Algorithm 1: Pool and Classifier Helper functions

procedure POOL-INITIALIZATION(*dataset*, *validation_ratio*, *test_ratio*, *initial_lb_size*)
    idx_abs = arange(len(*dataset*))
    idx_train, rest_data = train_test_split(idx_abs, test_size=*validation_ratio* + *test_ratio*)
    idx_test, idx_val = train_test_split(rest_data, test_size=*validation_ratio*/(*test_ratio*+*validation_ratio*))
    idx_ini_label = np.random.choice(idx_train, *initial_lb_size*)
end procedure
procedure GET-FOLDS(*initially_labeled_in_fold_size*)
    for *no_folds* **do**
        create a fold by randomly selecting *initially_labeled_in_fold_size* indecies from *idx_ini_label*
        add the *idx_newly_labeled* to this fold
    end for
    return the array of folds
end procedure
procedure CLASSIFIER-INITIALIZATION(*weight_decay*, *dropout_rate*)
    Define the layers
    Define the loss function
    Define the optimizer
    Define the metric (accuracy)
    Initialize weigths (using *xavier_uniform*)
end procedure
procedure GET-MODEL(*latest*)
    if not *latest* **then**
        ▷ This is the case when the saved model is requested for training
        if we do not have a copy of the model, initialize it using **Classifier-Initialization** and save the *initial_model_state*
        get a model from **Classifier-Initialization** and load the *initial_model_state*
        set the drop_out_rate for all drop out layers with the value saved in the class
        re-instantiate the model's optimizer with the weight_decay value saved in the class
    else
        ▷ This is the case when the best trained model is requested for prediction (by acquisition function)
        get a model from **Classifier-Initialization** and load the *latest_tuned_model_state*
    end if
    return the model
end procedure

Algorithm 2: HPO Helper functions

procedure FIT(*trainandvalidationloaders*, *model*)
  for *epochs* **do**
    Train the *model*
    Validate the *model*
  end for
  return the average validation loss over all *epochs*
end procedure
procedure OPTIMIZE
  Get the suggested drop_out_rate and weight_decay from *optuna* and save them inside the class
  for *train_fold* in **get-folds do**
    Get the model from **get-model**
    Get the validation loss from **fit**
    compute the average validation loss for the folds so far
    report the *averge_val_loss* so far with the fold number to *optuna*
  end for
  return the final *averge_val_loss*
end procedure
procedure HPO(*n_trials*)
  create an *Optuna* study (minimize)
  ▷ We use validation loss to report back to *optuna*
  HPO using *optuna* and **optimize** with *n_trials*
  return the best hyperparameters
end procedure
procedure OPT-MODEL-TESTING(*weight_decay*, *drop_out_rate*)
  Get the model from **get-model**
  traing and validate (using the test dataset) with **fit**
  set the *latest_tuned_model_state*
  return the test_avg_loss and test_metrics
end procedure

Algorithm 3: Active Learning

procedure ACTIVE LEARNING(*budget*,)
  Perform **Pool-Initialization**
  for *budget* **do**
    Get the best Hyperparameters by **HPO**
    Do **Opt-Model-Testing** using the best hyperparameters
    query the next index from *acquisition_function* and
      add it to the *idx_newly_labeled*
    Log the results
  end for
  Logging and Visualization
end procedure