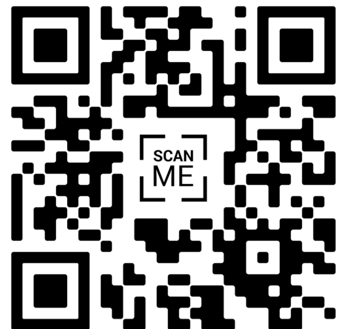


Grape🍇: Practical and Efficient Graph-based Executions for Dynamic Deep Neural Networks on GPUs

Bojian Zheng[1, 2, 3], Cody Hao Yu[4], Jie Wang[4], Yaoyao Ding[1, 2, 3], Yizhi Liu[4], Yida Wang[4], Gennady Pekhimenko[1, 2, 3]

1

2

3

4

Executive Summary

- Challenges posed by CUDA graphs:
  - Extra data movements into placeholders.
  - Huge GPU memory consumption on dynamic-shape workloads.
  - No support for data-dependent control flows.
- Grape🍇 addresses those challenges with: ① Alias Prediction, ② Metadata Compression, and ③ Predication Contexts.
- Key Result: Up to $\underline{\mathbf{1.41\times}}$ speedup over the prior state-of-the-art graph-based executor.

Outline

Background | What are CUDA graphs?

Challenges | of using CUDA graphs

Key Ideas | of Grape🍇 to resolve these challenges

Evaluation | on state-of-the-art workloads

Outline

Background What are CUDA graphs?

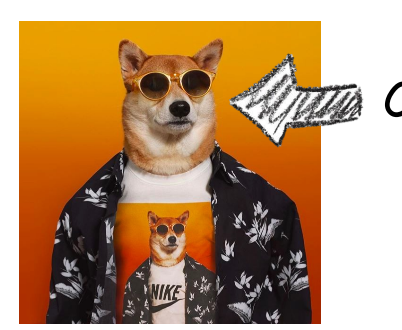
Challenges of using CUDA graphs

Key Ideas of Grape🍇 to resolve these challenges

Evaluation on state-of-the-art workloads

Deep Neural Networks (DNNs)

- State-of-the-art accuracies in many applications:

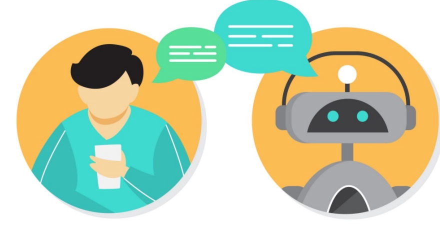


Cool Dog

Image Classification[1]

Machine Translation[2, 3]

Speech Recognition[4, 5]



Text Generation[6, 7]

[1] K. He et al. *Deep Residual Learning for Image Recognition*. CVPR 2016

[2] Y. Wu et al. *Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation*. arXiv 2016

[3] Ashish Vaswani et al. *Attention is All You Need*. NeurIPS 2017

[4] D. Amodei et al. *Deep Speech 2 : End-to-End Speech Recognition in English and Mandarin*. ICML 2016

[5] A. Baevski et al. *wav2vec 2.0: A Framework for Self-Supervised Learning of Speech Representations*. NeurIPS 2020

[6] A. Radford et al. *Language Models are Unsupervised Multitask Learners*. 2019

[7] W. Ben et al. *GPT-J-6B: A 6 Billion Parameter Autoregressive Language Model*. 2020

Machine Learning System Overview

Applications

Machine Learning Systems PyTorch[1] TensorFlow[2]

Hardware Accelerators

[1] A. Paszke et al. *PyTorch: An Imperative Style, High-Performance Deep Learning Library*. NeurIPS 2019

[2] M. Abadi et al. *TensorFlow: A System for Large-Scale Machine Learning*. OSDI 2016

Machine Learning System Overview

Applications

Machine Learning Systems

PyTorch[1]

TensorFlow[2]

Hardware Accelerators

[1] A. Paszke et al. *PyTorch: An Imperative Style, High-Performance Deep Learning Library*. NeurIPS 2019

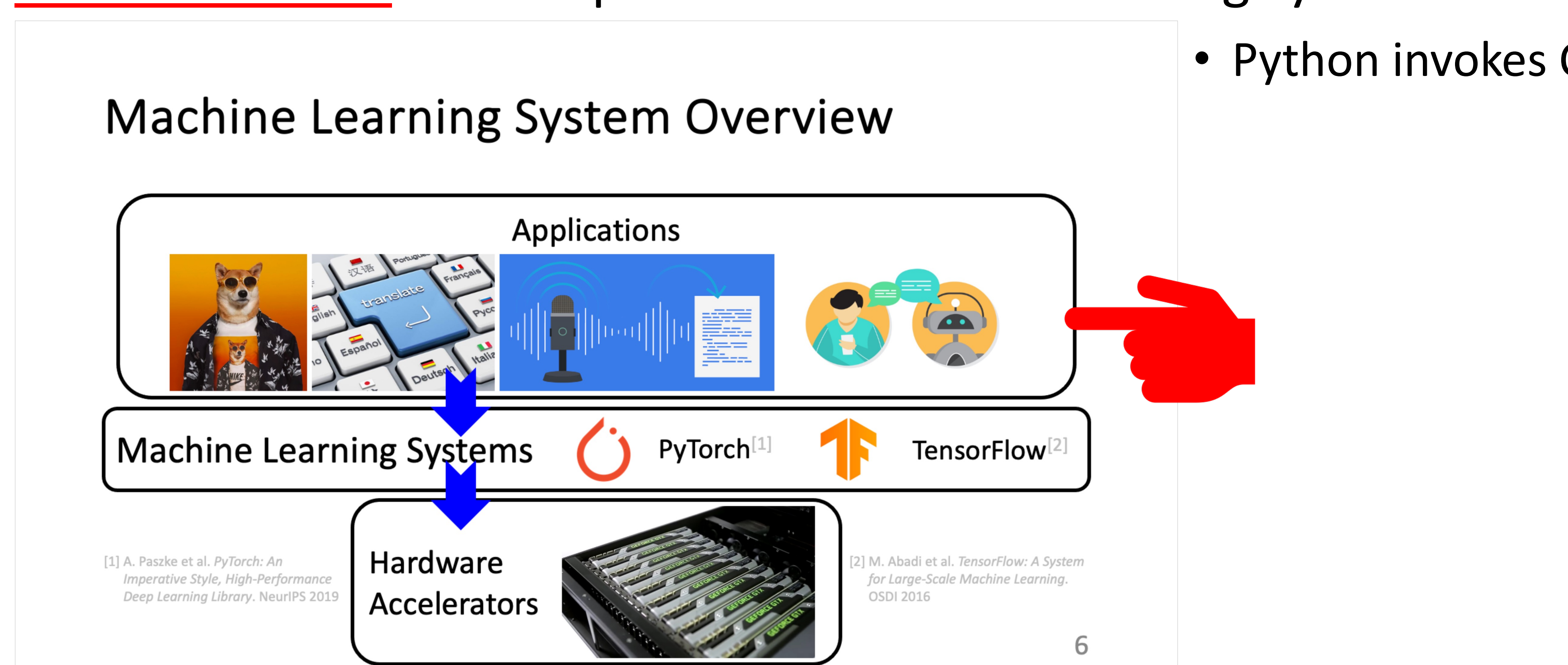
[2] M. Abadi et al. *TensorFlow: A System for Large-Scale Machine Learning*. OSDI 2016

Inefficiency: CPU Overheads

- **CPU overheads** are ubiquitous in machine learning systems.

Inefficiency: CPU Overheads

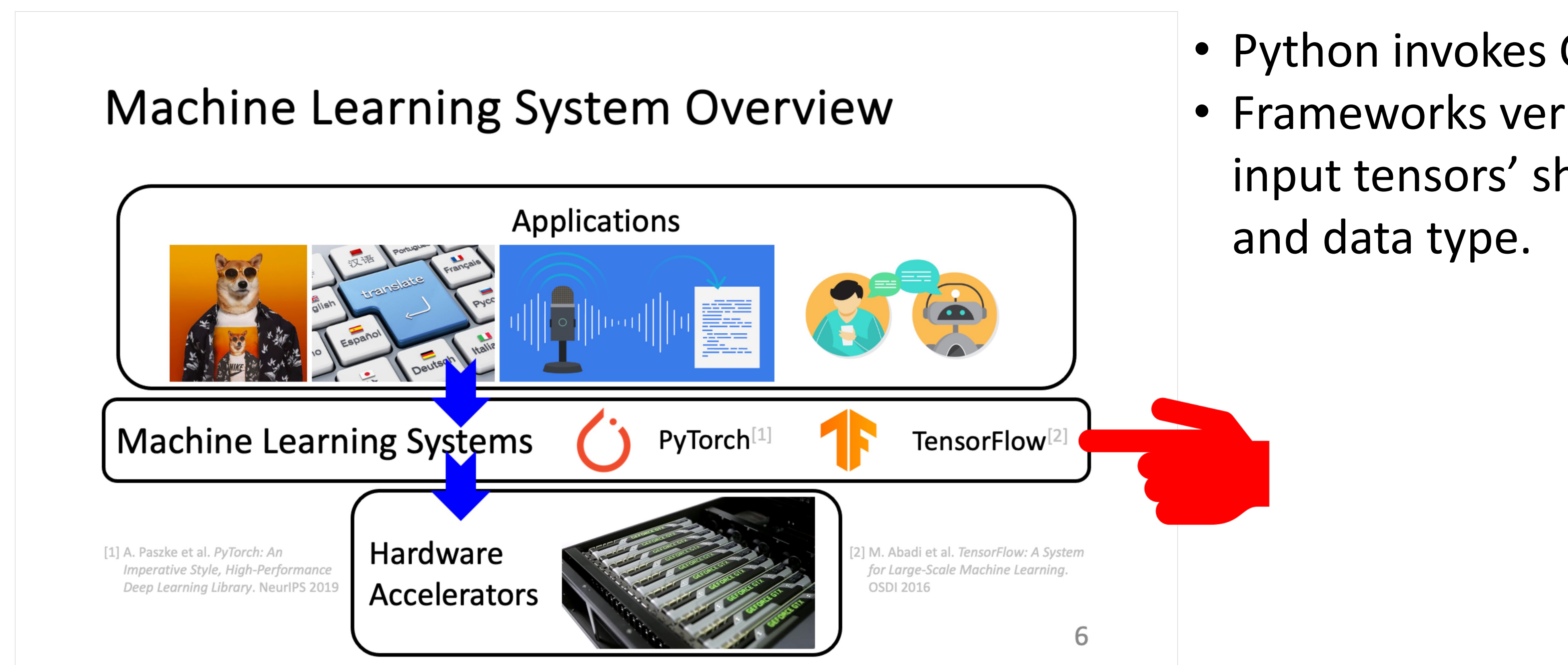
- **CPU overheads** are ubiquitous in machine learning systems.



- Python invokes C APIs.

Inefficiency: CPU Overheads

- **CPU overheads** are ubiquitous in machine learning systems.



- Python invokes C APIs.
- Frameworks verify input tensors' shape and data type.

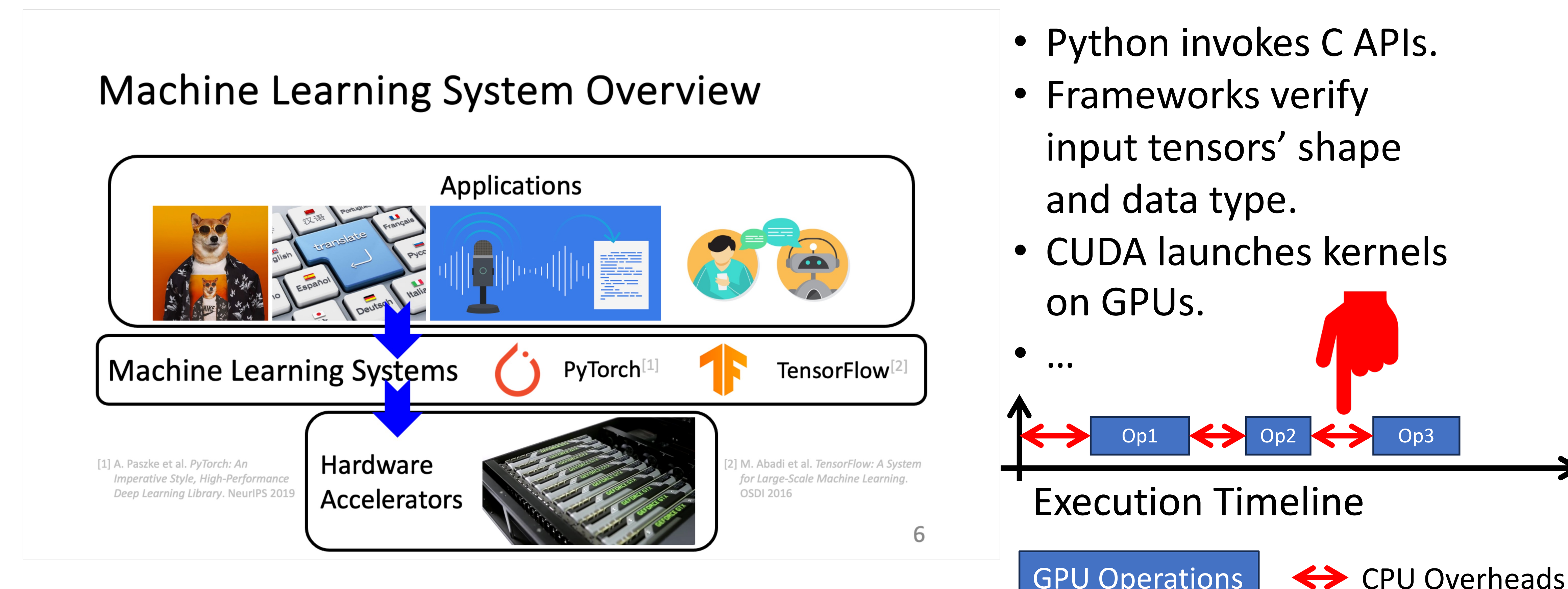
Inefficiency: CPU Overheads

- **CPU overheads** are ubiquitous in machine learning systems.

- Python invokes C APIs.
- Frameworks verify input tensors' shape and data type.
- CUDA launches kernels on GPUs.

Inefficiency: CPU Overheads

- **CPU overheads** are ubiquitous in machine learning systems.



CUDA Graphs

- Key Idea: **Capture** effective GPU computations in the first run and **replay** them in subsequent runs.

Capture

```
A = Tensor()
graph_ctx = CUDAGraph()
with graph_ctx:
  MyDNN(A)
```

Record

Op1 Op2 Op3

Execution Timeline

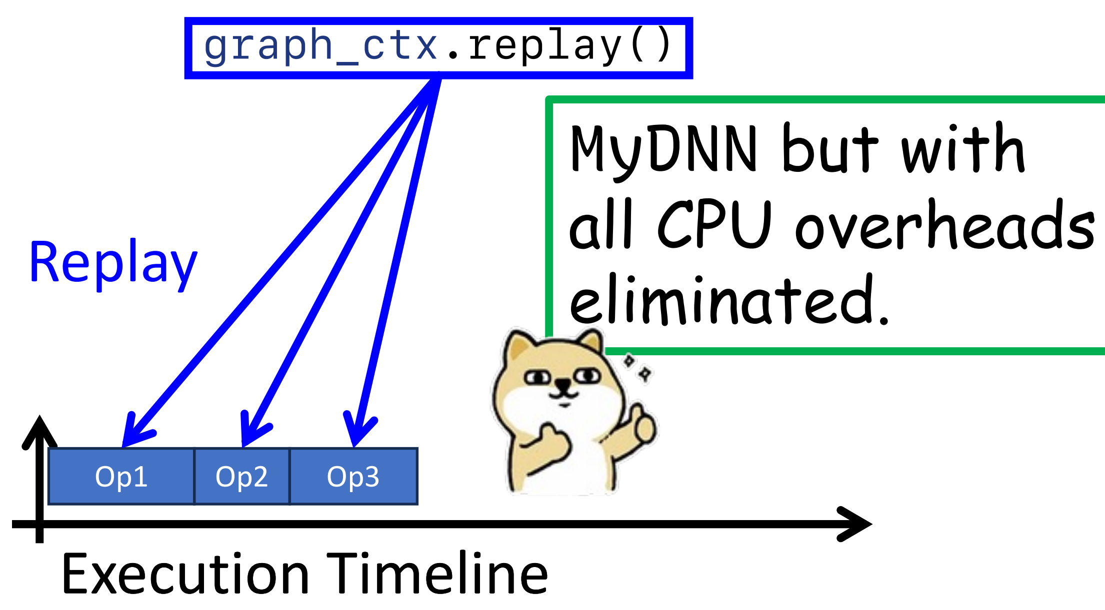
CUDA Graphs

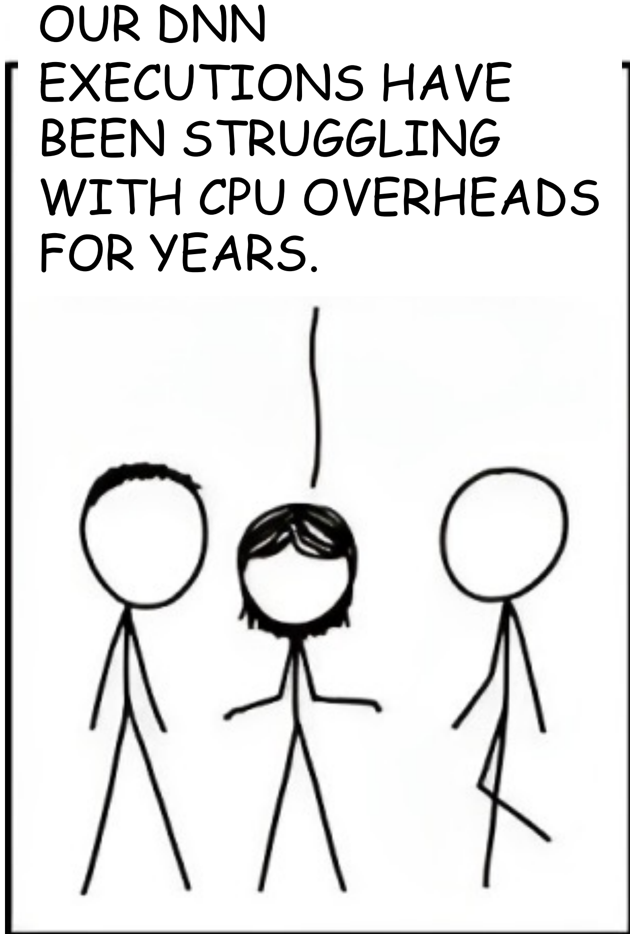
- Key Idea: **Capture** effective GPU computations in the first run and **replay** them in subsequent runs.

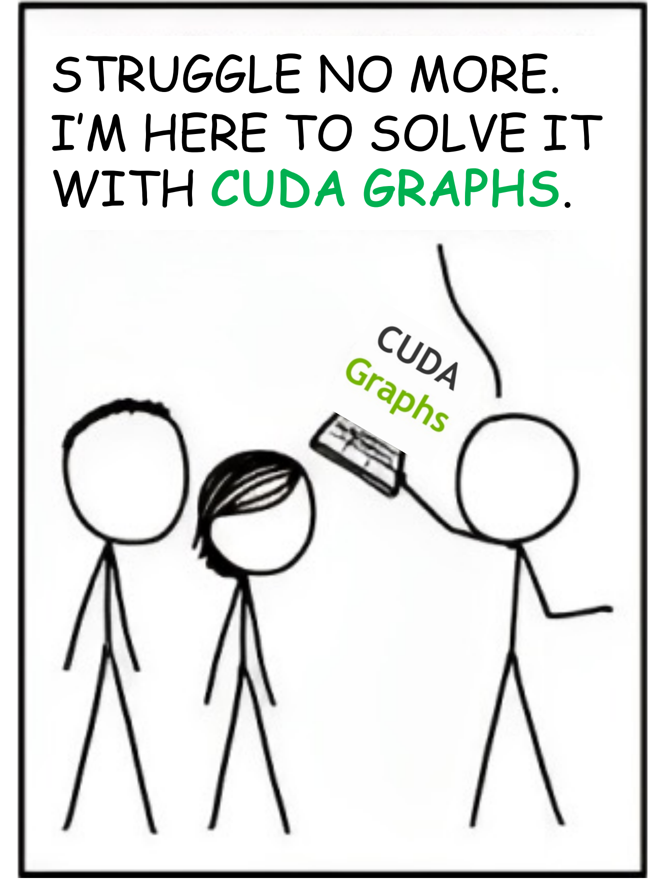
Capture

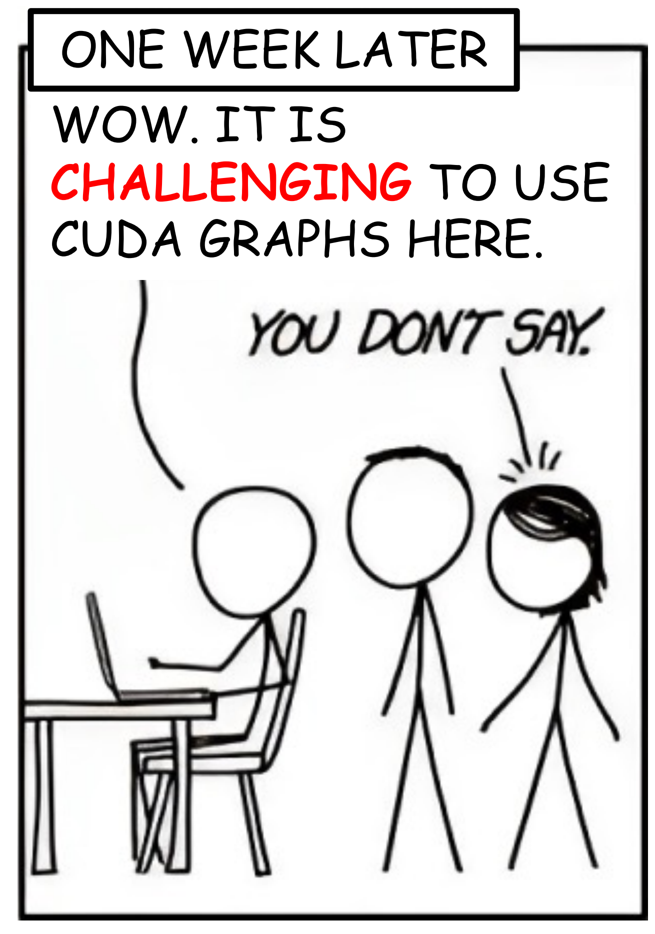
```
A = Tensor()
graph_ctx = CUDAGraph()
with graph_ctx:
  MyDNN(A)
```

Replay




OUR DNN EXECUTIONS HAVE BEEN STRUGGLING WITH CPU OVERHEADS FOR YEARS.


STRUGGLE NO MORE. I'M HERE TO SOLVE IT WITH CUDA GRAPHS.
CUDA Graphs


ONE WEEK LATER
WOW. IT IS CHALLENGING TO USE CUDA GRAPHS HERE.
YOU DON'T SAY.

Outline

Background What are CUDA graphs?

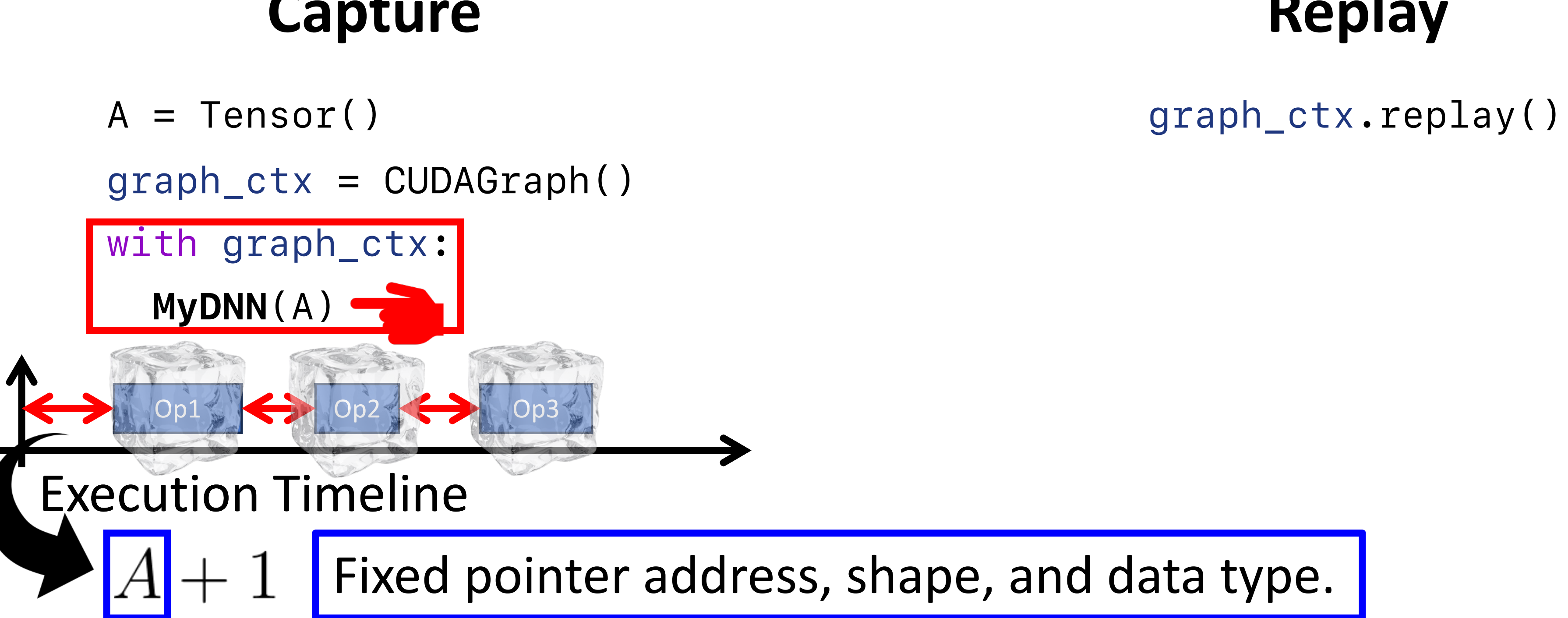
Challenges of using CUDA graphs

Key Ideas of Grape🍇 to resolve these challenges

Evaluation on state-of-the-art workloads

CUDA Graphs' Weaknesses

- All computations must be frozen.



CUDA Graphs' Weaknesses

- All computations must be frozen.

Capture

```
A = Tensor()
graph_ctx = CUDAGraph()
with graph_ctx:
  MyDNN(A)
```

Op1 Op2 Op3

Execution Timeline

$A + 1$ Constant.

Replay

```
graph_ctx.replay()
```

CUDA Graphs' Weaknesses

- All computations must be frozen.
- Every CUDA graph's creation consumes GPU memory.

Capture

```
A = Tensor()
graph_ctx = CUDAGraph()
with graph_ctx:
  MyDNN(A)
```

Replay

```
graph_ctx.replay()
```

Challenges posed by CUDA Graphs

- Implications:
  1. Synthetic inputs are used as *placeholders* at capture time and populated with real input values at runtime.

```
A = Tensor()
graph_ctx = CUDAGraph()
with graph_ctx:
  MyDNN(A)
```

```
graph_ctx.replay()
```

Weaknesses

- **All computations must be frozen.**
- Every CUDA graph's creation consumes GPU memory.

Challenges posed by CUDA Graphs

- Implications:
  1. Synthetic inputs are used as *placeholders* at capture time and populated with real input values at runtime.

Significant runtime overheads (up to **13%**).

```
ph_A = Tensor()
graph_ctx = CUDAGraph()
with graph_ctx:
  MyDNN(ph_A)
```

```
A = Tensor()
ph_A.copyFrom(A)
graph_ctx.replay()
```

Weaknesses

- **All computations must be frozen.**
- Every CUDA graph's creation consumes GPU memory.

Challenges posed by CUDA Graphs

```
ph_A = Tensor()
graph_ctx = CUDAGraph()
with graph_ctx:
  MyDNN(ph_A)
```

```
A = Tensor()
ph_A.copyFrom(A)
graph_ctx.replay()
```

- Implications:
  1. Synthetic inputs are used as *placeholders* at capture time and populated with real input values at runtime.
  2. To efficiently execute dynamic-shape workloads, all possible shapes have to be captured.

Weaknesses

- **All computations must be frozen.**
- Every CUDA graph's creation consumes GPU memory.

+

=

Huge GPU memory consumption (**20-100 GB**).

Challenges posed by CUDA Graphs

- Implications:
    1. Synthetic inputs are used as *placeholders* at capture time and populated with real input values at runtime.
    2. To efficiently execute dynamic-shape workloads, all possible shapes have to be captured.
    3. Cannot handle data-dependent control flows.

```
ph_A = Tensor()
graph_ctx = CUDAGraph()
with graph_ctx:
  MyDNN(ph_A)
```

```
A = Tensor()
ph_A.copyFrom(A)
graph_ctx.replay()
```

Weaknesses

- **All computations must be frozen.**
- Every CUDA graph's creation consumes GPU memory.

Challenges posed by CUDA Graphs

- CUDA Graphs' Challenges:
  1. Data movements into placeholders incur significant runtime overheads.
  2. Huge GPU memory consumption to efficiently execute dynamic-shape workloads.
  3. No support for data-dependent control flows.

- Grape🍇, a graph compiler that addresses those challenges with:

① Alias Prediction

② Metadata Compression

③ Predication Contexts

Outline

Background What are CUDA graphs?

Challenges of using CUDA graphs

Key Ideas of Grape🍇 to resolve these challenges

Evaluation on state-of-the-art workloads

Grape🍇's Key Ideas

① Alias Prediction

- If a Python code position yields a placeholder alias, the same position is likely to yield another alias in subsequent iterations.

```
A = Tensor()
ph_A.copyFrom(A)
graph_ctx.replay()
```

Grape🍇's Key Ideas

① Alias Prediction

- If a Python code position yields a placeholder alias, the same position is likely to yield another alias in subsequent iterations.

foo.py

```
42  A = Tensor()
```

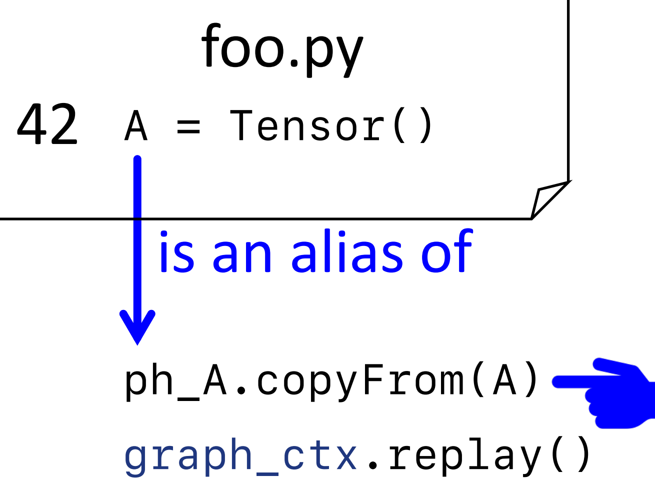
is an alias of

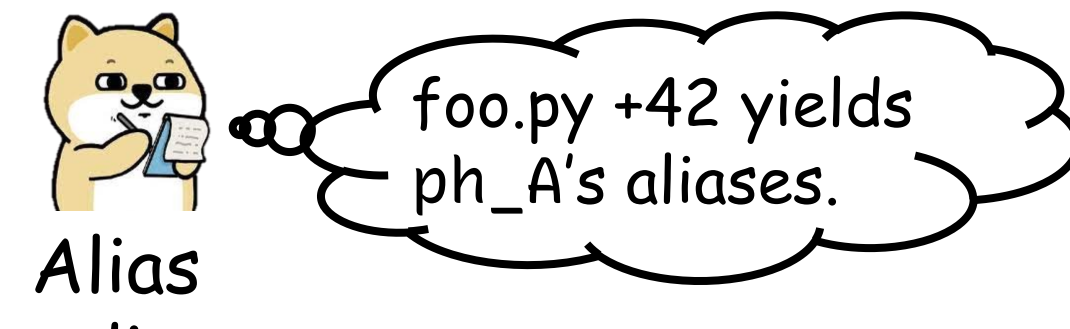
```
ph_A.copyFrom(A)
graph_ctx.replay()
```

Grape🍇's Key Ideas

① Alias Prediction

- If a Python code position yields a placeholder alias, the same position is likely to yield another alias in subsequent iterations.





Grape🍇's Key Ideas

① Alias Prediction

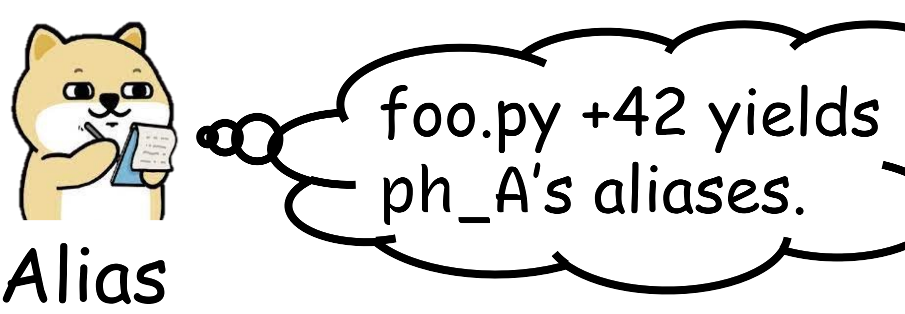
- If a Python code position yields a placeholder alias, the same position is likely to yield another alias in subsequent iterations.

foo.py

```
42  A = Tensor()

ph_A.copyFrom(A)
graph_ctx.replay()
```

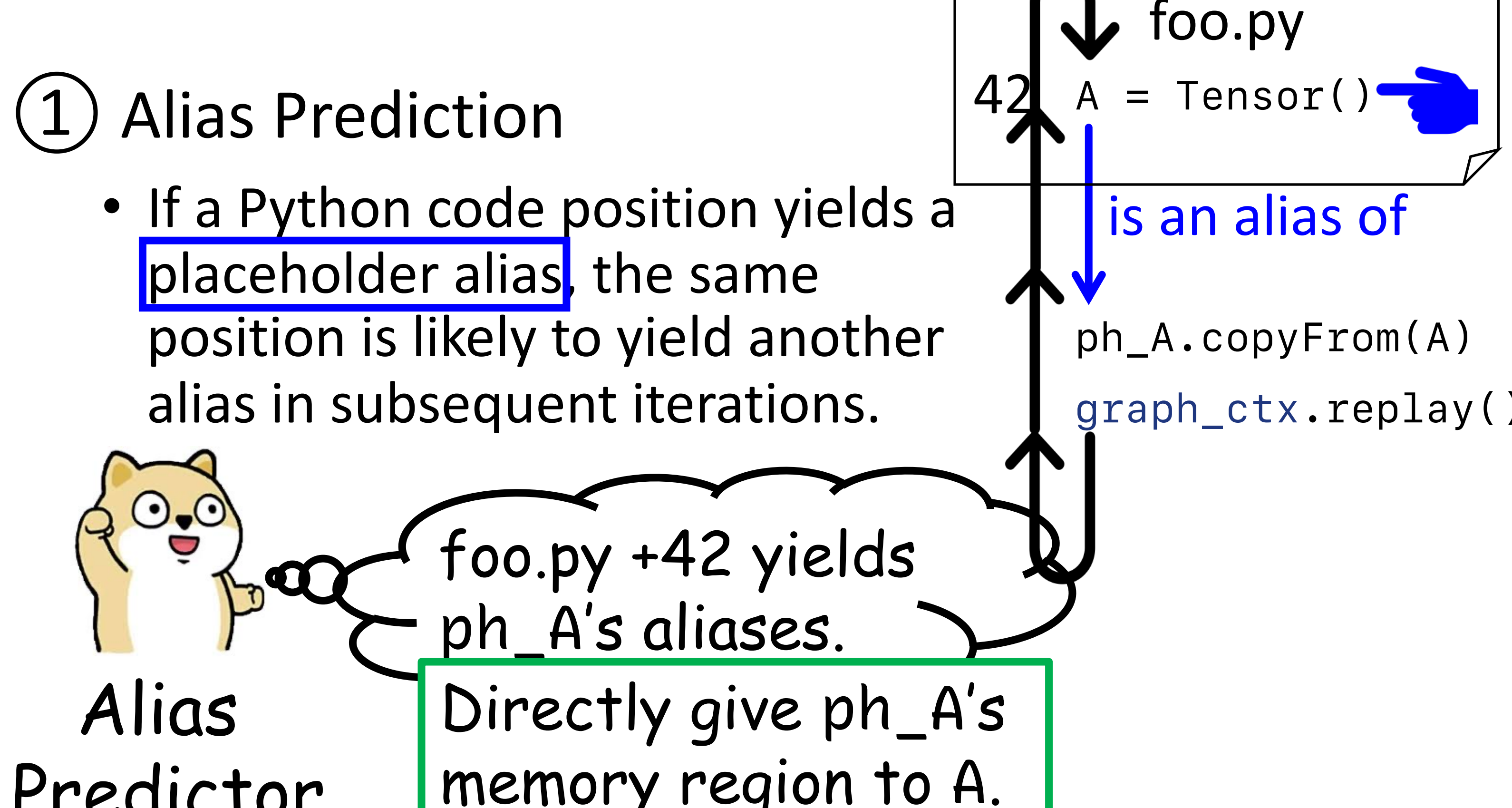
is an alias of



Grape🍇's Key Ideas

① Alias Prediction

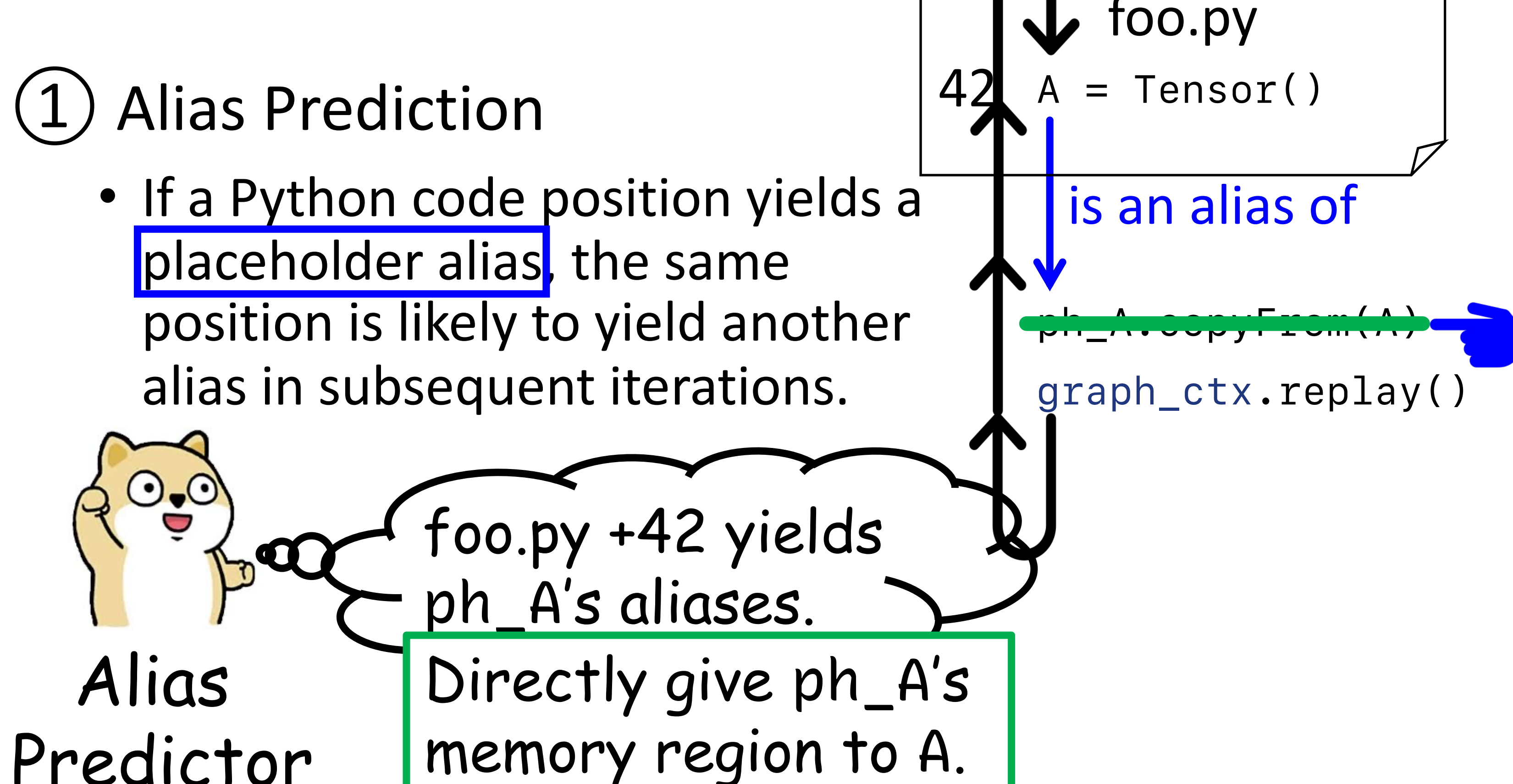
- If a Python code position yields a placeholder alias, the same position is likely to yield another alias in subsequent iterations.



Grape🍇's Key Ideas

① Alias Prediction

- If a Python code position yields a placeholder alias, the same position is likely to yield another alias in subsequent iterations.



Grape 🍇's Key Ideas

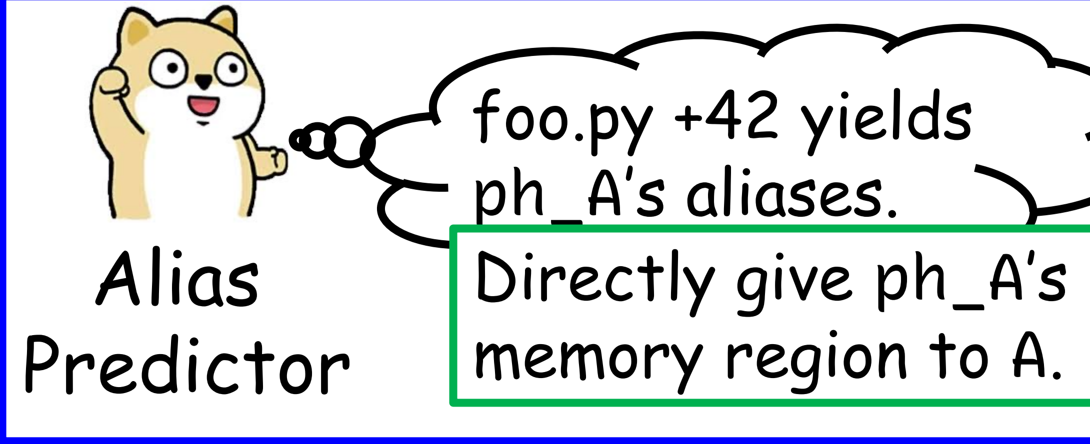
foo.py

```
42  A = Tensor()
```

① Alias Prediction

- If a Python code position yields a placeholder alias, the same position is likely to yield another alias in subsequent iterations.

```
ph_A.copyFrom(A)
graph_ctx.replay()
```

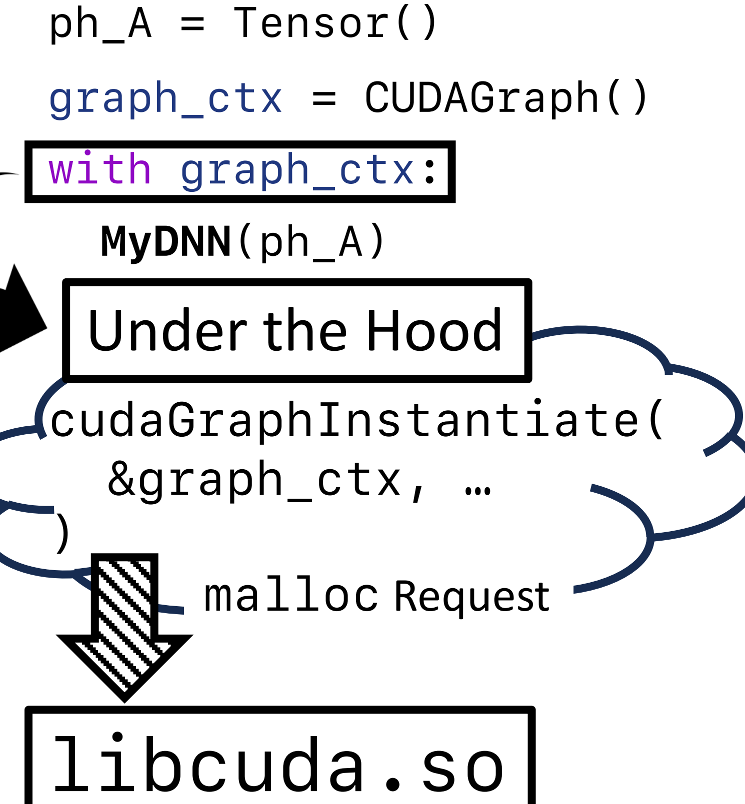


Transparent and Language-Independent

Grape🍇's Key Ideas

② Metadata Compression

- The memory allocations of CUDA graphs have high sparsity and value redundancy and hence are highly compressible.



Grape🍇's Key Ideas

② Metadata Compression

- The memory allocations of CUDA graphs have high sparsity and value redundancy and hence are highly compressible.

```
ph_A = Tensor()
graph_ctx = CUDAGraph()
with graph_ctx:
  MyDNN(ph_A)
```

Under the Hood

```
cudaGraphInstantiate(
  &graph_ctx, …
)
```

malloc Request

libcuda.so

nvidia.ko

Customized NVIDIA's open-gpu-kernel-modules

Grape🍇's Key Ideas

② Metadata Compression

- The memory allocations of CUDA graphs have high sparsity and value redundancy and hence are highly compressible.

```
ph_A = Tensor()
graph_ctx = CUDAGraph()
with graph_ctx:
  MyDNN(ph_A)
```

Under the Hood

```
cudaGraphInstantiate(
  &graph_ctx, …
)
```

malloc Request

Intercept

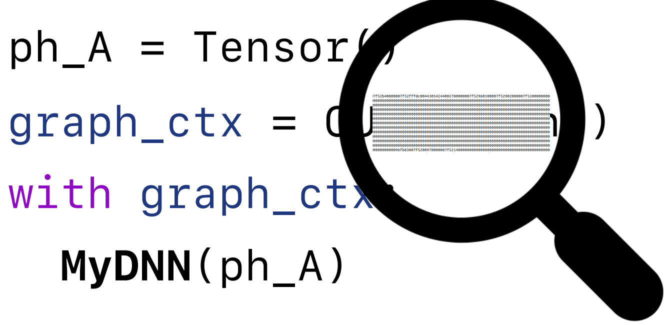
libcuda.so

nvidia.ko

Grape🍇's Key Ideas

② Metadata Compression

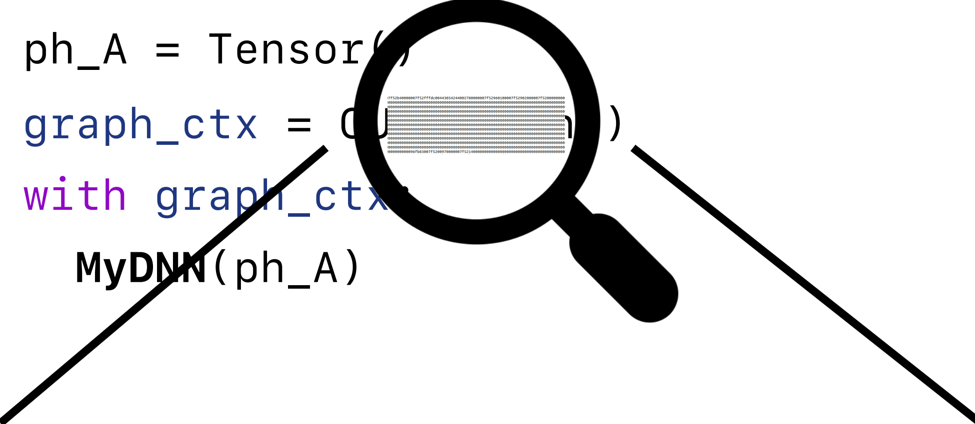
- The memory allocations of CUDA graphs have high sparsity and value redundancy and hence are highly compressible.



Grape🍇's Key Ideas

② Metadata Compression

- The memory allocations of CUDA graphs have high sparsity and value redundancy and hence are highly compressible.



Grape🍇's Key Ideas

② Metadata Compression

- The memory allocations of CUDA graphs have high sparsity and value redundancy and hence are highly compressible.

```
ph_A = Tensor()
graph_ctx = CUDAGraph()
with graph_ctx:
  MyDNN(ph_A)
```

Speculations:

- Sparsity: Underutilize the reserved function argument spaces.

E.g.,

```
__global__ void cudaKernelSample(
    const float *const input,
    float *const output
);
```

16 bytes used out of the reserved 4 KB by CUDA

Grape🍇's Key Ideas

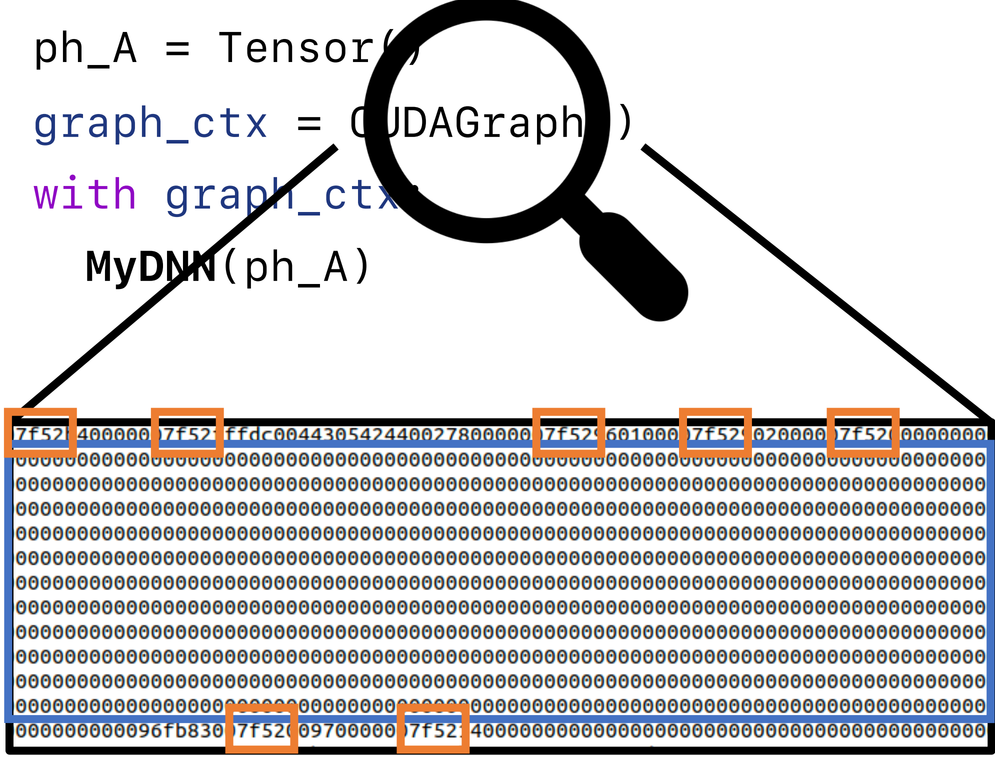
② Metadata Compression

- The memory allocations of CUDA graphs have high sparsity and value redundancy and hence are highly compressible.

Speculations:

- Sparsity: Underutilize the reserved function argument spaces.
- Redundancy: Pointer values.

E.g.,
```
__global__ void cudaKernelSample(
    const float *const input,
    float *const output
);
```



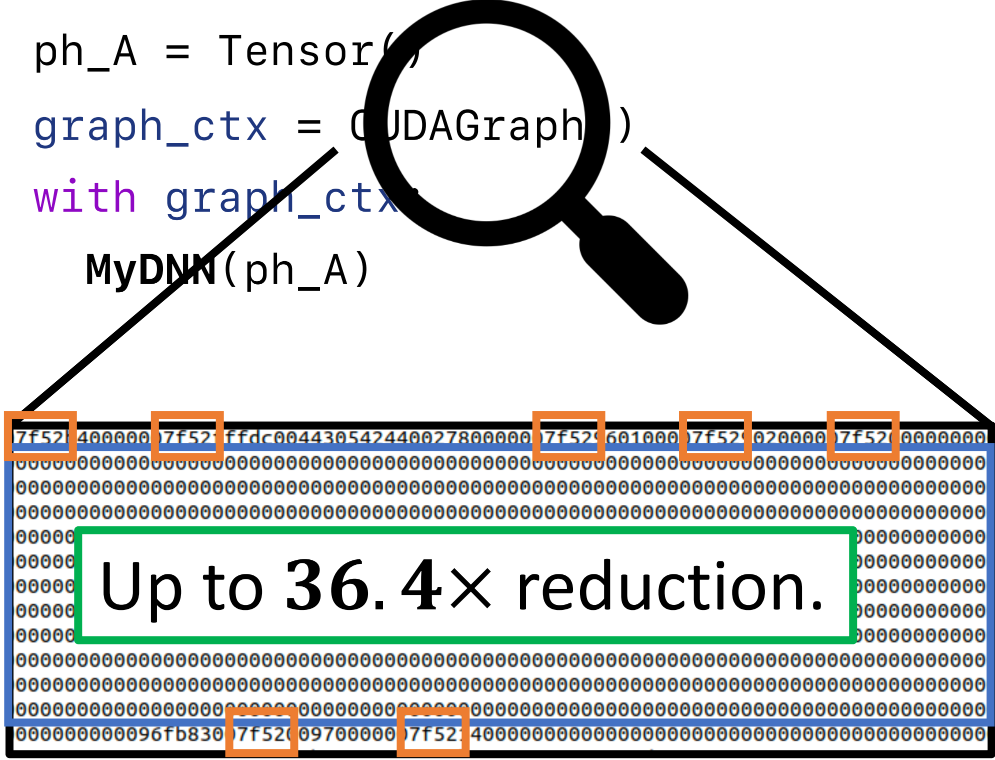
Grape's Key Ideas

② Metadata Compression

- The memory allocations of CUDA graphs have high sparsity and value redundancy and hence are highly compressible.

Speculations:

- Sparsity: Underutilize the reserved function argument spaces.
- Redundancy: Pointer values.



Grape🍇's Key Ideas

③ Predication Contexts

- Common data-dependent control flows can be replaced with predication contexts while fully preserving program semantics.

```
with Predicate(x):
  # do something
```

Grape🍇's Key Ideas

③ Predication Contexts

- Common data-dependent control flows can be replaced with predication contexts while fully preserving program semantics.

```
with Predicate(x):
  # do something
```

If x is true, all GPU operations within can proceed as normal.

Grape🍇's Key Ideas

③ Predication Contexts

- Common data-dependent control flows can be replaced with predication contexts while fully preserving program semantics.

```
with Predicate(x):
  # do something
```

Otherwise, all GPU operations within are nullified.

Grape🍇's Key Ideas

③ Predication Contexts

- Common data-dependent control flows can be replaced with predication contexts while fully preserving program semantics.

```
with Predicate(x):
  # do something
```

Implementation Details:

```
__global__ void cudaKernelInPyTorch(
  const float *const input,
  float *const output
) {
  // function body
}
```

Grape🍇's Key Ideas

③ Predication Contexts

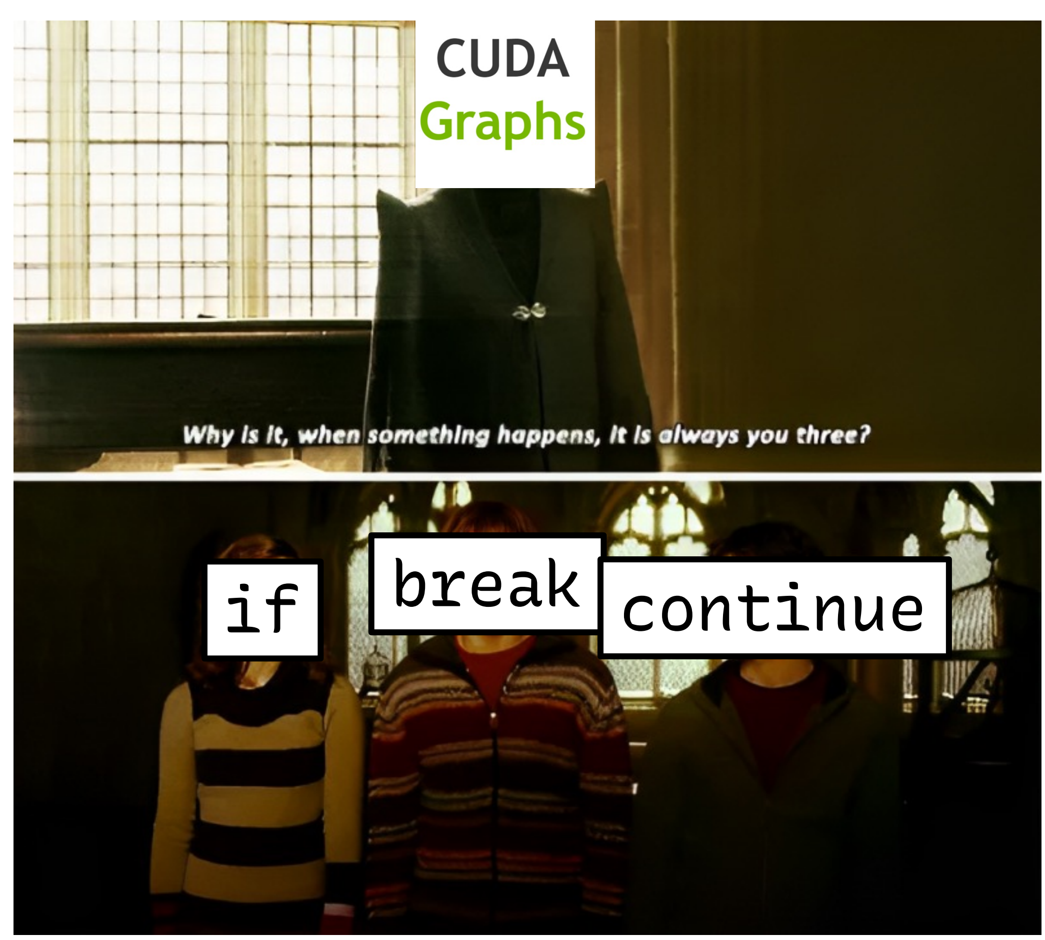
- Common data-dependent control flows can be replaced with predication contexts while fully preserving program semantics.

```
with Predicate(x):
  # do something
```

Implementation Details:

```
__global__ void cudaKernelInPyTorch(
  const float *const input,
  float *const output,
  const bool predicate
) {
  if (predicate) {
    // function body
  }
}
```

Grape🍇's Key Ideas



Grape🍇's Key Ideas

③ Predication Contexts

- Common data-dependent control flows can be replaced with predication contexts while fully preserving program semantics.

```
if x:
  # do something
else:
  # do other things
```

```
for ...:
  if x:
    break
  # do something
```

```
for ...:
  if x:
    continue
  # do something
```

Grape🍇's Key Ideas

③ Predication Contexts

- Common data-dependent control flows can be replaced with predication contexts while fully preserving program semantics.

Equivalent Forms:

```
if x:
  # do something
else:
  # do other things
```

```
with Predicate(x):
  # do something
with Predicate(!x):
  # do other things
```

```
for ...:
  if x:
    break
  # do something
```

```
flag = False
for ...:
  with Predicate(!flag):
    flag = x
  with Predicate(!flag):
    # do something
```

```
for ...:
  if x:
    continue
  # do something
```

```
for ...:
  with Predicate(!x):
    # do something
```

Grape🍇's Key Ideas

③ Predication Contexts

- Common data-dependent control flows can be replaced with predication contexts while fully preserving program semantics.

Strength: CUDA graph-optimizable

1.79× performance ↑↑↑ in GPT-2[1] beam search

[1] A. Radford et al. *GPT-2*. 2019

Equivalent Forms:

```
with Predicate(x):
  # do something
with Predicate(!x):
  # do other things
```

```
flag = False
for ...:
  with Predicate(!flag):
    flag = x
  with Predicate(!flag):
    # do something
```

```
for ...:
  with Predicate(!x):
    # do something
```

Outline

Background What are CUDA graphs?

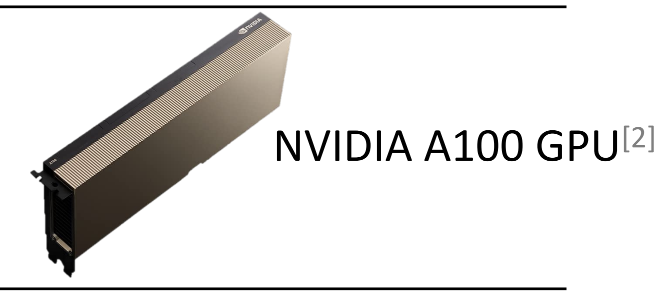
Challenges of using CUDA graphs

Key Ideas of Grape🍇 to resolve these challenges

Evaluation on state-of-the-art workloads

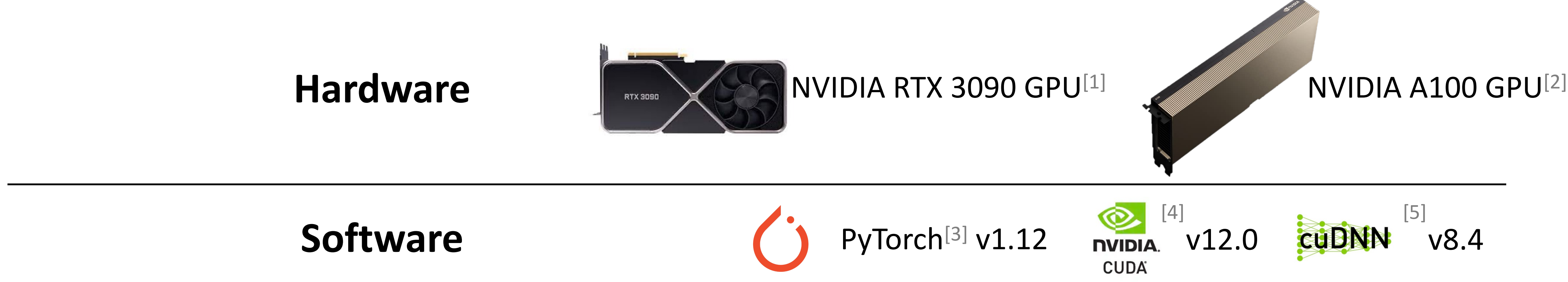
Evaluation

[1] https://www.nvidia.com/en-us/geforce/graphics-cards/30-series/rtx-3090-3090ti/
[2] https://www.nvidia.com/en-us/data-center/a100/

Hardware	NVIDIA RTX 3090 GPU[1]	 NVIDIA A100 GPU[2]

Evaluation

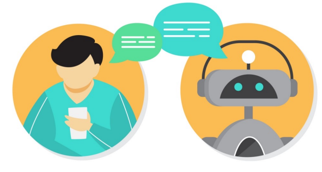
[1] https://www.nvidia.com/en-us/geforce/graphics-cards/30-series/rtx-3090-3090ti/
[2] https://www.nvidia.com/en-us/data-center/a100/



[3] A. Paszke et al. *PyTorch*. NeurIPS 2019
[4] https://docs.nvidia.com/cuda/archive/11.3.0/
[5] https://docs.nvidia.com/deeplearning/cudnn/developer-guide/index.html

Evaluation

[1] https://www.nvidia.com/en-us/geforce/graphics-cards/30-series/rtx-3090-3090ti/
[2] https://www.nvidia.com/en-us/data-center/a100/

Hardware	NVIDIA RTX 3090 GPU[1]	NVIDIA A100 GPU[2]
Software	PyTorch[3] v1.12	NVIDIA CUDA[4] v12.0 cuDNN[5] v8.4
Application	 GPT-2 137M, [6], GPT-J 6B, [7]	Wav2Vec2[8]
Number of Different Shapes:	1024	391

[3] A. Paszke et al. *PyTorch*. NeurIPS 2019
[4] https://docs.nvidia.com/cuda/archive/11.3.0/
[5] https://docs.nvidia.com/deeplearning/cudnn/developer-guide/index.html
[6] A. Radford et al. *GPT-2*. 2019
[7] W. Ben et al. *GPT-J*. 2020
[8] A. Baevski et al. *Wav2Vec2*. NeurIPS 2020

Evaluation

[1] https://www.nvidia.com/en-us/geforce/graphics-cards/30-series/rtx-3090-3090ti/
[2] https://www.nvidia.com/en-us/data-center/a100/

Hardware

NVIDIA RTX 3090 GPU[1]

NVIDIA A100 GPU[2]

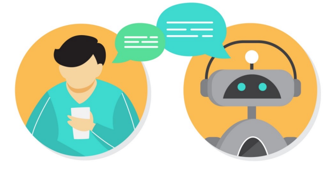
Software

PyTorch[3] v1.12

[4] v12.0

[5] v8.4

Application



GPT-2137M, [6], GPT-J6B, [7]

Wav2Vec2[8]

Baselines

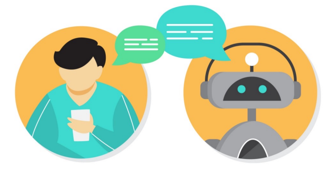
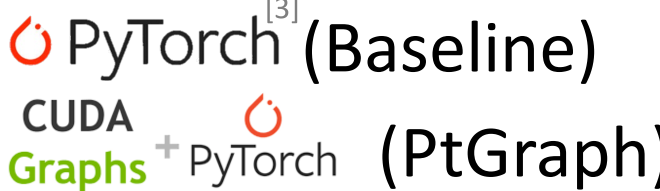
PyTorch[3] (Baseline)

CUDA Graphs + PyTorch (PtGraph)

[3] A. Paszke et al. *PyTorch*. NeurIPS 2019
[4] https://docs.nvidia.com/cuda/archive/11.3.0/
[5] https://docs.nvidia.com/deeplearning/cudnn/developer-guide/index.html
[6] A. Radford et al. *GPT-2*. 2019
[7] W. Ben et al. *GPT-J*. 2020
[8] A. Baevski et al. *Wav2Vec2*. NeurIPS 2020

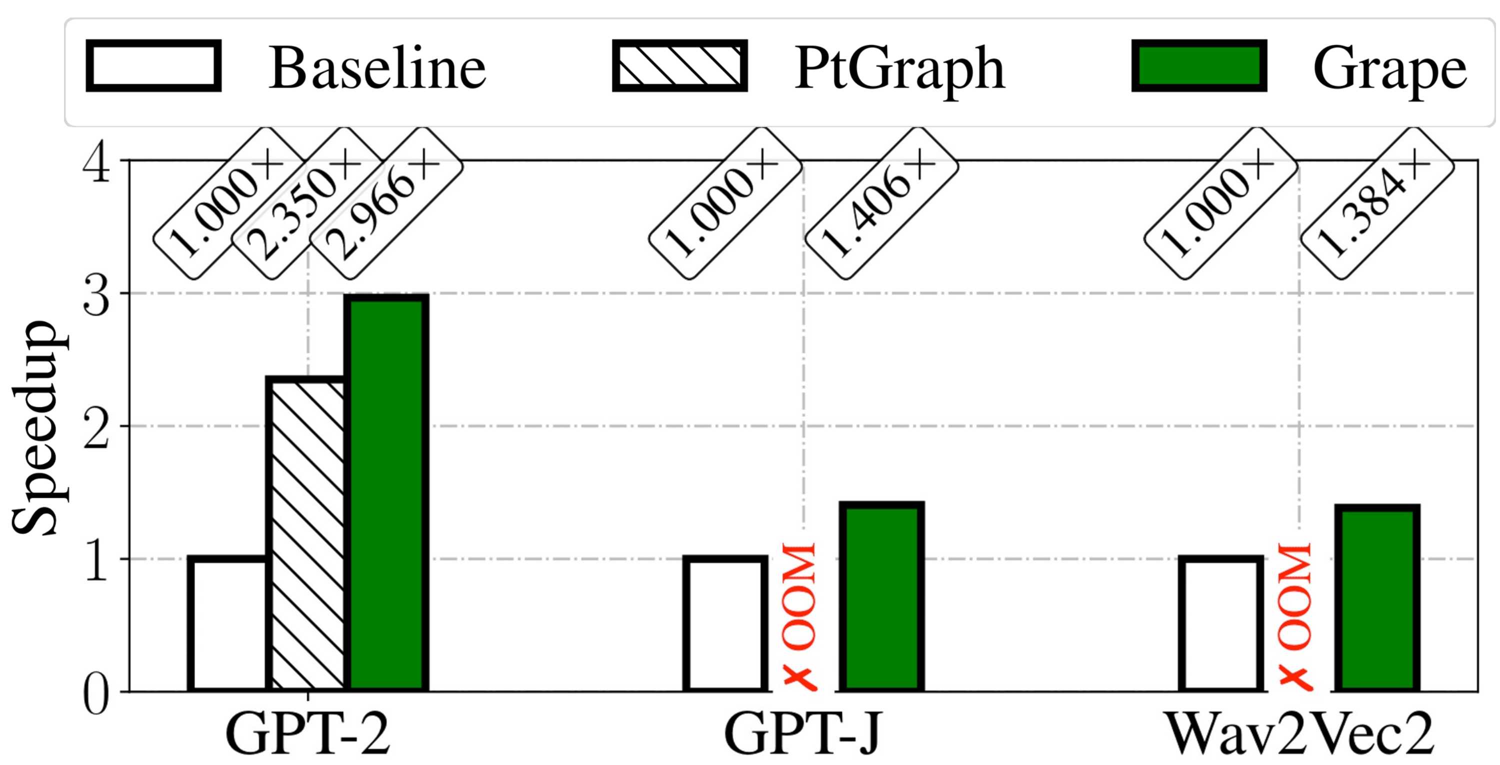
Evaluation

[1] https://www.nvidia.com/en-us/geforce/graphics-cards/30-series/rtx-3090-3090ti/
[2] https://www.nvidia.com/en-us/data-center/a100/

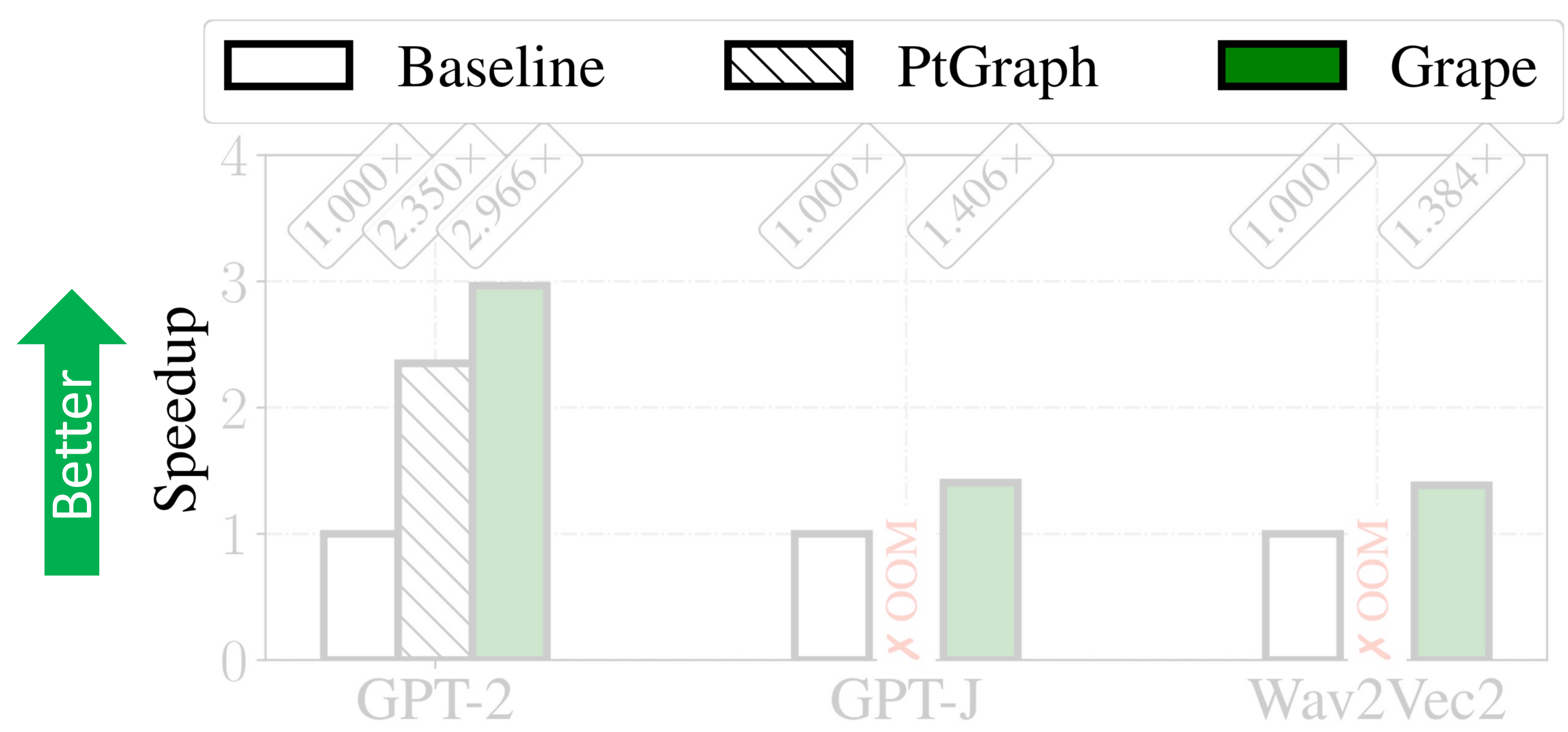
Hardware	NVIDIA RTX 3090 GPU[1] NVIDIA A100 GPU[2]
Software	PyTorch[3] v1.12 [4] v12.0 [5] v8.4
Application	 GPT-2[137M, [6]], GPT-J[6B, [7]] Wav2Vec2[8]
Baselines	

[3] A. Paszke et al. *PyTorch*. NeurIPS 2019
[4] https://docs.nvidia.com/cuda/archive/11.3.0/
[5] https://docs.nvidia.com/deeplearning/cudnn/developer-guide/index.html
[6] A. Radford et al. *GPT-2*. 2019
[7] W. Ben et al. *GPT-J*. 2020
[8] A. Baevski et al. *Wav2Vec2*. NeurIPS 2020

Performance vs. PyTorch/PtGraph



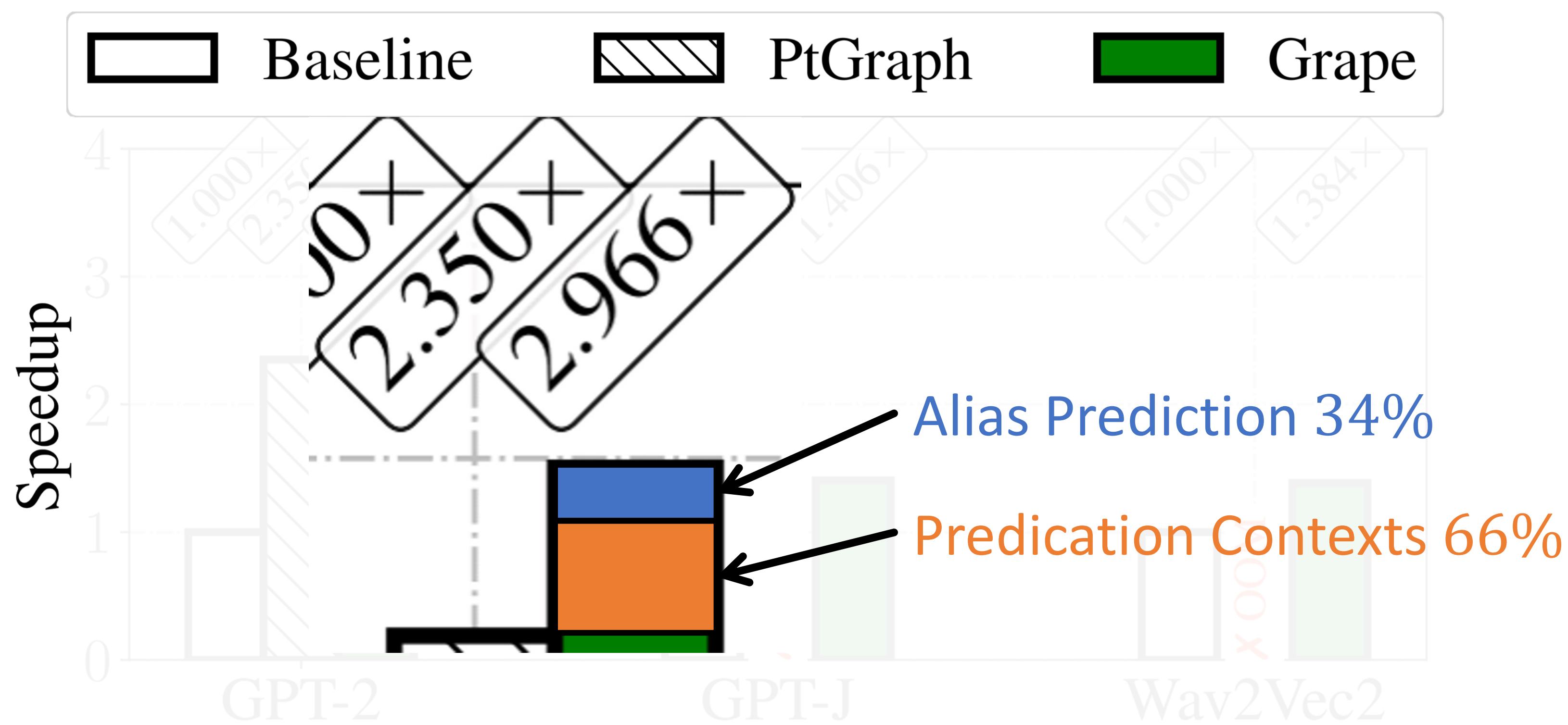
Performance vs. PyTorch/PtGraph



Performance vs. PyTorch/PtGraph

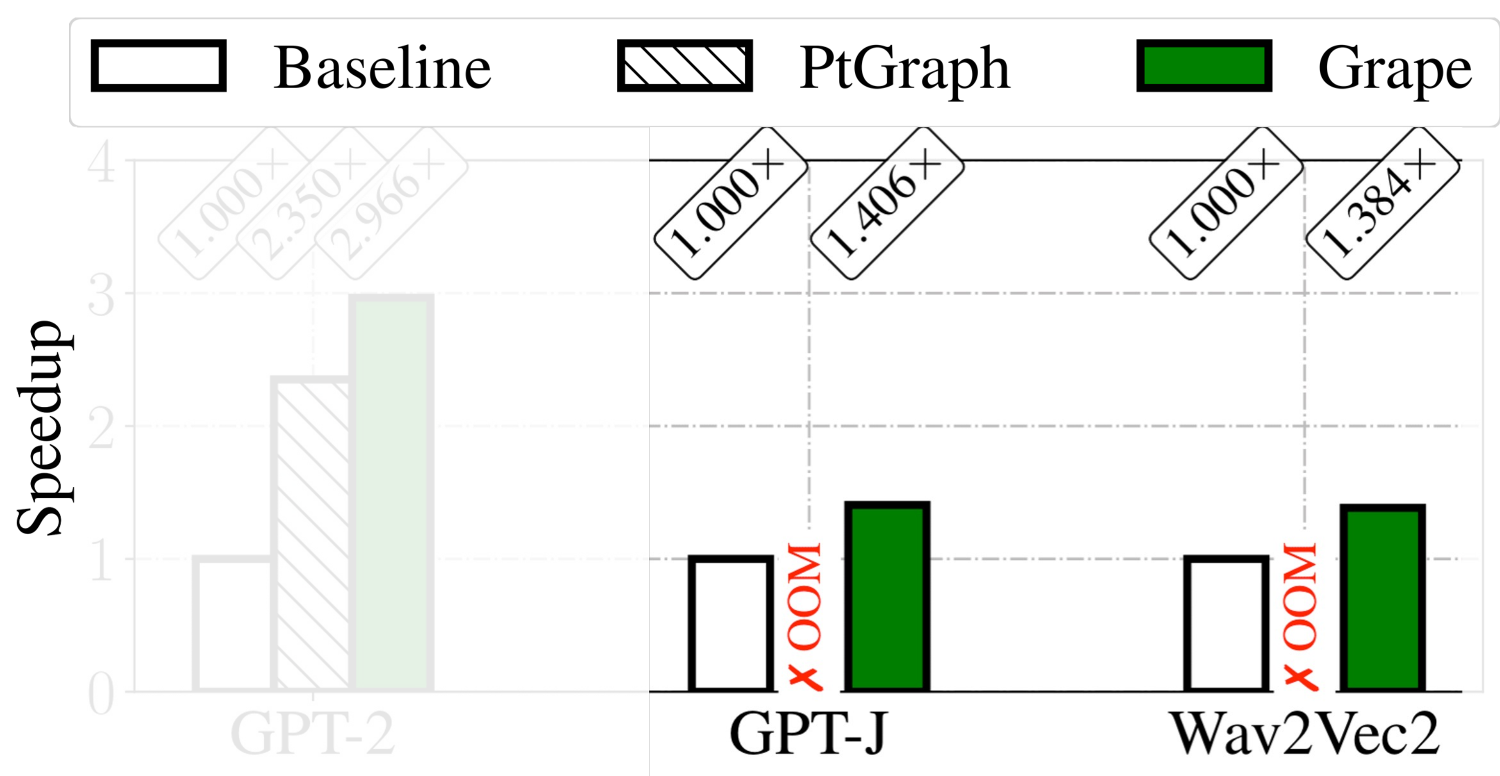
- **$2.97\times/1.26\times$** better than PyTorch/PtGraph on small workloads.

Performance vs. PyTorch/PtGraph



- **2.97×/1.26×** better than PyTorch/PtGraph on small workloads.

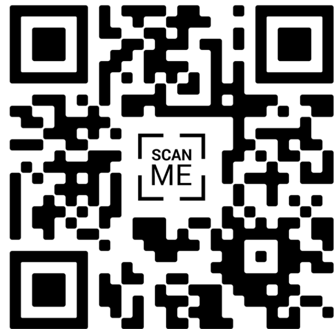
Performance vs. PyTorch/PtGraph



- $2.97\times/1.26\times$ better than PyTorch/PtGraph on small workloads.
- $1.41\times$ better than PyTorch on large workloads that are impractical for PtGraph.

Conclusion

- Challenges posed by CUDA graphs:
  - Extra data movements into placeholders.
  - Huge GPU memory consumption on dynamic-shape workloads.
  - No support for data-dependent control flows.
- Grape🍇 addresses those challenges with: ① Alias Prediction, ② Metadata Compression, and ③ Predication Contexts.
- Key Results:
  - On GPT-2, $1.26\times$ better performance than PtGraph.
  - On GPT-J and Wav2Vec2, up to $1.41\times$ better performance than PyTorch.



Grape🍇: Practical and Efficient Graph-based Executions for Dynamic Deep Neural Networks on GPUs

Bojian Zheng[1, 2, 3], Cody Hao Yu[4], Jie Wang[4], Yaoyao Ding[1, 2, 3], Yizhi Liu[4], Yida Wang[4], Gennady Pekhimenko[1, 2, 3]

1

2

3

4