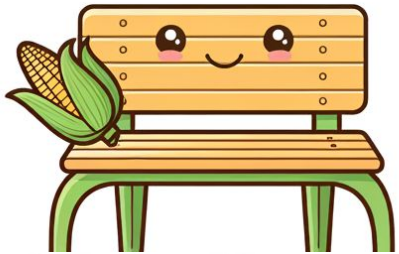
@ NVIDIA GTC



GPU MODE

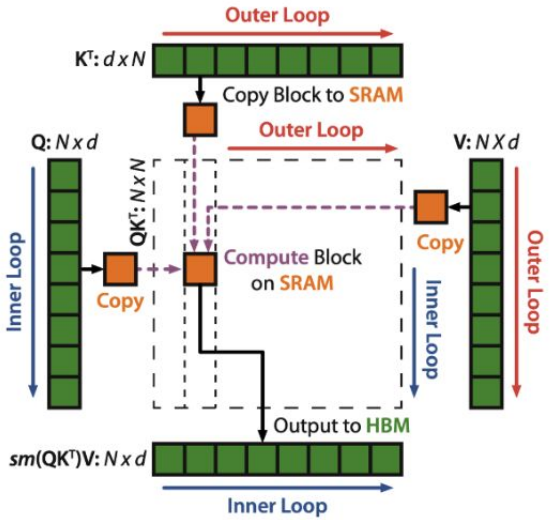
LELAND STANFORD JUNIOR UNIVERSITY
DIE LUFT DER FREIHEIT WEHT
1891

MISSION

Make **GPU** Programming more **accessible**

GPU programming is still too hard

2 years for performant **H100 kernels** for the single **most important** kernel on the market

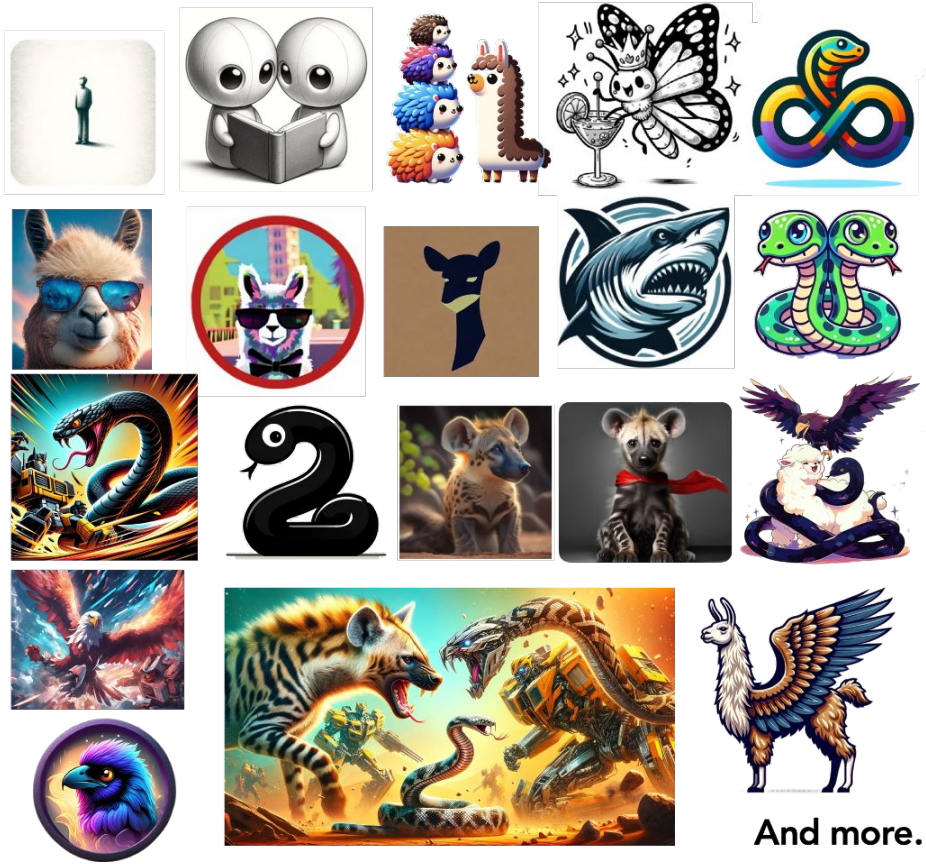


FlashAttention

Dao et al., 2022.

We're experiencing a Cambrian explosion of efficient foundation model architectures 🌍

Model / Paper	Complexity	Decode	Class
Memory Compressed (Liu et al., 2018)	$\mathcal{O}(N_c^2)$	✓	FP+M
Image Transformer (Parmar et al., 2018)	$\mathcal{O}(N.m)$	✓	FP
Set Transformer (Lee et al., 2019)	$\mathcal{O}(kN)$	✗	M
Transformer-XL (Dai et al., 2019)	$\mathcal{O}(N^2)$	✓	RC
Sparse Transformer (Child et al., 2019)	$\mathcal{O}(N\sqrt{N})$	✓	FP
Reformer (Kitaev et al., 2020)	$\mathcal{O}(N \log N)$	✓	LP
Routing Transformer (Roy et al., 2020)	$\mathcal{O}(N\sqrt{N})$	✓	LP
Axial Transformer (Ho et al., 2019)	$\mathcal{O}(N\sqrt{N})$	✓	FP
Compressive Transformer (Rae et al., 2020)	$\mathcal{O}(N^2)$	✓	RC
Sinkhorn Transformer (Tay et al., 2020b)	$\mathcal{O}(B^2)$	✓	LP
Longformer (Beltagy et al., 2020)	$\mathcal{O}(n(k+m))$	✓	FP+M
ETC (Ainslie et al., 2020)	$\mathcal{O}(N_g^2 + NN_g)$	✗	FP+M
Synthesizer (Tay et al., 2020a)	$\mathcal{O}(N^2)$	✓	LR+LP
Performer (Choromanski et al., 2020a)	$\mathcal{O}(N)$	✓	KR
Funnel Transformer (Dai et al., 2020)	$\mathcal{O}(N^2)$	✓	FP+DS
Linformer (Wang et al., 2020c)	$\mathcal{O}(N)$	✗	LR
Linear Transformers (Katharopoulos et al., 2020)	$\mathcal{O}(N)$	✓	KR
Big Bird (Zaheer et al., 2020)	$\mathcal{O}(N)$	✗	FP+M
Random Feature Attention (Peng et al., 2021)	$\mathcal{O}(N)$	✓	KR
Long Short Transformers (Zhu et al., 2021)	$\mathcal{O}(kN)$	✓	FP + LR
Poolingformer (Zhang et al., 2021)	$\mathcal{O}(N)$	✗	FP+M
Nyströmformer (Xiong et al., 2021b)	$\mathcal{O}(kN)$	✗	M+DS
Perceiver (Jaegle et al., 2021)	$\mathcal{O}(kN)$	✓	M+DS
Clusterformer (Wang et al., 2020b)	$\mathcal{O}(N \log N)$	✗	LP
Luna (Ma et al., 2021)	$\mathcal{O}(kN)$	✓	M
TokenLearner (Ryoo et al., 2021)	$\mathcal{O}(k^2)$	✗	DS
Adaptive Sparse Transformer (Correia et al., 2019)	$\mathcal{O}(N^2)$	✓	Sparse
Product Key Memory (Lample et al., 2019)	$\mathcal{O}(N^2)$	✓	Sparse
Switch Transformer (Fedus et al., 2021)	$\mathcal{O}(N^2)$	✓	Sparse
ST-MoE (Zoph et al., 2022)	$\mathcal{O}(N^2)$	✓	Sparse
GShard (Lepikhin et al., 2020)	$\mathcal{O}(N^2)$	✓	Sparse
Scaling Transformers (Jaszczur et al., 2021)	$\mathcal{O}(N^2)$	✓	Sparse
GLaM (Du et al., 2021)	$\mathcal{O}(N^2)$	✓	Sparse

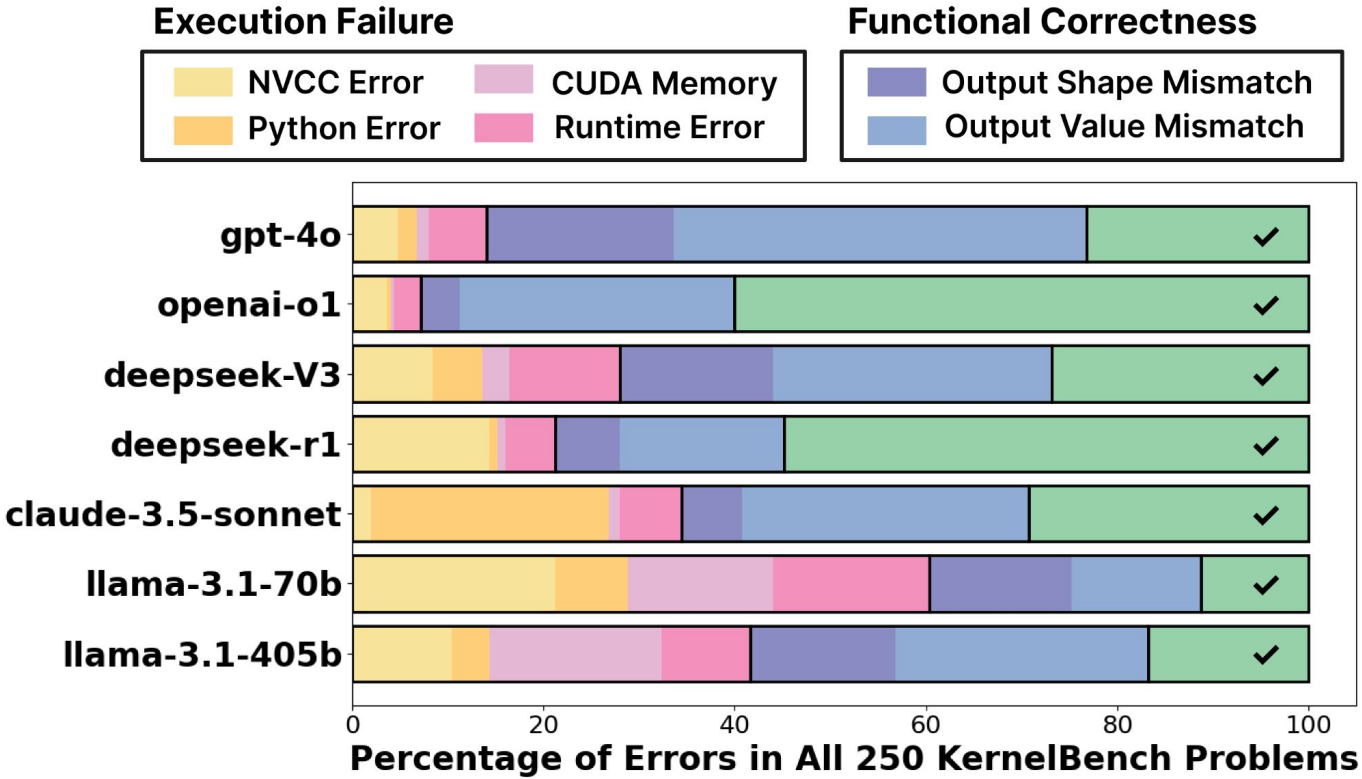


And more…!

Create an expert LLM GPU programmer 

That **anyone** can **use** or **reproduce**

KernelBench : Existing models are bad!



Work led by Anne Ouyang and Simon Guo https://scalingintelligence.stanford.edu/blogs/kernelbench/

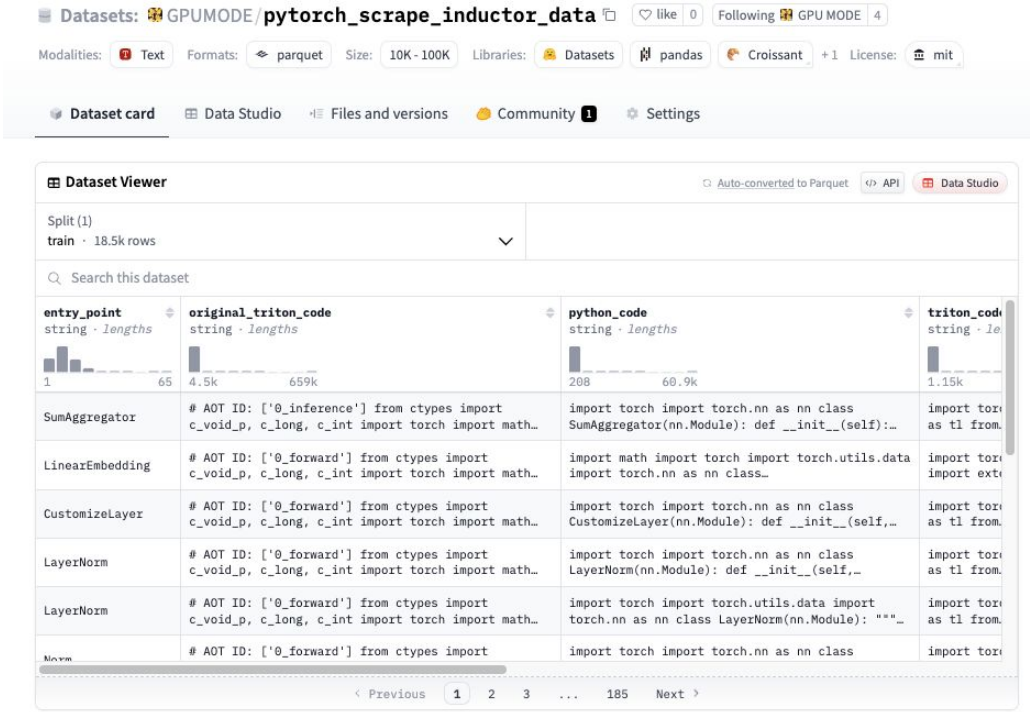
Why are LLMs so bad at writing kernels?

Hypothesis: Data starvation

KernelBook : The largest kernel dataset on earth

18K pairs of (torch, triton) kernels generated using **real** PyTorch models from Github

2k unique Triton kernels with permissive licenses from Github



Work led by Sahan Paliskara and Mark

https://huggingface.co/GPUMODE

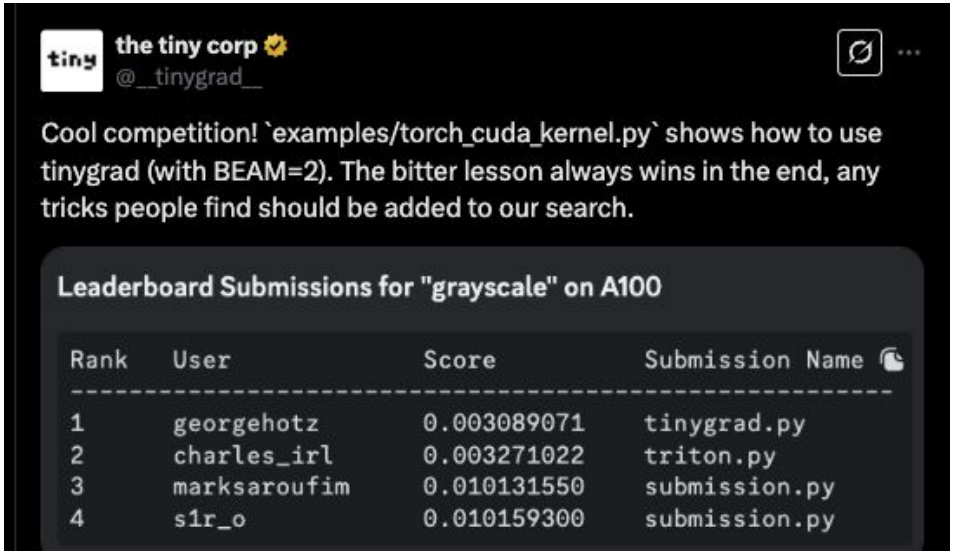
KernelBot  : How to get more high quality human data?

Host regular competitions on Discord directly with free access to NVIDIA H100 and AMD MI300

2,000+ submissions on practice round!

> Data will be released with a permissive license for anyone to study / use.

> To participate join discord.gg/gpumode and `/leaderboard submit`



Work led by: Matej Sirovatka, Alex Zhang, Erik Schultheis, Ben Horowitz, Mark Saroufim

https://github.com/gpu-mode/discord-cluster-manager

Thank you to our sponsors: **Modal, NVIDIA, AMD,** and **Nebius**

18K PyTorch models from Github

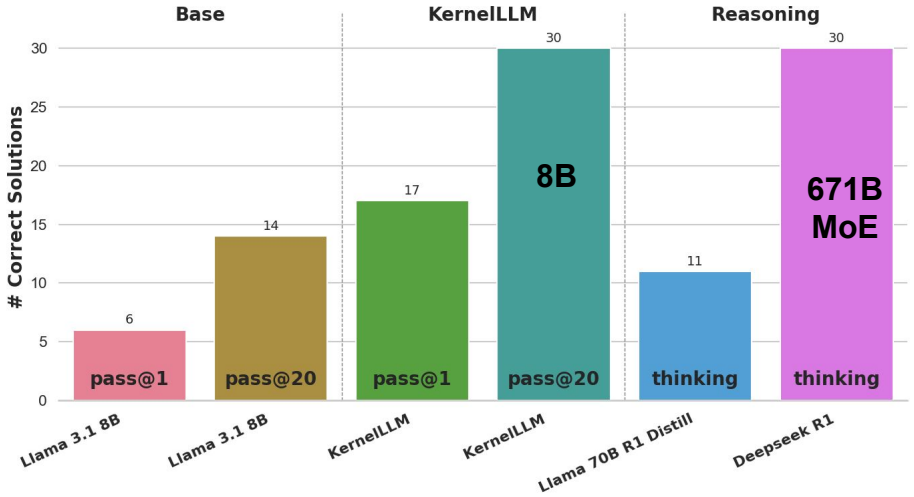
```python
import torch
from torch import nn


class Swish(nn.Module):

    def forward(self, x):
        return x.mul_(torch.sigmoid(x))


def get_inputs():
    return [torch.rand([4, 4, 4, 4])]


def get_init_inputs():
    return [[], {}]
```

TORCH_LOGS=
“output_code”

Torch.compiled Triton Code

```python
import torch
import triton
import triton.language as tl
from torch._inductor.runtime.triton_heuristics import grid
from torch._C import _cuda_getCurrentRawStream as get_raw_stream
from torch import nn
assert_size_stride = torch._C._dynamo.guards.assert_size_stride


@triton.jit
def triton_poi_fused_mul_sigmoid_0(in_ptr0, out_ptr1, xnumel, XBLOCK: tl.constexpr):
    xnumel = 256
    xoffset = tl.program_id(0) * XBLOCK
    xindex = xoffset + tl.arange(0, XBLOCK)[:]
    xmask = xindex < xnumel
    x0 = xindex
    tmp0 = tl.load(in_ptr0 + x0, xmask)
    tmp1 = tl.sigmoid(tmp0)
    tmp2 = tmp0 * tmp1
    tl.store(out_ptr1 + x0, tmp2, xmask)


def call(args):
    arg0_1, = args
    args.clear()
    assert_size_stride(arg0_1, (4, 4, 4, 4), (64, 16, 4, 1))
    with torch.cuda._DeviceGuard(0):
        torch.cuda.set_device(0)
        get_raw_stream(0)
        triton_poi_fused_mul_sigmoid_0[grid(256)](arg0_1, arg0_1, 256,
            XBLOCK=256, num_warps=4, num_stages=1)
    return arg0_1,


class SwishNew(nn.Module):

    def forward(self, input_0):
        arg0_1 = input_0
        output = call([arg0_1])
        return output[0]
```

KernelLLM : SOTA SFT model



A lot more work to be done

More tasks beyond translation

1. Minimal edit with maximal performance gain
2. Kernel discovery
3. Autotune heuristics
4. How to merge improvements in PyTorch
5. Abstraction discovery

These are hard problems and we want more people to work with!
https://gpu-mode.github.io/popcorn/

A lot more work to be done

More tasks beyond translation

1. Edit
2. Kernel discovery
3. Autotune heuristics
4. How to merge improvements in PyTorch
5. **Abstraction discovery**

These are hard problems and we want more people to work with!
https://gpu-mode.github.io/popcorn/

ThunderKittens: library for developing simple, fast, and adorable AI kernels

Workload	TK LoC	Reference LoC	Speed up (min-max)
H100 Attention inference	217	2325 (CUTLASS FA3)	0.87-1.14×
H100 BF16 GEMM	84	463 (CUTLASS)	0.98-2.05×
H100 FP8 GEMM	91	Closed Source (CuBLASLt)	0.93-1.10×
H100 Convolution	131	624 (CUDA FlashFFTConv)	4.6-4.7×
H100 Based linear attention	282	89 (Triton)	3.7-14.5×
H100 Hedgehog linear attention	316	104 (Triton)	4.0-6.5×
H100 Mamba-2	192	532 (Triton)	3.0-3.7×
H100 Rotary	101	119 (Triton)	1.1-2.3×
4090 Attention ($D = 64$)	93	1262 (CUTLASS FA2)	0.96-0.98×
4090 Attention ($D = 128$)	93	1262 (CUTLASS FA2)	0.89-0.95×
Apple M2 Attention ($D = 64$)	47	343 (Apple MLX)	1.12-1.15×
Apple M2 GEMM	27	412 (Apple MLX)	1.04-1.12×

Library sizes

Framework	Size	Date
CUTLASS	22 MB	10/22
Triton	12.6MB	10/22
TK	<1MB	10/22

TK kernels use few lines of code (~simple) and are fast.

Our contributions

KernelBench  : **The reference** kernel eval suite

KernelBook  : The **largest kernel dataset** ever published

KernelBot  : A viral platform for **competitive GPU coding**

KernelLLM  : A **SOTA** SFT **LLM Triton** programmer

ThunderKittens  : A **simple library** for **blazing fast** AI kernels

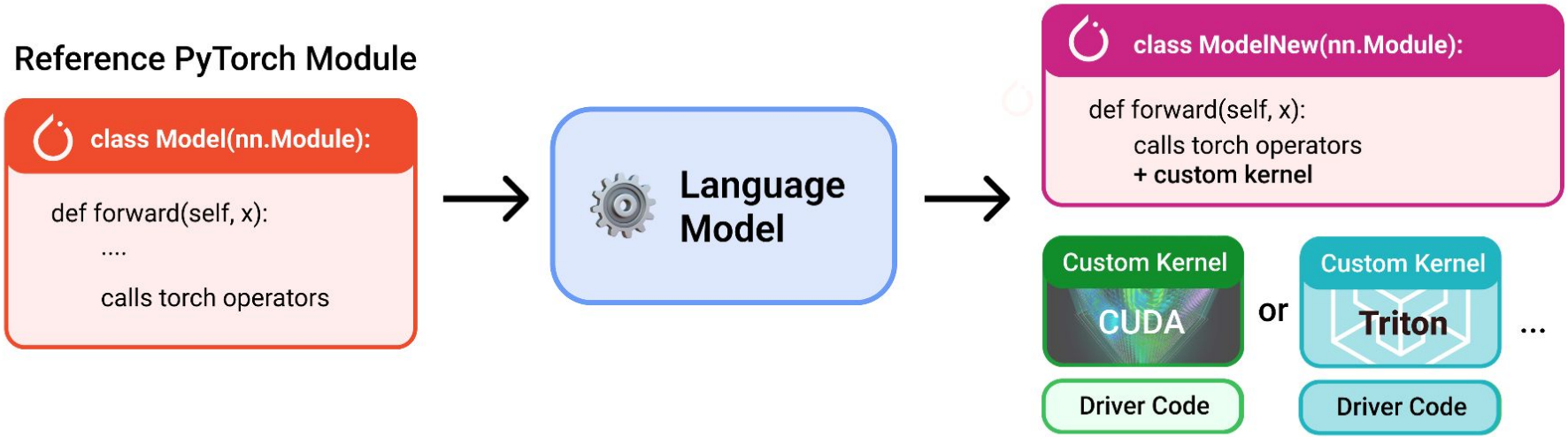
KernelBench

Can LLMs write GPU kernels?

Anne Ouyang*, Simon Guo*, Simran Arora, Alex L Zhang,
William Hu, Christopher Ré, Azalia Mirhoseini

Can we leverage LLM as a Compiler / Transpiler?

Task: PyTorch → Fast Custom Operators (in CUDA / Triton)



What kind of problems should we evaluate on?

KernelBench: Benchmark for Kernel Generations

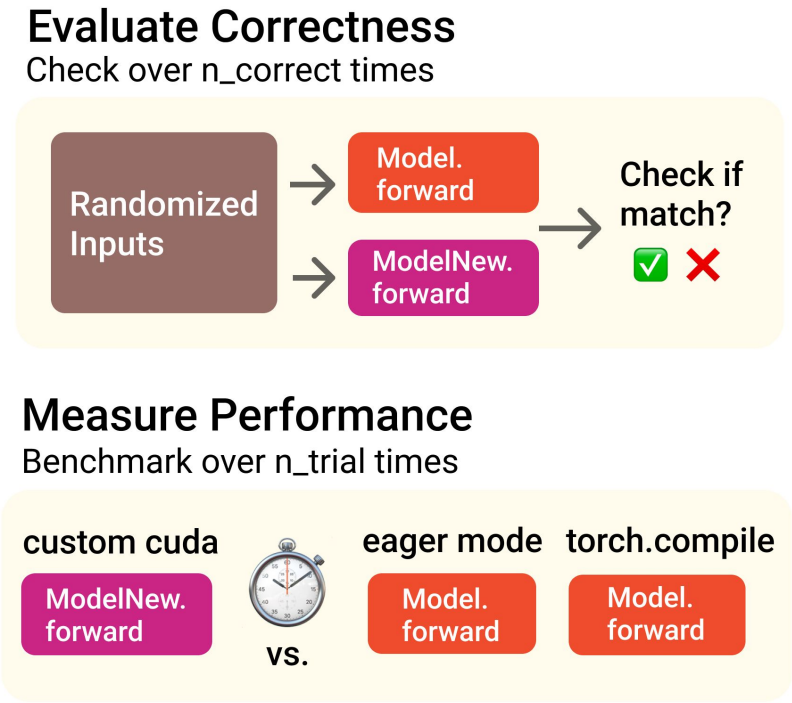
Collection of **250+** PyTorch Problems across **3 Levels**

Level 1 Single-kernel operator	**Level 2** Fused patterns	**Level 3** Full model architecture
Matrix Multiplications	Conv + Bias + ReLU	MiniGPT
Convolutions	Matmul + Scale + Sigmoid	Vision Transformer
Layer normalization		Mamba
Against highly-optimized hand-written kernels	Test ability to effectively perform kernel fusion	Long complex programs + algorithmic opportunities

Evaluating Generated GPU Kernels

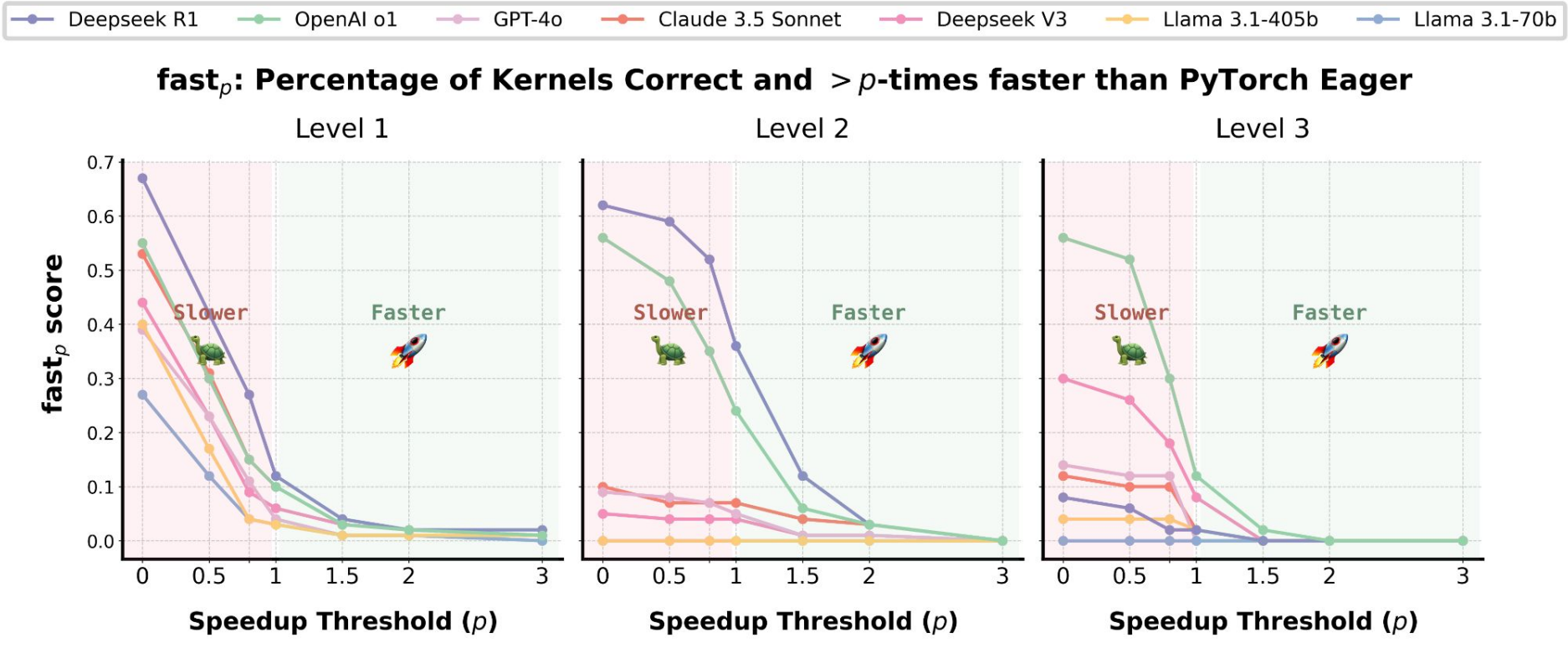
How many generated kernels are **correct** and **faster** than **PyTorch Eager***?

*torch.compile results available in paper

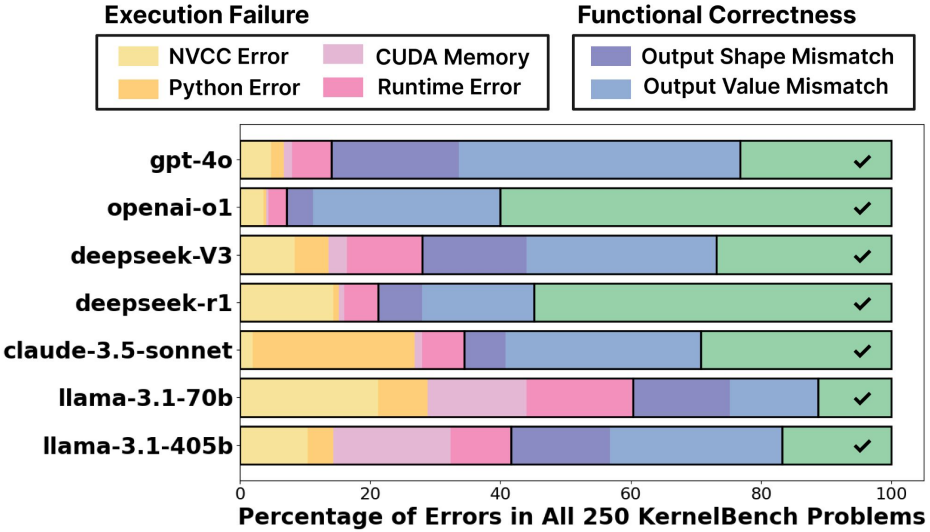


Most Generated Kernels are **Slow**

Frontier models outperform the PyTorch Eager <20% of the time



Understand the Gap



Syntax Errors and unrunnable programs

Hard to reason about logical errors

Do not effectively leverage hardware

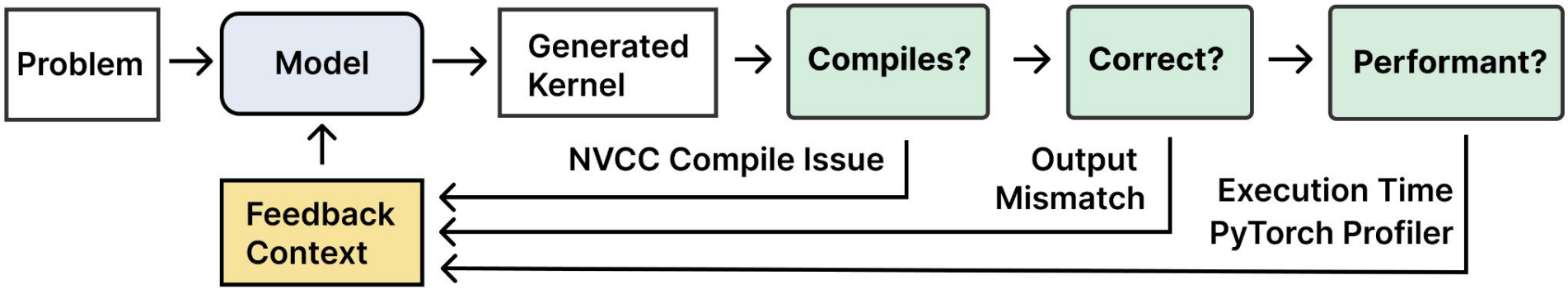
GPU Programming scarce in training data

0.073% CUDA in The Stack (code corpus)

2k Unique Triton programs on GitHub

KernelBench Provides a **Rich Environment**

A playground to leverage ground-truth signals



We started exploring the following in the paper

Scaling test-time compute

Generate labelled trajectories

Effectively leverage tool use

… many ideas to try!

Exciting Work Targeting KernelBench

NVIDIA DEVELOPE

Technical Blog

Searc

Generative AI

Automating with DeepS Scaling

Feb 12, 2025

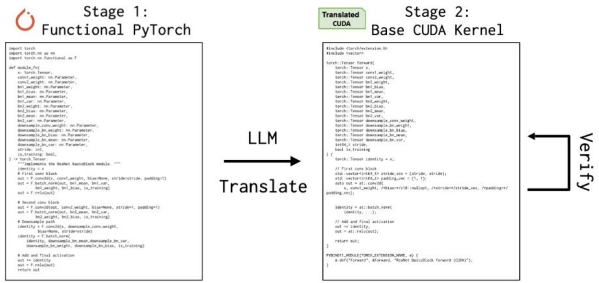
By Terry Chen, Bing Xu and Kirthi Devleke

sakana.ai

The AI CUDA Engineer: Agentic CUDA Kernel Discovery, Optimization and Composition

February 20, 2025

An End-to-End Agent for Optimizing CUDA Kernels



Then, the AI CUDA Engineer 👷
translates torch to CUDA

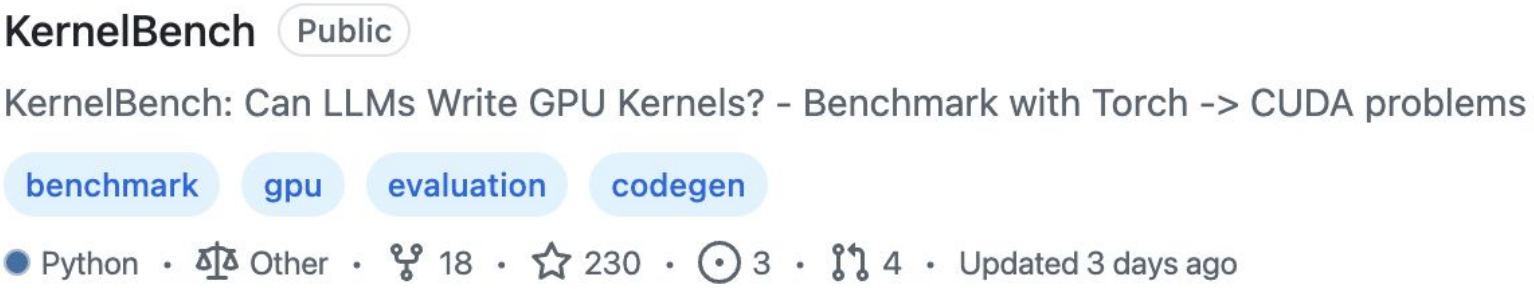
Tao Lin

d Kernel Engineering

f frontier models at writing GPU
caffolding, we found that the best
eedup on KernelBench of 1.8x.

Get Involved!

New Problems? Other Backends? Cool approaches? Contribute!



GitHub: github.com/ScalingIntelligence/KernelBench

🤗 Hugging Face: datasets/ScalingIntelligence/KernelBench

📄 Paper: arxiv.org/abs/2502.10517

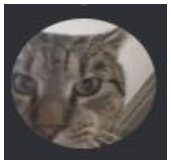
Special thanks to the PyTorch team for Triton + Eval support!

KernelBot 

The “Kaggle” of GPU Programming

The Core OSS Team

“Ben” (Ben H.)



“s1ro” (Matej S.)

“az” (Alex Z.)

“ngc92” (Erik S.)

“Seraphim” (Mark S.)

Is GPU Code Low-Resource?

KernelBench  results show that LLMs struggle to generate both **correct** and **performant** GPU code:

reference.py reference.py (6-21)

Write this kernel (a 2D convolution) in Triton. Make it really really fast, taking advantage of tensor cores.

Triton Conv2D Implementation with Tensor Core Acceleration

I'll create a fast 2D convolution implementation using Triton that leverages tensor cores for acceleration. Let's start with the implementation:

```
# Initialize accumulator with zeros
acc = tl.zeros((BLOCK_M, BLOCK_N), dtype=tl.float32)
```

⋮

```
# Perform matrix multiplication for this tile using tensor cores
acc[i:i+TILE_M, :] += tl.dot(x_block, k_block, allow_tf32=True)
```

Collecting High-Quality GPU Kernel Data

Hypothesis: GPU kernel generation is a data scarcity problem. We want to apply LIMO, which requires a strong knowledge base (*next talk*) and **a small number of high-quality examples!**

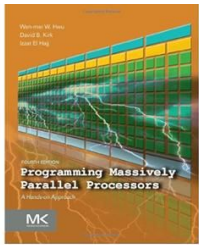
Less-Is-More Reasoning (LIMO) Hypothesis (Ye et al., 2025). This hypothesis identifies two critical factors that determine the **elicitation threshold for complex reasoning**:

(1) the **latent presence of prerequisite knowledge** within the model's parameter space

(2) the effectiveness of **minimal exemplars** in demonstrating systematic problem-solving processes that encourage extended deliberation.

A First Look at the Competition: **Practice Round**

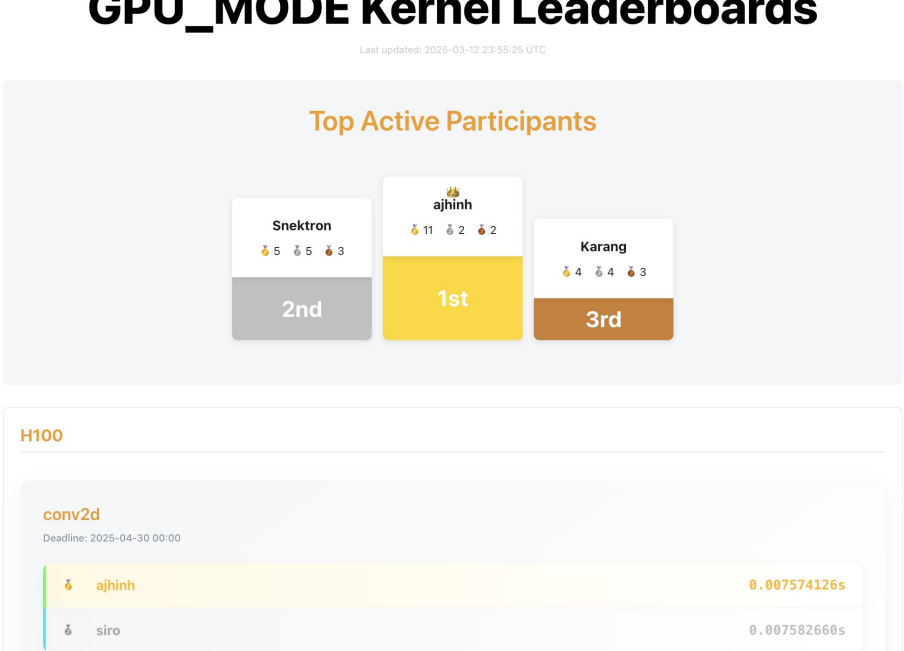
Launched with **8 problems from**



⭐ **2000+** user submissions!

Participants submit / on **Discord**

Entirely OSS – open to feedback and adapting eval over time



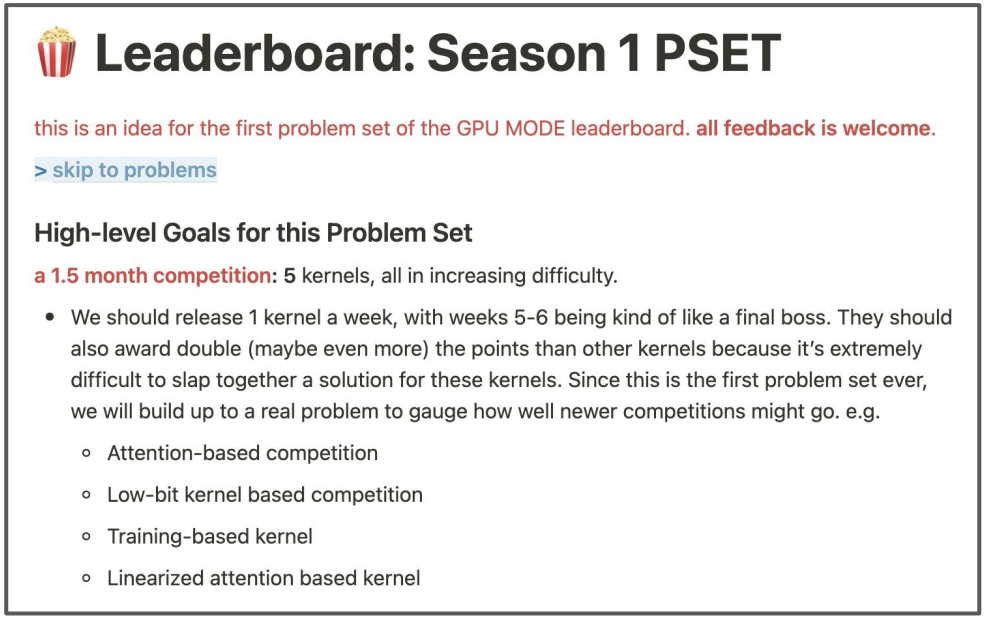
Round 1 and Beyond

🏆 Compete for **real prizes!**

🍂 Seasonal problem sets with focus on NVIDIA H100 and AMD MI300!

1️⃣ Per feedback, competition is structured as **one problem per week / bi-weekly**.

Companies, labs, individuals can **suggest new problem sets** to place “bounties” on particular kernels!



🍿 Leaderboard: Season 1 PSET

this is an idea for the first problem set of the GPU MODE leaderboard. **all feedback is welcome.**

> skip to problems

High-level Goals for this Problem Set

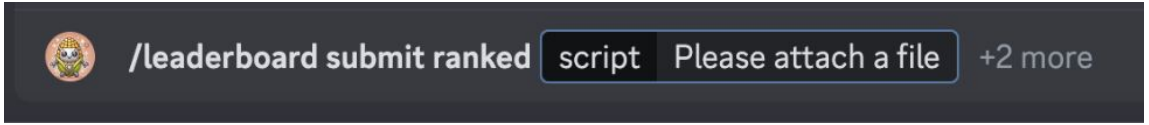
a 1.5 month competition: 5 kernels, all in increasing difficulty.

- We should release 1 kernel a week, with weeks 5-6 being kind of like a final boss. They should also award double (maybe even more) the points than other kernels because it’s extremely difficult to slap together a solution for these kernels. Since this is the first problem set ever, we will build up to a real problem to gauge how well newer competitions might go. e.g.
  - Attention-based competition
  - Low-bit kernel based competition
  - Training-based kernel
  - Linearized attention based kernel

To participate join discord.gg/gpumode and `/leaderboard submit`

Come participate now – It’s super easy!

To participate, join discord.gg/gpumode and `/leaderboard submit` !

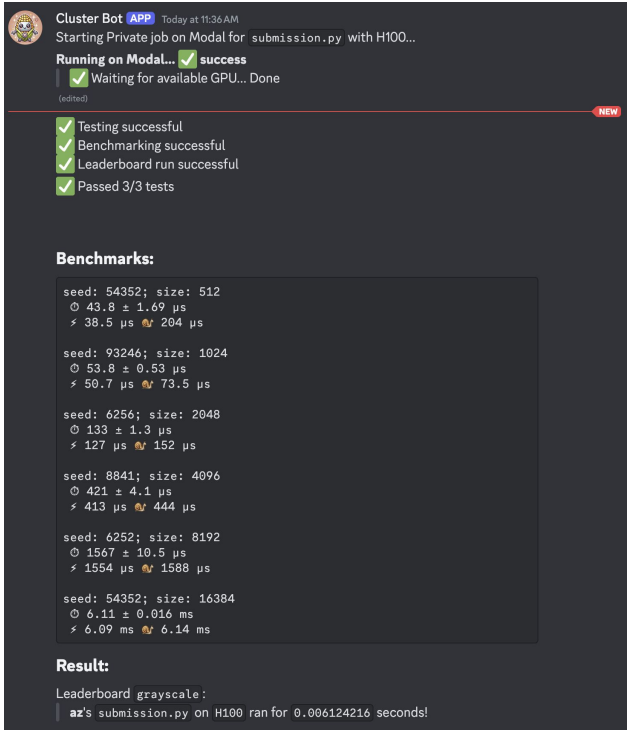


We also have:

Documentation!

A Website!

CLI submissions!



KernelBook

The Largest GPU Kernel Dataset On Earth

Sources of Data

- 📉Kernels on Github - NOT ENOUGH
  - 2k unique triton kernels on Github
  - 0.1% of the stack is CUDA Kernels
- 🔮 Kernel Leaderboard - Data coming soon!
- 🤖 Torch.compile generated data
  - Scrape Github for PyTorch
  - Synthetically Generate PyTorch

PyTorch models from Github

```python
import torch
from torch import nn


class Swish(nn.Module):

    def forward(self, x):
        return x.mul_(torch.sigmoid(x))


def get_inputs():
    return [torch.rand([4, 4, 4, 4])]


def get_init_inputs():
    return [[], {}]
```

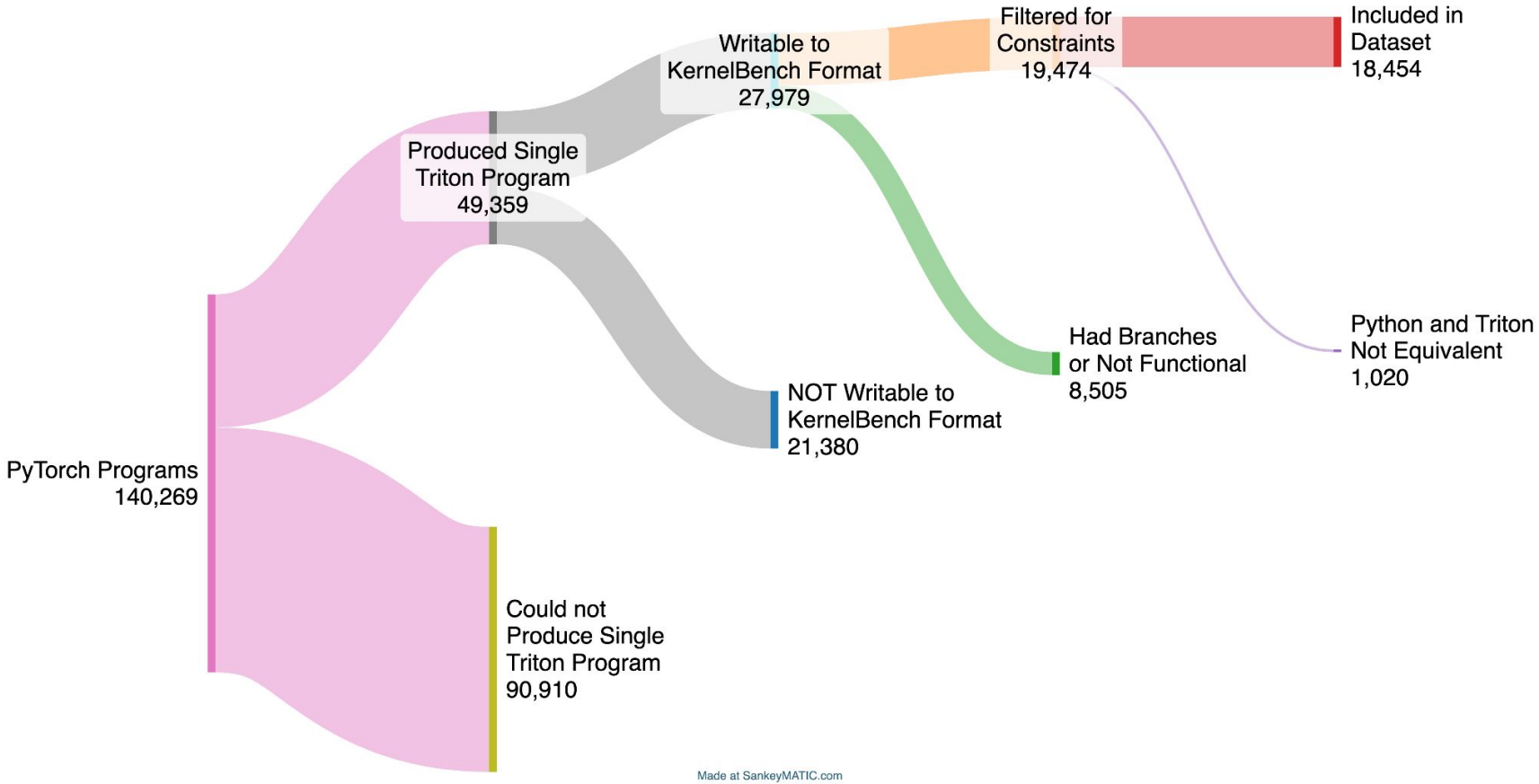
TORCH_LOGS=
“output_code”

More tricks

Torch.compiled Triton Code

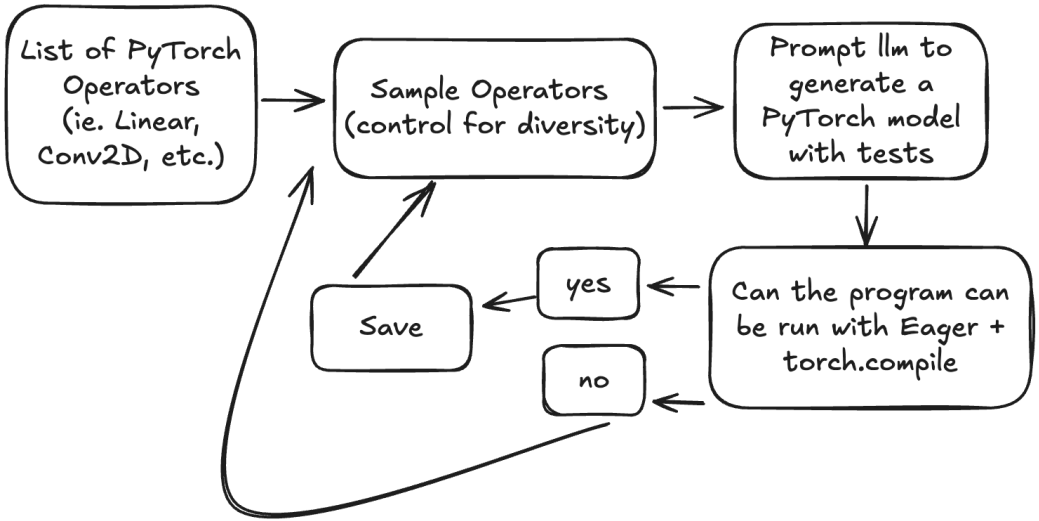
```python
import torch
import triton
import triton.language as tl
from torch._inductor.runtime.triton_heuristics import grid
from torch._C import _cuda_getCurrentRawStream as get_raw_stream
from torch import nn
assert_size_stride = torch._C._dynamo.guards.assert_size_stride


@triton.jit
def triton_poi_fused_mul_sigmoid_0(in_ptr0, out_ptr1, xnumel, XBLOCK: tl.constexpr):
    xnumel = 256
    xoffset = tl.program_id(0) * XBLOCK
    xindex = xoffset + tl.arange(0, XBLOCK)[:]
    xmask = xindex < xnumel
    x0 = xindex
    tmp0 = tl.load(in_ptr0 + x0, xmask)
    tmp1 = tl.sigmoid(tmp0)
    tmp2 = tmp0 * tmp1
    tl.store(out_ptr1 + x0, tmp2, xmask)


def call(args):
    arg0_1, = args
    args.clear()
    assert_size_stride(arg0_1, (4, 4, 4, 4), (64, 16, 4, 1))
    with torch.cuda._DeviceGuard(0):
        torch.cuda.set_device(0)
        get_raw_stream(0)
        triton_poi_fused_mul_sigmoid_0[grid(256)](arg0_1, arg0_1, 256,
            XBLOCK=256, num_warps=4, num_stages=1)
    return arg0_1,


class SwishNew(nn.Module):

    def forward(self, input_0):
        arg0_1 = input_0
        output = call([arg0_1])
        return output[0]
```

Human Github Dataset Yield



Synthetic Data - Overview

- Generated **10k** samples
- Synthetic data yield **75%**
- Human data yield **12%**



🙏 Thanks to @simon and @alex :)

What we have and beyond

- 🤗 What we have
  - 18k programs scraped from github
  - 7.5k synthetically generated examples
- 🔮 Moving forward:
  - Improve yield of the pipeline but we will **still** be data starved
  - The best stuff is from y'all! Take a crack at the Kernel Leaderboard

https://huggingface.co/GPUMODE

KernelLLM

Zacharias Fisches

Meta Fundamental AI Research (FAIR)

Why are LLMs so meh at writing Triton kernels?

Why are LLMs so meh at writing Triton kernels?

KernelBench-Triton

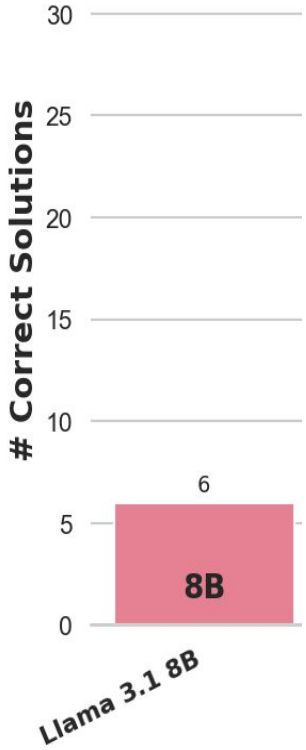
Level 1
Single-kernel operators

The foundational building blocks of neural nets.

- Convolutions
- Matrix multiplies
- Layer normalization

Why are LLMs so meh at writing Triton kernels?

KernelBench-Triton



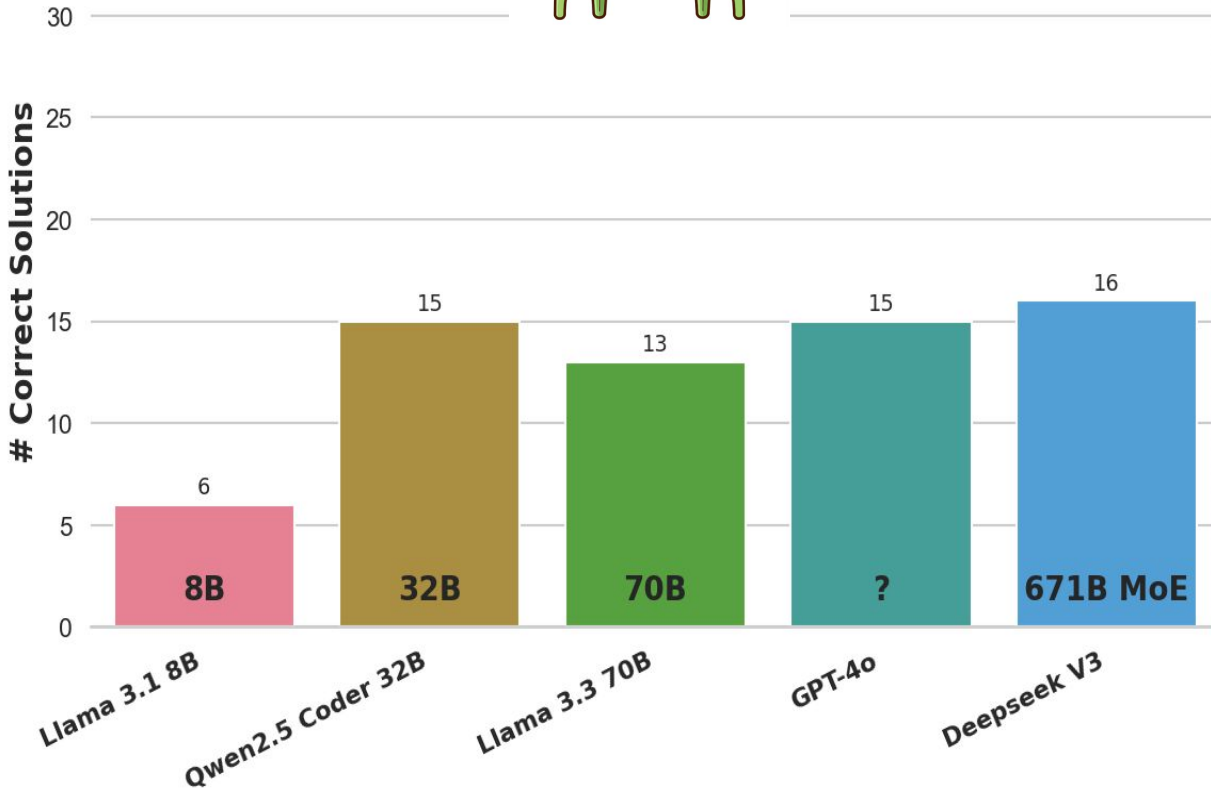
Level 1
Single-kernel operators

The foundational building blocks of neural nets.

- Convolutions
- Matrix multiplies
- Layer normalization

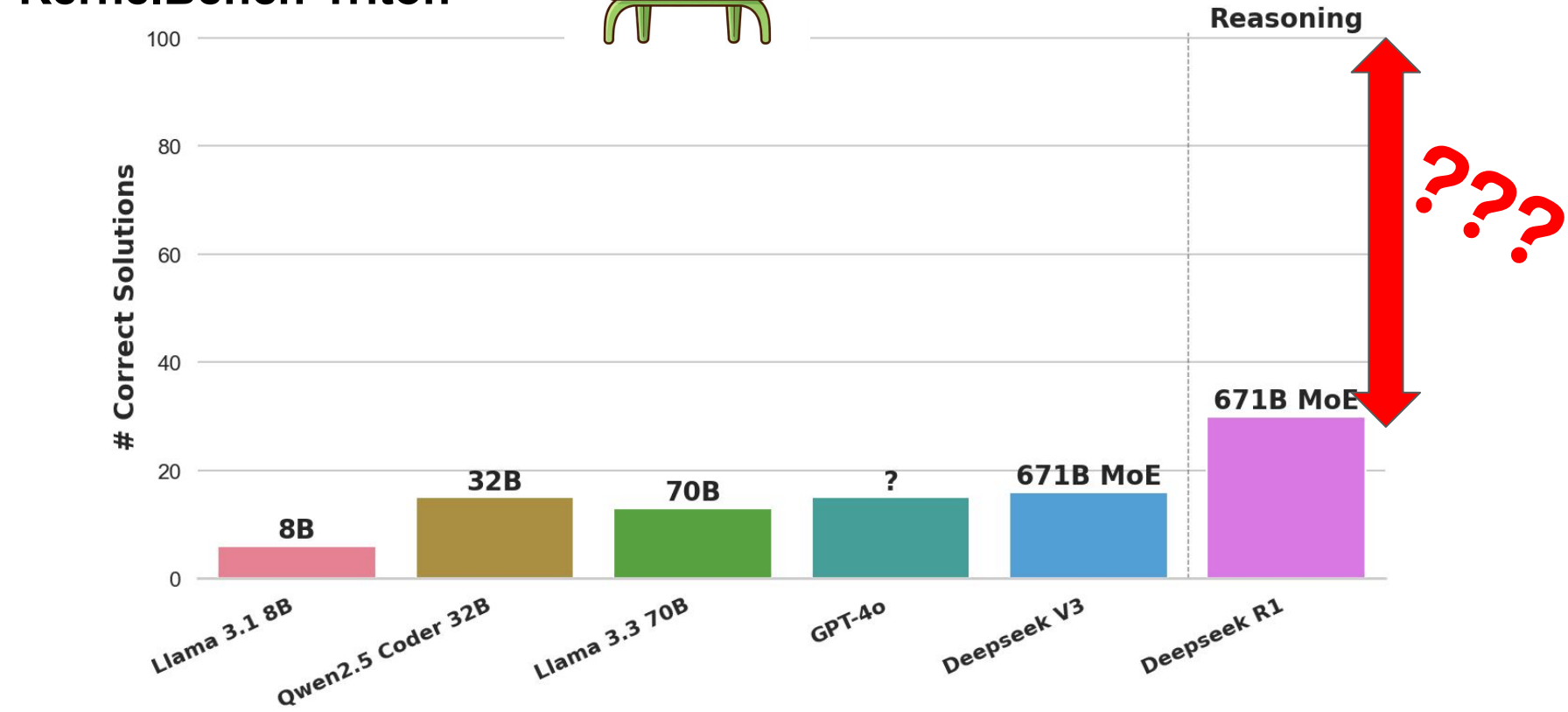
Why are LLMs so meh at writing Triton kernels?

KernelBench-Triton



Why are LLMs so meh at writing Triton kernels?

It's really bad…



Related Work

Evaluation study targeting CUDA:

Ensemble of SOTA reasoning models,

Scaffolding, **Pass @ 20$** per problem.

Related Work

METR

Evaluation study targeting CUDA:

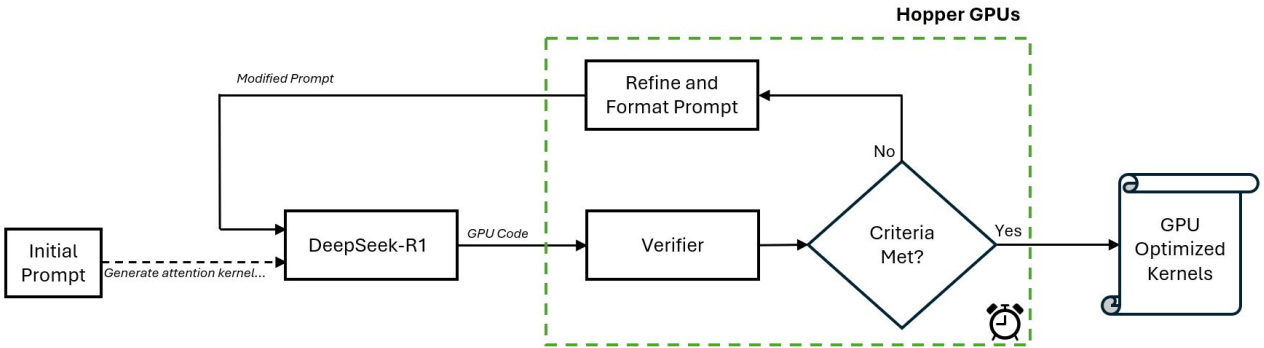
Ensemble of SOTA reasoning models,

Scaffolding, **Pass @ 20$** per problem.

NVIDIA

DeepSeek R1 based verifier-in-the-loop scaffold,

Pass @ 20 minutes.



Related Work

METR

Evaluation study targeting CUDA:

Ensemble of SOTA reasoning models,

Scaffolding, **Pass @ 20$** per problem.

NVIDIA

DeepSeek R1 based verifier-in-the-loop scaffold,

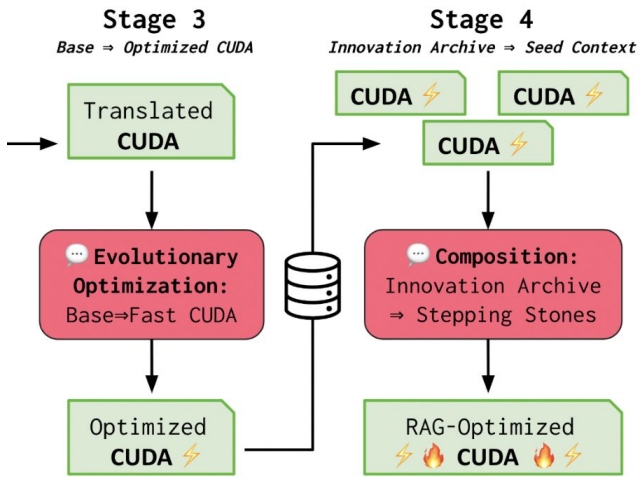
Pass @ 20 minutes.

sakana.ai

Test-time evolutionary optim,

Recombination prompting,

Results WIP.



Related Work

METR

Evaluation study targeting CUDA:

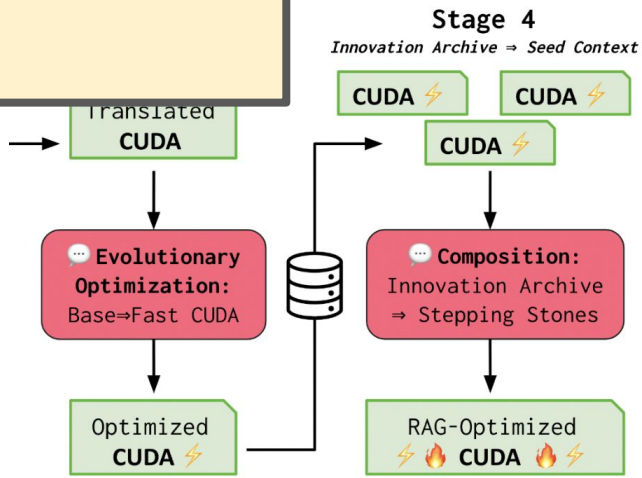
Ensemble of SOTA reasoning models,

Scaffolding, **Pass @ 20$** per problem.

NVIDIA

Test-Time Compute Augmentations

affold,



Test-time evolutionary optim,

Recombination prompting,

Results WIP

sakana.ai

Why are LLMs so meh at writing Triton kernels?

Hypothesis:

LLMs are highly data-starved.

Why are LLMs so meh at writing Triton kernels?

Hypothesis:

LLMs are highly data-starved.

But can be trained

📖 Methods

Translation Task (Torch → Triton)

```python
# === Torch (Reference) ===
import torch.nn as nn

class Model(nn.Module):
    ...
    def forward(self, x, y):
        return x + y
```

torch.compile() ⇓

```python
# === Triton (Generated) ===
import torch.nn as nn
import triton
import triton.language as tl

@triton.jit
def kernel(x_ptr,
           y_ptr,
           out_ptr, ...):
    pid = tl.program_id(0)
    ...
    x = tl.load(...)
    y = tl.load(...)
    tl.store(..., x + y)

class ModelNew(nn.Module):
    def forward(self, x, y):
        kernel[grid, 1024](
            x.data_ptr(), ...)
```

📖 Methods

+

Translation Task (Torch → Triton)

Llama 3.1 8B Instruct

```python
# === Torch (Reference) ===
import torch.nn as nn

class Model(nn.Module):
    ...
    def forward(self, x, y):
        return x + y
```

torch.compile() ⇓

```python
# === Triton (Generated) ===
import torch.nn as nn
import triton
import triton.language as tl

@triton.jit
def kernel(x_ptr,
           y_ptr,
           out_ptr, ...):
    pid = tl.program_id(0)
    ...
    x = tl.load(...)
    y = tl.load(...)
    tl.store(..., x + y)

class ModelNew(nn.Module):
    def forward(self, x, y):
        kernel[grid, 1024](
            x.data_ptr(), ...)
```

📖 Methods

+

Translation Task (Torch → Triton)

Llama 3.1 8B Instruct

+

Standard instruction SFT recipe

```python
# === Torch (Reference) ===
import torch.nn as nn

class Model(nn.Module):
    ...
    def forward(self, x, y):
        return x + y
```

torch.compile() ⇓

```python
# === Triton (Generated) ===
import torch.nn as nn
import triton
import triton.language as tl

@triton.jit
def kernel(x_ptr,
           y_ptr,
           out_ptr, ...):
    pid = tl.program_id(0)
    ...
    x = tl.load(...)
    y = tl.load(...)
    tl.store(..., x + y)

class ModelNew(nn.Module):
    def forward(self, x, y):
        kernel[grid, 1024](
            x.data_ptr(), ...)
```

📖 Methods

Translation Task (Torch → Triton)

Llama 3.1 8B Instruct

Standard instruction SFT recipe

KernelBench-Triton Eval Task

+

+

+

=

```python
# === Torch (Reference) ===
import torch.nn as nn

class Model(nn.Module):
    ...
    def forward(self, x, y):
        return x + y
```

torch.compile() ⇓

```python
# === Triton (Generated) ===
import torch.nn as nn
import triton
import triton.language as tl

@triton.jit
def kernel(x_ptr,
           y_ptr,
           out_ptr, ...):
    pid = tl.program_id(0)
    ...
    x = tl.load(...)
    y = tl.load(...)
    tl.store(..., x + y)

class ModelNew(nn.Module):
    def forward(self, x, y):
        kernel[grid, 1024](
            x.data_ptr(), ...)
```

📖 Methods

Translation Task (Torch → Triton)

Llama 3.1 8B Instruct

Standard instruction SFT recipe

KernelBench-Triton Eval Task

Behold the KernelLLM

+

+

+

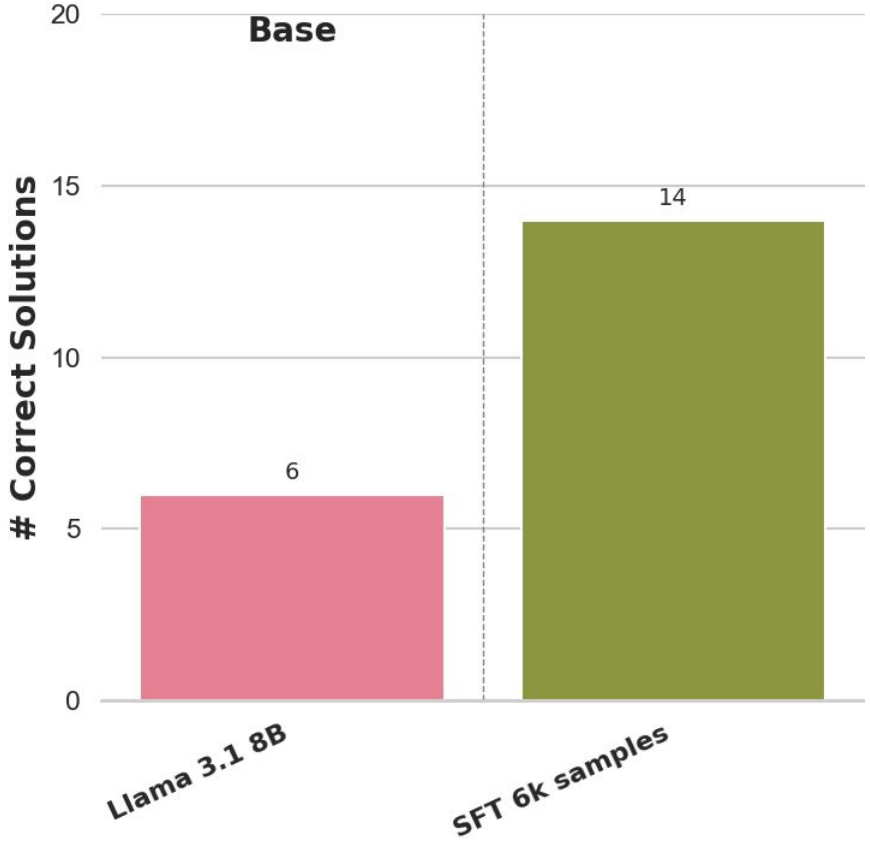
=

```python
# === Torch (Reference) ===
import torch.nn as nn

class Model(nn.Module):
    ...
    def forward(self, x, y):
        return x + y
```

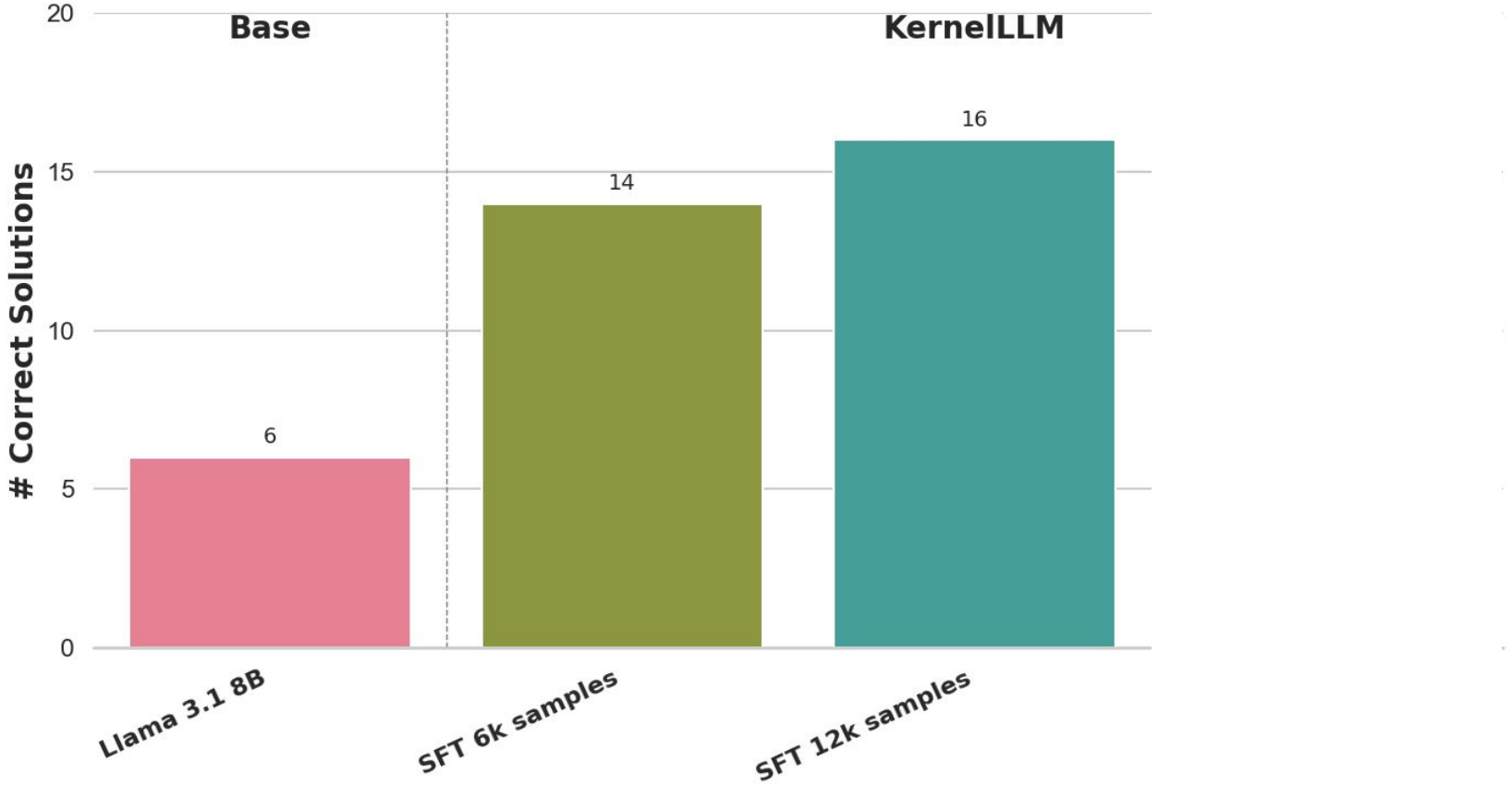
torch.compile() ⇓

```python
# === Triton (Generated) ===
import torch.nn as nn
import triton
import triton.language as tl

@triton.jit
def kernel(x_ptr,
           y_ptr,
           out_ptr, ...):
    pid = tl.program_id(0)
    ...
    x = tl.load(...)
    y = tl.load(...)
    tl.store(..., x + y)

class ModelNew(nn.Module):
    def forward(self, x, y):
        kernel[grid, 1024](
            x.data_ptr(), ...)
```

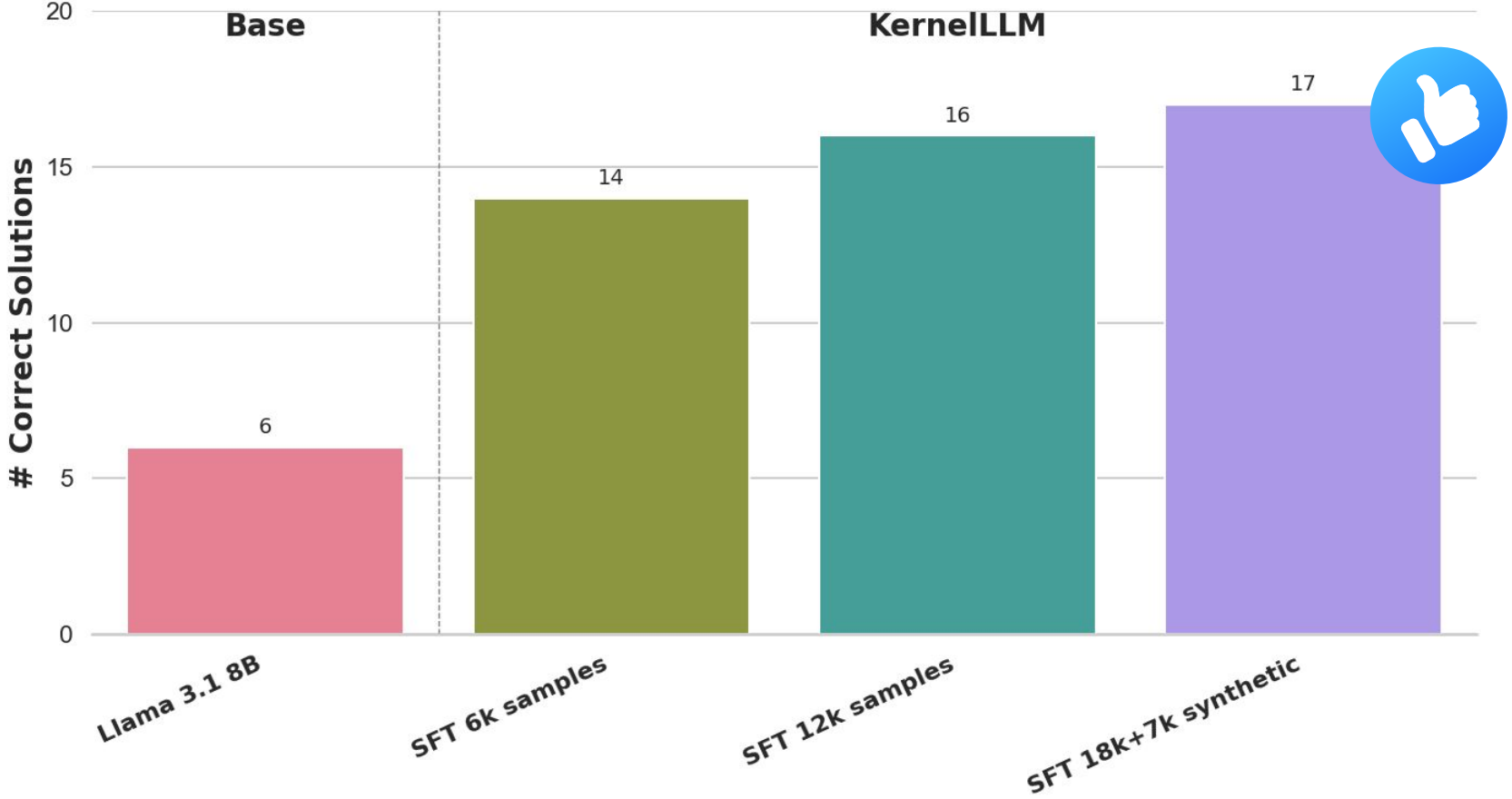
Results: Dataset Scaling



Results: Dataset Scaling

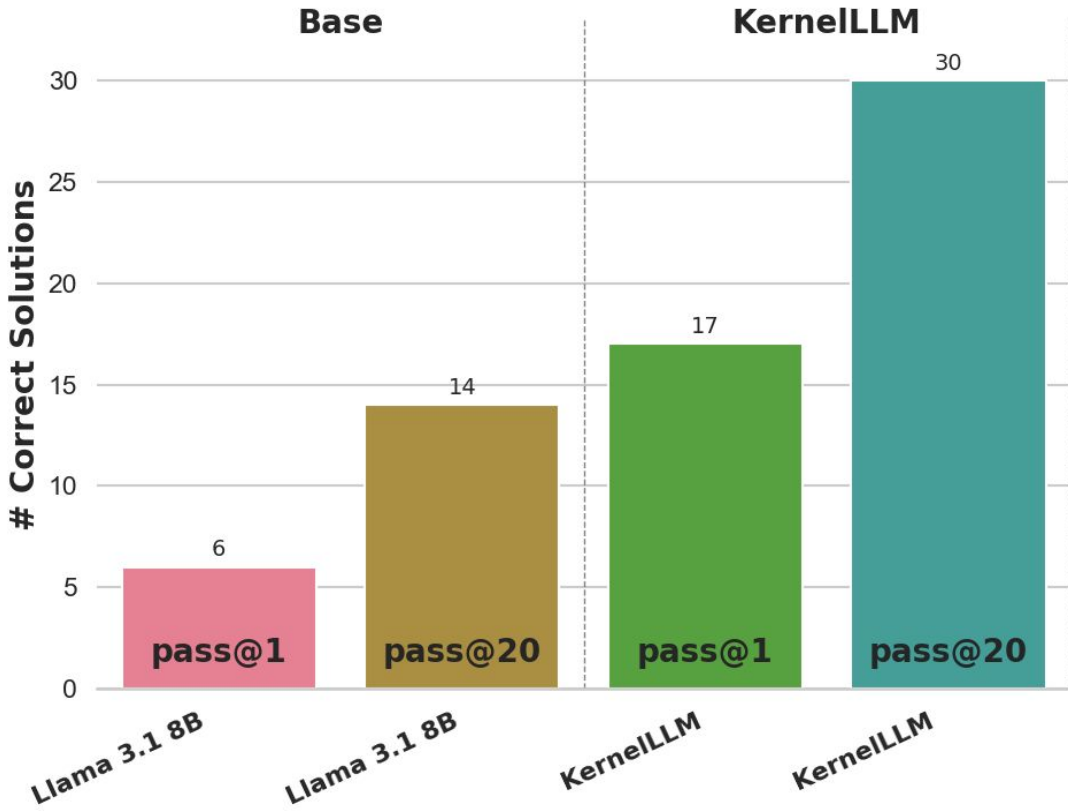


Results: Dataset Scaling

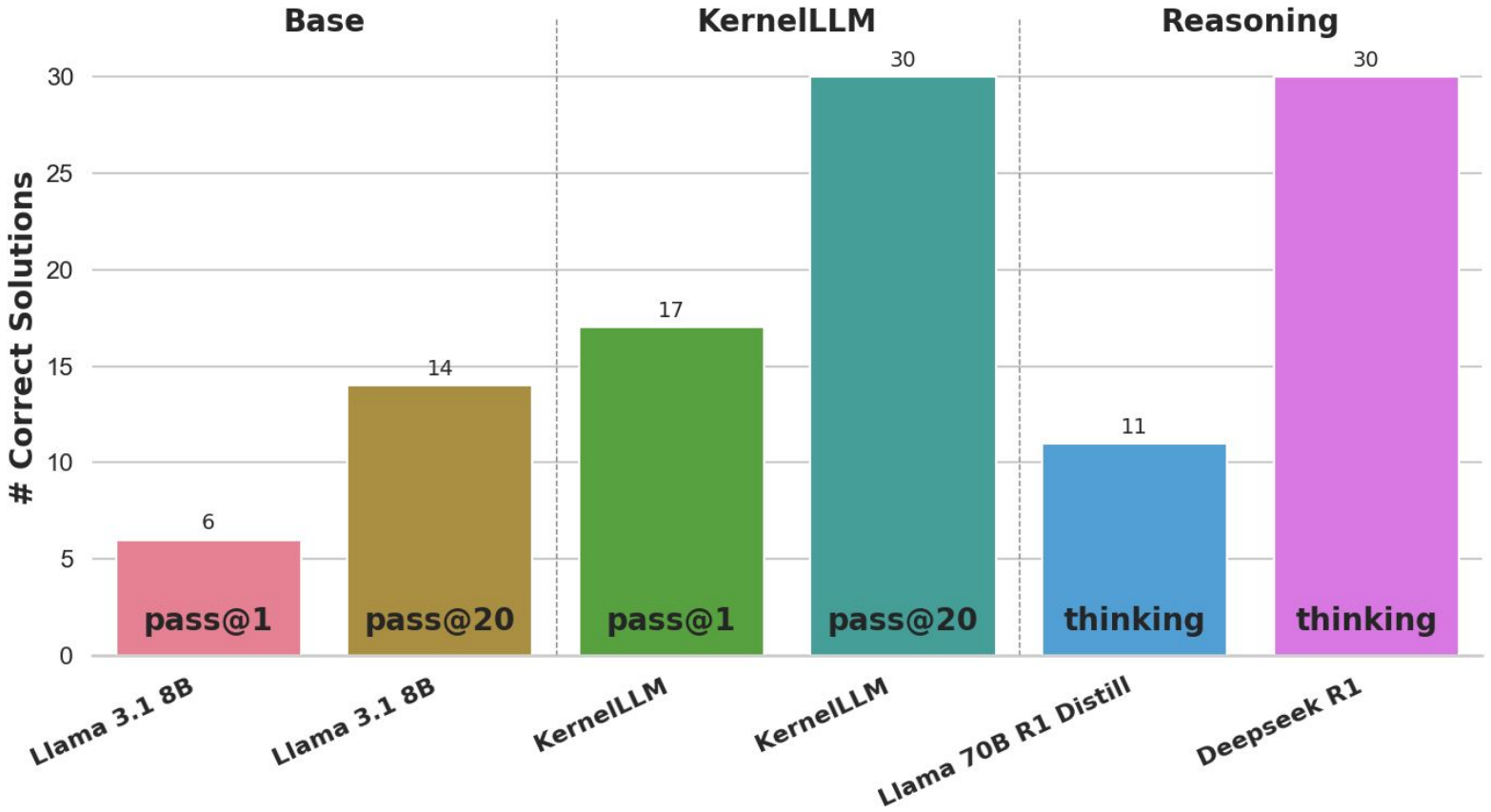


Results: Pass@k Test Time Scaling

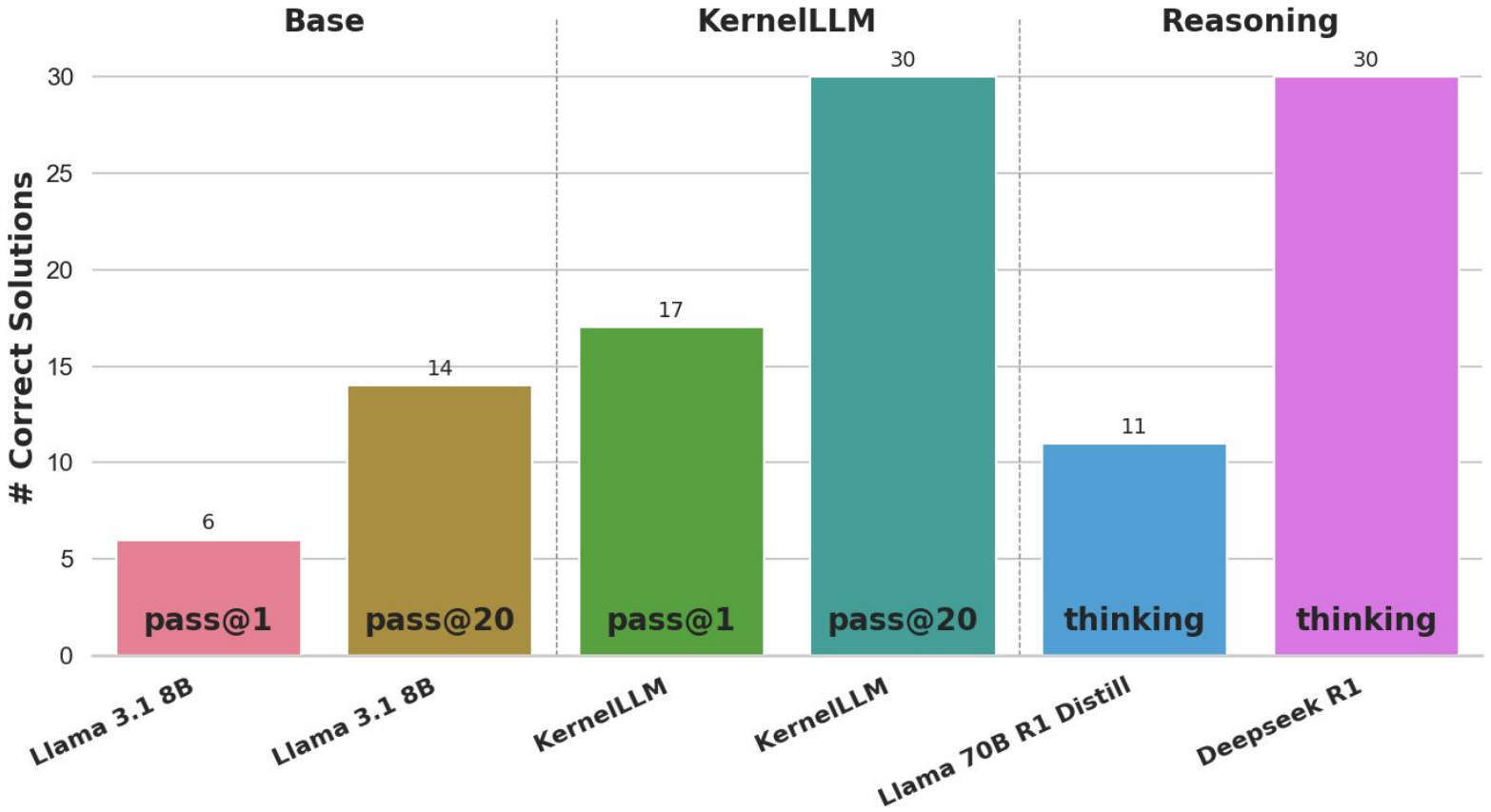
Results: Pass@k Test Time Scaling



Results: Pass@k Test Time Scaling

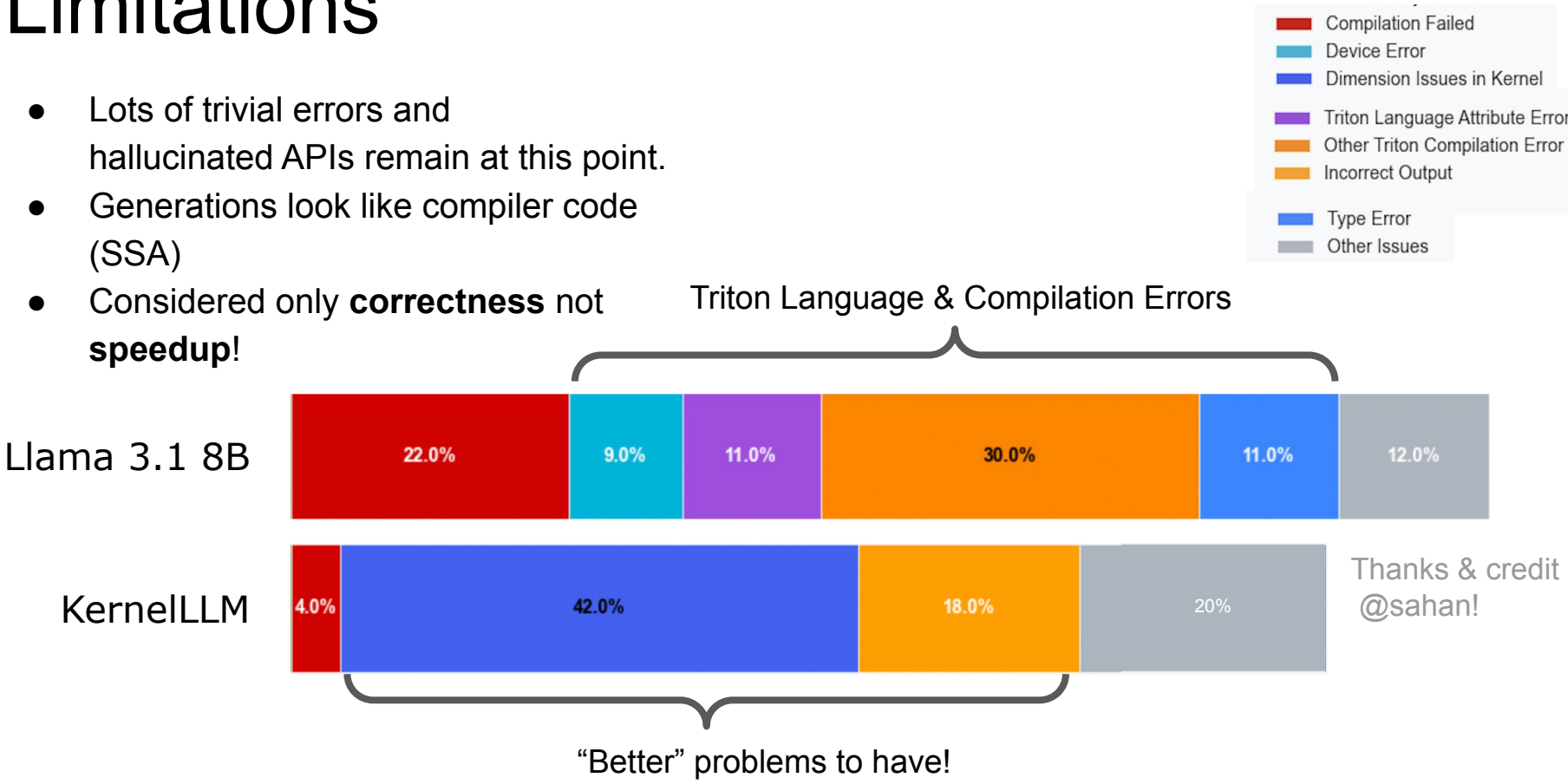


Results: Pass@k Test Time Scaling



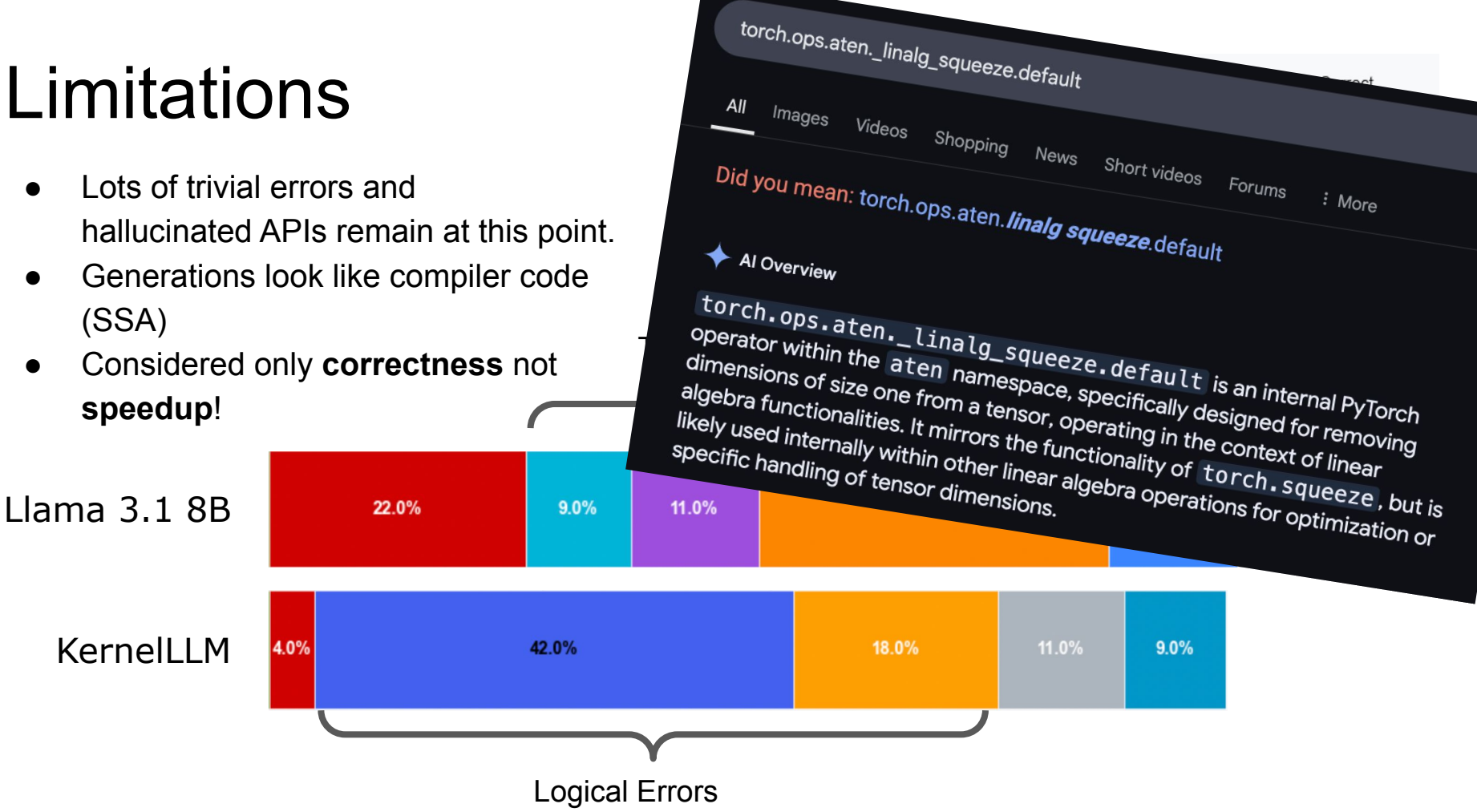
Limitations

- Lots of trivial errors and hallucinated APIs remain at this point.
- Generations look like compiler code (SSA)
- Considered only **correctness** not **speedup**!



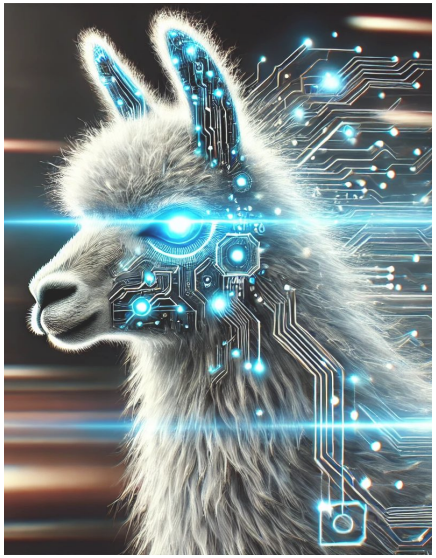
Limitations

- Lots of trivial errors and hallucinated APIs remain at this point.
- Generations look like compiler code (SSA)
- Considered only **correctness** not **speedup**!



Summary

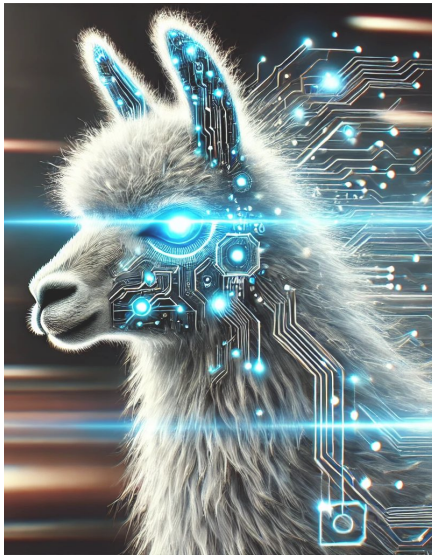
KernelLLM goes beyond test-time eval tricks for the first time.



Summary

KernelLLM goes beyond test-time eval tricks for the first time.

8B model with SFT matches 671B SOTA.

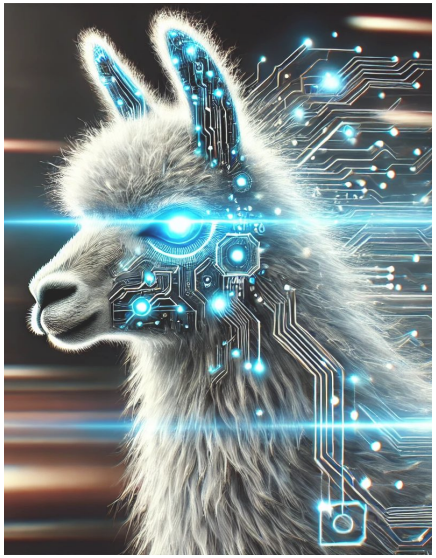


Summary

KernelLLM goes beyond test-time eval tricks for the first time.
8B model with SFT matches 671B SOTA.

Lots more left to do !!!

- **Scaling** to larger models.
- **Better datasets**.
- **Method research** to make breakthroughs on **speedup!**



Summary

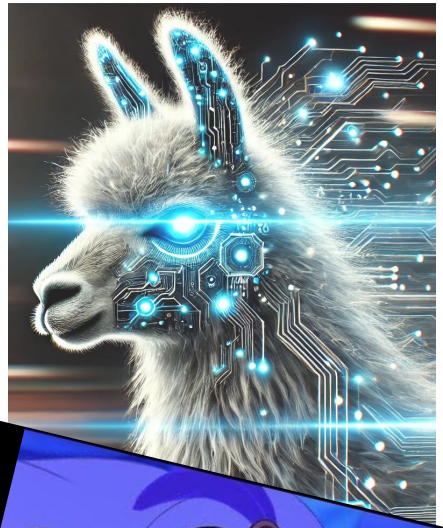
KernelLLM goes beyond test-time eval tricks for the first time.

8B model with SFT matches 671B SOTA.

Lots more left to do !!!

- **Scaling** to larger models.
- **Better datasets**.
- **Method research** to make breakthroughs on **speedup!**

Please reach out to collaborate!



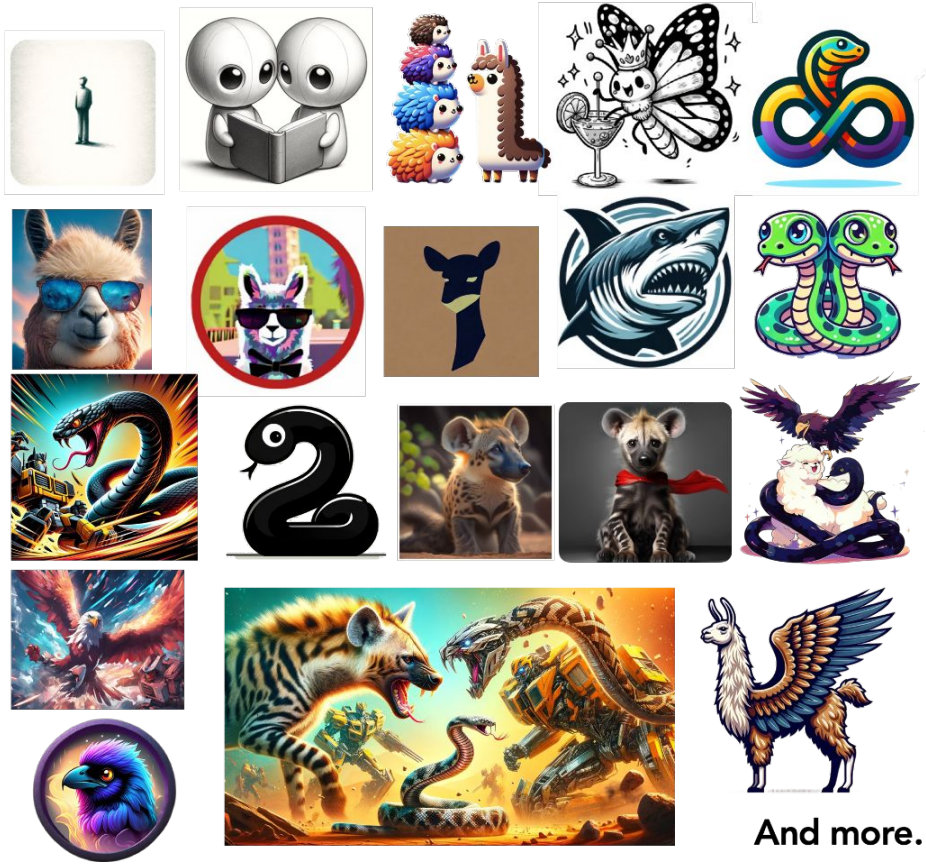
Thunderkittens: Simple, Fast, and Adorable AI kernels

Many online resources:
Arxiv: [2410.20399] ThunderKittens: Simple, Fast, and Adorable AI Kernels
GitHub: https://github.com/HazyResearch/ThunderKittens/tree/main
Blog posts: https://hazyresearch.stanford.edu/blog

With: Benjamin Spector, Aaryan Singhal, Dan Fu, Christopher Ré

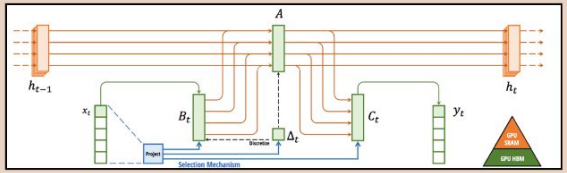
We're experiencing a Cambrian explosion of efficient foundation model architectures 🌍

Model / Paper	Complexity	Decode	Class
Memory Compressed (Liu et al., 2018)	$\mathcal{O}(N_c^2)$	✓	FP+M
Image Transformer (Parmar et al., 2018)	$\mathcal{O}(N.m)$	✓	FP
Set Transformer (Lee et al., 2019)	$\mathcal{O}(kN)$	✗	M
Transformer-XL (Dai et al., 2019)	$\mathcal{O}(N^2)$	✓	RC
Sparse Transformer (Child et al., 2019)	$\mathcal{O}(N\sqrt{N})$	✓	FP
Reformer (Kitaev et al., 2020)	$\mathcal{O}(N \log N)$	✓	LP
Routing Transformer (Roy et al., 2020)	$\mathcal{O}(N\sqrt{N})$	✓	LP
Axial Transformer (Ho et al., 2019)	$\mathcal{O}(N\sqrt{N})$	✓	FP
Compressive Transformer (Rae et al., 2020)	$\mathcal{O}(N^2)$	✓	RC
Sinkhorn Transformer (Tay et al., 2020b)	$\mathcal{O}(B^2)$	✓	LP
Longformer (Beltagy et al., 2020)	$\mathcal{O}(n(k+m))$	✓	FP+M
ETC (Ainslie et al., 2020)	$\mathcal{O}(N_g^2 + NN_g)$	✗	FP+M
Synthesizer (Tay et al., 2020a)	$\mathcal{O}(N^2)$	✓	LR+LP
Performer (Choromanski et al., 2020a)	$\mathcal{O}(N)$	✓	KR
Funnel Transformer (Dai et al., 2020)	$\mathcal{O}(N^2)$	✓	FP+DS
Linformer (Wang et al., 2020c)	$\mathcal{O}(N)$	✗	LR
Linear Transformers (Katharopoulos et al., 2020)	$\mathcal{O}(N)$	✓	KR
Big Bird (Zaheer et al., 2020)	$\mathcal{O}(N)$	✗	FP+M
Random Feature Attention (Peng et al., 2021)	$\mathcal{O}(N)$	✓	KR
Long Short Transformers (Zhu et al., 2021)	$\mathcal{O}(kN)$	✓	FP + LR
Poolingformer (Zhang et al., 2021)	$\mathcal{O}(N)$	✗	FP+M
Nyströmformer (Xiong et al., 2021b)	$\mathcal{O}(kN)$	✗	M+DS
Perceiver (Jaegle et al., 2021)	$\mathcal{O}(kN)$	✓	M+DS
Clusterformer (Wang et al., 2020b)	$\mathcal{O}(N \log N)$	✗	LP
Luna (Ma et al., 2021)	$\mathcal{O}(kN)$	✓	M
TokenLearner (Ryoo et al., 2021)	$\mathcal{O}(k^2)$	✗	DS
Adaptive Sparse Transformer (Correia et al., 2019)	$\mathcal{O}(N^2)$	✓	Sparse
Product Key Memory (Lample et al., 2019)	$\mathcal{O}(N^2)$	✓	Sparse
Switch Transformer (Fedus et al., 2021)	$\mathcal{O}(N^2)$	✓	Sparse
ST-MoE (Zoph et al., 2022)	$\mathcal{O}(N^2)$	✓	Sparse
GShard (Lepikhin et al., 2020)	$\mathcal{O}(N^2)$	✓	Sparse
Scaling Transformers (Jaszczur et al., 2021)	$\mathcal{O}(N^2)$	✓	Sparse
GLaM (Du et al., 2021)	$\mathcal{O}(N^2)$	✓	Sparse

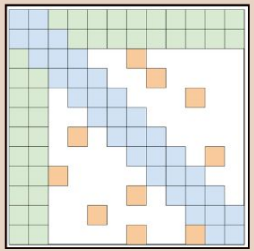


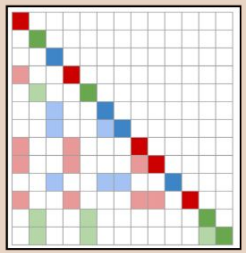
And more…!

Two challenges of hardware-aware AI.



Parallel scans

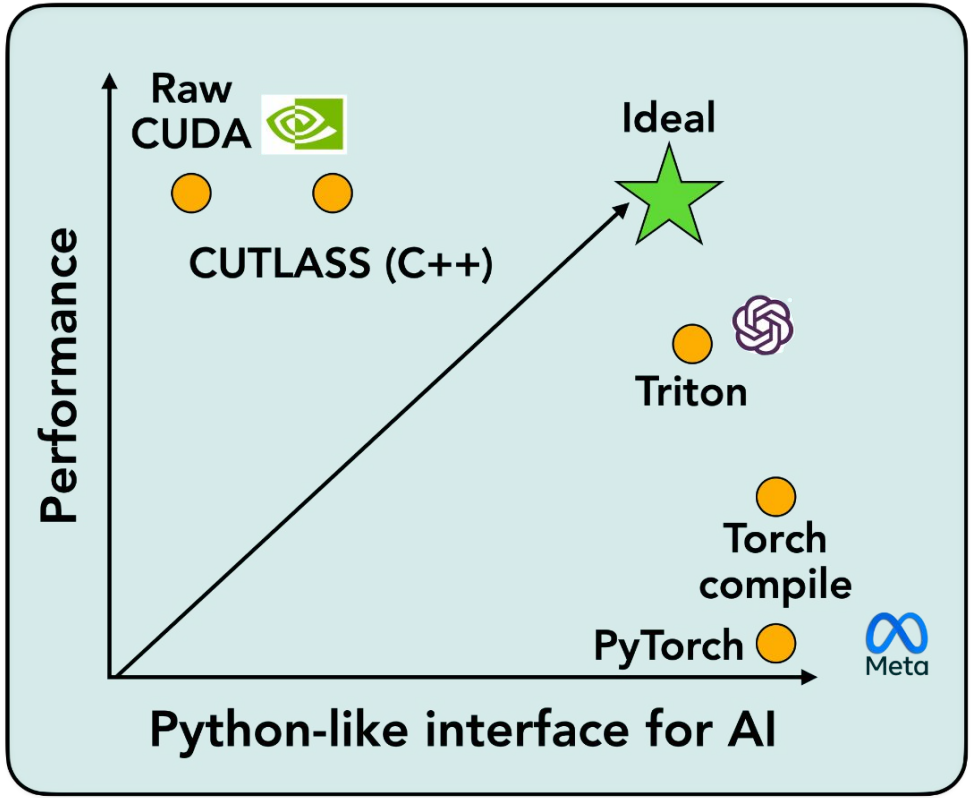




Sparse, uncoalesced accesses

And more!

Architectural choices



Programming abstraction choices

Research question: How concise a set of programming abstractions can we use, to support fast, simple kernels for a breadth of AI workloads?

ThunderKittens: library for developing simple, fast, and adorable AI kernels

Workload	TK LoC	Reference LoC	Speed up (min-max)
H100 Attention inference	217	2325 (CUTLASS FA3)	0.87-1.14×
H100 BF16 GEMM	84	463 (CUTLASS)	0.98-2.05×
H100 FP8 GEMM	91	Closed Source (CuBLASLt)	0.93-1.10×
H100 Convolution	131	624 (CUDA FlashFFTConv)	4.6-4.7×
H100 Based linear attention	282	89 (Triton)	3.7-14.5×
H100 Hedgehog linear attention	316	104 (Triton)	4.0-6.5×
H100 Mamba-2	192	532 (Triton)	3.0-3.7×
H100 Rotary	101	119 (Triton)	1.1-2.3×
4090 Attention ($D = 64$)	93	1262 (CUTLASS FA2)	0.96-0.98×
4090 Attention ($D = 128$)	93	1262 (CUTLASS FA2)	0.89-0.95×
Apple M2 Attention ($D = 64$)	47	343 (Apple MLX)	1.12-1.15×
Apple M2 GEMM	27	412 (Apple MLX)	1.04-1.12×

Library sizes

Framework	Size	Date
CUTLASS	22 MB	10/22
Triton	12.6MB	10/22
TK	<1MB	10/22

TK kernels use few lines of code (~simple) and are fast.

ThunderKittens library

We identify a small, opinionated set of primitives that suffices for peak performance.

Includes/

Types/ → **Memory: 16 x 16 tiles** as the primitive data structure

Register/

Shared/

Global/

Ops/ → **Interface: PyTorch-like library functions** for familiarity

Warp/

Group/

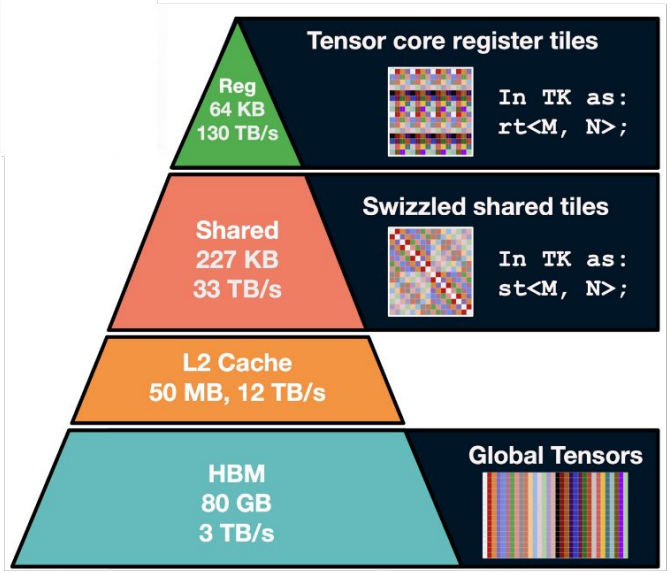
Common/

Prototype/ → **Compute:** *Single* template for **asynchronous execution**

Data abstractions: Tiles with managed memory layouts to minimize bank conflicts and encourage tensor core use

Includes/
- Types/ → **Memory: 16 x 16 tiles** as the primitive data structure
  - Register/
  - Shared/
  - Global/
- Ops/
  - Warp/
  - Group/
- Common/

Prototype/



Function abstractions: Templated NumPy and PyTorch like library functions over the tiles

Includes/
- Types/
  - Register/
  - Shared/
  - Global/
- Ops/
  - Warp/
  - Group/
- Common/

Prototype/

Interface: PyTorch-like library functions for familiarity

```
// Data tiles in registers and shared memory
rt_bf<16,16> x;
rt_fl <16,64> o;
st_bf<64,16> (&y_t) = al.allocate<st_bf<64,16>>();

// Multiply tiles using tensor cores
warpgroup::mma_ABt(o, x, y_t);

// Arithmetic operations
mul(o, o, 0.25);
add(o, o, o);
exp(o);
```

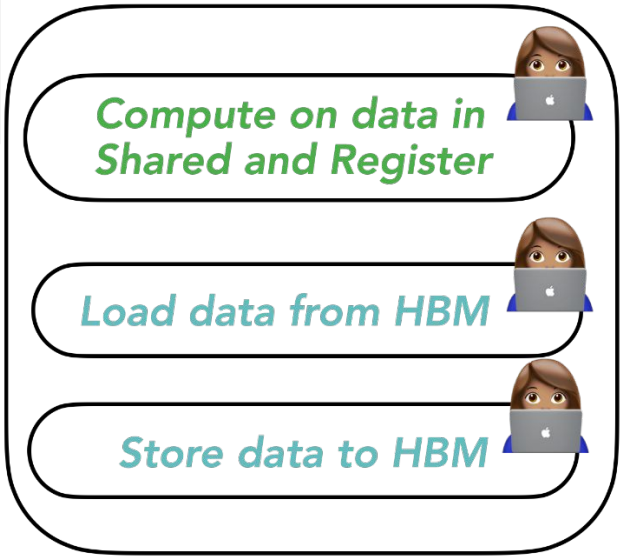
Template abstraction: Kernel template to manage asynchronous execution across threads

Includes/
  Types/
    Register/
    Shared/
    Global/
  Ops/
    Warp/
    Group/
  Common/
Prototype/

Compute: *Single* template for **asynchronous execution**

Compute workers

Load and store workers

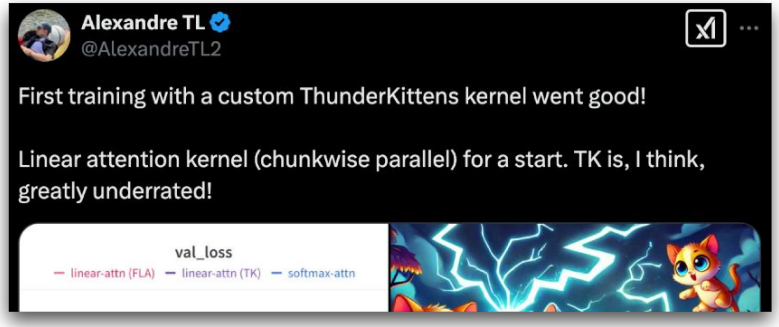


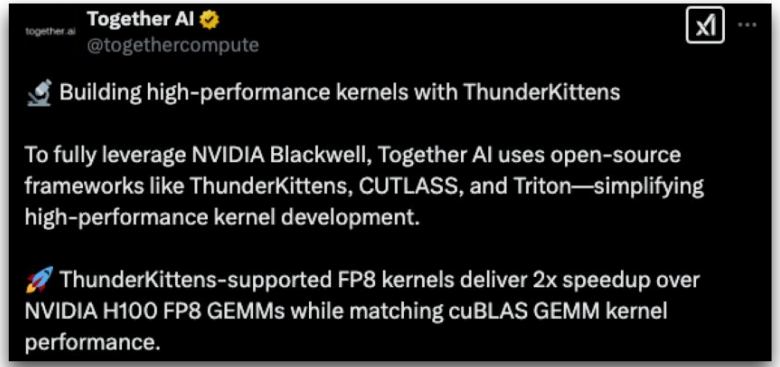
ThunderKittens Impact

- **Open-source:**
  - 2.2K+ GitHub stars in < 1 year
  - Facilitated rapid Blackwell support
  - Tool of choice in GPU MODE's kernel competition!

- **Research:**
  - ICLR 2025 Spotlight (top 5.1% of 12k papers)
  - In published work at Stanford, UCSD, UC Berkeley, University of Italy, etc.

- **Industry**:
  - Growing adoption including at Cruise, Jump Trading, Together AI.
  - Led a wave of simplified documentation and more pythonic DSLs (CUTLASS 4.0, TileLang)!





jumptrading

together.ai

GPU MODE

cruise

Meta

x

Calls to action! 🚀

- **Leaderboard:**
  - TK is one of 3 DSLs supported on the GPU MODE / Meta kernels leaderboard!
  - Look out for our new competition leaderboard focused on efficient model kernels; Progress saves BILLIONS OF DOLLARS of compute costs!!! Real world impact :)

- **Please reach out to collaborate! (simarora@stanford.edu)**
  - Do the abstractions that *people prefer* correspond to the abstractions that *LLMs prefer*?
  - How can we train LLMs to use new low-resource abstractions like TK's?

Our contributions

KernelBench  : **The reference** kernel eval suite

KernelBook  : The **largest kernel dataset** ever published

KernelBot  : A viral platform for **competitive GPU coding**

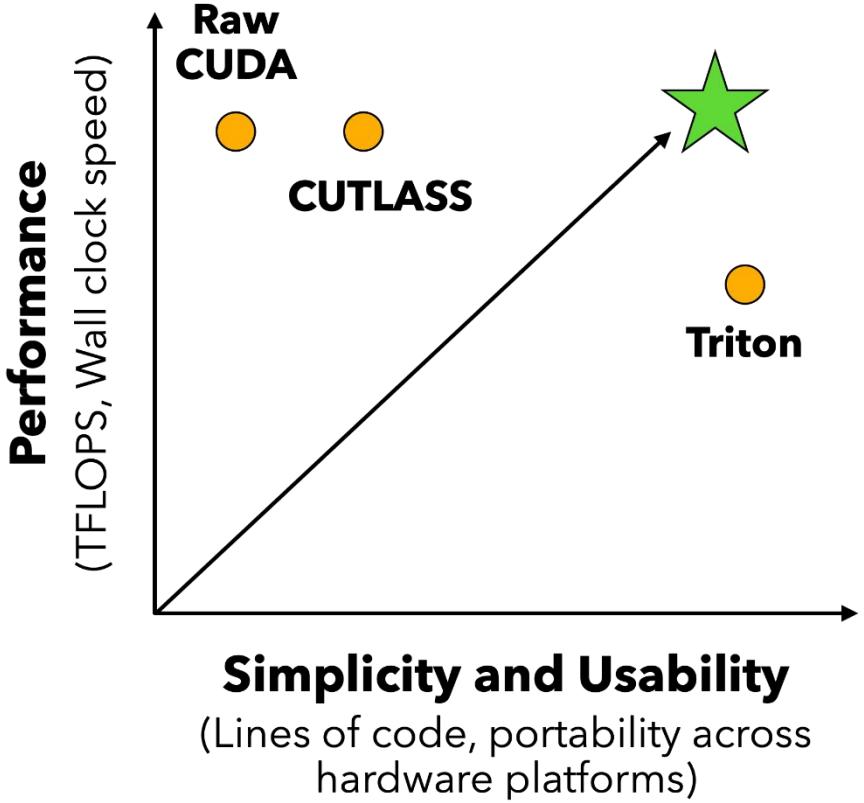
KernelLLM  : A **SOTA** SFT **LLM Triton** programmer

ThunderKittens  : A **simple library** for **blazing fast** AI kernels

@ NVIDIA GTC

Extra slides

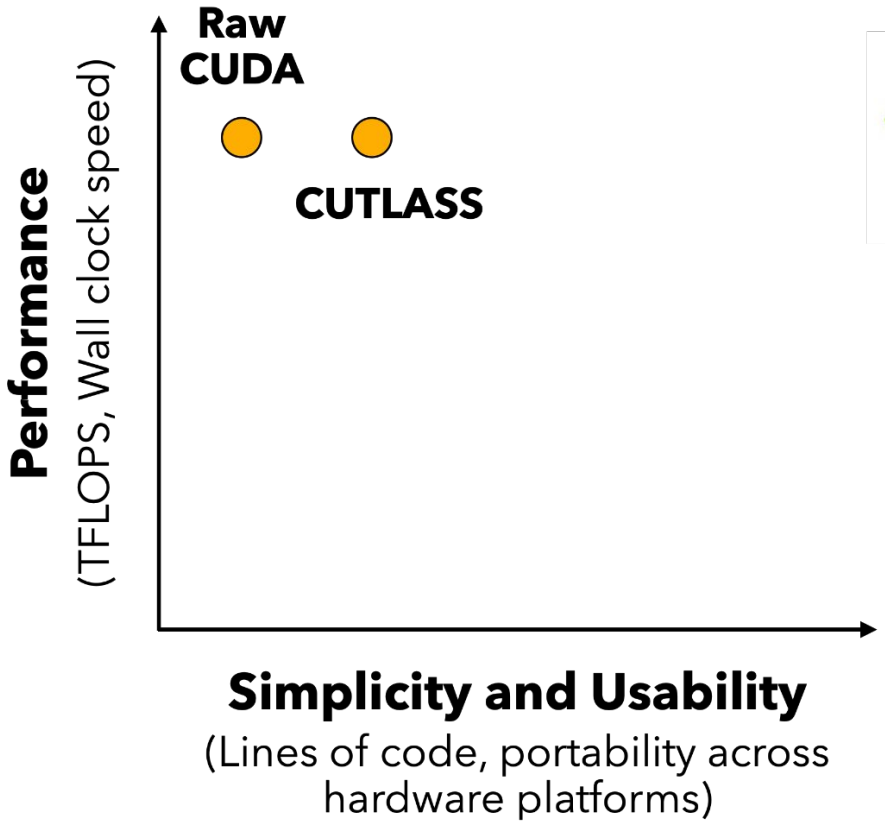
ThunderKittens: a library of programming abstractions for fast and simple AI kernels



ThunderKittens

Performant	✓ Peak performance
Python-like for AI users	✓ Few primitives ✓ Familiar to ML folks
Extensibile	✓ C++ embedded ✓ Hardware-platform agnostic

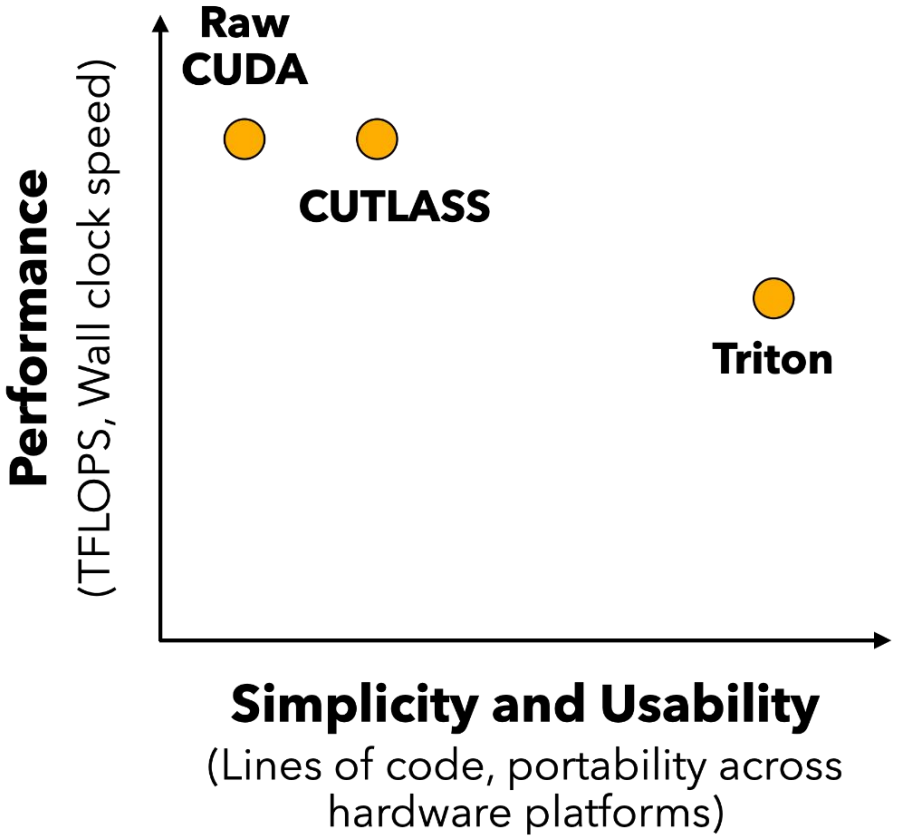
Ideally, we would have a framework for hardware programming that is **familiar to AI users, performant, and extensible.**



CUTLASS

Performant	✓
Python-like for AI users	✗
Extensible	✓ C++ embedded ✗ NVIDIA-specific

Ideally, we would have a framework for hardware programming that is **familiar to AI users, performant, and extensible**



Meta

Triton & Torch compile

Performant	✗
Python-like for AI users	✓
Extensibile	✗ Not C++ embedded ✓ Hardware-platform agnostic