

---

# Pyro API

*Release 0.0*

May 21, 2020



---

## Contents:

---

<b>1 Dispatch</b>	<b>3</b>
1.1 Generic Modules . . . . .	4
<b>2 Testing</b>	<b>5</b>
<b>3 Indices and tables</b>	<b>7</b>
<b>Python Module Index</b>	<b>9</b>
<b>Index</b>	<b>11</b>



The `pyroapi` package dynamically dispatches among multiple Pyro backends, including standard `Pyro`, `NumPyro`, `Funsor`, and custom user-defined backends. This package includes both **dispatch** mechanisms for use in model and inference code, and **testing** utilities to help develop and test new Pyro backends.



Dispatching allows you to dynamically set a backend using `pyro_backend()` and to register new backends using `register_backend()`. It's easiest to see how to use these by example:

```
from pyroapi import distributions as dist
from pyroapi import infer, ops, optim, pyro, pyro_backend

# These model and guide are backend-agnostic.
def model():
    locs = pyro.param("locs", ops.tensor([0.2, 0.3, 0.5]))
    p = ops.tensor([0.2, 0.3, 0.5])
    with pyro.plate("plate", len(data), dim=-1):
        x = pyro.sample("x", dist.Categorical(p))
        pyro.sample("obs", dist.Normal(locs[x], 1.), obs=data)

def guide():
    p = pyro.param("p", ops.tensor([0.5, 0.3, 0.2]))
    with pyro.plate("plate", len(data), dim=-1):
        pyro.sample("x", dist.Categorical(p))

# We can now set a backend at inference time.
with pyro_backend("numpyro"):
    elbo = infer.Trace_ELBO(ignore_jit_warnings=True)
    adam = optim.Adam({"lr": 1e-6})
    inference = infer.SVI(model, guide, adam, elbo)
    for step in range(10):
        loss = inference.step(*args, **kwargs)
        print("step {} loss = {}".format(step, loss))
```

**pyro\_backend**(\*aliases, \*\*new\_backends)

Context manager to set a custom backend for Pyro models.

Backends can be specified either by name (for standard backends or backends registered through `register_backend()`) or by providing kwargs mapping module name to backend module name. Standard backends include: `pyro`, `minipyro`, `funsor`, and `numpy`.

**register\_backend** (*alias*, *new\_backends*)

Register a new backend alias. For example:

```
register_backend("minipyro", {
    "infer": "pyro.contrib.minipyro",
    "optim": "pyro.contrib.minipyro",
    "pyro": "pyro.contrib.minipyro",
})
```

### Parameters

- **alias** (*str*) – The name of the new backend.
- **new\_backends** (*dict*) – A dict mapping standard module name (*str*) to new module name (*str*). This needs to include only nonstandard backends (e.g. if your backend uses torch ops, you need not override ops)

## 1.1 Generic Modules

- `pyro` - The main pyro module.
- `distributions` - Includes `distributions.transforms` and `distributions.constraints`.
- `handlers` - Generalizing the original `pyro.poutine`.
- `infer` - Inference algorithms.
- `optim` - Optimization utilities.
- `ops` - Basic tensor operations (like `numpy` or `torch`).



The pyroapi package includes tests to ensure new backends conform to the standard API, indeed these tests serve as the formal API description. To add tests to your new backend say in `project/test/` follow these steps (or see the [example](#) in `funsor`):

1. Create a new directory `project/test/pyroapi/`.
2. Create a file `project/test/pyroapi/conftest.py` and a hook to treat missing features as `xfail`:

```
import pytest

def pytest_runtest_call(item):
    try:
        item.runtest()
    except NotImplementedError as e:
        pytest.xfail(str(e))
```

3. Create a file `project/test/pyroapi/test_pyroapi.py` and define a backend fixture:

```
import pytest
from pyroapi import pyro_backend
from pyroapi.tests import * # noqa F401

@pytest.yield_fixture
def backend():
    with pyro_backend("my_backend"):
        yield
```

4. Test your backend with `pytest`

```
pytest -vx test/pyroapi
```



## CHAPTER 3

---

### Indices and tables

---

- `genindex`
- `modindex`
- `search`



**p**

`pyroapi.dispatch`, 3



## P

`pyro_backend()` (*in module `pyroapi.dispatch`*), 3  
`pyroapi.dispatch` (*module*), 3

## R

`register_backend()` (*in module `pyroapi.dispatch`*),  
3