# WhiteFox

# White-box Compiler Fuzzing Empowered by Large Language Models

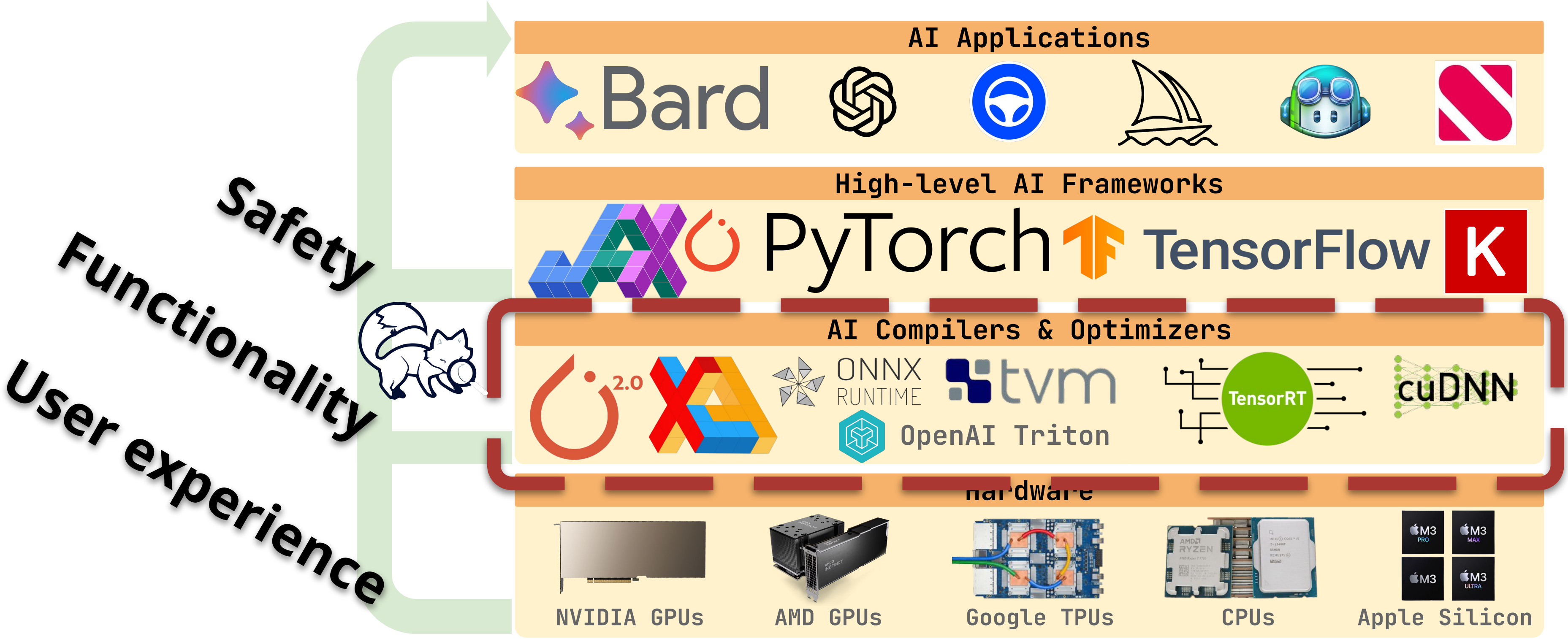
**Chenyuan Yang**, Yinlin Deng, Runyu Lu, Jiayi Yao, Jiawei Liu
Reyhaneh Jabbarvand, Lingming Zhang

GitHub: ise-uiuc/WhiteFox arXiv: 2310.15991

UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN

SPLASH PASADENA 2024

# DL Correctness is Crucial



# DL Compiler/Optimizer is the Trend

**PYTORCH 2.X: FASTER, MORE PYTHONIC AND AS DYNAMIC AS EVER**

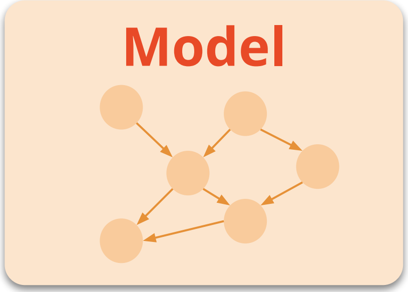
Today, we announce `torch.compile`, a feature that pushes PyTorch performance to new heights and starts the move for parts of PyTorch from C++ back into Python. We

https://pytorch.org/get-started/pytorch-2.0/

**Fast and scalable**

- **XLA Compilation.** We are focusing on XLA compilation and aim to make most model training and inference workflows faster on GPU and CPU, building on XLA's performance wins on TPU. We intend for XLA to become the industry-standard deep learning compiler,

https://blog.tensorflow.org/2022/10/building-the-future-of-tensorflow.html




2.0

ONNX RUNTIME

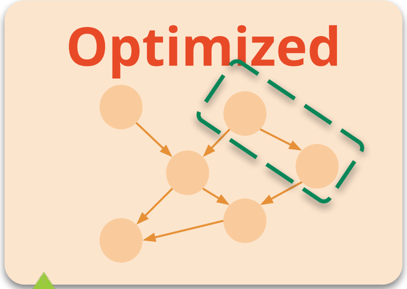
tvm

OpenAI Triton

Uncovered

LinearReLU

Uncovered




Performance

FuzzGPT [ICSE 24]
TitanFuzz [ISSTA 23]
NNSmith [ASPLOS 23]
**Black-box**

**Failed** to cover the **optimization** efficiently

# Limitations of Existing Work

```python
def permute_linear_fusion(module:GraphModule):
  for node in module.graph.nodes:
    if (
      node.op == "call_function"
      and node.target==torch.nn.functional.linear
    ):
      if len(node.args) > 0:
        input_node = node.args[0]
      else:
        input_node = node.kwargs["input"]
      if (
        input_node.op == "call_method"
        and input_node.target == "permute"
        and check_permute(input_node)
      ):
        # BUG IS HERE! (Code omitted for brevity)
  return module
def check_permute(node: torch.fx.Node):
  ranks = len(node.meta["tensor_meta"].shape)
  ...
  allowed_permutation = list(range(ranks))
  allowed_permutation[-1] = ranks - 2
  allowed_permutation[-2] = ranks - 1
  return permutation == allowed_permutation
```

**Random** models (e.g., by existing black-box DL fuzzer NNSmith) can *rarely* trigger ***permute_linear_fusion***

```python
class Model(nn.Module):
    def forward(self, v5):
        v6 = torch.neg(v5)
        return self.linear(v6)
```

## Black-/grey-box fuzzing

- Unaware of source code implementation
- Struggle with reaching deep paths

## Traditional white-box fuzzing

- Optimization techniques are too complex
- Path exploration and hard to model


@cy1yang cy54@illinois.edu

# Insights: White-Box & LLMs

```python
def permute_linear_fusion(module:GraphModule):
  for node in module.graph.nodes:
    if (
      node.op == "call_function"
      and node.target==torch.nn.functional.linear
    ):
      if len(node.args) > 0:
        input_node = node.args[0]
      else:
        input_node = node.kwargs["input"]
      if (
        input_node.op == "call_method"
        and input_node.target == "permute"
        and check_permute(input_node)
      ):
        # BUG IS HERE! (Code omitted for brevity)
  return module
def check_permute(node: torch.fx.Node):
  ranks = len(node.meta["tensor_meta"].shape)
  ...
  allowed_permutation[-1] = ranks - 2
  allowed_permutation[-2] = ranks - 1
  return permutation == allowed_permutation
```

**Optimization Source Code**

**Analysis Agent**

The permute method is invoked on an input tensor with more than 2 dimensions, and it swaps the last two dimensions of this tensor.

**NL Description**

```python
t1 = input_tensor.permute(...) # Permute input
t2 = torch.nn.functional.linear(t1, ...) # ...
```

**Pseudo Code**

**Requirement Summarization**

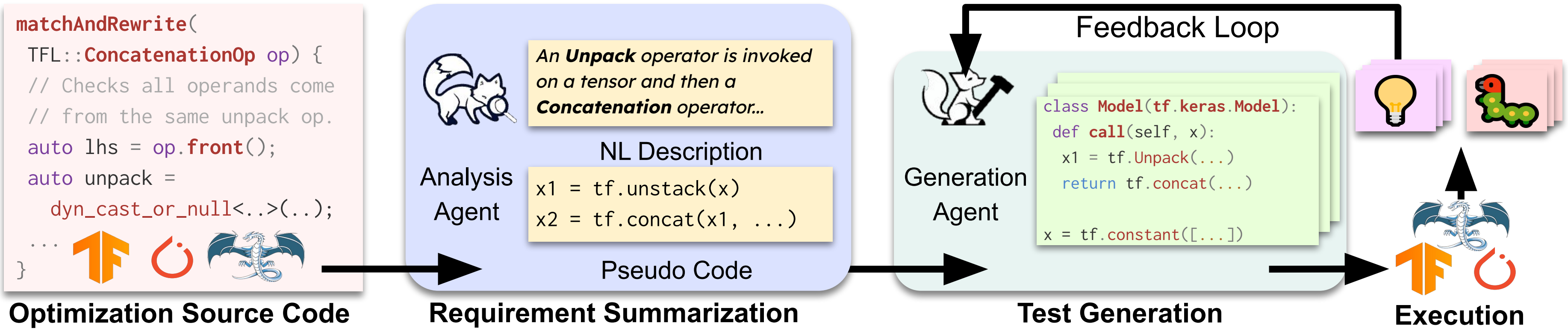
**Generation Agent**

```python
def forward(self, x1, weight, bias):
  v1 = x1.permute(0, 2, 1).resize_(1, 1, 2)
  return torch.F.linear(v1, weight, bias)
```

**Test Generation**

# WhiteFox: White-box Fuzzing via LLMs

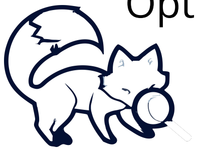
- **Dual-agent framework**
  - **Analysis Agent**: summarize source code of compiler passes
  - **Generation Agent**: construct tests to specifically test the target pass



# Requirement Summarization

```
# Code of the optimization
def permute_linear_fusion(module: torch.fx.GraphModule):
  for node in module.graph.nodes:
    if node.op == "call_function":
      if node.target == torch.nn.functional.linear:
        ...
```

**Analysis Agent**



Optimization source code

⇩

Input pattern

```
# Description
The model should contain the following pattern:
```
t1 = input_tensor.permute(...) # Permute the input tensor
t2 = torch.nn.functional.linear(t1, ...) # Apply linear transformation to the permuted tensor.
```
The permute method is invoked on an input tensor with more than 2 dimensions, and it swaps the last two dimensions of this tensor.
```

PSEUDO CODE

NL DESCRIPTION

- Analyze the **optimization source code** to infer the **input pattern** to trigger it
- Few-shot prompting

> Please describe the **PyTorch model** that can trigger the `**permute_linear_fusion**` optimization. Use code to illustrate patterns or constraints as needed.

TARGET INPUT

OPTIMIZATION NAME

- The description
  - Pseudo code
  - NL description

# Test Generation

```
# Description
The model should contain the following pattern:
```
t1 = input_tensor.permute(...) # Permute the input tensor
t2 = torch.nn.functional.linear(t1, ...) # Apply linear transformation to the permuted tensor.
```
The permute method is invoked on an input tensor with more
than 2 dimensions, and it swaps the last two dimensions of
this tensor.
```

**Generation Agent**

Input pattern

↓

Input

```
# Model
class Model(torch.nn.Module):
  def forward(self, x1):
      v1 = x1.permute(0, 2, 1)
      v2 = F.linear(v1, self.weight, self.bias)
      return v2
```

- Generate diverse inputs based on the pattern
- Few-shot prompting

```
Please generate a valid PyTorch model example with
public PyTorch APIs meets the specified requirements.
```

TARGET

INPUT FORMAT

# Feedback Loop

*Few-shot Example*

```
### Please generate a valid TARGET INPUT example with INPUT SPECIFICATION meets the specified requirements.
```

```
# Description
The TARGET INPUT should contain the following pattern:
```
PSEUDO CODE
```
This pattern characterizes scenarios where NL DESCRIPTION.
```

```
# TARGET INPUT
EXAMPLE TRIGGERING INPUT #1
```

```
# TARGET INPUT
EXAMPLE TRIGGERING INPUT #2
```

```
# TARGET INPUT
EXAMPLE TRIGGERING INPUT #3
```

```
# TARGET INPUT [TO BE GENERATED]
```

- Incorporating execution feedback:
  - Use successful triggering tests to guide future generations **for the same target**
- Balancing exploration and exploitation
  - Use example selection algorithm

# Implementation

- Targeted DL compilers
  - PyTorch Inductor, TensorFlow XLA, TensorFlow Lite
  - Collected the compiler source code specific for optimization
- Agent
  - GPT-4 as the *analysis* agent
  - StarCoder as the *generation* agent
- Budget: 24 hour fuzzing / 1,000 tests for each optimization

| | # Optimizations | Source Language | Language of tests | Baselines |
|---|---|---|---|---|
| **PyTorch Inductor** | **61** | **Python** | **Python** | **TitanFuzz, NNSmith** |
| TensorFlow Lite | 13 | C++ | | |
| TensorFlow XLA | 49 | C++ | | |

# Comparison with Baselines – 24-Hour

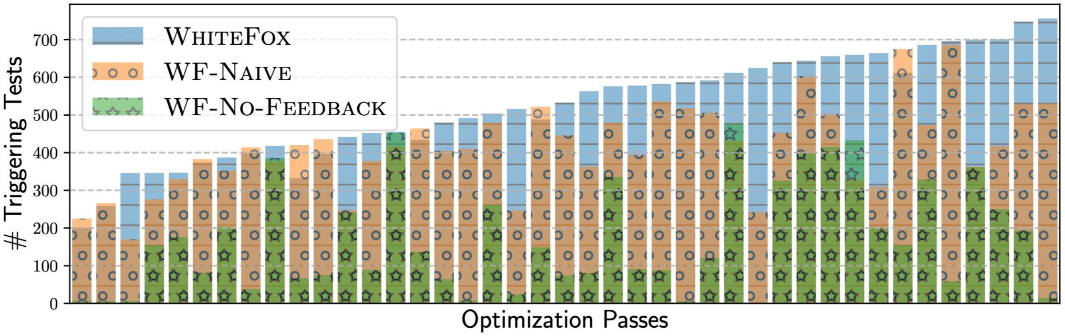
- **High efficiency:** Covers ~8x optimizations
- **Minimal manual effort:** Only a few fixed handwritten demonstrations for prompting

| | # Triggered optim. | # Triggered tests / Total | Coverage |
|---|---|---|---|
| **WhiteFox** | **41** | **12,127 / 35,380** | **54,819** |
| TitanFuzz (LLM-based) | 4 | 1,697 / 167,521 | 53,592 |
| NNSmith (Symbolic) | 5 | 233 / 57,664 | 49,910 |

**# Total Optim.:** 61

# Ablation Studies

| Approach | #Triggered |
| --- | --- |
| WhiteFox-**Mix** | **39** |
| WhiteFox-NL-Only | 37 |
| WhiteFox-PseudoCode-Only | 32 |
| WhiteFox-Impl-Code | 32 |
| WhiteFox-StarCoder | 32 |




## Analysis phase

Mixed format > single format > raw source code (no analysis)

## Generation phase

- Feedback loop: **2.1x** more triggering tests

## Replacing GPT-4 with StarCoder:

- Performance degradation

# Bug Detection by WhiteFox

- WhiteFox detects **101** bugs for DL compilers
  - **95** confirmed
  - **73** Fixed
  - Contributed to **11 high-priority** bugs for PyTorch
  - Only **12%** can be detected by state-of-the-art baselines

| | Total | Confirmed | Fixed | Won't fix |
|---|---|---|---|---|
| PyTorch | 79 | 76 | 71 | 3 |
| TensorFlow Lite | 11 | 8 | 0 | 3 |
| TensorFlow XLA | 11 | 11 | 2 | 0 |
| Total | 101 | 95 | 73 | 6 |

# Buggy Optimization: Cat + Slice + Cat => Cat

```
def forward(self, x):
    cat_output = torch.cat([x, x[:, :5]], dim=1)
    res = torch.cat([cat_output, cat_output[:, :-3]], dim=1)
    return res
```

**Naive Execution** `res.shape = [2, 9, 16, 16]`

**W/ Optimization** `res.shape = [2, 6, 16, 16]`

```
# Verify we fallback to non-optimal path for negative `end`.
def fn(a, b):
    cat_1 = torch.ops.aten.cat.default([a, b], 1)
```

Added Unit Test from #100828

```
_cat_1 = CallFunction(aten.cat, Arg(), 1, _users=2)
@register_lowering_pattern(
    CallFunction(
        aten.cat, [_cat_1, CallFunction(
                        aten.slice,
                        _cat_1,
                        1,
                        0,
                        KeywordArg("size"),
                    )], 1))
def cat_slice_cat(match, cat_input, size, dim=1):
    ...
```

Optimization Source Code

Issue #100807, PR: #100828

# Root Cause - Cat + Slice + Cat

```
def cat_slice_cat(match, cat_input, size, dim=1):
```

Developers have a concrete example in mind (`slice_size=19`)

```
    multiple times inside the pattern.  We fold 2 calls to cat into one.

    Matches:
        cat_1: f32[1024, 4077] = torch.ops.aten.cat.default([add_26, primals_217], 1)
        slice_1: f32[1024, 4077] = torch.ops.aten.slice.Tensor(cat_1, 0, 0, 9223372036854775807)
        slice_2: f32[1024, 19] = torch.ops.aten.slice.Tensor(slice_1, 1, 0, 19)
        cat_2: f32[1024, 4096] = torch.ops.aten.cat.default([cat_1, slice_2], 1)
    Rewrite to:
        slice_2 = torch.ops.aten.slice.Tensor(add_26, 1, 0, 19)
        cat_2 = torch.ops.aten.cat.default([add_26, primals_217, slice2], 1)
    """
```

But overlooked a special case (`slice_size<0`), which causes a bug

```
    # optimization is optional, because we can just not fold the cat
    if V.graph.sizevars.statically_known_leq(size, first.get_size()[dim]):
    if size >= 0 and V.graph.sizevars.statically_known_leq(size, first.get_size()[dim]):
        # fold 2 cats into 1 cat
        ...
```

wrongly assumes the comparison is between two positive sizes

# Bug Example - Fused Attention

**Bug-triggering Test**

```
class Attention(torch.nn.Module
  ...
  def forward(self, inputs, attn_mask):
    q = self.query(inputs).view(...)
    k = self.key(inputs).view(...)
    v = self.value(inputs).view(...)
    scores = q @ k.transpose(-2,-1) / sqrt(self.hidden_dim)
    scores = scores + attn_mask
    attn_weights = torch.softmax(scores, dim=-1)
    return attn_weights @ v
attn_mask = torch.zeros(...).bool()
```

| | |
|---|---|
| **Naive Execution** | tensor([0.026, 0.017, ...]) |
| **W/ Optimization** | tensor([NaN, NaN, ...]) |

Issue: 100318 PR: 100619

The model should contain the following pattern:

```
qk = query @ key.transpose(-2, -1) / math.sqrt(query.size(-1))
qk = qk + attn_mask
attn_weight = torch.softmax(qk, dim=-1)
attn_weight = torch.dropout(attn_weight, dropout_p, True)
output = attn_weight @ value
```

This pattern characterizes the attention mechanism in transformer models, where the attention weights are computed as the softmax of the scaled dot product of the query and key (plus an attention mask), followed by a dropout operation. The output is then computed as the dot product of these attention weights and the value.

Description generated by WhiteFox

# Bug Example - Linear + ReLU

**Bug-triggering Test**

```python
class Net(nn.Module):
  def __init__(self, inplace):
      self.relu = nn.ReLU(inplace=inplace)
      self.f = nn.Sequential(nn.Linear(10, 20), self.relu,
                             nn.Linear(20, 30), self.relu,
                             nn.Linear(30, 40), self.relu)
  def forward(self, x):
    return self.f(x)
```

| | |
|---|---|
| **Naive Execution** | `tensor([0.07, 0.16, 0.01, ...])` |
| **W/ Optimization** | `tensor([0.07, 0.18, -0.01...])` 🐛 |
| **W/ Optimization + bfloat16** input | `IOT Instruction Crash` 🐛 |

```diff
-        if dtype in [torch.bfloat16]:
-            assert opt_ctx.is_load_bf16_as_fp32 or opt_ctx.is_bf16_mem_copy
         proposed_dtype = opt_ctx.dtype
         if val == float("inf"):
             assert proposed_dtype == torch.float
@@ -648,6 +645,10 @@ def to_dtype(x, dtype):
             return f"vec_convert_to_mask({x})"
         if opt_ctx_x.dtype == torch.bool and dtype in (torch.float, torch.float32):
             return f"mask_convert_to_float({x})"
+        if opt_ctx_x.dtype in (torch.float, torch.float32) and dtype == torch.bfloat16:
+            return f"cvt_fp32_to_bf16({x})"
+        if opt_ctx_x.dtype == torch.bfloat16 and dtype in (torch.float, torch.float32):
+            return f"cvt_bf16_to_fp32({x})"
```

Fixing memory copy error for bf16 #101042

high priority

Issues: 98852, 100830, PRs: 98880, 101042

# WhiteFox: White-box Compiler Fuzzing via LLM

- A new dimension of white-box compiler fuzzing
  - The first practical white-box compiler fuzzer by LLM-based agents
  - **Analysis agent**: summarize source code of compiler passes for requirements
  - **Generation agent**: construct tests to specifically test the target pass
- Detected **101** bugs
  - For PyTorch Inductor, TensorFlow-XLA, and TensorFlow Lite
  - with **95** confirmed and **73** already fixed

**Scan for code!**

**Scan for paper!**

# Observations

- Developers usually **overlook** certain **optimization triggering cases**
  - Invalid models are mis-accepted
    - out-of-bound read, wrong argument validation
  - Valid models that should not be optimized are incorrectly optimized
    - mischeck for special dtypes, overlooking negative index
  - Valid models are optimized in a wrong way
    - memory copy issues, precision
- **Multiple** bugs might exist in one single optimization 😱

```python
class Model(torch.nn.Module)
   def __init__(self):
       super().__init__()
       self.conv_transpose = torch.nn.ConvTranspose2d(3, 6, 3, stride=1, padding=1, output_padding=1)
   def forward(self, input_tensor):
       x = self.conv_transpose(input_tensor)
       output = torch.tanh(x)
       return output
```

**Naive Execution**

```
RuntimeError: output padding must be smaller than either stride or dilation
```

**W/ Optimization**

```
tensor([...])
```

🐛