Initialize:

student latent sizes: L = [48, 40, 32, 24, 16, 12, 8, 4]

currentStage = 1

batchBufferX, batchBufferY = empty

batchSize = <large, e.g., 64>

maxEpochs = <safe halting value>

convergenceTolerance = <hyperparameter>

learningRate = <initial LR>

Initialize MLP Encoder with first stage latent dimension

Initialize Adam optimizer state

Loop over stages (while currentStage <= length(L)):

stageLatentDim = L[currentStage]

If stage change:

Expand MLP Encoder for new latent dimension

Reinitialize Adam optimizer (moments reset)

Reset batch buffers, step counters, epoch counters

epoch = 0

stageConverged = false

While not stageConverged and epoch < maxEpochs:

epoch += 1

Loop over input signal frames (or streaming samples):

1. Add current frame to batch buffer

batchBufferX.append(inputFrame)

batchBufferY.append(teacherFilteredFrame)

2. If batch buffer full:

a. Forward Pass:

- studentTaps = MLP Encoder(teacherTaps)

- studentFilteredOutput = Apply student filter to batch input

- teacherFilteredOutput = Apply teacher filter to same input

b. Compute Batch Loss:

- batchPerformanceLoss = MSE(studentFilteredOutput, teacherFilteredOutput)

c. Backpropagate:

- Compute gradients through student filter back to MLP Encoder

- Update MLP weights using Adam optimizer

d. Clear batch buffers

End of epoch

Check Stage Convergence:

- Compare current epoch performance loss with previous epoch

- If |loss\_epoch - loss\_previous\_epoch| < convergenceTolerance:

stageConverged = true

- Else:

continue to next epoch

End While (stage loop)

Move to next stage:

currentStage += 1

End Loop (all latent stages done)

Outputs:

- Trained MLP Encoder for all latent dimensions

- Student filters producing near-teacher performance