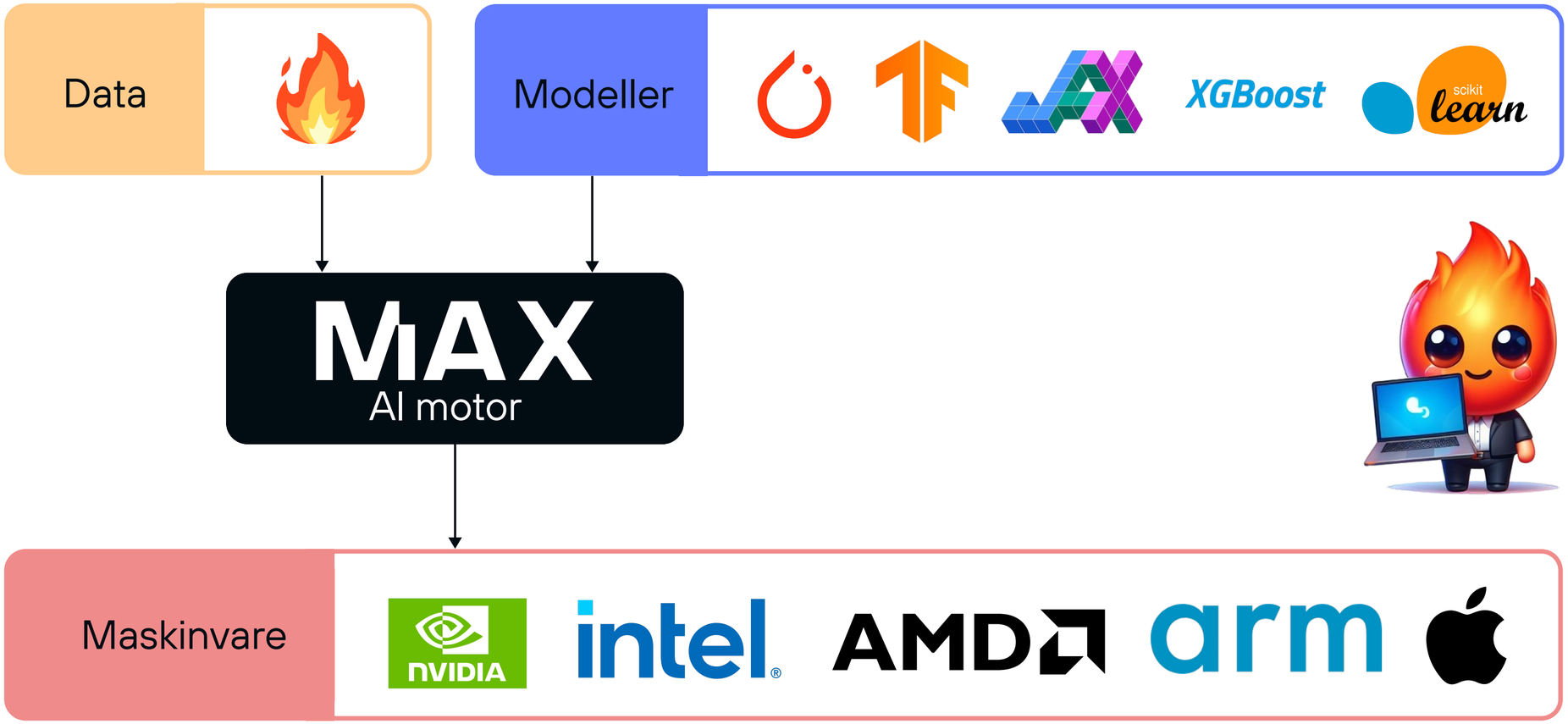
Implementering

Redusert kompleksitet ved distribusjon til produksjonsmiljø

+

Møt ventetid- og kostnadsbudsjetter

Forent Verktøyskjede



MAX

Enkelt å Integrere inferensemotoren i programvare

01

C-bibliotek

Bakoverkompatibel API/ABI

C/C++ programmer

For bruk med progarmmeringsspråket du foretrekker

MAX

Enkelt å Integrere inferensemotoren i programvare

01

C-bibliotek

Bakoverkompatibel API/ABI

C/C++ programmer

For bruk med progarmmeringsspråket du foretrekker

02

Python- og Mojo 🔥-pakker

Bindninger for C-biblioteket

Python og Mojo🔥 kode

Jupyter-notatbøker

Grensesnitt med Python diksjonarer, tupler, lister og numpy matriser.

MAX

Enkelt å Integrere inferensemotoren i programvare

01

C-bibliotek

Bakoverkompatibel API/ABI

C/C++ programmer

For bruk med progarmmeringsspråket du foretrekker

02

Python- og Mojo 🔥-pakker

Bindninger for C-biblioteket

Python og Mojo🔥 kode

Jupyter-notatbøker

Grensesnitt med Python diksjonarer, tupler, lister og numpy matriser.

03

Modular baksystem for Triton Inference Server

Åpen kildekode fra nVidia

KServe API: HTTP & gRPC

Flere ML- og DL-rammeverk

Avanserte funksjoner: Ensembles, forretningslogikk-skripting, osv.

MAX inferensemotor – ytesle

CPU			
intel	3x OPP TIL	2x OPP TIL	2x OPP TIL
AMD	4x OPP TIL	2x OPP TIL	2x OPP TIL
arm	5x OPP TIL	3x OPP TIL	2x OPP TIL

FLOAT32 PÅ DE NYESTE PROGRAMVAREVERSJONER

Modular

Mojo 🔥

Last ned og
eksperimenter nå!

modul.ar/mojo

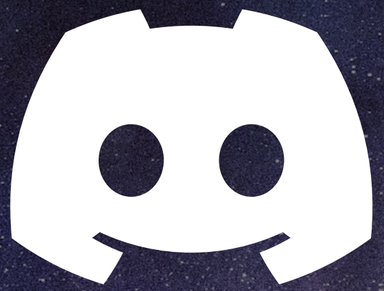
MAX

Kommer våren 2024!

modul.ar/max

Velkommen til vårt utviklergruppe!

modul.ar/github



modul.ar/discord

Takk!