



CONFERENCE

—NORTH AMERICA 2026—

Nested Graph Breaks

Reducing the cost of graph breaks in torch.compile

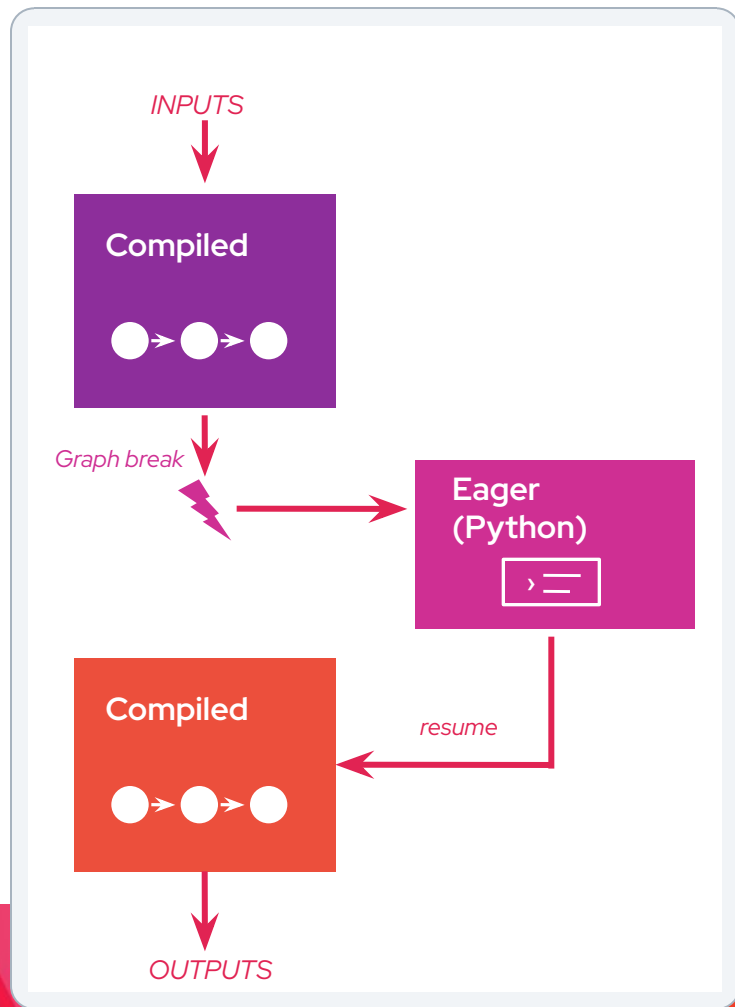
William Wen (Meta)

Graph breaks

The beloved `torch.compile` “feature”

Eager fallback for something untraceable

`torch.compile`’s advantage: don’t **error** or be **unsound** when something is untraceable, but **fallback to eager** (`fullgraph=False`)



How does a graph break work?

Original foo

```
@torch.compile
def foo(x):
    y = torch.sin(x)
    unsupported_function()
    return y + 1
```

Compiled foo (Python equivalent)

```
def compiled_foo(x):
    y = graph_1(x)
    unsupported_function()
    return resume_foo(x, y)
```

1 Compile what we have so far



How does a graph break work?

Original foo

```
@torch.compile
def foo(x):
    y = torch.sin(x)
    unsupported_function()
    return y + 1
```

Compiled foo (Python equivalent)

```
def compiled_foo(x):
    y = graph_1(x)
    unsupported_function()
    return resume_foo(x, y)
```

- 1 Compile what we have so far



- 2 Run unsupported code in eager



How does a graph break work?

Resume foo ("Python" equivalent)

```
def resume_foo(x, y):  
    goto resume  
    y = torch.sin(x)  
    unsupported_function()  
  
resume:  
    return y + 1
```

Compiled resume foo (Python equivalent)

```
def compiled_resume_foo(x, y):  
    z = graph_2(y)  
    return z
```

1 Compile what we have so far



2 Run unsupported code in eager



3 Resume tracing



The nested graph break problem

torch.compile could only resume from the top-level function!

This was due to early implementation decisions. A graph break deeper in the function call stack was handled by bubbling up to the top-level function and graph breaking at the next function call, which gets recursively traced.

A graph break N function calls deep resulted in

$O(N)$

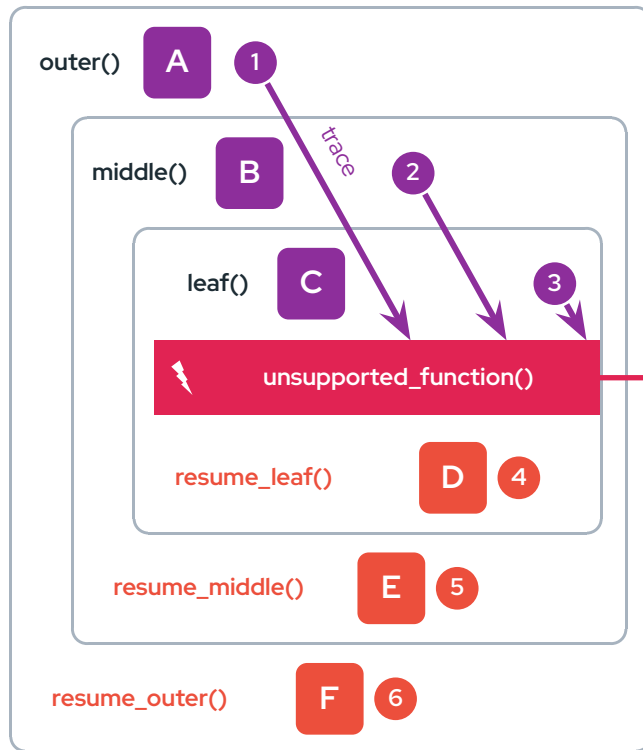
Duplicate graph breaks

$O(N^2)$

Functions traced

$O(N)$

Graphs produced



Tracing attempt	What is traced	What is compiled
1	A B C	A
2	B C	B
3	C	C
4	D	D
5	E	E
6	F	F

Nested graph break support

torch.compile can now resume arbitrarily deep!

Capture work across all nested frames. After the eager call, trace the nested resume chain together.

Now, a graph break N function calls deep results in

$O(1)$

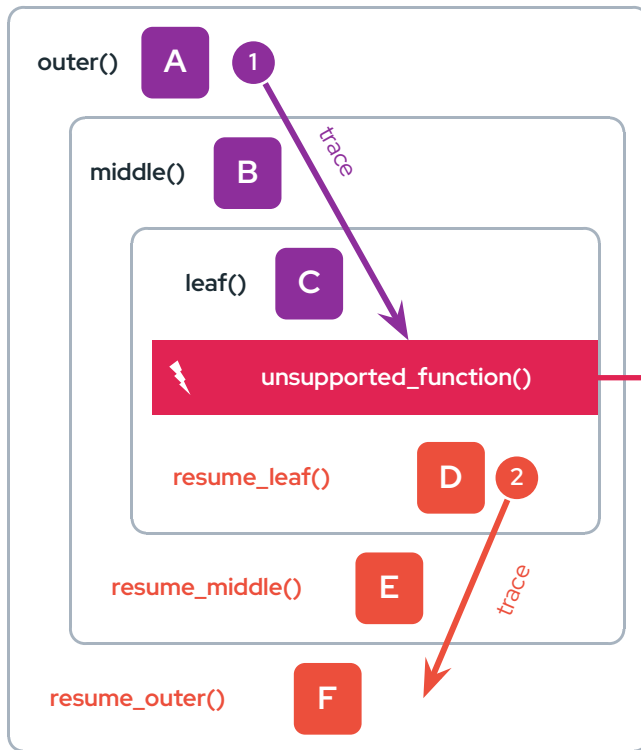
Duplicate graph breaks

$O(N)$

Functions traced

$O(1)$

Graphs produced



Tracing attempt	What is traced	What is compiled
1	A B C	A B C
	⚡ unsupported_function()	
2	D E F	D E F

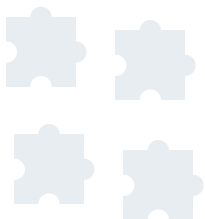
How?

```
torch._dynamo.config.nested_graph_breaks = True
```

Benefits of nested graph breaks

Larger graphs

Before



After



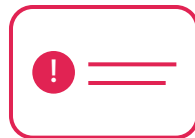
Larger graphs create more opportunities to fuse work.

Easier debugging

Before



After



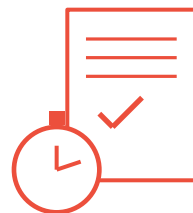
Find the real graph break without duplicate reports.

Less repeated tracing

Before



After

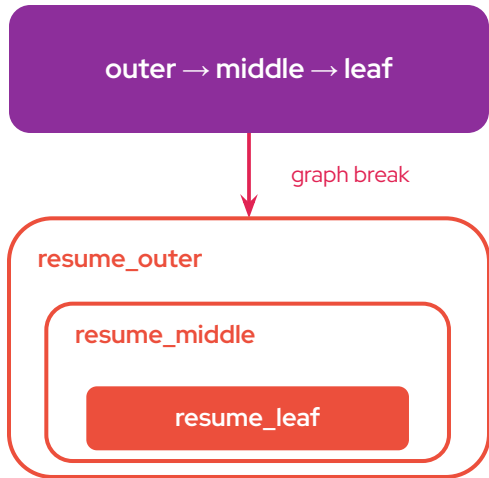


Avoid tracing the same code again.

Why nested graph breaks took work

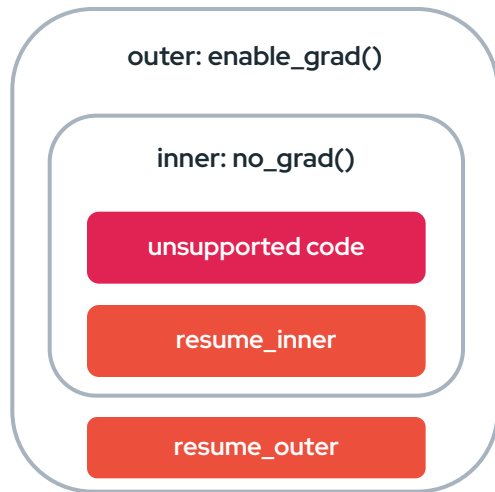
3 implementation highlights

1. Nested resume creation



Create a resume for every function.
Resumes call each other to
recreate the call stack.

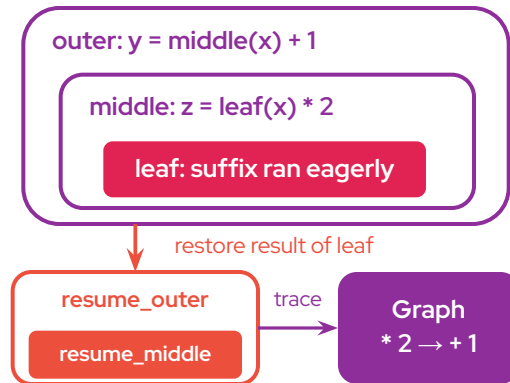
2. Restore contexts



Restore contexts for unsupported
code and every resume function.

3. Unresumable graph breaks

We trace a meaningful graph to compile,
but we are unable to resume tracing
the rest of the leaf function.



Compile the prefix; skip leaf tracing.
Trace caller resumes into a new graph.

Takeaways

Try it out! Set `nested_graph_breaks = True`

The feature is still experimental, but you can turn on nested graph break support on the PyTorch nightlies with

```
torch._dynamo.config.nested_graph_breaks = True
```

Report any issues to the PyTorch GitHub repo!

