Making Your Jobs Run Faster With System-level Optimizations So You Never Miss Your (Submission) Deadline

Vector Institute Workshop, Feb 10, 2023

Presenters:
Yubo Gao <ybgao@centml.ai>
Xin Li <xin@centml.ai>
CentML Inc.

Todos

Let user configure port in case of port collision

Slide numbers (Qs)

“System-level” consistent

Potential blog post from intro

Why underutilization

“Why are my jobs so slow? ”

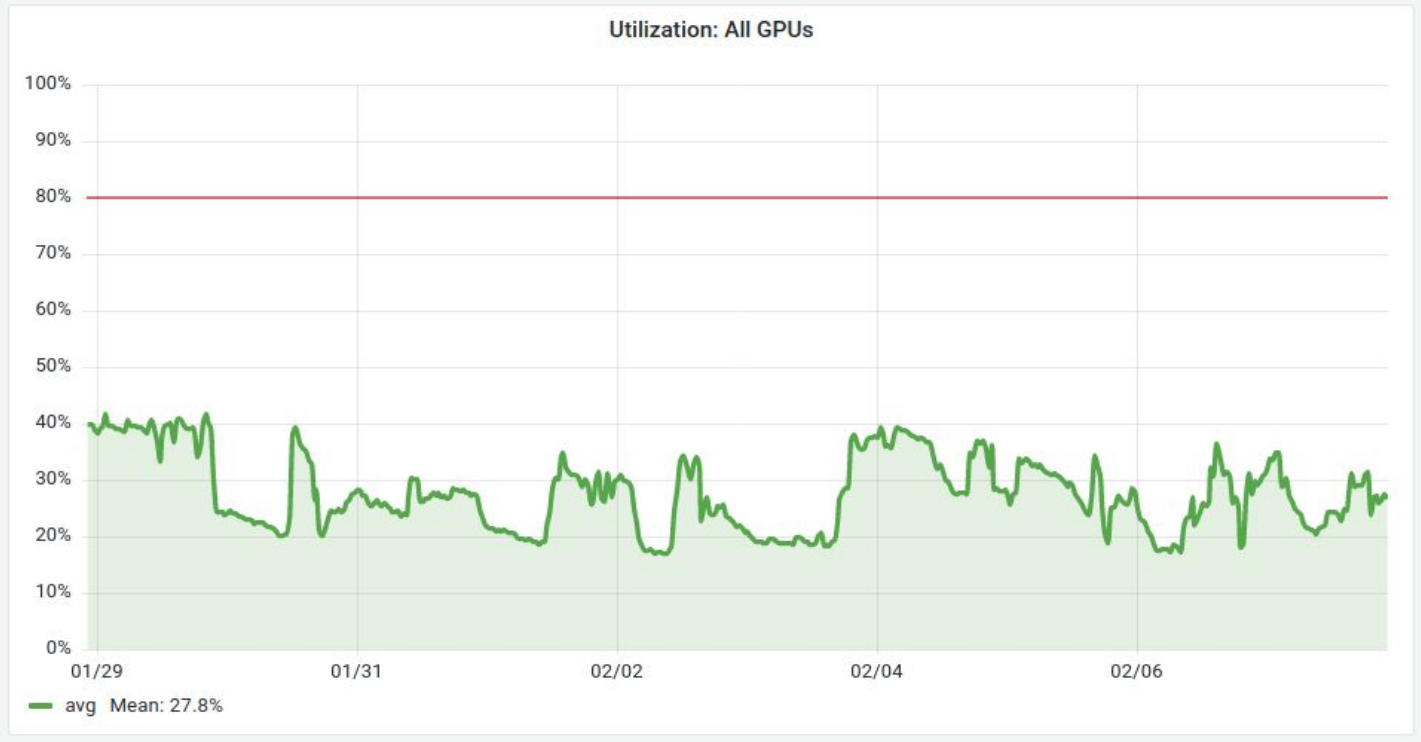
Performance engineering is complex and multifaceted

- Job scheduler
- Host (CPU) side bottleneck
- **Device (GPU) side bottleneck**

Low throughput is bad for everyone:

a. Leads to resource underutilization and resource waste
b. Training jobs runs slower

How much can under-utilization cost us?



A lot!

How to improve the utilization of my DL jobs?

1. System level optimizations
    a. Profiling to understand the performance bottlenecks
    b. Optimize the bottlenecks with different techniques
    c. Iterate
2. Use the right hardware resources and GPU

(this only scratches the surface of all possible optimizations that one can apply)

Agenda

1. Preparation: Install VSCode extension for profiling and performance prediction
   - We are going to use **Skyline** and **Habitat**, open-source DL tools integrated with VSCode
2. Hands-on iterative profiling and optimization
   - Using **Habitat** to select the right hardware
   - Walk through an example of image classification training (link to repo)
   - Profile, optimize (*x6)*
3. Additional optimizations and ideas
4. Q/A + Discussion

Preparation

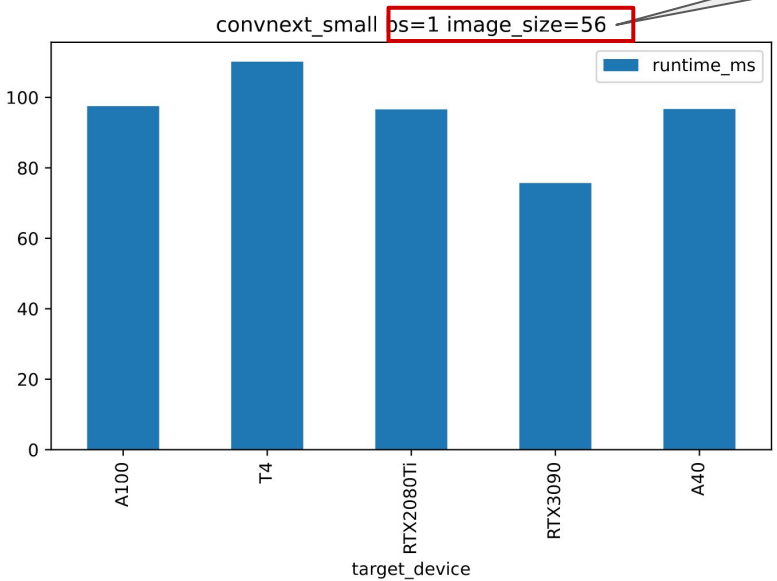
- If you have VSCode with Remote SSH access to your own workstation
  - Follow the instructions found in docs.centml.ai
- If you want to use the tools on Vector's cluster
  - Follow the instructions on github
- If you want to check it out
  - Use a VM hosted on centml-vms.dev

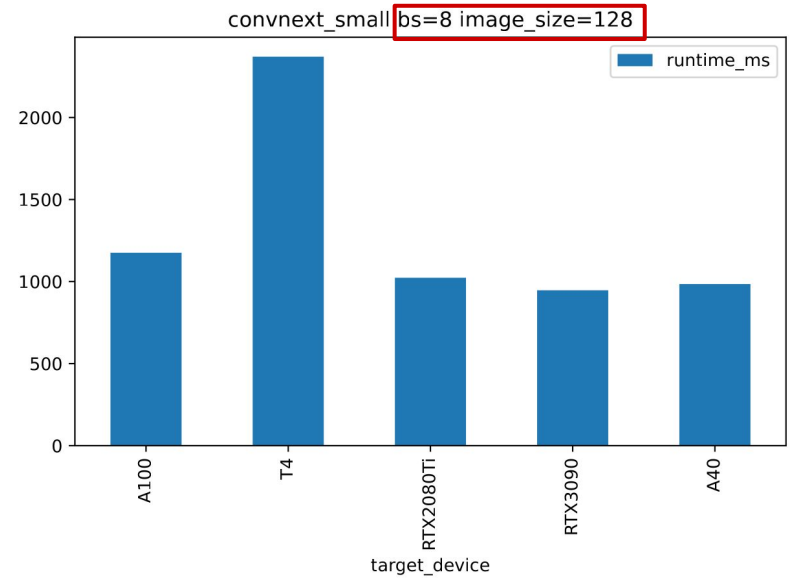
Demo

Aren’t more powerful GPUs better?

Small **input** under-utilizes GPU.

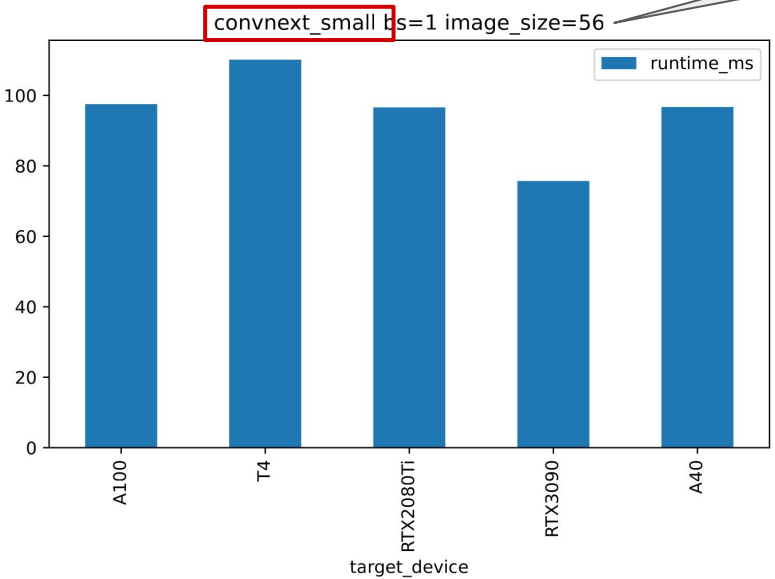
Not all the time! Not always a good idea to wait for the A40.

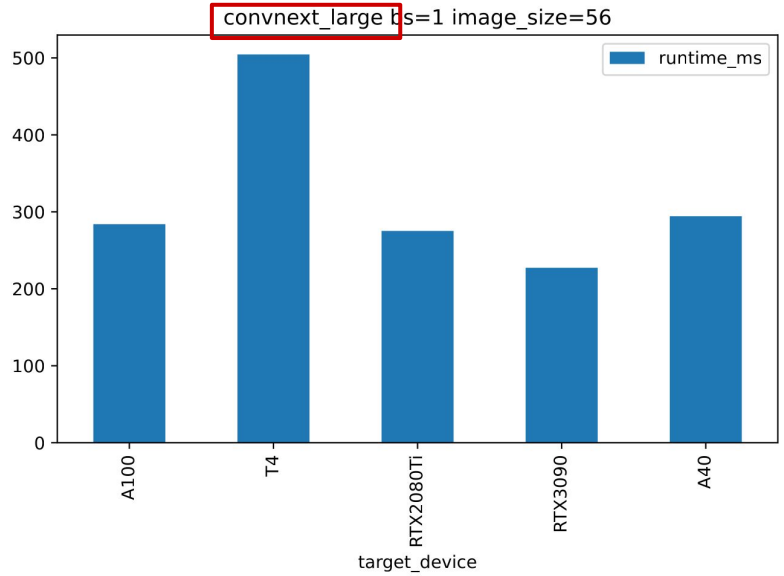




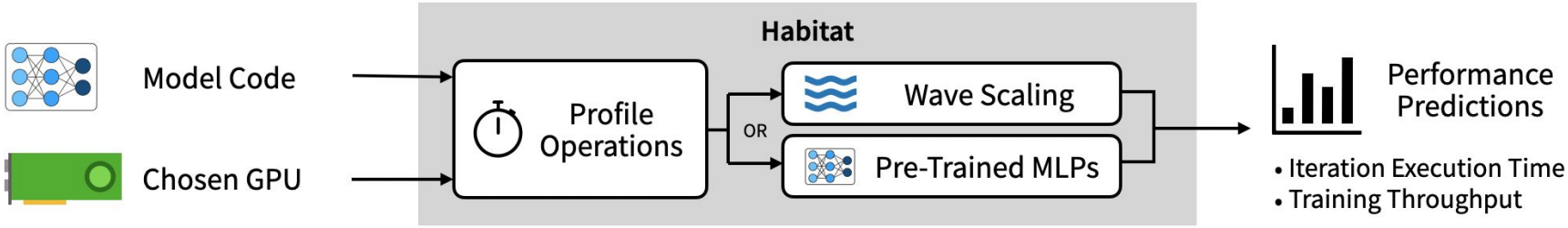
Aren’t more powerful GPUs better?

Small **model** under-utilizes GPU.





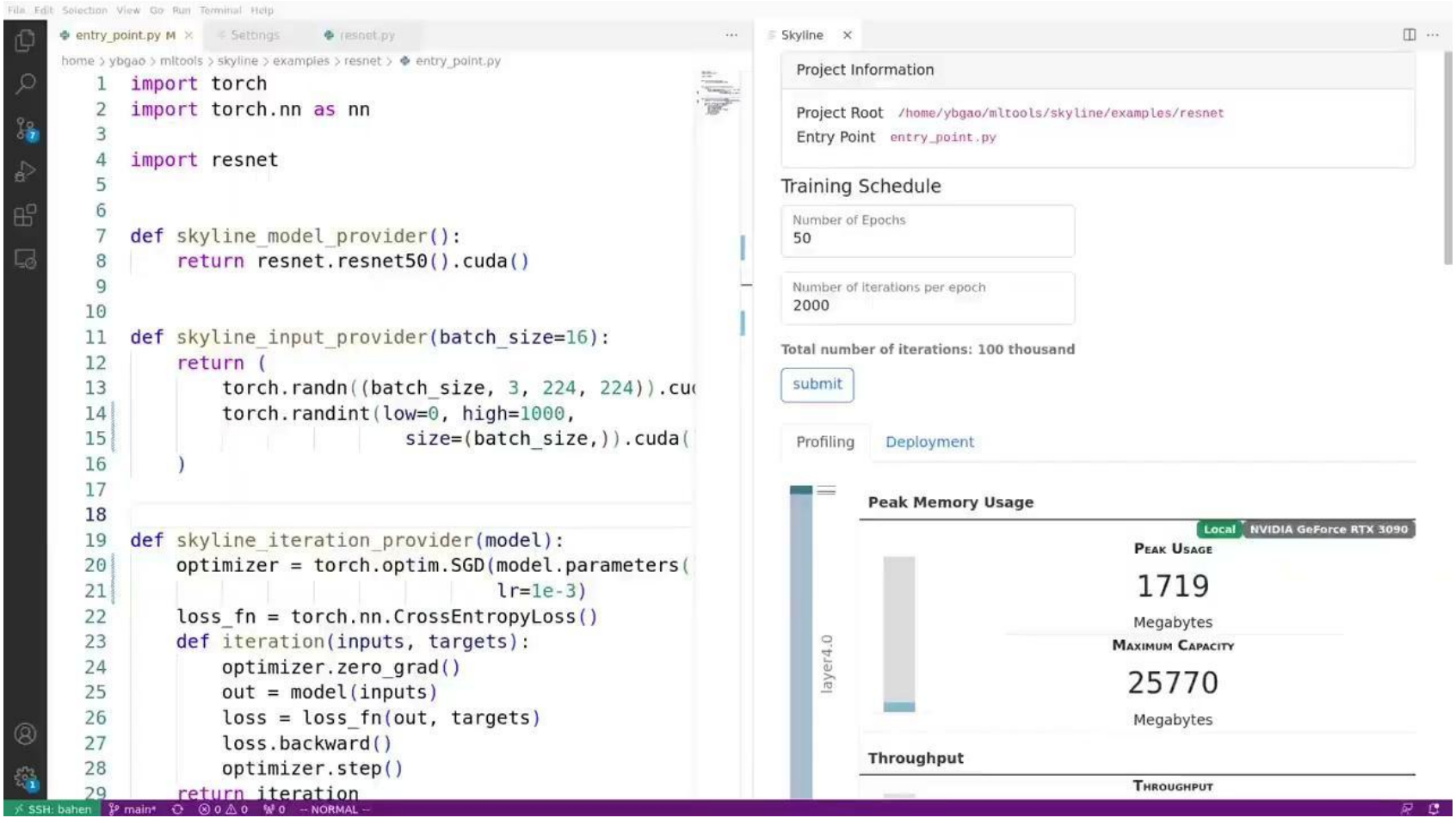
GPU Runtime Predictor (Habitat)



Pick the best GPU for your training job, whether it is:

- Deciding which new GPU to purchase for your local workstation
- Which cloud GPU instance to pick
- **Which cluster GPU to queue up for at Vector**

Tedious to benchmark model on all the available GPUs.


File Edit Selection View Go Run Terminal Help
entry_point.py M
Settings
resnet.py
home > ybgao > mltools > skyline > examples > resnet > entry_point.py
1 import torch
2 import torch.nn as nn
3
4 import resnet
5
6
7 def skyline_model_provider():
8 return resnet.resnet50().cuda()
9
10
11 def skyline_input_provider(batch_size=16):
12 return (
13 torch.randn((batch_size, 3, 224, 224)).cu
14 torch.randint(low=0, high=1000,
15 size=(batch_size,)).cuda(
16)
17
18
19 def skyline_iteration_provider(model):
20 optimizer = torch.optim.SGD(model.parameters(
21 lr=1e-3)
22 loss_fn = torch.nn.CrossEntropyLoss()
23 def iteration(inputs, targets):
24 optimizer.zero_grad()
25 out = model(inputs)
26 loss = loss_fn(out, targets)
27 loss.backward()
28 optimizer.step()
29 return iteration
SSH: bahen
main*
0 0
0
-- NORMAL --
Skyline
Project Information
Project Root /home/ybgao/mltools/skyline/examples/resnet
Entry Point entry_point.py
Training Schedule
Number of Epochs
50
Number of iterations per epoch
2000
Total number of iterations: 100 thousand
submit
Profiling
Deployment
Peak Memory Usage
Local
NVIDIA GeForce RTX 3090
Peak Usage
1719
Megabytes
Maximum Capacity
25770
Megabytes
layer4.0
Throughput
Throughput

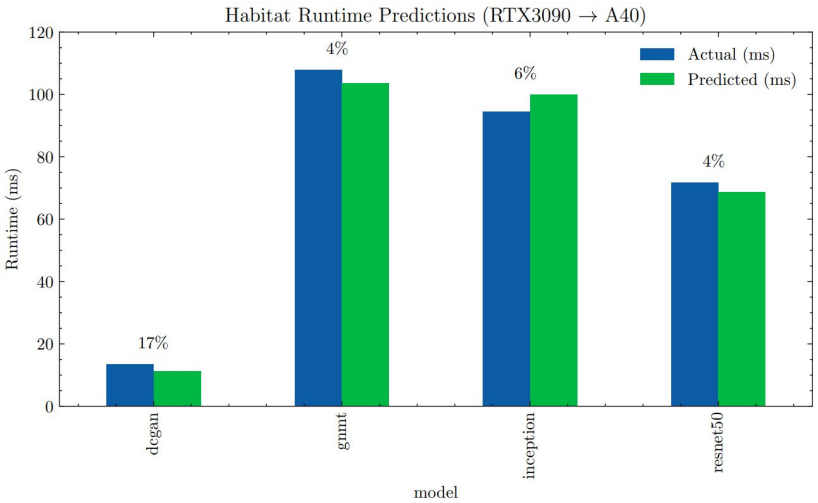
Which GPUs are supported?

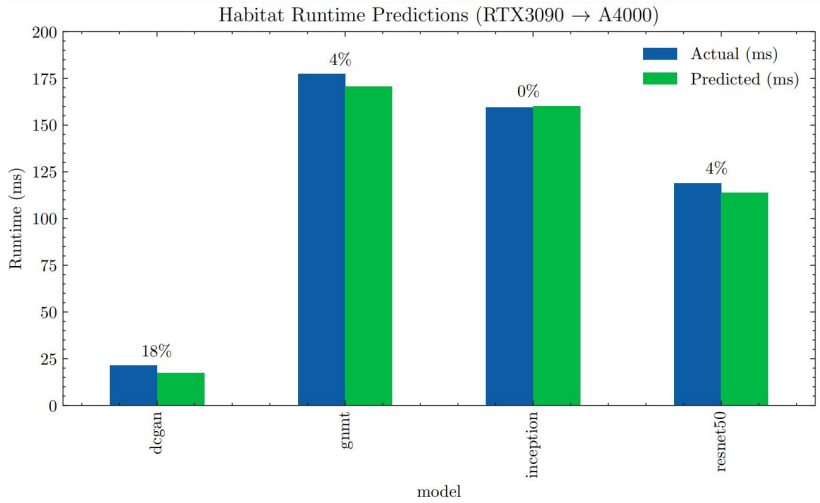
Generation \ Use Case	Desktop/Consumer	Workstation/Server
Pascal	GTX1080Ti	Quadro P4000 P4 P100
Volta		V100
Turing	RTX2070 RTX2080Ti	Quadro RTX4000
Tesla		T4
Ampere	RTX3090	A100 A40 A4000
Hopper		H100 (soon?)

- Beta

- Deprecated / Not available

How accurate is Habitat?





Deploying to the cloud

Close integration with Skyline

Performance predictions provided as part of the VSCode extension.

<Add screenshot of Habitat here>

CentML Tools (DeepView)

Interactive Profiler (Skyline)

Identifies performance bottlenecks

Enables rapid iterative profiling

Quantifies energy consumption and environmental impacts of training jobs.

Runtime Predictor (Habitat)

Predicts a deep neural network's training iteration execution time on a different GPU.

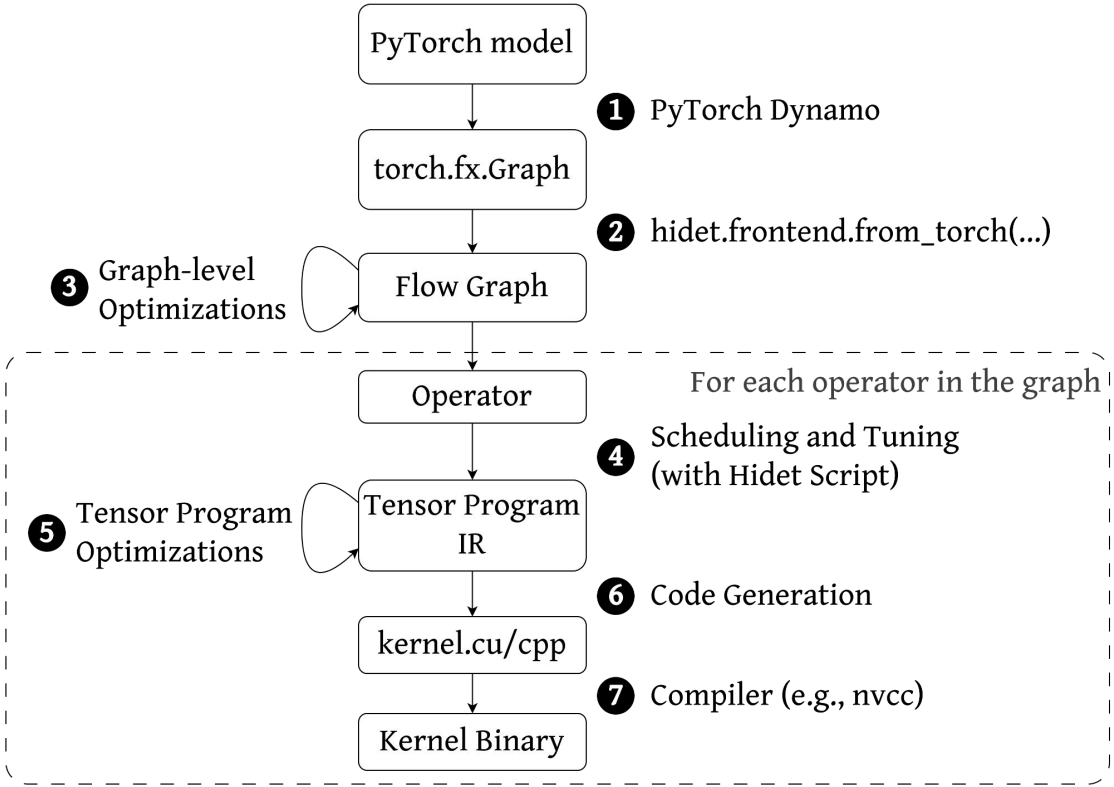
Recommends the most cost/time effective hardware option for your workload

Additional Optimizations: Horizontal Fusion

Horizontally fused training array for efficient training

- Best for training small models + hyper-parameter tuning

Additional Optimizations: Tensor Compiler (Hidet)



GitHub

> `github.com/hidet-org/hidet`

Documentation

> `docs.hidet.org`

Academic Paper

> `Hidet: Task-Mapping Programming Paradigm for Deep Learning Tensor Programs (to appear in ASPLOS '23)`

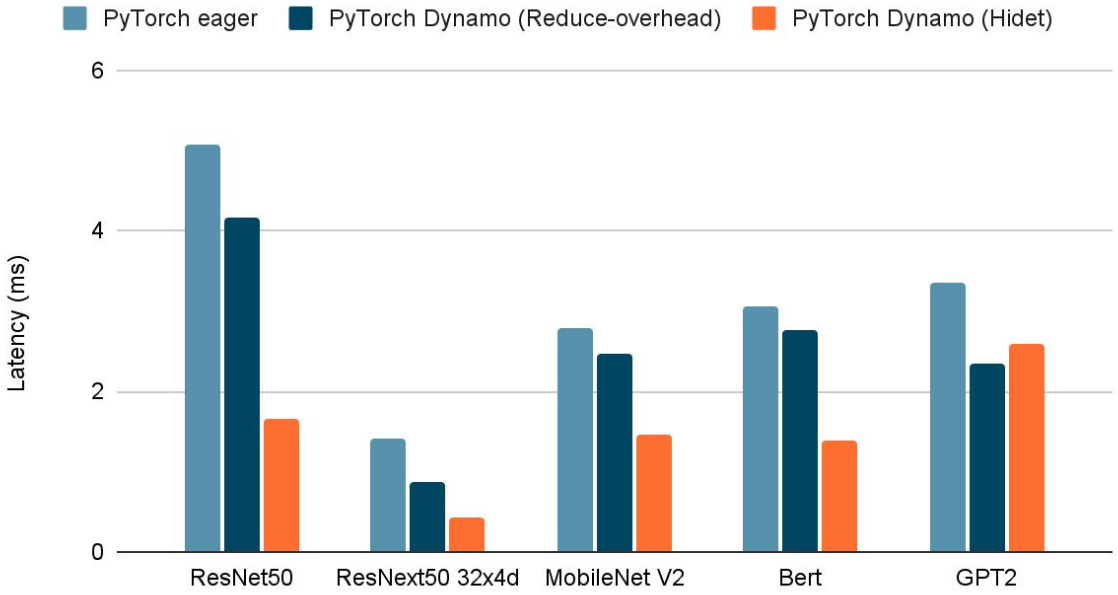
Thank you!

Check us out at
centml.ai

Email us at
ybgao@centml.ai
xin@centml.ai

Backup Slides

Comparisons of Different Torch Dynamo Backends



Models: Torch Hub
- torchvision: all CNNs
- transformers: bert & gpt2
- bert: bert-base-uncased

Data Type: float32

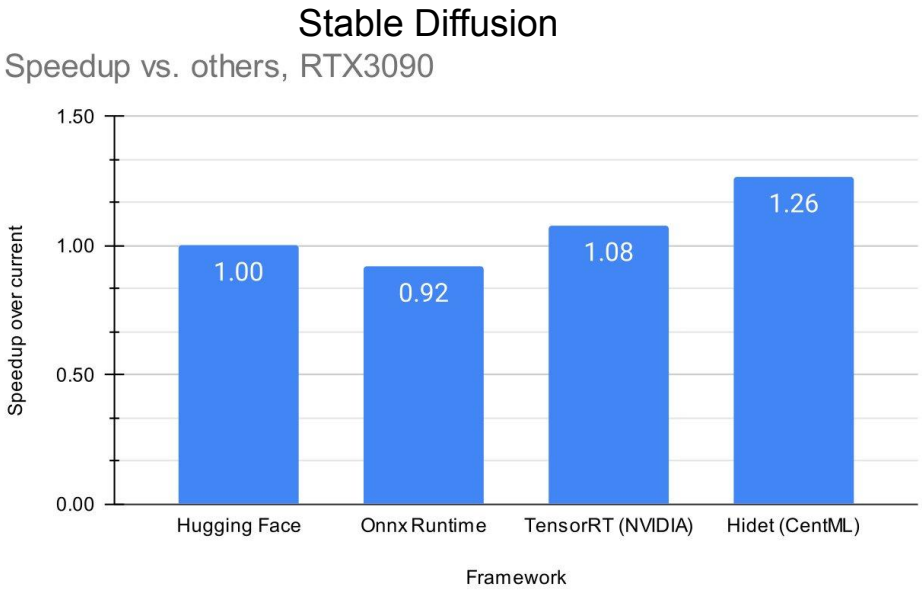
Input Size:
- Batch Size = 1
- Sequence length = 128 (bert & gpt2)

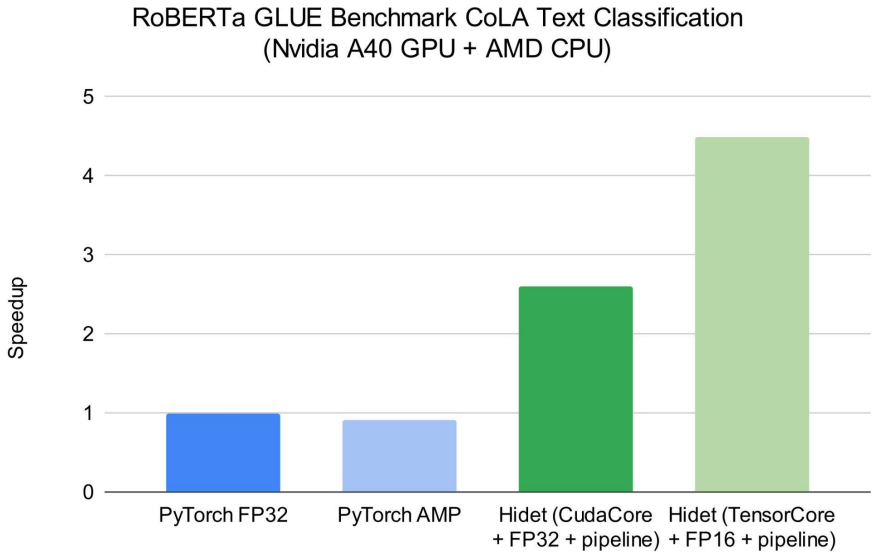
Hardware: NVIDIA RTX 3090

We are still actively working on optimizing the performance.

```
# optimize the model with hidet provided backend 'hidet'
model_hidet = torch.compile(model, backend='hidet')
```

Customer Success





Hidet Script

```
with hidet.lang.script_module() as script_module:

    @hidet.lang.script
    def matmul_kernel(
        a_ptr: ~float32,  # ~ means "pointer to", similar to "*"
        b_ptr: ~float32,
        c_ptr: ~float32,
        m_size: int32,
        n_size: int32,
        k_size: int32,
    ):
        attr.func_name = 'matmul_kernel'
        attr.cuda_block_dim = num_threads
        attr.cuda_grid_dim = (
            (m_size + block_m_size - 1) // block_m_size,
            (n_size + block_n_size - 1) // block_n_size,
        )

        a = as_tensor_pointer(a_ptr, float32, [m_size, k_size])
        b = as_tensor_pointer(b_ptr, float32, [k_size, n_size])
        c = as_tensor_pointer(c_ptr, float32, [m_size, n_size])

        smem_a = tensor('shared', float32, shape=[block_m_size,
        smem_b = tensor('shared', float32, shape=[block_k_size,
        regs_c = tensor(
            scope='register',
            dtype=float32,
```

Snapshot of Hidet Script

(See the documentation for a complete example)

Compared with Triton:

- Both translate python ast;
- Both support tuning;
- Programming model:
  - Hidet Script: thread
  - Triton: thread block
- Complexity:
  - CUDA C: +++
  - Hidet Script: ++
  - Triton: +
- Optimization Expression Ability:
  - CUDA C with PTX asm: +++
  - Hidet Script: ++
  - Triton: +

Task-mapping is the main contribution of our paper, which greatly simplifies tensor program writing and has implemented in Hidet Script.

What We Need

1. Not necessarily a PyTorch “feature”
   a. But we’d still appreciate feedback from PyTorch developers & community.
2. Co-marketing with PyTorch 2.0 release
   a. Promoting both Hidet and Dynamo (and how flexible it is to add custom backend)
   b. Provide another backend for PyTorch users to choose.
3. Backend registration API
   a. https://github.com/pytorch/pytorch/issues/91824