```
net = Network()  # define your neural network (torch.nn.Module)

inference_method = 'vi'  # 'mc_dropout', 'sgld', 'la'

#---- Variational Inference ----#
if inference_method == 'vi':

    model = vi.Model(net, ND=1000, prior_sig=1.0, kld=1.0)
        # ND = training data size, prior_sig = prior's sigma,
        # kld = KL discount factor

    # training
    optim = torch.optim.SGD(model)  # optimizer for the posterior model
    vi.train(model, optim, train_data_loader)

    # test prediction with report error calibration scores
    vi.evaluate(model, test_data_loader, nst=5)
        # nst = number of MC samples from the posterior (test predictive distribution)
    calibration.analyze(model, test_data_loader, num_bins=20, temperature=1)
        # calibration (ECE, MCE, NLL, reliability plot)


#---- MC-Dropout ----#
elif inference_method == 'mc_droppout':

    model = mc_dropout.Model(net, ND=1000, prior_sig=1.0, pdrop=0.1, kld=1.0)
        # pdrop = dropout prob

    # training
    optim = torch.optim.SGD(model)  # optimizer for the posterior model
    mc_dropout.train(model, optim, train_data_loader)

    # test prediction with report error calibration scores
    mc_dropout.evaluate(model, test_data_loader, nst=5)
    calibration.analyze(model, test_data_loader, num_bins=20, temperature=1)


#---- SGLD ----#
elif inference_method == 'sgld':

    model = sgld.Model(net, ND=1000, prior_sig=1.0, Ninflate=1.0, nd=1.0, burnin=5, thin=10)
        # Ninflate = data inflation factor, nd = noise discount factor,
        # burnin = number of burn-in epochs, thin = number of thinning steps

    # training
    optim = torch.optim.SGD(model)  # optimizer for the posterior model
    sgld.train(model, optim, train_data_loader)

    # test prediction with report error calibration scores
    sgld.evaluate(model, test_data_loader, nst=5)
    calibration.analyze(model, test_data_loader, num_bins=20, temperature=1)


#---- Laplace Approximation ----#
elif inference_method == 'la':

    model = la.Model(net, ND=1000, prior_sig=1.0, Ninflate=1.0)

    # training
    optim = torch.optim.SGD(model)  # optimizer for the posterior model
    post_var = la.train(model, optim, train_data_loader)
        # find the MAP first; then compute the posterior variance

    # test prediction with report error calibration scores
    la.evaluate(model, post_var, test_data_loader, nst=5)
    calibration.analyze(model, test_data_loader, num_bins=20, temperature=1)
```