



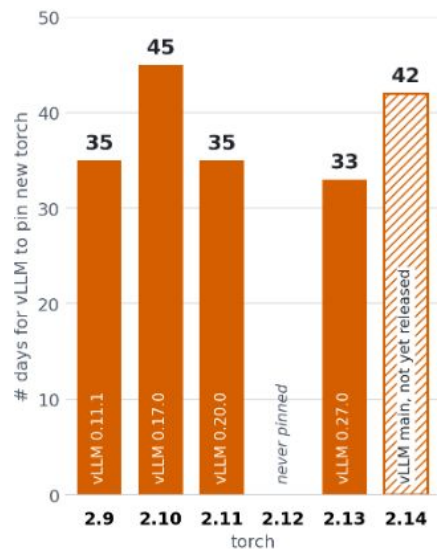
CONFERENCE

—NORTH AMERICA 2026—

Clearing the Path Towards an ABI Stable PyTorch C++ Extension Ecosystem

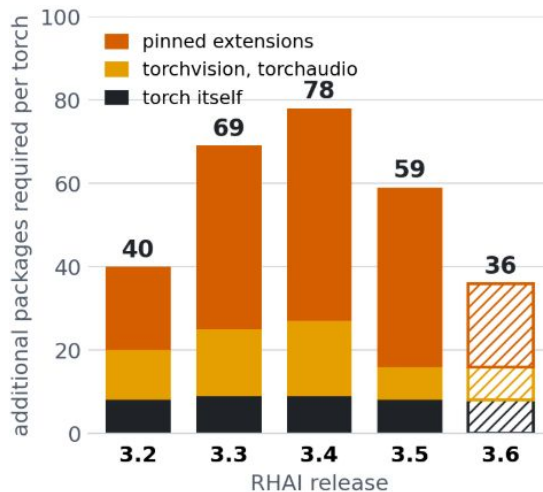
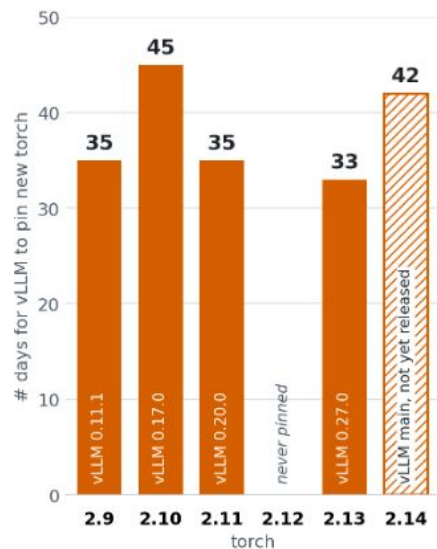
Sean McGovern & Chris Leonard, Red Hat;
Jane Xu, Meta

A new PyTorch release! Yay! . . . right?



maintainers of custom op libraries
(e.g. vLLM) must update their kernel
binaries to support the new version

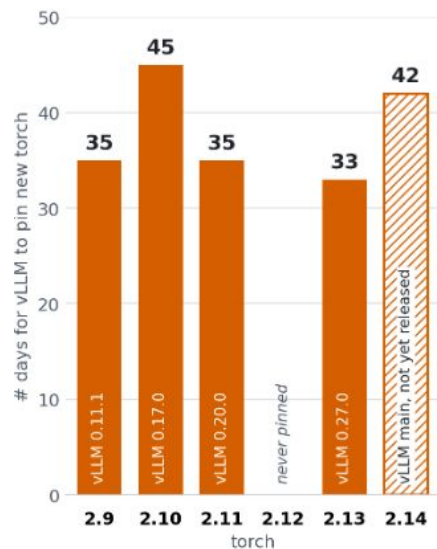
A new PyTorch release! Yay! ... right?



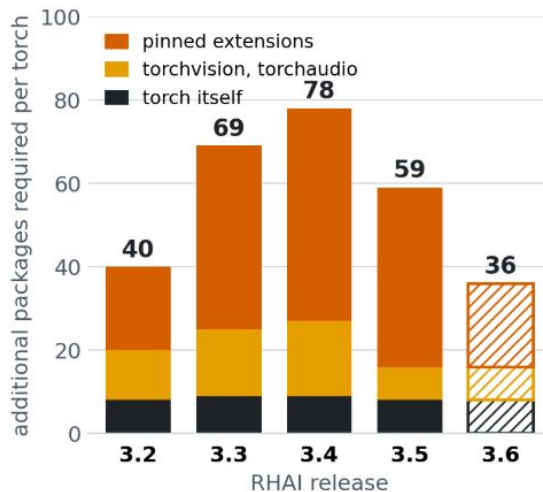
maintainers of custom op libraries (e.g. vLLM) must update their kernel binaries to support the new version

then, **package distributors** must rebuild and release the wheels of all the updated dependencies

A new PyTorch release! Yay! ... right?



maintainers of custom op libraries (e.g. vLLM) must update their kernel binaries to support the new version



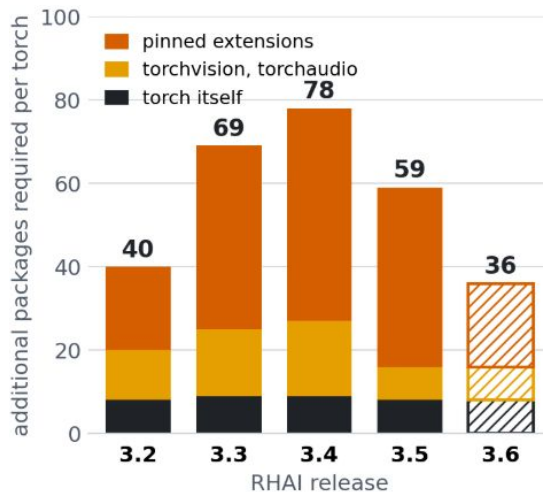
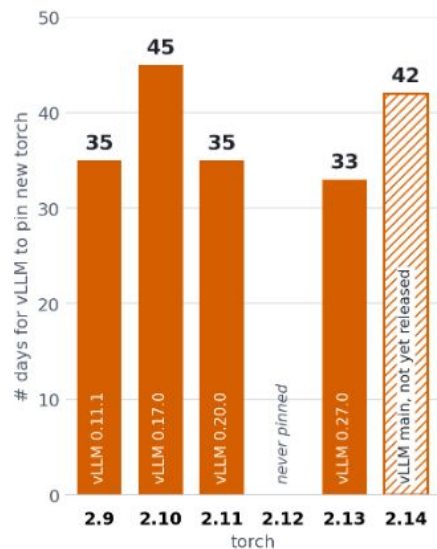
then, **package distributors** must rebuild and release the wheels of all the updated dependencies

```
$ python -m pip install torch==2.14.1
... 25 lines: Collecting / Downloading / Uninstalling torch
2.13.0 ...
ERROR: pip's dependency resolver does not currently take
into account all the packages that are installed. This
behaviour is the source of the following dependency
conflicts.
vllm 0.31.0 requires torch==2.13.0, but you have torch
2.14.1 which is incompatible.
torchvision 0.28.0 requires torch==2.13.0, but you have
torch 2.14.1 which is incompatible.
Successfully installed nvidia-cudnn-cu13-9.24.0.43
nvidia-nccl-cu13-2.30.7 torch-2.14.1 triton-3.8.0

$ python -c "from vllm import LLM; print('ok!')"
Traceback (most recent call last):
  File "<string>", line 1, in <module>
  File "<frozen importlib._bootstrap>", line 1412, in
_handle_fromlist
  File "/root/vllm-demo/venv/lib/python3.12/site-packages/
```

then, **you (the developer)** must download all the updated dependencies when updating torch

A new PyTorch release! Yay! ... right?



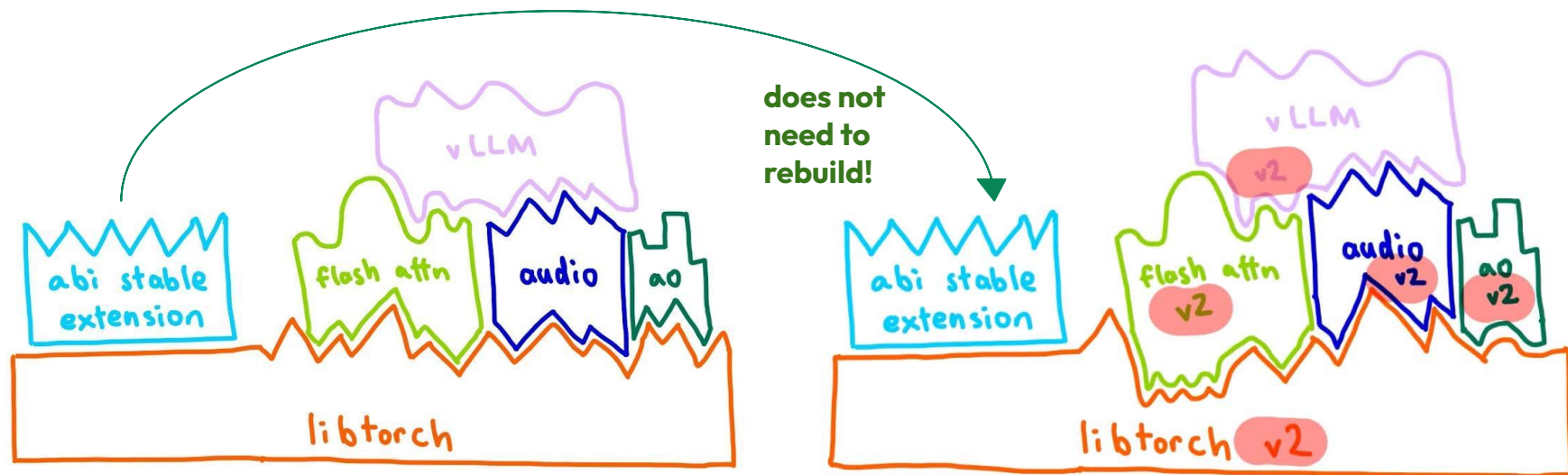
```
$ python -m pip install torch==2.14.1
... 25 lines: Collecting / Downloading / Uninstalling torch
... 2.13.0 ...
ERROR: pip's dependency resolver does not currently take
into account all the packages that are installed. This
behaviour is the source of the following dependency
conflicts.
vllm 0.31.0 requires torch==2.13.0, but you have torch
2.14.1 which is incompatible.
torchvision 0.28.0 requires torch==2.13.0, but you have
torch 2.14.1 which is incompatible.
Successfully installed nvidia-cudnn-cu13-9.24.0.43
nvidia-nccl-cu13-2.30.7 torch-2.14.1 triton-3.8.0

$ python -c "from vllm import LLM; print('ok!')"
Traceback (most recent call last):
  File "<string>", line 1, in <module>
  File "<frozen importlib._bootstrap>", line 1412, in
_handle_fromlist
  File "/root/vllm-demo/venv/lib/python3.12/site-packages/
```

Being tangled to LibTorch = a new release requires work from everyone :(

If only there was a way to detangle from torch releases...

If only there were an ABI Stable LibTorch...



Then, extensions relying on only stable APIs would be free from LibTorch upgrades.

Good news: there is a limited LibTorch stable ABI:

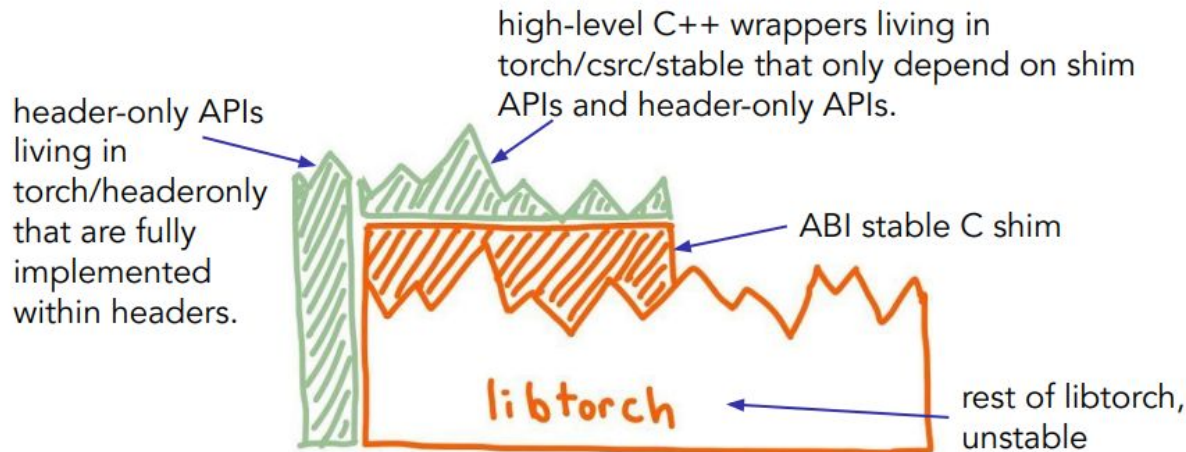
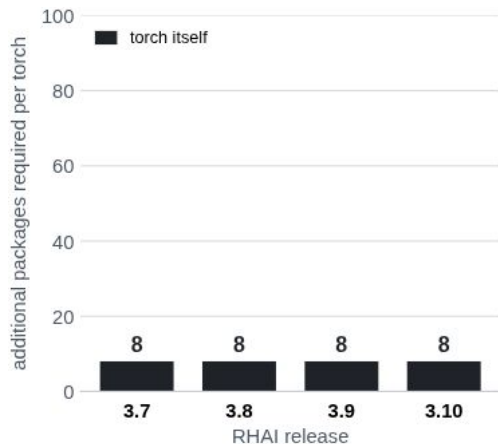
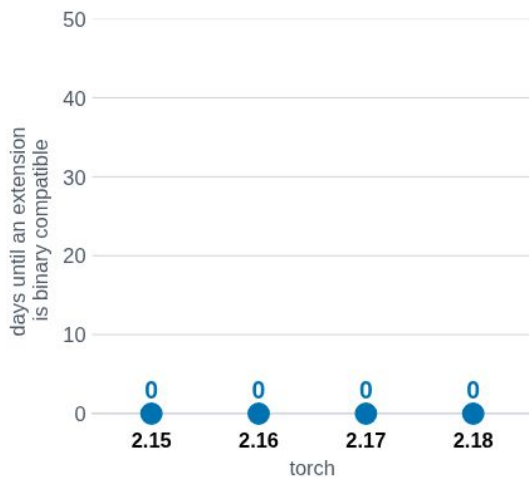


Image from "A Stable Limited LibTorch ABI? How?! (and Why?)" talk at PTC US 2025

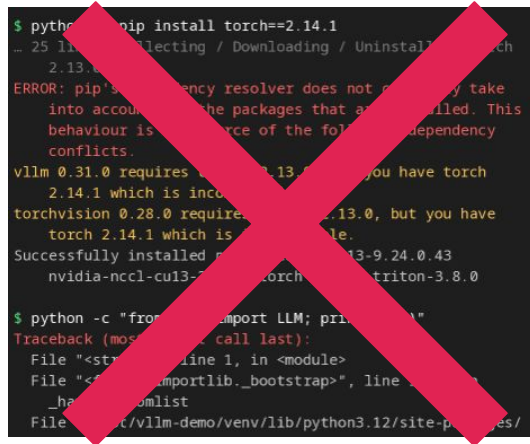
And as of today, everyone from the previous slide uses it!

What would freedom look like for a stable extension?

Well, when the next exciting new PyTorch release rolls around...



NB: This slide is for illustrative purposes about ecosystem impact for a hypothetical new torch release. No claim is made about any project's future releases.



maintainers no longer need to worry about binary incompatibilities

package distributors need not rebuild any stable dependencies

and, **you (the developer)** only need to download the new torch!

Freedom (ABI Stability) does come at a (small) cost...

2.10+ Needed for Build

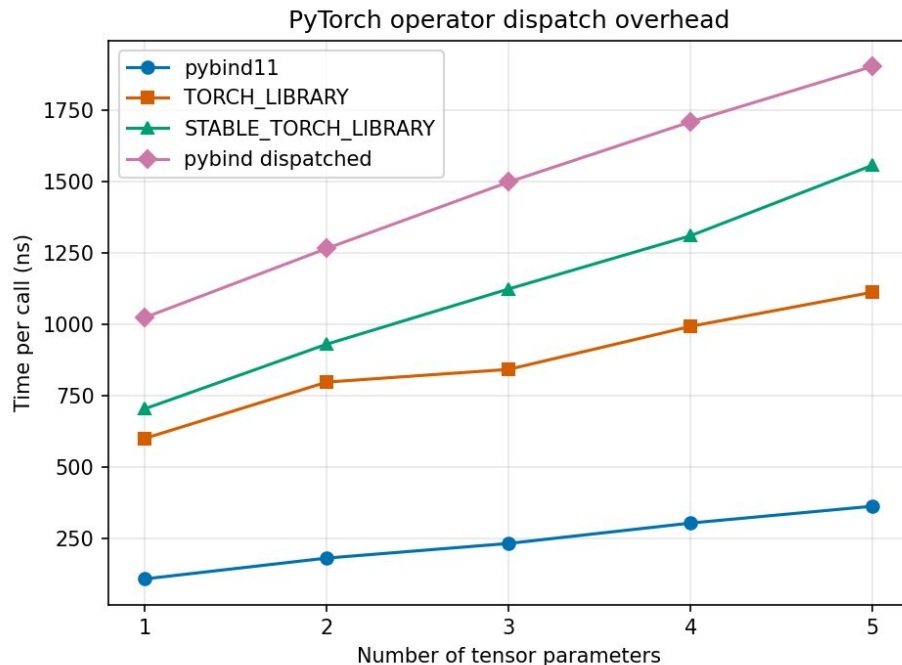
Raises minimum supported PyTorch version at build time

Limited API Surface

Does not contain all the APIs of LibTorch

Dispatch Cost

Minor runtime overhead associated with dispatcher and boxing of StableValues



**vLLM counted the cost and decided ABI stability was worth.
Let's dive into the migration!**

Steps for migrating a library:

I. Assess the situation

- a. can you do the migration all in one go?
- b. does your library need unsupported unstable APIs?

Steps for migrating a library:

1. Assess the situation
 - a. can you do the migration all in one go?
 - b. does your library need unsupported unstable APIs?
2. **Set up necessary scaffolding**
 - a. if you cannot migrate all at once,
set up a stable and unstable extension.

Steps for migrating a library:

1. Assess the situation
 - a. can you do the migration all in one go?
 - b. does your library need unsupported unstable APIs?
2. Set up necessary scaffolding
 - a. if you cannot migrate all at once,
set up a stable and unstable extension.
3. **Migrate AutogradFunctions to Python**
4. **Migrate meta functions to Python**

Steps for migrating a library:

1. Assess the situation
 - a. can you do the migration all in one go?
 - b. does your library need unsupported unstable APIs?
2. Set up necessary scaffolding
 - a. if you cannot migrate all at once,
set up a stable and unstable extension.
3. Migrate AutogradFunctions to Python
4. Migrate meta functions to Python
5. **Make sure you register ops through the dispatcher**

Steps for migrating a library:

1. Assess the situation
 - a. can you do the migration all in one go?
 - b. does your library need unsupported unstable APIs?
2. Set up necessary scaffolding
 - a. if you cannot migrate all at once,
set up a stable and unstable extension.
3. Migrate AutogradFunctions to Python
4. Migrate meta functions to Python
5. Make sure you register ops through the dispatcher
6. **Migrate each source file to use stable APIs**

Steps for migrating a library:

1. Assess the situation
 - a. can you do the migration all in one go?
 - b. does your library need unsupported unstable APIs?
2. Set up necessary scaffolding
 - a. if you cannot migrate all at once,
set up a stable and unstable extension.
3. Migrate AutogradFunctions to Python
4. Migrate meta functions to Python
5. Make sure you register ops through the dispatcher
6. Migrate each source file to use stable APIs
7. **Verify!**

Steps for migrating vLLM:

for vLLM

1. Assess the situation ☒
2. Set up necessary scaffolding ☐
3. Migrate AutogradFunctions to Python ☒
4. Migrate meta functions to Python ☒
5. Make sure you register ops through the dispatcher ☒
6. Migrate each source file to use stable APIs ☐
7. Verify! ☐

FYI: AI is pretty good

1. Assess the situation
2. Set up necessary scaffolding
3. Migrate AutogradFunctions to Python
4. Migrate meta functions to Python
5. Make sure you register ops through the dispatcher
6. Migrate each source file to use stable APIs
7. Verify!

use the assess-abi-migration skill



then use the
orchestrate-stable
-abi-migration skill
which will dispatch
to subskills for steps
2-6

do still verify!

<https://github.com/janeyx99/abi-stable-skills/tree/main/skills>

or try one-shotting it :p

Assessing the situation



Migrating vLLM, how hard can it be?

175+

.cpp/.cu source files

120+

header files

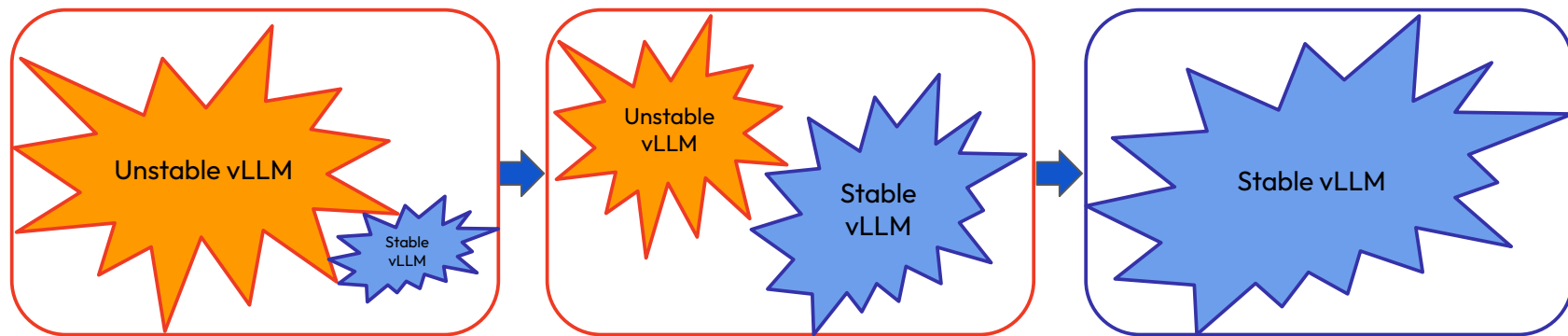
7

third-party libraries using
LibTorch

REAL-WORLD IMPACT

vLLM powers production systems serving millions of users

Setting up the scaffolding

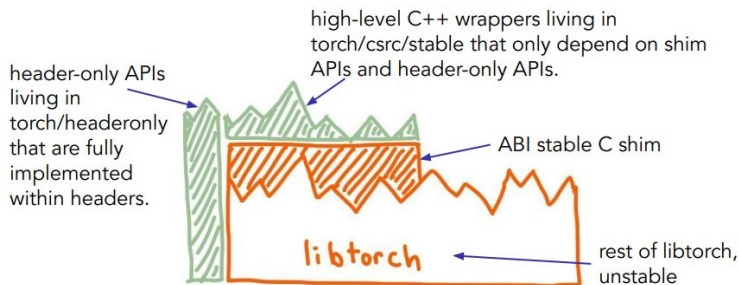


- As we migrate, the LibTorch ABI stable files move to the **stable library**
- Everything else stays in the **unstable library**
- Use **TORCH_TARGET_VERSION** compiler flag to prevent regressions & ensure nothing unstable gets brought in

Migrate each source file to use stable APIs

Use only stable headers

- torch/csrc/stable
- torch/headeronly
- torch/csrc/stable/c/shim.h



& Update APIs

```
torch::Tensor → torch::stable::Tensor  
t.view({...}) → torch::stable::view(t, {...})  
at::kInt → torch::headeronly::ScalarType::Int  
t.data_ptr<T>() → t.const_data_ptr<T>() (read only)  
t.data_ptr<T>() → t.mutable_data_ptr<T>() (read-write)  
TORCH_CHECK → STD_TORCH_CHECK  
TORCH_LIBRARY → STABLE_TORCH_LIBRARY
```

We attempted a deterministic clang based tool to do this, but AI was still simpler.

<https://github.com/TorchedHat/pytorch-stable-abi-transform>

Verify with torch-abi-audit!

<https://pypi.org/project/torch-abi-audit/> – Special thanks to Ralf Gommers!

torch-abi-audit traces .so files and categorizes **stable** vs. **unstable** PyTorch symbols

Use it to check PyTorch and CPython ABI compliance, track migration progress, detect unstable symbol regressions, and easily enforce stability in CI.

```
-- extensions --
[STABLE] [abi3-ok] _C_stable_libtorch.abi3.so (stable_shim=87, unstable=0) min-torch=2.11.0
[UNSTABLE] [abi3-ok] _flashmla_C.abi3.so (stable_shim=66, unstable=0)
[UNSTABLE] [abi3-ok] _flashmla_extension_C.abi3.so (stable_shim=63, unstable=0)
[STABLE] [abi3-ok] _moe_C.abi3.so (stable_shim=72, unstable=0) min-torch=2.11.0
[UNSTABLE] [abi3-ok] _qutlass_C.abi3.so (stable_shim=0, unstable=59)
[NO-TORCH] [abi3-ok] cumem_allocator.abi3.so
[NO-TORCH] [abi3-ok] spinloop.abi3.so
[UNSTABLE] [uses-private-api] third_party/deep_gemm/_C.cpython-312-x86_64-linux-gnu.so (stable_shim=0,
unstable=59)
[UNSTABLE] [abi3-ok] vllm_flash_attn/_vllm_fa2_C.abi3.so (stable_shim=0, unstable=85)
[UNSTABLE] [abi3-ok] vllm_flash_attn/_vllm_fa3_C.abi3.so (stable_shim=0, unstable=81)
```

Don't forget about the dependencies!

```
-- extensions --
[STABLE] [abi3-ok] ] _C_stable_libtorch.abi3.so (stable_shim=87, unstable=0) min-torch=2.11.0
[UNSTABLE] [abi3-ok] ] _flashmla_C.abi3.so (stable_shim=66, unstable=0)
[UNSTABLE] [abi3-ok] ] _flashmla_extension_C.abi3.so (stable_shim=63, unstable=0)
[STABLE] [abi3-ok] ] _moe_C.abi3.so (stable_shim=72, unstable=0) min-torch=2.11.0
[UNSTABLE] [abi3-ok] ] _qutlass_C.abi3.so (stable_shim=0, unstable=59)
[NO-TORCH] [abi3-ok] ] cumem_allocator.abi3.so
[NO-TORCH] [abi3-ok] ] spinloop.abi3.so
[UNSTABLE] [uses-private-api] third_party/deep_gemm/_C.cpython-312-x86_64-linux-gnu.so (stable_shim=0,
unstable=59)
[UNSTABLE] [abi3-ok] ] vllm_flash_attn/_vllm_fa2_C.abi3.so (stable_shim=0, unstable=85)
[UNSTABLE] [abi3-ok] ] vllm_flash_attn/_vllm_fa3_C.abi3.so (stable_shim=0, unstable=81)
```

C++ dependencies also need to be made ABI stable with LibTorch in order for the whole wheel to be ABI stable.

Each dependency is its own beast

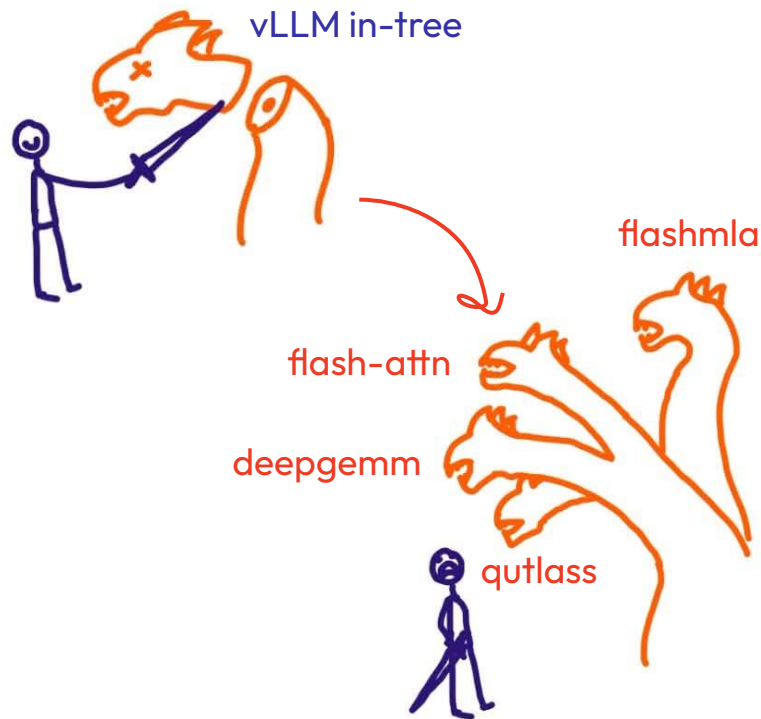
Migrating each dependency is a mini project with its own

build system

different dependencies have varying strategies for binding operators, compilation, and packaging into vLLM

governance

every project has its own maintainers who we'd need to convince to migrate or migrate onto a fork



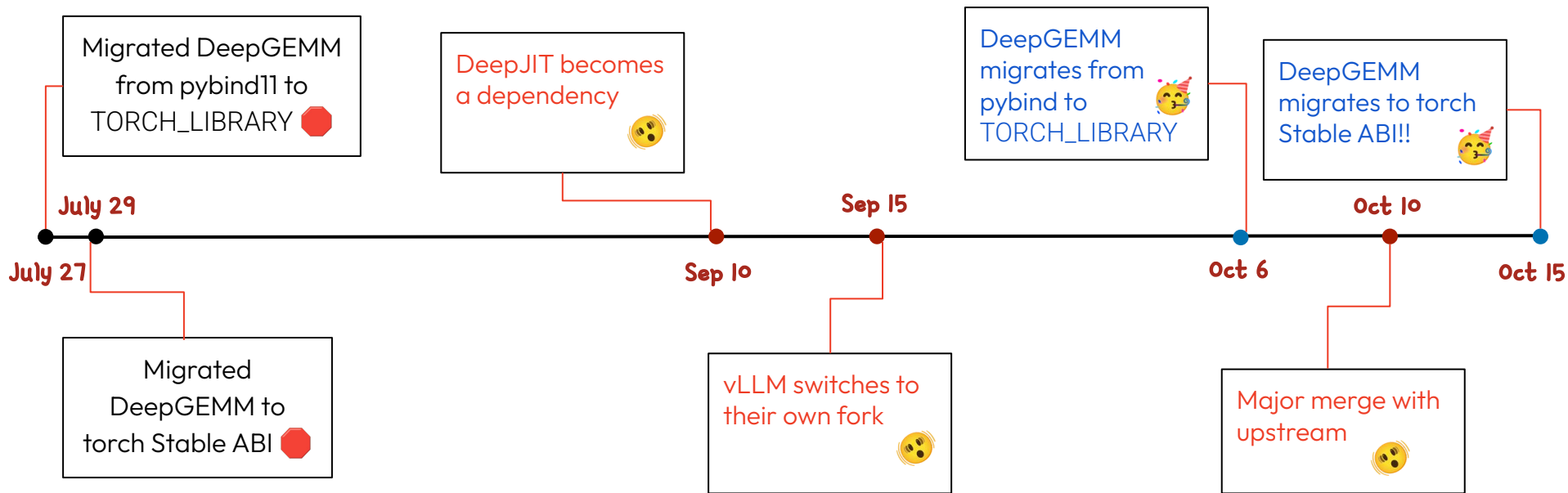
For example: DeepGEMM, a timeline





ONE MONTH
LATER...

For example: DeepGEMM, the real timeline



After migrating all dependencies...success!

```
-- extensions --
[STABLE] [abi3-ok] _C_stable_libtorch.abi3.so (stable_shim=87, unstable=0) min-torch=2.11.0
[STABLE] [abi3-ok] _deepselect_C.abi3.so (stable_shim=40, unstable=0) min-torch=2.10.0
[STABLE] [abi3-ok] _flashmlda_C.abi3.so (stable_shim=43, unstable=0) min-torch=2.11.0
[STABLE] [abi3-ok] _flashmla_C.abi3.so (stable_shim=66, unstable=0) min-torch=2.11.0
[STABLE] [abi3-ok] _flashmla_extension_C.abi3.so (stable_shim=63, unstable=0) min-torch=2.11.0
[STABLE] [abi3-ok] _moe_C_stable_libtorch.abi3.so (stable_shim=72, unstable=0) min-torch=2.11.0
[STABLE] [abi3-ok] _qutlass_C.abi3.so (stable_shim=63, unstable=0) min-torch=2.13.0
[NO-TORCH] [abi3-ok] cumem_allocator.abi3.so
[NO-TORCH] [abi3-ok] fs_io_C.abi3.so
[NO-TORCH] [abi3-ok] spinloop.abi3.so
[STABLE] [abi3-ok] third_party/deep_gemm/_C_extension.abi3.so (stable_shim=78, unstable=0) min-torch=2.11.0
[STABLE] [abi3-ok] vllm_flash_attn/_vllm_fa2_C.abi3.so (stable_shim=69, unstable=0) min-torch=2.10.0
[STABLE] [abi3-ok] vllm_flash_attn/_vllm_fa3_C.abi3.so (stable_shim=63, unstable=0) min-torch=2.10.0
```

!! HUGE thanks to [@Harry-Chen](#) for getting our code merged from the vLLM side !!

Also thanks to the vLLM, DeepSeek & QuTLASS maintainers who helped land them as well :D

vLLM CUDA is now LibTorch ABI stable!

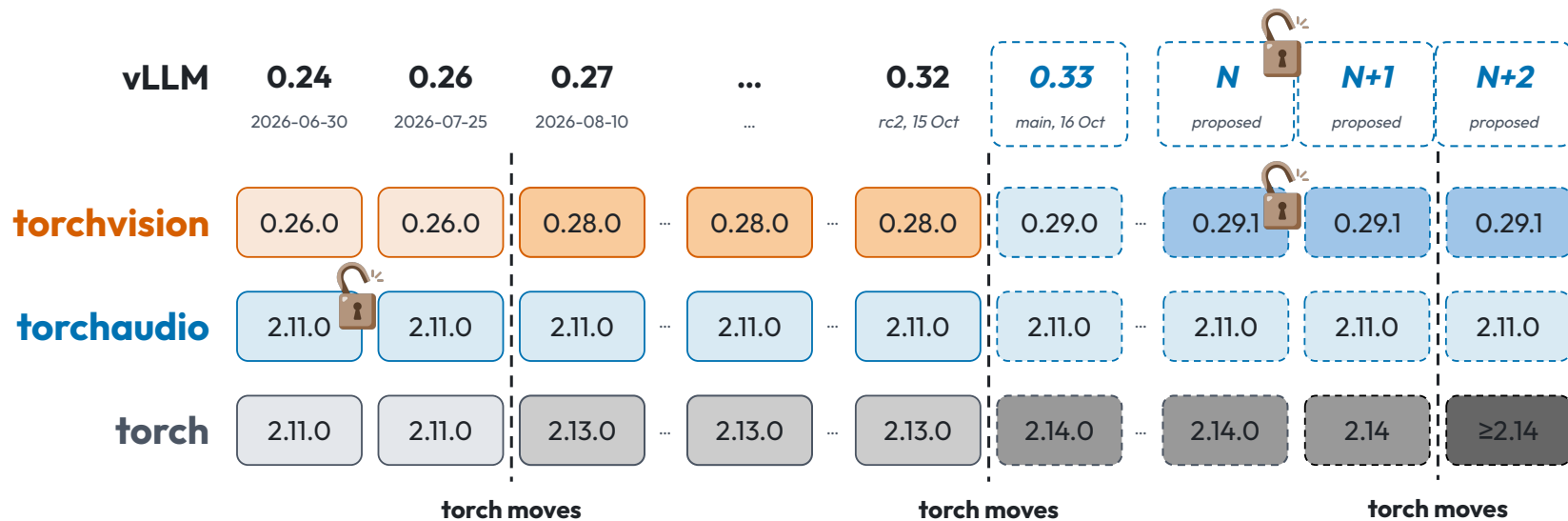
Before

		Python version		
		3.10	3.11	3.12
torch version	2.10	vllm-0.17.0-cp38-abi3-manylinux_2_35_x86_64.whl		
	2.11	vllm-0.20.0-cp38-abi3-manylinux_2_31_x86_64.whl		
	2.13	vllm-0.27.0-cp38-abi3-manylinux_2_28_x86_64.whl		

After

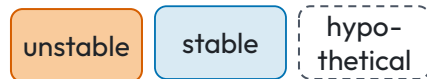
		Python version		
		3.10	3.11	3.12
torch version	2.11	vllm-0.33.0-cp38-abi3-manylinux_2_28_x86_64.whl		
	2.12			
	2.13			

vLLM joins the ranks of other stable libraries

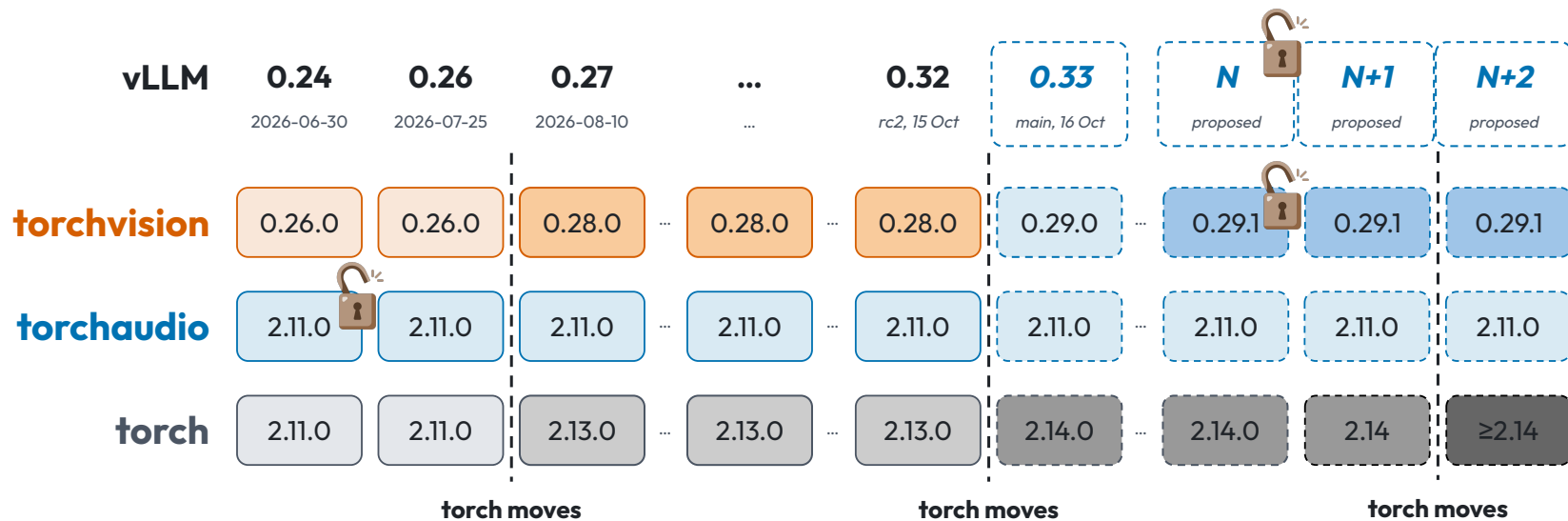


Thanks to [@NicolasHug](#) and [@adabeyta](#) for their recent efforts to get torchvision onto the stable ABI!

stops pinning torch at runtime

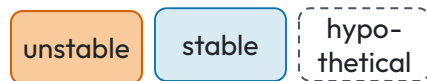


Towards a stable PyTorch extension ecosystem



ABI stability lets libraries stop pinning torch at runtime if desired, so the more untangled the ecosystem is, the faster it can blossom 🌸

stops pinning torch at runtime



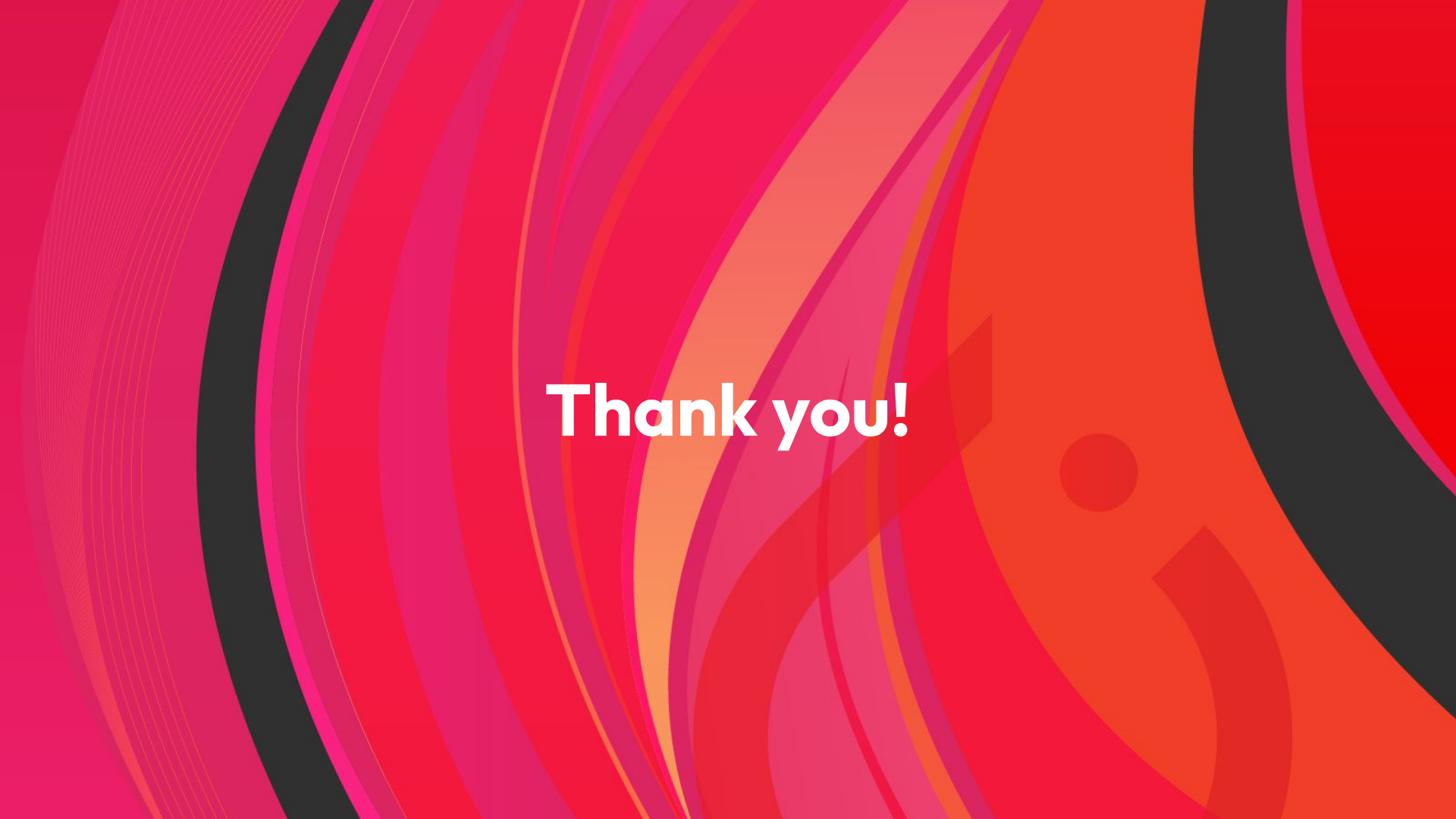
It's your turn!



With our tools, honed skills, and expanded ABI coverage, you and your fave agent should be able to migrate your library in < a few hours.

👁️ at the unmigrated libs in:
<https://github.com/tiran/agent-skills/wiki/Torch-Ecosystem-stable-ABI-status>

Please holler if you have questions or suggestions!



Thank you!

