



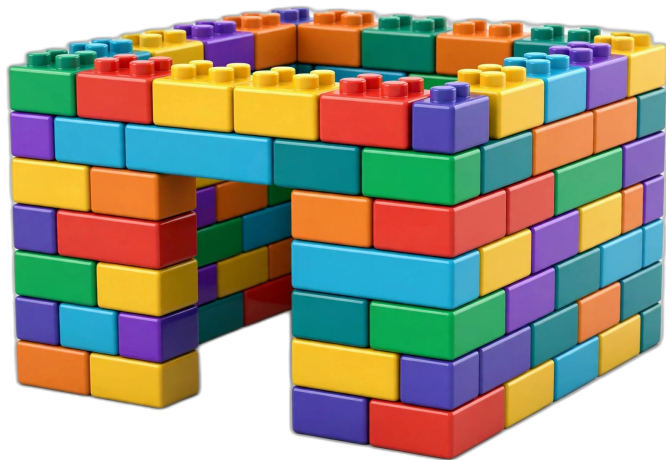
CONFERENCE

—NORTH AMERICA 2026—

Static Tensor Shapes with Pyrefly

Avik Chaudhuri and Steven Troxler
Programming Foundations, Meta

Programming often feels like building with lego bricks.



Programming often feels like building with lego bricks.

One of the main benefits of
a *static type system*
is that it gives you

Programming often feels like building with lego bricks.



One of the main benefits of
a *static type system*
is that it gives you

**instantaneous feedback on how
these lego bricks fit together.**

Programming often feels like building with lego bricks.



One of the main benefits of
a *static type system*
is that it gives you

instantaneous feedback on how
these lego bricks fit together.

Say you want to call a function.

What kind of inputs does it expect?

What kind of outputs does it return?

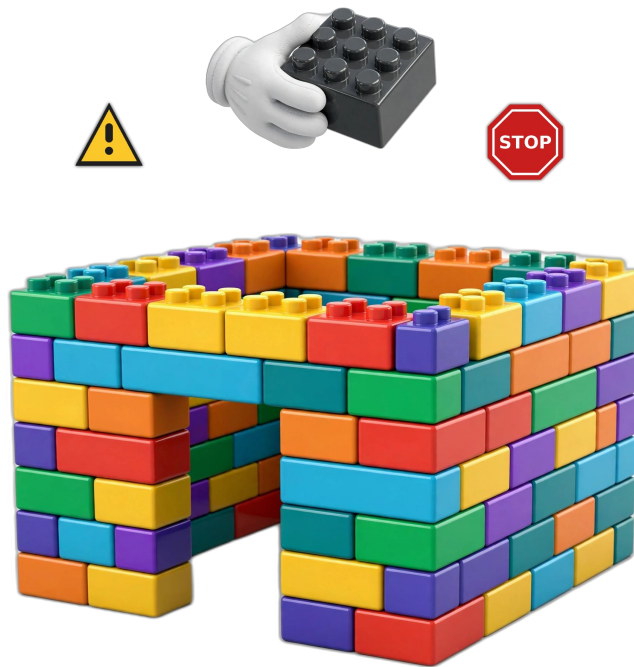
Programming often feels like building with lego bricks.

Another great benefit of
a *static type system*
is that it

Programming often feels like building with lego bricks.

Another great benefit of
a *static type system*
is that it

**automatically checks whether you
composed these lego bricks
correctly.**

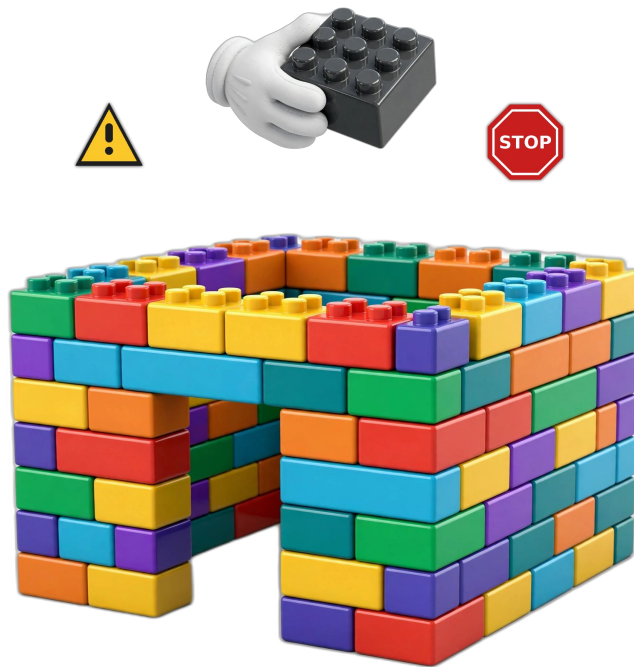


Programming often feels like building with lego bricks.

Another great benefit of
a *static type system*
is that it

automatically checks whether you
composed these lego bricks
correctly.

Because if you didn't?
You'd get a runtime error.



Programming often feels like building with lego bricks.

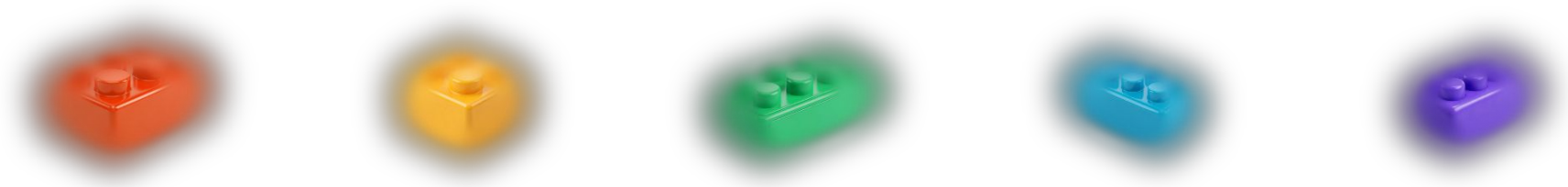
Unfortunately, for a large class of modern programs that manipulate tensors

you get practically no benefit from a traditional type system.

Programming often feels like building with lego bricks.

Unfortunately, for a large class of modern programs that manipulate tensors

you get practically no benefit from a traditional type system.



Because all they tell you is?

Something is a tensor.

Both **developers** and
coding agents have
to **carry shapes in**
their head.

```
def forward(self, query: Tensor, context: Tensor) -> Tensor:
    batch, query_length, _ = query.shape
    context_length = context.shape[1]

    q = self.query(query).reshape(
        batch, query_length, self.num_heads, self.head_dim
    ).transpose(1, 2)
    k = self.key(context).reshape(
        batch, context_length, self.num_heads, self.head_dim
    ).transpose(1, 2)
    v = self.value(context).reshape(
        batch, context_length, self.num_heads, self.head_dim
    ).transpose(1, 2)

    scores = q @ k.transpose(1, 2) / self.head_dim**0.5
    weights = F.softmax(scores, dim=-1)
    attended = weights @ v
    merged = attended.transpose(1, 2).reshape(
        batch, query_length, self.model_width
    )
    return self.output(merged)
```



This is a **huge cognitive burden.**

```
def forward(self, query: Tensor, context: Tensor) -> Tensor:
    batch, query_length, _ = query.shape
    context_length = context.shape[1]

    q = self.query(query).reshape(
        batch, query_length, self.num_heads, self.head_dim
    ).transpose(1, 2)
    k = self.key(context).reshape(
        batch, context_length, self.num_heads, self.head_dim
    ).transpose(1, 2)
    v = self.value(context).reshape(
        batch, context_length, self.num_heads, self.head_dim
    ).transpose(1, 2)
```



This is a **huge cognitive burden**.

```
RuntimeError                                Traceback (most recent call last)
Cell In[7], line 1
----> 1 att(query, context)

File ~/code/shape-types-oct-2026/.venv/lib/python3.13/site-packages/torch/nn/modules/module.py:1778, in Module._wrapped_call_impl(self, args, **kwargs)
   1776     return self._compiled_call_impl(*args, **kwargs) # type: ignore[misc]
   1777 else:
-> 1778     return self._call_impl(*args, **kwargs)

File ~/code/shape-types-oct-2026/.venv/lib/python3.13/site-packages/torch/nn/modules/module.py:1789, in Module._call_impl(self, args, **kwargs)
   1784 # If we don't have any hooks, we want to skip the rest of the logic in
   1785 # this function, and just call forward.
   1786 if not (self._backward_hooks or self._backward_pre_hooks or self._forward_hooks or self._forward_pre_hooks
   1787         or _global_backward_pre_hooks or _global_backward_hooks
   1788         or _global_forward_hooks or _global_forward_pre_hooks):
-> 1789     return forward_call(*args, **kwargs)
   1791 result = self._forward_call(*args, **kwargs)
   1792 called_always_called_hooks = set()

Cell In[2], line 38, in CrossAttention.forward(self, query, context)
   31 k = self.key(context).reshape(
   32     batch, context_length, self.num_heads, self.head_dim
   33 ).transpose(1, 2)
   34 v = self.value(context).reshape(
   35     batch, context_length, self.num_heads, self.head_dim
   36 ).transpose(1, 2)
----> 38 scores = self.attention(q, k, v)
   39 weights = F.softmax(scores, dim=-1)
   40 attended = weights @ v

RuntimeError: The size of tensor a (4) must match the size of tensor b (5) at non-singleton dimension 1
```



And there is a **huge risk of breaking code** as **changes are made** (and realizing only much later).

Pyrefly *statically* tracks your
Tensor Shapes!



Pyrefly *statically* tracks your
Tensor Shapes!



Pyrefly *statically* tracks your
Tensor Shapes!



Motivated by PyTorch. Inspired by PyTorch compiler.



At Meta, we have a lot of experience we **PyTorch users**, and we have seen them **struggle with shapes**



We know that the **PyTorch compiler** internally tracks shapes for optimization, by tracing a program **"just-in-time"**



In **Pyrefly**, we do a similar analysis, but **"ahead-of-time"**

On most typical
PyTorch models, we
can get **end-to-end
shape coverage** with
just a **few annotations**
at the interfaces.

```
class AutoEncoder[InDim: IntVar, H1: IntVar, H2: IntVar](nn.Module):  
  
    def __init__(  
        self,  
        in_dim: Int[InDim],  
        h1: Int[H1],  
        h2: Int[H2],  
        act_fn: ShapePreservingActivation = nn.SELU,  
        dp_drop_prob: float = 0.0,  
    ) -> None:  
        super().__init__()  
        self.act: ELU | LeakyReLU | ReLU | SELU | Sigmoid | ... = act_fn()  
        self.dp_drop_prob = dp_drop_prob  
        if dp_drop_prob > 0:  
            self.drop = nn.Dropout(dp_drop_prob)  
  
        # Encoder layers  
        self.enc1: Linear[InDim, H1] = nn.Linear(in_dim, h1)  
        self.enc2: Linear[H1, H2] = nn.Linear(h1, h2)  
  
        # Decoder layers (mirror of encoder)  
        self.dec1: Linear[H2, H1] = nn.Linear(h2, h1)  
        self.dec2: Linear[H1, InDim] = nn.Linear(h1, in_dim)  
  
    def encode[B: IntVar](self, x: Tensor[[B, InDim]]) -> Tensor[[B, H2]]:  
        h: Tensor[[B, H1]] = self.act(self.enc1(x))  
        z: Tensor[[B, H2]] = self.act(self.enc2(h))  
        if self.dp_drop_prob > 0:  
            z: Tensor[[B, H2]] = self.drop(z)  
        return z
```

```
def forward[B: IntVar, Q: IntVar, K: IntVar](
    self, query: Tensor[[B, Q, D]], context: Tensor[[B, K, D]]
) -> Tensor[[B, Q, D]]:
    batch: Int[B], query_length: Int[Q], _: Int[D] = query.shape
    context_length: Int[K] = context.shape[1]

    q: Tensor[[B, NHead, Q, D // NHead]] = self.query(query).reshape(
        batch, query_length, self.num_heads, self.head_dim
    ).transpose(1, 2)
    k: Tensor[[B, NHead, K, D // NHead]] = self.key(context).reshape(
        batch, context_length, self.num_heads, self.head_dim
    ).transpose(1, 2)
    v: Tensor[[B, NHead, K, D // NHead]] = self.value(context).reshape(
        batch, context_length, self.num_heads, self.head_dim
    ).transpose(1, 2)

    scores: Tensor = q @ k.transpose(1, 2) / self.head_dim**0.5
    weights: Tensor = F.softmax(scores, dim=-1)
    attended: Tensor = weights @ v
    merged: Tensor[[B, Q, D]] = attended.transpose(1, 2).reshape(
        batch, query_length, self.model_width
    )
    return self.output(merged)
```



We can check for bugs.

```
def forward[B: IntVar, Q: IntVar, K: IntVar](
  self, query: Tensor[[B, Q, D]], context: Tensor[[B, K, D]]
) -> Tensor[[B, Q, D]]:
  batch: Int[B], query_length: Int[Q], _: Int[D] = query.shape
  context_length: Int[K] = context.shape[1]

  q: Tensor[[B, NHead, Q, D // NHead]] = self.query(query).reshape(
    batch, query_length, self.num_heads, self.head_dim
  ).transpose(1, 2)
  k: Tensor[[B, NHead, K, D // NHead]] = self.key(context).reshape(
    batch, context_length, self.num_heads, self.head_dim
  ).transpose(1, 2)
  v: Tensor[[B, NHead, K, D // NHead]] = self.value(context).reshape(
    batch, context_length, self.num_heads, self.head_dim
  ).transpose(1, 2)

  scores: Tensor = q @ k.transpose(1, 2) / self.head_dim**0.5
  weights: Tensor = F.softmax(scores, dim=-1)
  attended: Tensor = weights @ v
  merged: Tensor[[B, Q, D]] = attended.transpose(1, 2).reshape(
    batch, query_length, self.model_width
  )
  return self.output(merged)
```



We can **check for bugs**.



And we can **show inferred shapes** through the IDE.

Demo



The core subsystem
that tracks shapes is
separated from a
pluggable library
specification.

```

@type_shape_dsl_function
def tile_shape(shape: IntTuple, repeats: IntTuple) -> IntTuple:
    if len(repeats) >= len(shape):
        return repeat_shape(shape, repeats)
    if any(dsl.is_concrete_int(repeat) and repeat < 0 for repeat in repeats):
        return dsl.Invalid("repeat dimensions must be non-negative")
    extra: int = len(shape) - len(repeats)
    return dsl.IntTuple(
        (
            shape[index] if index < extra else shape[index] * repeats[index - extra]
            for index in range(len(shape))
        )
    )

def tile[Shape: IntTuple, Repeats: IntTuple](
    self: Tensor[Shape], dims: Repeats
) -> Tensor[tile_shape(Shape, Repeats)]:
    """Tile tensor. Shape inference via type-level DSL."""
    ...

```

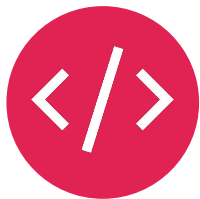


Define **how the APIs**
exposed by a library
transform shapes.



This can be done **purely in**
Python!

GPU kernels in Python? Static types for tensor layouts!



We now have **Python DSLs for GPU kernel-authoring**, e.g., Triton and CuTe DSL, and they care about **tensor layouts**.



Bugs in layout composition can go undetected and cause **hardware crashes** or produce **incorrect results**.



We are now designing a **type system for tensor layouts**.

Try it out!

<https://pyrefly.org/en/docs/tensor-shapes/>

Why tensor shapes?

When you write PyTorch code, the hardest thing to keep track of is shapes in non-trivial ways, and shape mistakes don't always crash. With Pyrefly's tensor shape support, you get **automatic inlay type** so you can see exactly what shape every intermediate tensor has via statements.

Here's the same NanoGPT forward method, without and with tensor

Without tensor shapes With tensor shapes

```
b : Int[B], t : Int[T] = idx.size()
assert t <= self.config.block_size, (
    f"Cannot forward sequence of length {t}, block size is only {self.config.block_size}"
)
pos : Tensor[[T]] = torch.arange(0, t, dtype=torch.long, device=device)

# forward the GPT model itself
tok_emb : Tensor[[B, T, NEmbedding]] = self.transformer.wte(idx)
pos_emb : Tensor[[T, NEmbedding]] = self.transformer.wpe(pos) # pos_emb is [T, NEmbedding]
tok_pos_emb : Tensor[[B, T, NEmbedding]] = tok_emb + pos_emb
x : Tensor[[B, T, NEmbedding]] = self.transformer.drop(tok_pos_emb)
for block in self.transformer.h:
    x : Tensor[[B, T, NEmbedding]] = block(x)

x : Tensor[[B, T, NEmbedding]] = self.transformer.ln_f(x)
```



A fast type checker and language server for Python with powerful IDE features

```
$ pip install pyrefly && pyrefly init
```

Get VSCode Extension

Get OpenVSX Extension

Tensor Shapes > Getting Started

Getting Started with Pyrefly Tensor Shapes

This page walks you through configuring Pyrefly for tensor shape checking and getting shape-annotated code running.

Installation

Pyrefly publishes shape-aware stub packages for these libraries:

- **PyTorch:** `pyrefly-torch-stubs`
- **JAX:** `jax-pyrefly-stubs` (not `pyrefly-jax-stubs`)
- **NumPy:** `pyrefly-numpy-stubs`
- **einops:** `pyrefly-einops-stubs` (currently for PyTorch tensors)

Tensor Shapes > Tutorials > 1. Your First Port

Tutorial 1: Your First Port

In this tutorial, you'll add tensor shape annotations to a simple multi-layer perceptron (MLP) model. By the end, you'll understand `Int`, `Tensor[...]`, class-level type parameters, and method-level type parameters.



TIP

Try it yourself

Open [this example in the Pyrefly sandbox](#) — Try porting a simple MLP with class-level and method-level type parameters.

Agent Skill for Tensor Shape Porting

Pyrefly includes a skill for **Claude Code** that can automatically port PyTorch models to use tensor shape annotations. The skill runs a structured, multi-step workflow — auditing ops, inventorying parameters, probing shapes with `reveal_type`, containing tracking boundaries, and verifying the result — that produces an annotated model and a report of what could not be tracked.

Where the skill lives

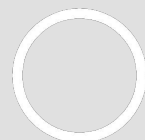
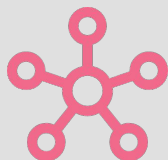
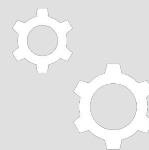
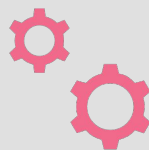
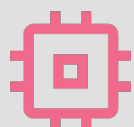
The skill is a directory in the Pyrefly source tree:

```
tensor-shapes/skills/add-shape-types-to-torch-model/
```


Backup icons



PyTorch



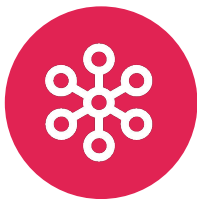
The AI Infrastructure Challenge Has Evolved



From Static LLMs to Agentic AI

Requiring new training paradigms

*Reinforcement learning, post-training,
online & async RL,
Large-scale training.*



From Single GPU to 100K+ Clusters

Frontier models require
distributed systems that span
thousands of GPUs with fault
tolerance built-in



From One Vendor to Many

Multi-accelerator reality demands
hardware-portable software



Usability over Performance

Flexibility first.

We optimize without restricting
what you can build.



Simple Over Easy

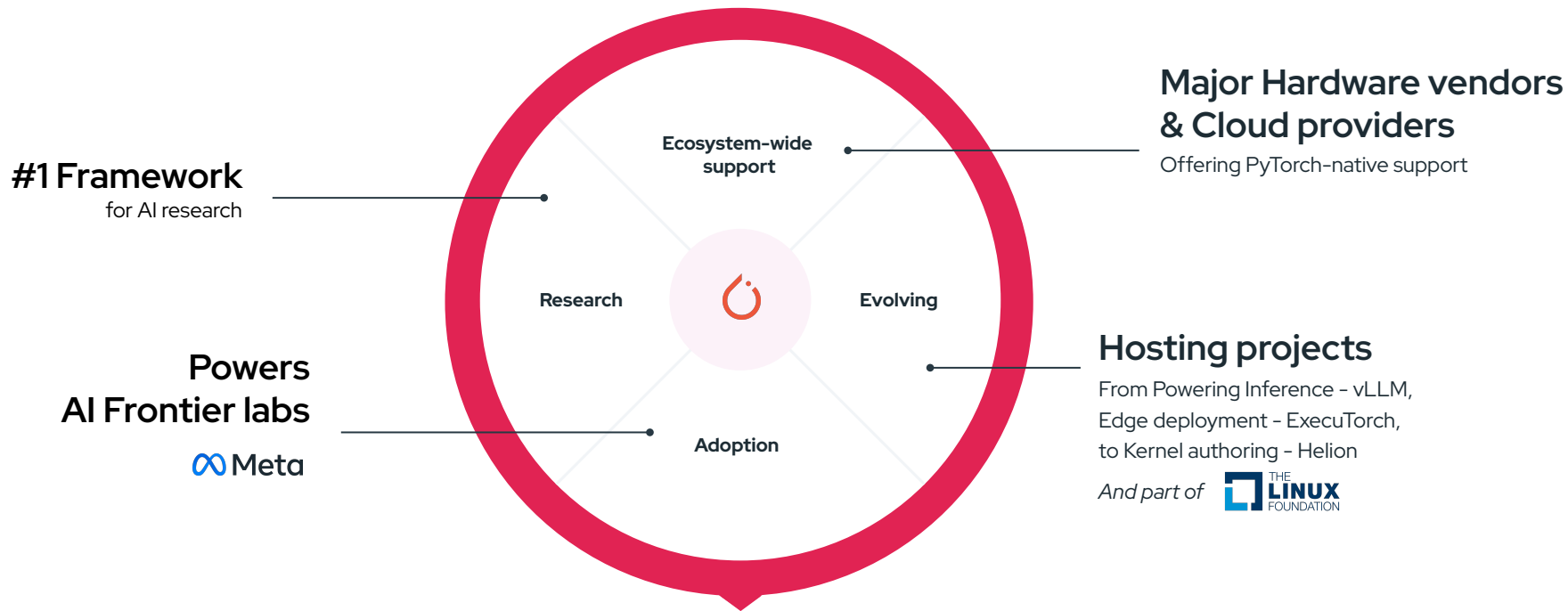
Explicit, debuggable, hackable.

No black-box magic.



Pythonic

Write AI like you think—imperative,
debuggable, Pythonic



Helion: Kernel authoring DSL

Write once, run fast everywhere

Built for Developer Productivity

PyTorch syntax you know.

4x less code than Triton.

Improved performance.

Intelligent Autotuning

Automatically explores configuration space to identify optimal kernel parameters.

Hardware heterogeneity

Growing External Ecosystem with industry contributions
incl. vLLM integration.

10x

Faster Dev Time

95%

Hand-Tuned Perf

HIGH LEVEL
Hardware-agnostic

Abstraction level

PyTorch

Helion

Triton

TLX / Gluon / CUDA

PTX

LOW LEVEL
Hardware-specific

ExecuTorch: AI at the Edge

Run PyTorch models on devices

Physical AI

For phones, glasses, wearables,
personal computers...

Lightweight & Portable

Minimal runtime footprint
design for resource-constrained environments

GA and Production ready

Powers all Meta Technologies on-device inference.
External adoption and ecosystem integration.

12+

Hardware backends

Incl. CUDA, Cortex-M MCU, Cortex-A CPU, Vulkan,
Metal, Qualcomm NPU, Arm Ethos-U



CONFERENCE

—NORTH AMERICA 2026—



Our complete Stack

From kernels to agents, a unified Production Workflow



Built for Scale

Architected to handle massive workloads, supporting thousands of GPUs



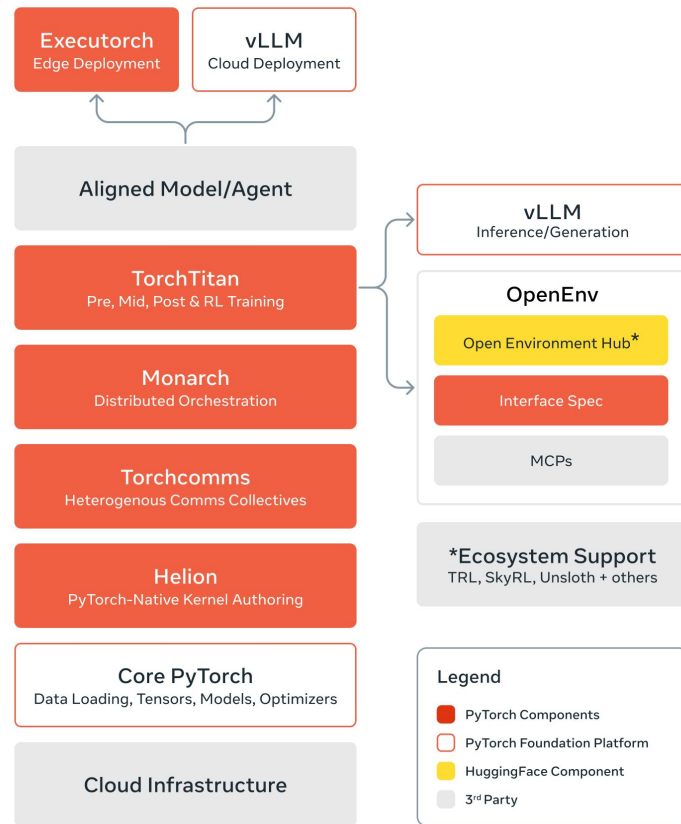
PyTorch-Native

Deeply integrated into the PyTorch ecosystem. Writing scalable AI code stays pythonic



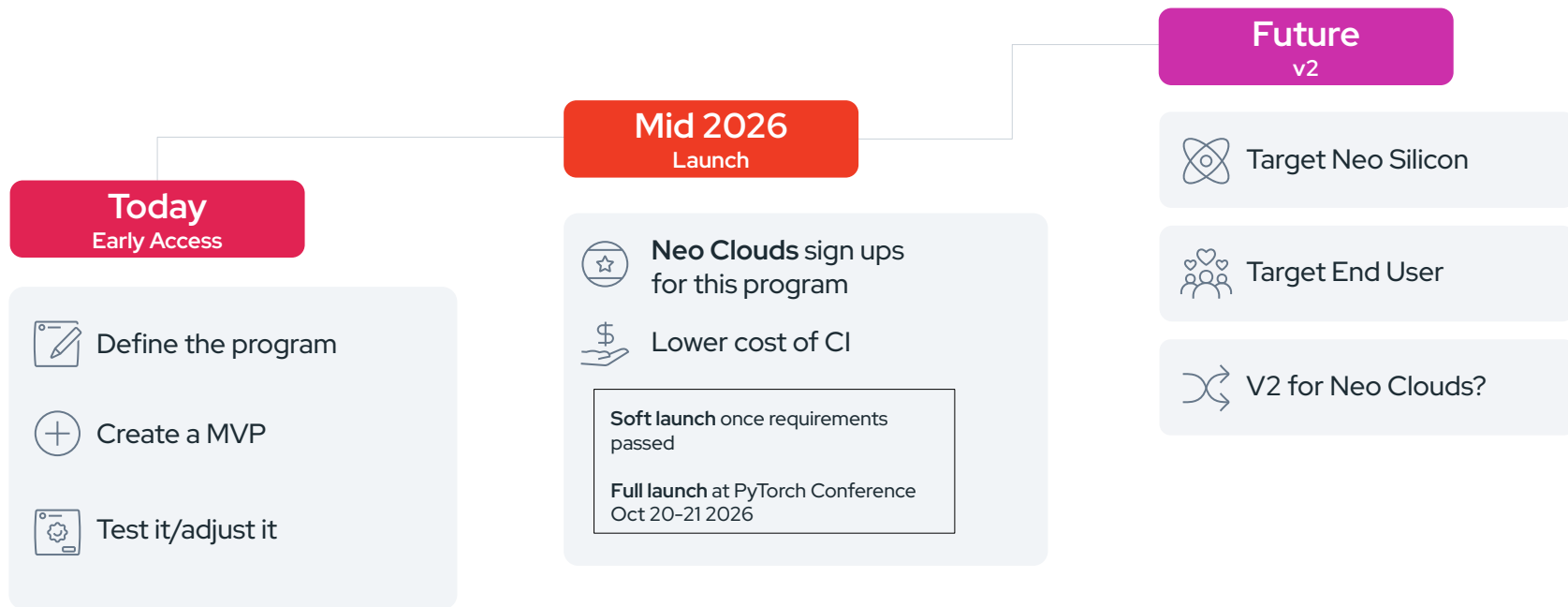
Heterogeneous Hardware Ready

The stack is built for a diverse hardware future, spanning edge devices, data centers.



PyTorch Verified

For Meta and PyTorch Foundation



Keeping the momentum

With 360 strategy on all dimensions

		Project 0.X <i>Early 2025</i>	Project 1.0 <i>April 2026 - PTC Europe</i>	Project 1.X <i>To be defined</i>
Open Governance		Meta PyTorch	PyTorch Foundation Platform Project Core maintainers: 2 org.	Other Foundation? Core maintainers: 5+ org. <i>Nvidia, Amd, Intel...</i>
Developers target/adoption		General adoption	Frontier labs?	AI-native/adopting companies?
Ecosystem Integration		Lorem ipsum	Lorem ipsum	Lorem ipsum
GTM	In-person	PyTorch events	Major Dev Tech events Hackathon	Lorem ipsum
	Online	Blog posts - webinars	Performance benchmark recurring reports?	Research papers?
	Community	GitHub	Lorem ipsum	
Developer experience <i>How to lower the barrier?</i>			Playground? Libraries?	Lorem ipsum

Headline to illustrate where we are heading

Allows us to paint a broad picture



Innovation topic #1

State the objective of what this innovation aims to achieve over time



Innovation topic #2

State the objective of what this innovation aims to achieve over time

Now

STARTING
QUARTER

Name the initiative that is underway today and ongoing

Name the initiative that is underway today and ongoing

Next

LAUNCHING
QUARTER

Name the initiative that
is in the pipeline

Name the initiative that will
be launched at a later date

Code example

For when you have to show some code

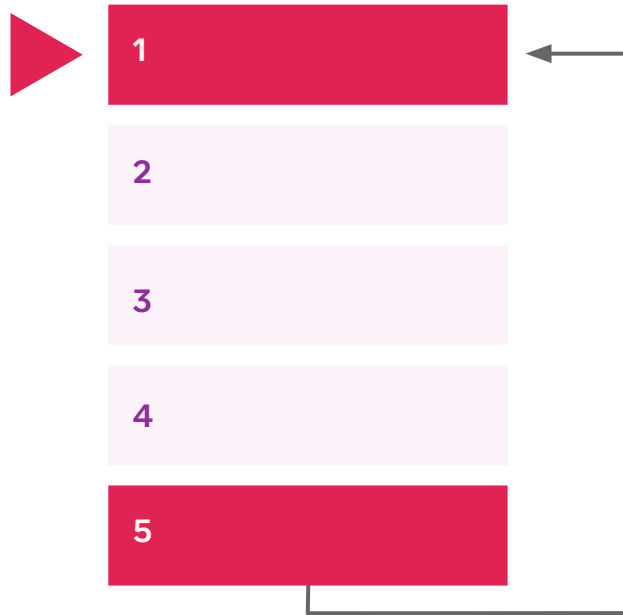
Function Foo:

```
buffer[bufferIdx] = hash("Foo")  
bufferIdx = (bufferIdx + 1) % SIZE  
// Rest of the function
```

Function Bar:

```
buffer[bufferIdx] = hash("Foo")  
bufferIdx = (bufferIdx + 1) % SIZE  
// Rest of the function
```

EXAMPLE



HEADER

Lorem ipsum dolor sit
amet, vel unum dignissim
contentiones et. Ei pro
integre apeirian honestatis.



Ne tempor audiam gubergren his, probo
volumus commune ex pri. Ei solum choro.



Ne tempor audiam gubergren his, probo
volumus commune ex pri. Ei solum choro.



Ne tempor audiam gubergren his, probo
volumus commune ex pri. Ei solum choro.



Ne tempor audiam gubergren his, probo
volumus commune ex pri. Ei solum choro.

Information-heavy layouts for read-only decks.

Lorem ipsum dolor sit amet, consectetur adipiscing elit.

These slides are designed for read-only decks. These layouts are not ideal for speaker presentations. For presentations remember to consider your audience's experience. Summarize visual content and use the 'presenter notes' feature.

At vero eos et accusamus et iusto odio dignissimos.

voluptatum deleniti atque corrupti quos dolores et quas molestias excepturi sint occaecati cupiditate non provident, similique sunt in culpa qui officia deserunt mollitia animi, id est laborum et dolorum fuga. Et harum quidem rerum facilis est et expedita distinctio. Nam libero tempore, cum soluta nobis est eligendi optio cumque.

Sed ut perspiciatis unde omnis iste natus error sit voluptatem.

Sed ut perspiciatis unde omnis iste natus error sit voluptatem accusantium doloremque laudantium, totam rem aperiam, eaque ipsa quae ab illo inventore veritatis et quasi architecto beatae vitae dicta sunt explicabo. Nemo enim ipsam voluptatem quia voluptas sit aspernatur aut odit aut fugit.

Takeaways

Takeaway item 01

Missio est scriptor Meta ad populum potestatem aedificare civitatem et propius.

Takeaway item 02

Missio est scriptor Meta ad populum potestatem aedificare civitatem et propius.

Takeaway item 03

Missio est scriptor Meta ad populum potestatem aedificare civitatem et propius.

pytorch.org

Get started!



github.com/pytorch

 PyTorch

CONFERENCE

—NORTH AMERICA 2026—