

01 · THE PROBLEM

Why did my graph break?

your code

```
DEFAULTS = {"scale": 1.0, "bias": 0.0}

@torch.compile(fullgraph=True)
def f(x, opts):
    cfg = DEFAULTS | opts
    return x * cfg["scale"] + cfg["bias"]
```

What does `|` mean here? CPython answers with one rule: look up `nb_or` in the slot table of `dict` and call it. Defining `__or__` on a class fills that slot.

Dynamo used to answer with a hand-written `dict.__or__` that guessed at subclasses and `__ror__`, and a separate `if name == ...` chain for every other type. Each gap between those guesses and CPython's rule was a **graph break** or a wrong result.

A · HOW CPYTHON RUNS A | B

- 1
- a | b → `PyNumber_Or(a, b)`
- 2
- `type(a)->tp_as_number->nb_or` ? Call it
- 3
- NotImplemented ? Try `type(b)` 's reflected slot (`__ror__`)
- 4
- Still nothing → `TypeError`

Check the pointer, then call it: that's the whole protocol.

Over time, Dynamo's core logic became less about well-defined rules and more about accumulating special cases.

02 · CPYTHON → DYNAMO

Same slots, traced

In CPython, every operation *on an object* is a slot lookup on its type. Dynamo now works the same way: every `VariableTracker` exposes CPython's slot tables. An empty slot raises the same `TypeError` as eager Python, so traced errors match eager ones word for word. The goal of this work is to move Dynamo from an implementation that works by accident to a model we can reason about and validate against CPython's source and tests.

B · THE SAME DISPATCH, SIDE BY SIDE

CPython · Objects/dictobject.c (simplified)

```
PyObject *dict_or(PyObject *self, PyObject *other) {
    if (!PyDict_Check(self) || !PyDict_Check(other))
        Py_RETURN_NOTIMPLEMENTED;
    PyObject *new = PyDict_Copy(self);
    dict_update_arg(new, other);
    return new;
}
```

Dynamo · variables/dicts.py

```
def nb_or_impl(self, tx, other):
    if not (pydict_check(self) and pydict_check(other)):
        return ConstantVariable.create(NotImplemented)
    new = self.call_method(tx, "copy", [], {})
    new.call_method(tx, "update", [other], {})
    return new
```

Same checks, same order, same `NotImplemented`. The C is the spec; the Python is a translation.

2026 · WHO BUILT IT

119

Pull Requests

26

Contributors

55

PyTypeObject Slots Modeled

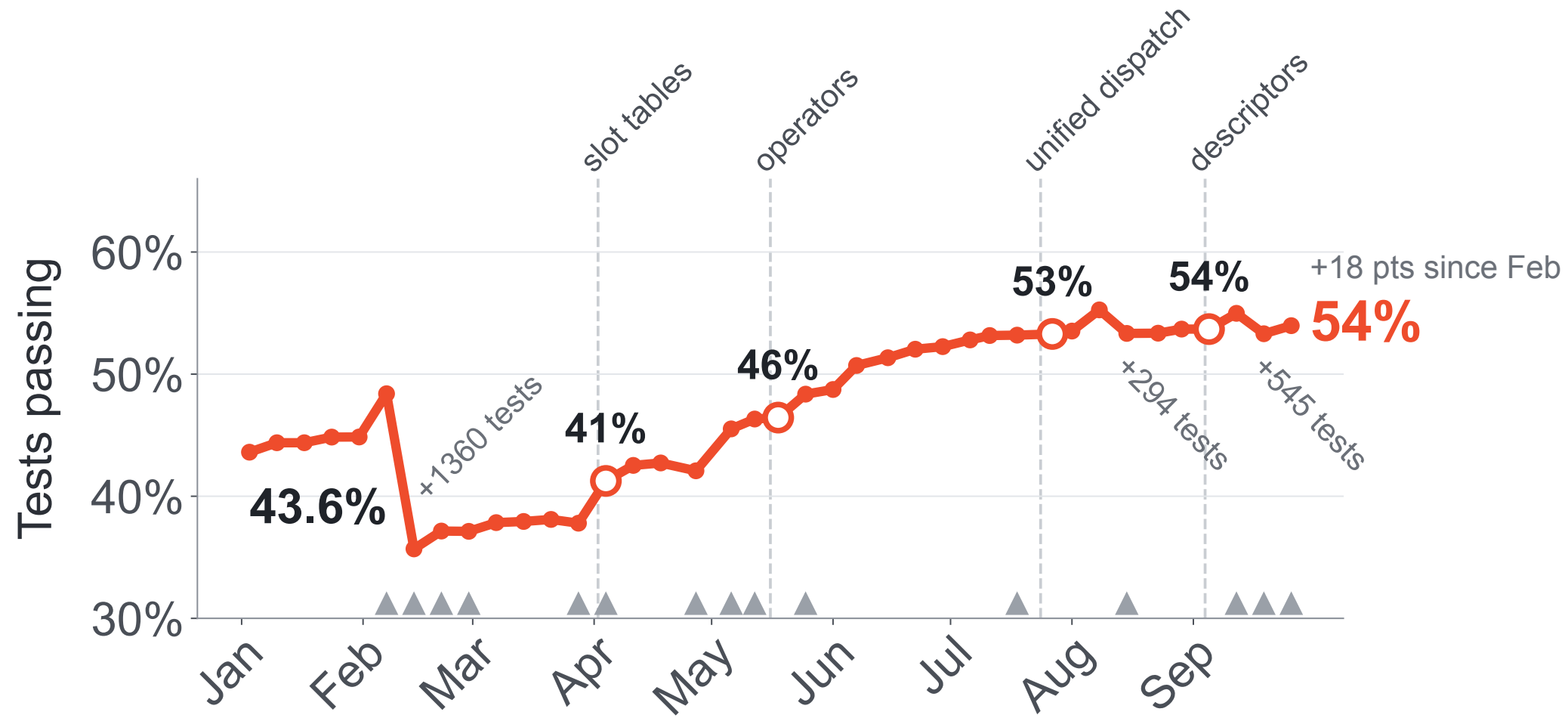
Meta, OpenTeams, Red Hat and the open-source community.

03 · RESULTS

CPython tests

43.6→54% of CPython's own tests pass under `torch.compile`

C · WEEKLY PASS RATE, 2026



6765 tests from 83 CPython test modules, run weekly under `torch.compile`.
▲ marks weeks when new test files were added (+N tests); the dips follow them.

D · GRAPH BREAKS THAT WENT AWAY

	Jan 3	Sep 26	
Unsupported hasattr()	77	0	gone
Untraceable unittest method	60	0	gone
Untraceable builtin operator	139	26	-81%
Unsupported function call	57	19	-67%
Call to skipped function	496	323	-35%

E · MOST IMPROVED TEST FILES

test_cmath	12%	→ 85%
test_userdict	28%	→ 92%
test_userlist	35%	→ 90%
test_operator	30%	→ 82%