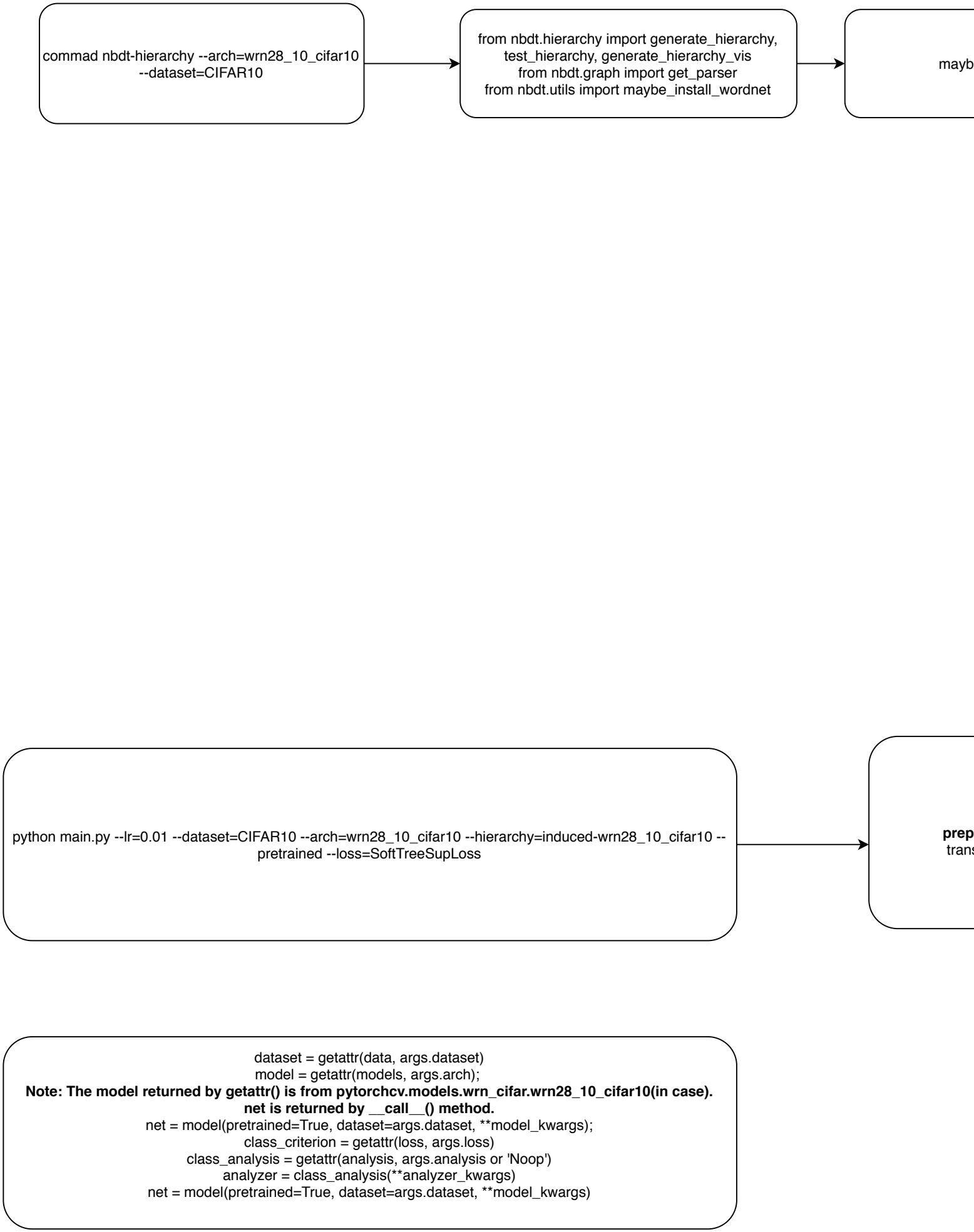
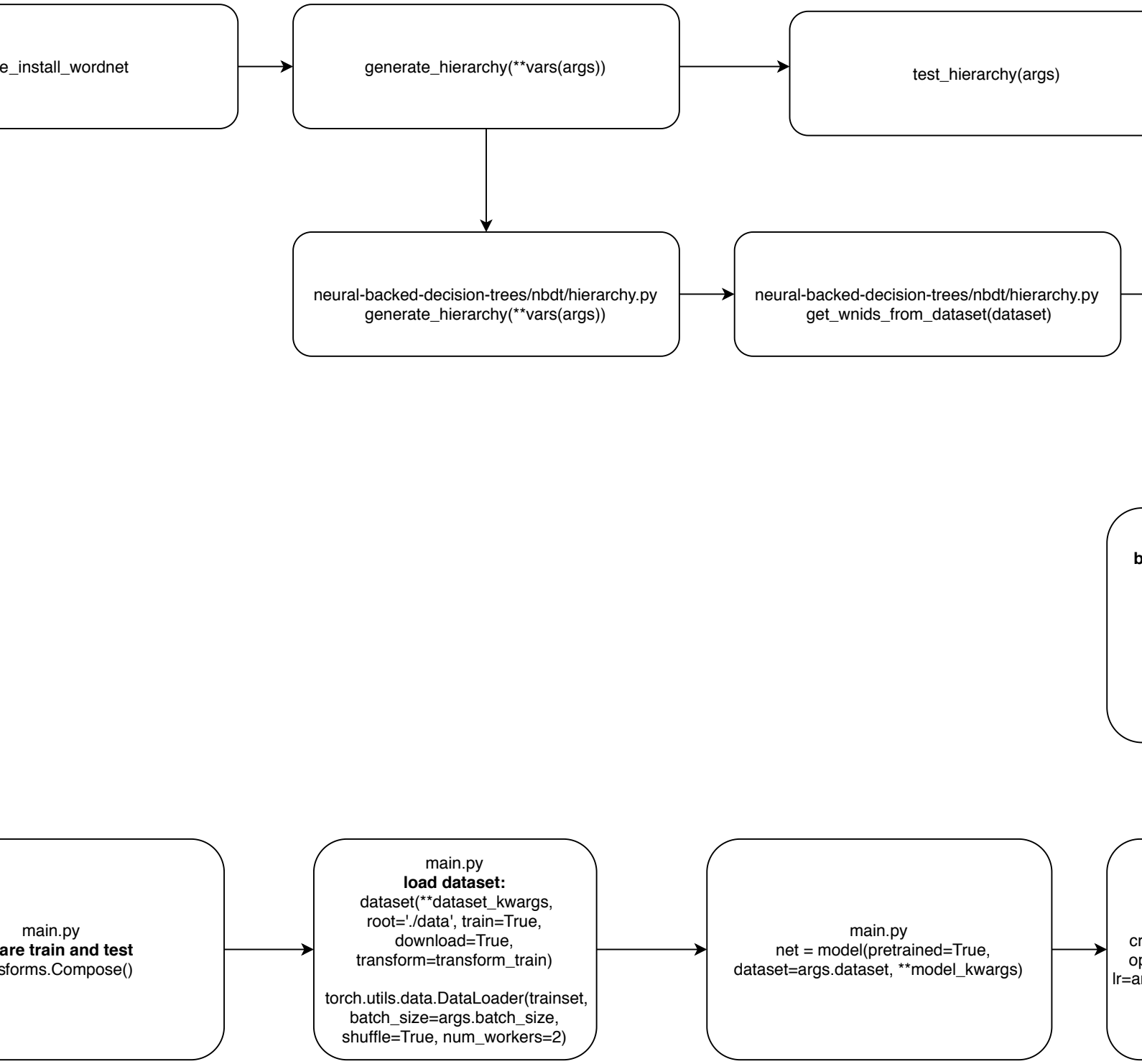
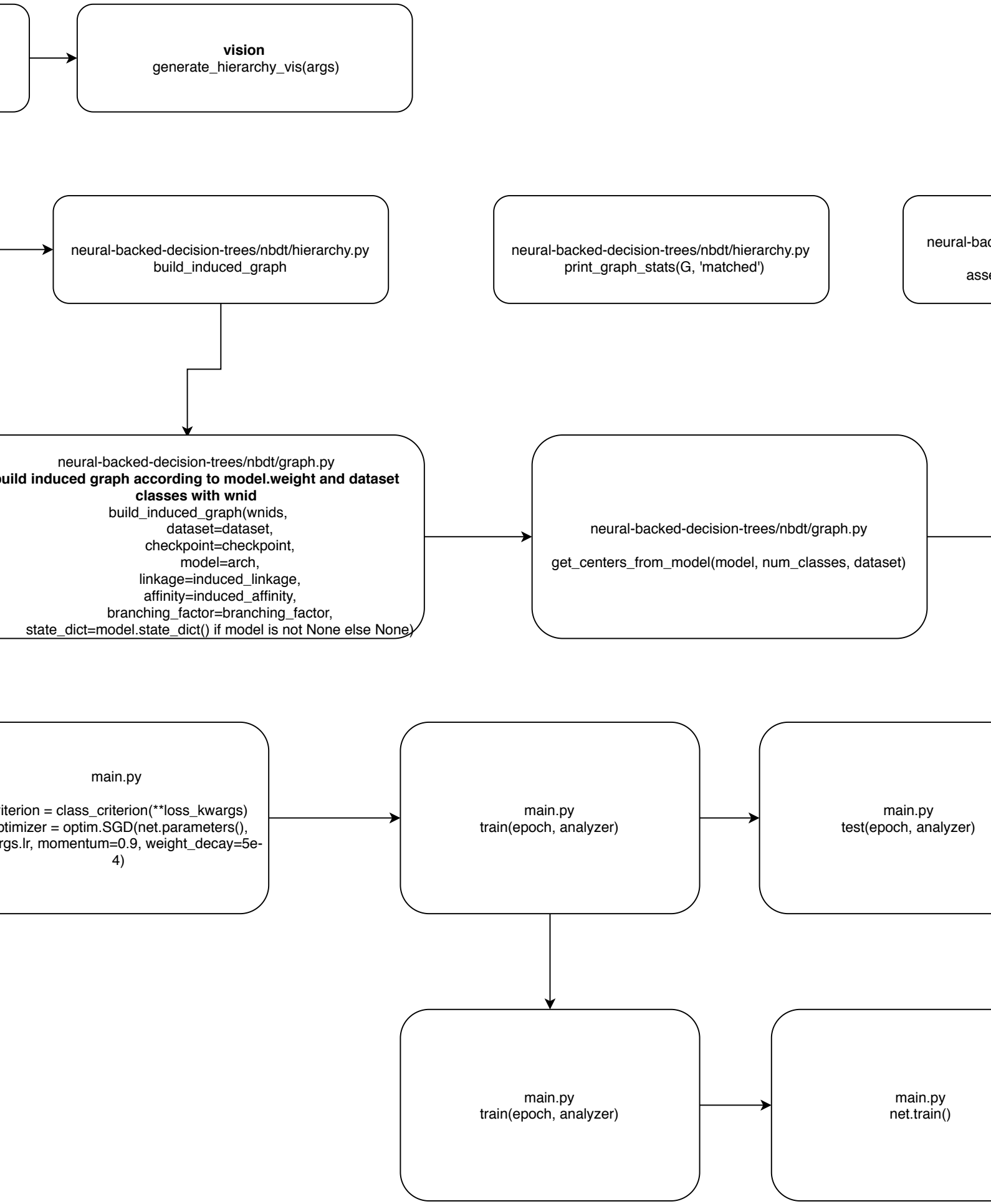
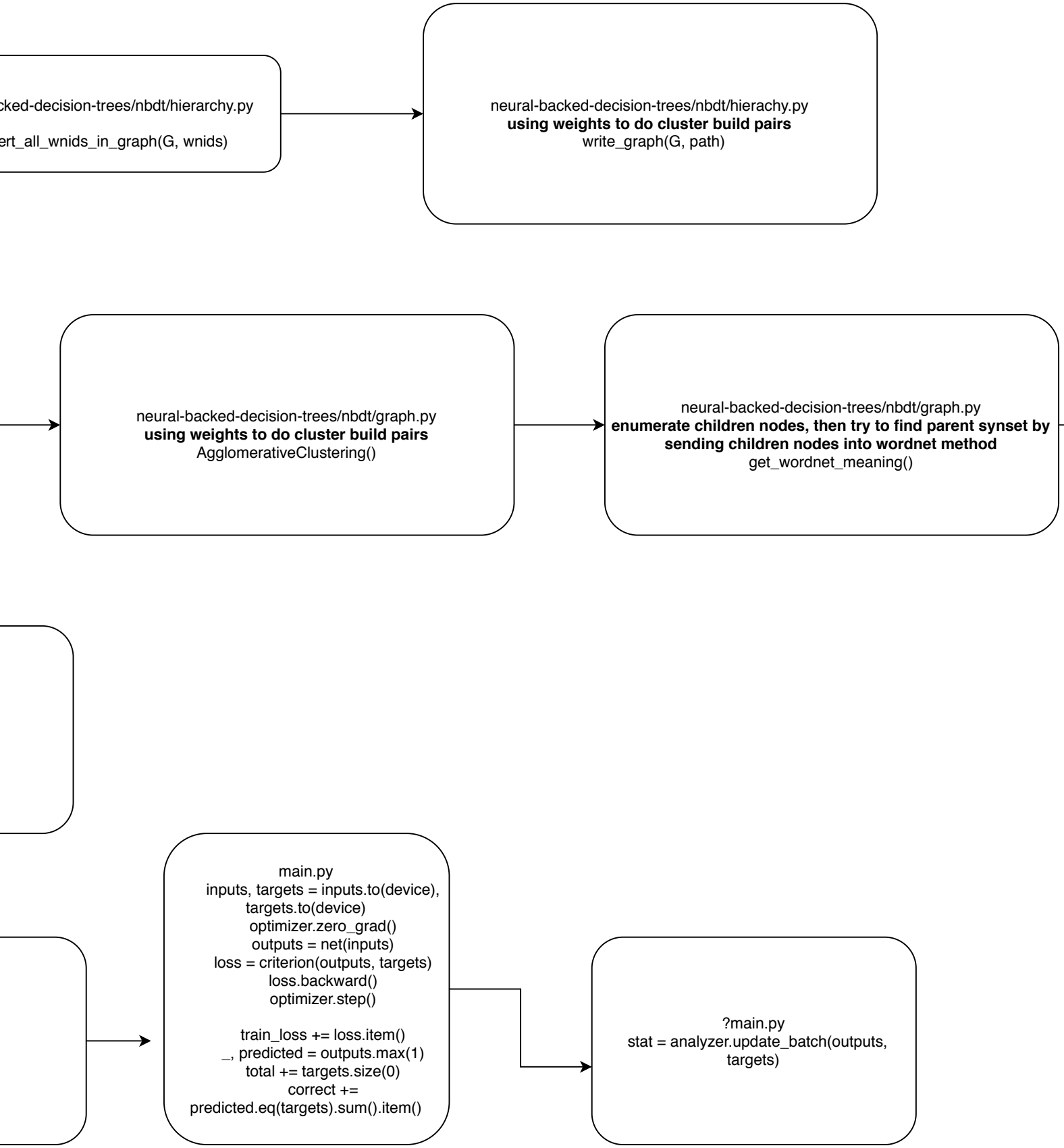

commad nbdt-hierarchy --arch=wrn28_10_cifar10 --dataset=CIFAR10
from nbdt.hierarchy import generate_hierarchy, test_hierarchy, generate_hierarchy_vis
from nbdt.graph import get_parser
from nbdt.utils import maybe_install_wordnet
maybe
python main.py --lr=0.01 --dataset=CIFAR10 --arch=wrn28_10_cifar10 --hierarchy=induced-wrn28_10_cifar10 --pretrained --loss=SoftTreeSupLoss
prep
trans
dataset = getattr(data, args.dataset)
model = getattr(models, args.arch);
Note: The model returned by getattr() is from pytorchcv.models.wrn_cifar.wrn28_10_cifar10(in case).
net is returned by __call__() method.
net = model(pretrained=True, dataset=args.dataset, **model_kwargs);
class_criterion = getattr(loss, args.loss)
class_analysis = getattr(analysis, args.analysis or 'Noop')
analyzer = class_analysis(**analyzer_kwargs)
net = model(pretrained=True, dataset=args.dataset, **model_kwargs)


e_install_wordnet
generate_hierarchy(**vars(args))
test_hierarchy(args)
neural-backed-decision-trees/nbdt/hierarchy.py
generate_hierarchy(**vars(args))
neural-backed-decision-trees/nbdt/hierarchy.py
get_wnids_from_dataset(dataset)
main.py
are train and test
sforms.Compose()
main.py
load dataset:
dataset(**dataset_kwargs,
root='./data', train=True,
download=True,
transform=transform_train)
torch.utils.data.DataLoader(trainset,
batch_size=args.batch_size,
shuffle=True, num_workers=2)
main.py
net = model(pretrained=True,
dataset=args.dataset, **model_kwargs)


vision
generate_hierarchy_vis(args)
neural-backed-decision-trees/nbdt/hierarchy.py
build_induced_graph
neural-backed-decision-trees/nbdt/hierarchy.py
print_graph_stats(G, 'matched')
neural-backed-decision-trees/nbdt/graph.py
build induced graph according to model.weight and dataset classes with wnid
build_induced_graph(wnids,
dataset=dataset,
checkpoint=checkpoint,
model=arch,
linkage=induced_linkage,
affinity=induced_affinity,
branching_factor=branching_factor,
state_dict=model.state_dict() if model is not None else None)
neural-backed-decision-trees/nbdt/graph.py
get_centers_from_model(model, num_classes, dataset)
main.py
iterion = class_criterion(**loss_kwargs)
optimizer = optim.SGD(net.parameters(),
rgs.lr, momentum=0.9, weight_decay=5e-4)
main.py
train(epoch, analyzer)
main.py
test(epoch, analyzer)
main.py
train(epoch, analyzer)
main.py
net.train()


cked-decision-trees/nbdt/hierarchy.py
ert_all_wnids_in_graph(G, wnids)
neural-backed-decision-trees/nbdt/hierachy.py
using weights to do cluster build pairs
write_graph(G, path)
neural-backed-decision-trees/nbdt/graph.py
using weights to do cluster build pairs
AgglomerativeClustering()
neural-backed-decision-trees/nbdt/graph.py
enumerate children nodes, then try to find parent synset by sending children nodes into wordnet method
get_wordnet_meaning()
main.py
inputs, targets = inputs.to(device), targets.to(device)
optimizer.zero_grad()
outputs = net(inputs)
loss = criterion(outputs, targets)
loss.backward()
optimizer.step()
train_loss += loss.item()
_, predicted = outputs.max(1)
total += targets.size(0)
correct += predicted.eq(targets).sum().item()
?main.py
stat = analyzer.update_batch(outputs, targets)

neural-backed-decision-trees/nbdt/graph.py
return G

1. fbresnet152 last_linear.weight.
CIFAR10. output.weight 640*10
2.net = model = wrn28_10_cifar10 from pytorchcv
3. ?生成的tree哪里用到了。
net.train()