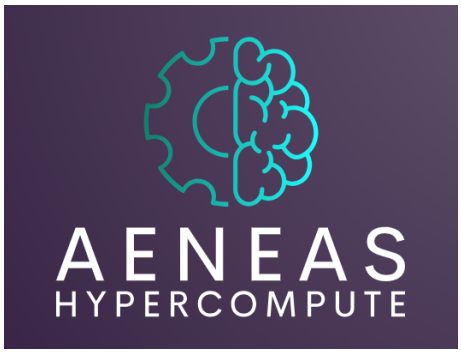

Aeneas HyperCompute Reference Manual

Version 3.0.0

Marco Angioli[1], Saeid Jamili[2]

[1]Contact: marco.angioli@uniroma1.it
[2]Contact: saeid.jamili@uniroma1.it

Contents

1 Library Documentation

AeneasHDC offers a rich array of configuration options for crafting HDC classification models. The python class encompasses versatile settings for the HV size, type, and density, allowing for fine-tuned hyperspace characteristics. Multiple encoding techniques for classification tasks are provided, including record-based, n-gram, and random-projection methods. LevelVectors can be generated using Linear, Approximately Linear, or Thermometer techniques. Moreover, various post-training clippings can be applied to the resulting ClassHVs, including options to make them binary, ternary, powers of two, and quantized. Additionally, AeneasHDC facilitates retraining, supporting state-of-the-art (SOA) learning rate strategies. Finally, users can choose from different similarity measures in the inference stage.

Attribute	**Description**
Dimensionality	Size of the HV.
HV_type	HV type (e.g., bipolar, binary).
density	HVs density (e.g., dense or sparse).
sparsity_factor	HV percentage sparsity (number of ones)
Number_of_features	Number of features in the dataset.
Number_of_levels	Number of levels for encoding.
num_classes	Number of classes in the dataset.
LevelTechnique	Technique used for level encoding.
similarity	Adopted Similarity measure (e.g., Hamming distance).
encoding_technique	Spatial encoding technique
n_gram	Temporal encoding (0: disabled, 1: enabled)
n_gram_size	Size of the n-grams.
clipping_encoding	Clipping technique used for encoding.
clipping_class	Clipping technique used for classes.
quantization_range	Range for quantization (min, max).
epochs	Number of retraining epochs.
lr_max	Maximum learning rate.
lr_decay	Learning rate decay method.
beta_lr	Beta learning rate.

Table 1: Attributes of the HDC Class

1.1 Class Methods

All features and settings of the AeneasHDC model are specified in the ‘config.py‘ file. This file determines the behavior of the class methods outlined

below. Ensure that the ‘config.py‘ file is configured to your requirements before using these methods.

1.1.1 generate_LevelVector

Description: Generate the LevelHVs used for encoding the scalar value of a feature.
Configurable Parameters: HD_LV_TYPE in config.py file.

```
model.generate_LevelVector()
```

1.1.2 Similarity

Description: Computes the similarity between two hypervectors.
Configurable Parameters: HD_SIMI_METHOD in config.py file.
Input Parameters:

- HV1, HV2: Hypervectors whose similarity is to be computed.

```
model.similarity(HV1, HV2)
```

1.1.3 Bind

Description: Performs the binding operation between two hypervectors.
Configurable Parameters: None specific in config.py for this method. The arithmetic implementation of this operation depends by the HD_DATA_TYPE
Input Parameters:

- HV1, HV2: Hypervectors to be bound together.

```
model.bind(HV1, HV2)
```

1.1.4 Permutation

Description: Applies a circular shift (permutation) to a hypervector.
Configurable Parameters: None specific in config.py for this method.
Input Parameters:

- HV: The hypervector to be permuted.
- positions: Number of positions to shift.

```
model.permutation(HV, positions)
```

1.1.5 Bundle

Description: Combines two hypervectors using the bundling operation.
Configurable Parameters: None specific in config.py for this method. The arithmetic implementation of this operation depends by the HD_DATA_TYPE.
Input Parameters:

- HV1, HV2: Hypervectors to be bundled.

```
model.bundle(HV1, HV2)
```

1.1.6 Clip

Description: Performs clipping on a hypervector, supporting various clipping types.
Configurable Parameters: CLIPPING_TYPE in config.py file.
Input Parameters:

- HV1: The hypervector to be clipped.
- min_val, max_val: Range for quantized clipping.
- clipping_type: Type of clipping to be applied.
- bundled_HVs: Number of hypervectors bundled before clipping.

```
model.clip(HV1, min_val, max_val, clipping_type, bundled_HVs)
```

1.1.7 Context_dependent_thinning

Description: Applies Context-Dependent Thinning (CDT) to a hypervector.
Configurable Parameters: None. Always performed when HD_MODE=SPARSE.

Input Parameters:

- Z: The hypervector to be thinned.
- bundled_HVs: Number of hypervectors bundled.
- thinning_steps: Number of steps for the thinning process.

```
model.context_dependent_thinning(Z, bundled_HVs, thinning_steps)
```

1.1.8 Encoding

Description: Encodes a feature vector into a hypervector using selected encoding techniques. It performs both the temporal encoding and the spatial one.
Configurable Parameters: ENCODING_TECHNIQUE, N_GRAM, N_GRAM_SIZE
Input Parameters:

- feature_vector: The feature vector to be encoded.
- BaseHVs: Base Hypervectors for encoding.
- LevelHVs: Level Hypervectors for encoding.
- quant_levels: Quantization levels for feature values.
- clipping_type: Clipping type to be used post-encoding.
- encoding_technique: The encoding technique to be used.

```
model.encoding(feature_vector, BaseHVs, LevelHVs, quant_levels,
    clipping_type, encoding_technique)
```

1.1.9 train

Description: Trains the HDC model using training data and labels.
Configurable Parameters: N_GRAM, DS_TRAIN_SIZE, CLIPPING_CLASS.
Input Parameters:

- train_data: Training data array.
- train_labels: Corresponding labels for the training data.
- verbose: Verbosity level.

- ds_min_value: Minimum value in the dataset for quantization.
- ds_max_value: Maximum value in the dataset for quantization.
- clip_class: Clipping type for classes.

```
model.train(train_data, train_labels, verbose, ds_min_value,
    ds_max_value, clip_class=config.CLIPPING_CLASS)
```

1.1.10 predict

Description: Makes predictions on test data using the trained HDC model.
Configurable Parameters: N_GRAM, DS_TEST_SIZE.
Input Parameters:

- test_data: Test data array.
- test_labels: Corresponding labels for the test data.
- BaseHVs: Base Hypervectors.
- LevelHVs: Level Hypervectors.
- ClassHVs: Class Hypervectors.
- quant_levels: Quantization levels.

```
model.predict(test_data, test_labels, BaseHVs, LevelHVs, ClassHVs,
    quant_levels, verbose=1)
```

1.1.11 retrain

Description: Retrains the HDC model using additional data to improve accuracy. The method adapts the model to better fit the data by adjusting the Class Hypervectors based on retraining data and labels.
Configurable Parameters: epochs, lr_max, lr_decay.
Input Parameters:

- retrain_data: Retraining data array.
- retrain_labels: Corresponding labels for the retraining data.
- test_data: Test data array for validation.

- test_label: Corresponding labels for the test data.
- BaseHVs: Base Hypervectors.
- LevelHVs: Level Hypervectors.
- ClassHVs: Class Hypervectors.
- quant_levels: Quantization levels.
- starting_accuracy: Initial accuracy before retraining.

```
model.retrain(retrain_data, retrain_labels, test_data, test_label,
    BaseHVs, LevelHVs, ClassHVs, quant_levels, verbose=1,
    starting_accuracy=0)
```

2 Test Environment

This document details the ‘change_config‘ function and various test functions in the Aeneas project, designed for configuring and evaluating Hyperdimensional Computing (HDC) models.

2.1 change_config Function

Description The ‘change_config‘ function dynamically modifies the HDC model’s configuration, enabling adjustments to parameters such as HDV size, sparsity, hardware parallelism, and more. Table 2 list the function parameters and their description.

Parameter	Description
dim	Hyperdimensional Vector (HDV) size.
hd_mode	Mode of the HDV.
sparsity_factorx10	Sparsity factor for the HDV.
parallel_features	Number of features processed in parallel.
parallel_classes	Number of classes compared in parallel.
frame	Number of HV elements processed concurrently.
hvtype	Type of HDV.
similarity	Similarity measure used.
clip	Clipping technique.
lv_mode_model	Level Vector mode on the model.
bv_mode	Base Vector generation mode.
lv_mode	Level Vector generation mode.
cv_mode	Class Vector generation mode.
hw_train	Enable or disable hardware training.
retrain_in_hw	Enable or disable retraining on hardware.
encoding_technique	Encoding technique used.
n_gram_size	Size of n-grams for temporal encoding.
lr	Learning rate.
lr_decay	Learning rate decay.

Table 2: Input parameters for the change_config function

2.2 Test Functions

The test functions evaluate the HDC model under different configurations, providing insights into performance and behaviour. Each test outputs a

report including metrics like resource utilization, accuracy, processing speed, and other key performance indicators. These reports help identify optimal configurations and understand the model's behaviour under various settings. Table 3 summarizes all the actual implemented tests.

Test Name	Tested Feature
Test 1	HDV Size (dim)
Test 2	HV Sparsity (sparsity_factorx10)
Test 3	HV Type (hvtype)
Test 4	Learning Rate (lr)
Test 5	Frame Parallelism (frame)
Test 6	Feature Parallelism (parallel_features)
Test 7	Class Parallelism (parallel_classes)
Test 8	LV Mode on Model (lv_mode_model)
Test 9	Base Vector Generation Mode (bv_mode)
Test 10	Class Vector Generation Mode (cv_mode)
Test 11	Training on Hardware (hw_train)
Test 12	Retraining on Hardware (retrain_in_hw)
Test 13	Spatial Encoding mode (encoding_technique)
Test 14	Temporal Encoding N-Gram Size (n_gram_size)
Test 15	Clipping Technique (clip)
Test 16	Combined Parallelism Effects
Test 17	Various Vector Generation Modes
Test 18	Clipping and Similarity Combinations
Test 19	HV Types, Clipping, and Similarity Measures

Table 3: Overview of Test Functions

Test 1: HV Dimension (HD_DIM) *Purpose*: Assesses the impact of different HDV sizes.
Options: Varying HDV sizes, such as 512, 1024, 2048.

```
test_hd_dim(aeneas, hd_dim_range=[512, 1024, 2048])
```

Test 2: HV Sparsity (sparsity_factorx10)
Purpose: Evaluates different sparsity levels.
Options: Varying sparsity factors, like 1, 5, 10 (multiplied by 10).

```
test_hd_sparsity(aeneas, sparsity_range=[1, 5, 10])
```

Test 3: HV Type (hvtype)
Purpose: Analyzes performance with different HV types.
Options: ‘BINARY‘ (0) and ‘BIPOLAR‘ (1).

```
test_hd_type(aeneas, hd_type_options=['BINARY', 'BIPOLAR'])
```

Test 4: Learning Rate (LR)
Purpose: Investigates the effect of learning rates.
Options: Various learning rates, e.g., 0.1, 0.01, 0.001.

```
test_learning_rate(aeneas, lr_values=[0.1, 0.01, 0.001])
```

Test 5: Frame Parallelism (FRAME)
Purpose: Tests different numbers of HV elements processed in parallel.
Options: Varying frame sizes, e.g., 256, 512.

```
test_frame_parallelism(aeneas, frame_range=[256, 512], dim=1000)
```

Test 6: Feature Parallelism (PARALLELISM_FEATURES)
Purpose: Assesses parallel processing of features.
Options: Different levels of feature parallelism, e.g., 1, 2.

```
test_feature_parallelism(aeneas, parallel_features_range=[1, 2])
```

Test 7: Class Parallelism (PARALLELISM_CLASS)
Purpose: Evaluates parallel processing of classes.
Options: Different levels of class parallelism, e.g., 2, 4.

```
test_class_parallelism(aeneas, parallel_classes_range=[2, 4])
```

Test 8: Level Vector Mode on Model (LV_MODE_MODEL)
Purpose: Tests different Level Vector modes.
Options: Modes like ‘LINEAR‘, ‘APPROX‘, ‘THERMOMETER‘.

```
test_lv_mode_model(aeneas, lv_mode_options=['LINEAR', 'APPROX'])
```

Test 9: Base Vector Generation Mode (BV_MODE)
Purpose: Investigates Base Vector generation methods.
Options: Modes like ‘INT_MEM‘, ‘EXT‘, ‘PERMUTATION‘, ‘RND_GEN‘.

```
test_bv_mode(aeneas, bv_mode_options=['INT_MEM', 'EXT'])
```

Test 10: Class Vector Generation Mode (CV_MODE)
Purpose: Analyzes different Class Vector generation modes.
Options: Modes like ‘INT_MEM‘, ‘EXT‘, ‘HDL_GEN‘.

```
test_cv_mode(aeneas, cv_mode_options=['INT_MEM', 'EXT'])
```

Test 11: Training on Hardware (TRAIN_ON_HW)
Purpose: Assesses hardware training’s effect.
Options: Enable (1) or disable (0) hardware training.

```
test_hw_train(aeneas, hw_train_options=[0, 1])
```

Test 12: Retraining on Hardware (RETRAIN_ON_HW)
Purpose: Tests the impact of retraining on hardware.
Options: Enable (1) or disable (0) retraining on hardware.

```
test_retrain_in_hw(aeneas, retrain_in_hw_options=[0, 1])
```

Test 13: Spatial Encoding (encoding_technique)
Purpose: Evaluates different spatial encoding techniques.
Options: Techniques like ‘ENCODING_RECORD‘, ‘ENCODING_NGRAM‘.

```
test_spatial_encoding(aeneas, encoding_technique_options=['
    ENCODING_RECORD', 'ENCODING_NGRAM'])
```

Test 14: Temporal Encoding (N_GRAM_SIZE)
Purpose: Investigates different n-gram sizes for temporal encoding.
Options: Various n-gram sizes, e.g., 2, 3.

```
test_temporal_encoding(aeneas, n_gram_size_options=[2, 3])
```

Test 15: Clipping Techniques (clip)
Purpose: Analyzes effects of different clipping techniques.
Options: Methods like ‘CLIPPING_BINARY‘, ‘CLIPPING_TERNARY‘, ‘CLIPPING_QUANTIZED‘.

```
test_clipping_techniques(aeneas, clipping_options=['CLIPPING_BINARY',
     'CLIPPING_TERNARY', 'CLIPPING_QUANTIZED'])
```

Test 16: Combined Parallelism
Purpose: Evaluates the combined effect of class, feature, and element parallelism.
Options: Various combinations of parallelism degrees.

```
test_comb_parallelism(aeneas, parallel_features_range=[1, 2],
    parallel_classes_range=[2, 4], frame_range=[256, 512])
```

Test 17: Generation Techniques

Purpose: Explores vector generation modes and their impact.
Options: Combinations of different generation techniques.

```
test_generation(aeneas, dim_range=[1000, 2000], bv_mode_range=['
    INT_MEM', 'EXT'], lv_mode_range=['INT_MEM', 'EXT'], cv_mode_range
    =['INT_MEM', 'EXT'])
```

Test 18: Clipping and Similarity Measures

Purpose: Evaluates combinations of clipping techniques and similarity measures.
Options: Various combinations of clipping and similarity measures.

```
test_clipping_similarity(aeneas, clip_range=['CLIPPING_BINARY', '
    CLIPPING_TERNARY'], sim_range=['SIMI_COS', 'SIMI_DPROD'])
```

Test 19: HV Type, Clipping, and Similarity Measures

Purpose: Assesses the combined influence of HV types, clipping techniques, and similarity measures.
Options: Combinations of HV types, clippings, and similarities.

```
test_type_clipping_similarity(aeneas, hvtype_range=['BINARY', '
    BIPOLAR'], clip_range=['CLIPPING_BINARY', 'CLIPPING_TERNARY'],
    sim_range=['SIMI_COS', 'SIMI_DPROD'])
```