

## Espam-AI manual

### Requirements & Installation:

See README.md at the espamAI root folder or at <https://git.liacs.nl/lerc/espam>

### Execution from command line (after the tool is configured):

1. cd <project root directory>
2. cd bin
3. ./espam [options]

### Execution from executable jar (after espam.jar is made by make jar-exec command):

1. cd <jar directory>
2. java -jar espam.jar [options]

### Input:

- Path to single DNN model in .onnx or .json format + options (see below) *or*
- Path to single CSDF model in .json format *or*
- Directory with several DNN models (not recommended for large DNN models)

### Output:

- DNN model evaluation results in .json format printed to console *or/and*
- Files, generated from DNN model and intended for CSDF-models for CSDF-models processing tools such as Sesame/DARTS/SDF3

### Possibilities:

- evaluation : Evaluation of DNN/CSDF model in terms of power/performance
- generation : Generation of files intended for CSDF-models processing tools such as Sesame/DARTS/SDF3\

### Tool running progress:

(1) *Model(s) reading*: read input model(s)

(2) *Model(s) conversion*:

- conversion of input .onnx/json DNN model(s) to internal Network model(s) (initiated automatically after onnx/json model reading)
- conversion of internal Network model(s) to CSDF model(s): (initiated automatically, if evaluation flag is set *or/and* one or multiple \*-csdf output files generation flags are set)

(3) *Output files generation [optional]*: generates for input model output files, according to set files generation options.

(4) *Model evaluation [optional]*: evaluates one or several input models in terms of power/performance, by means of the DARTS/SDF3 tool.

**TODO direct interface to SDF3 tool is not presented for current moment**

## Command-line options

Command-line options that take arguments

option	abbr	arguments	example
<b>General options</b>			
--generate	-g	Path to input DNN model	--generate ./tests/lenet.json
--evaluate	-e	Path to input DNN model	--evaluate ./tests/lenet.json
<b>Model representation options (by default --layer-based option is set)</b>			
--layer-based	-lb	none	-lb
--neuron-based	-nb	none	-nb
--block-based	-bb	number of blocks	--bb 20
--split-step [optional], for -bb models only	none	Number of child nodes, obtained after one layer splitting	--split-step 4
<b>Hardware specification options</b>			
--time-spec	none	Path to operators times specification	--time-spec ./time_spec.json
--energy-spec		Path to model energy specification	--energy-spec.json ./energy_spec.json
<b>Additional options</b>			
--dot-w	none	Width of generated .dot images in pixels	--dot-w 7000

Command-line flags (The command-line options that are either present or not).

flag	abbr	description
<b>General flags</b>		
--help	-h	Printout help
--version	-v	Printout program version
--verbose	-V	Printout program progress information
--copyright	none	Printout copyright
Input model type flag (--in-dnn is set by default)		
--in-dnn	none	Input model is dnn model in .onnx or .json format
--in-csdf	none	Input model is csdf model in .json format
<b>Files generation flags</b>		
--json	none	Generate DNN graph in .json format
--dot	none	Generate DNN graph image in .dot format
--json-csdf	none	Generate CSDF graph in .json format for DARTS
--xml-csdf	none	Generate CSDF graph in .xml format for SDF3

<code>--dot-csdf</code>	none	Generate CSDF graph image in .dot format
<code>--sesame</code>	none	Generate code templates for Sesame
<code>--multiple-models</code>	-m	Process not a single model, but multiple models. In this case path after –generate or –evaluate general flag should be the directory with several models. <i>Note: not recommended, especially for large models</i>

### Files generation examples

For more information about files generation see comments, output files.

#### Example 1

*Command:*

```
./espam --generate ./tests/json/bvlc_alexnet_LB.json --sesame -lb
```

*Description:*

Generate layer-based CSDF graph for ./tests/json/bvlc\_alexnet\_LB.json DNN model. Provide it with Sesame application.

*Expected output:* files folder

<bvlc\_alexnet\_LB>

-app //sesame application in layer-based mode

#### Example 2

*Command:*

```
./espam --generate ./tests/json/lenet_NB.json --sesame --dot --json-csdf --xml-csdf --dot-csdf -nb
```

*Description:*

Generate neuron-based CSDF graph for ./tests/json/lenet\_NB.json DNN model. Provide it with Sesame application, .dot images of DNN and CSDF model, .json and .xml files of CSDF model.

*Expected output:* files folder

<lenet\_NB>

-app //contains sesame application in -nb mode

-dot //contains lenet\_NB.dot picture of DNN model

-sdfg //corresponding CSDF graph directory

-dot //contains lenet\_NB.dot picture CSDF model

-json //contains lenet\_NB.json CSDF model suitable for DARTS\

-xml //contains lenet\_NB.xml CSDF model suitable for SDF3

#### Example 3

*Command:*

```
./espam --generate ./tests/onnx/mnist.onnx --sesame --dot --json-csdf --xml-csdf --dot-csdf -bb -100 --split-step 4
```

*Description:*

Generate block-based CSDF graph for ./tests/onnx/mnist.onnx DNN model. Split model until it reaches <=100 blocks with –split-step = 4 (For more information about layers splitting see Comments, block-based models). Provide it with Sesame application, .dot images of DNN and CSDF model, .json and .xml files of CSDF model.

*Expected output:* files folder

<CNTKGraph\_NB>

- app //contains sesame application in -bb mode
- dot //contains CNTKGraph.dot picture of DNN model
- sdfg //corresponding CSDF graph directory
  - dot //contains CNTKGraph.dot picture of CSDF model
  - json //contains CNTKGraph.json CSDF model suitable for DARTS\
  - xml //contains CNTKGraph.xml CSDF model suitable for SDF3

## Model evaluation examples

### Example 1

*Command:*

```
./espam --evaluate ./tests/json/bvlc_alexnet_LB.json
```

*Description:*

Evaluate ./tests/json/bvlc\_alexnet\_LB.json DNN model in terms of power/performance in -lb mode, set by default.

*Expected output:*

```
@svetlana-Latitude-7280:/home/svetlana/jar_for_simon] $ java -jar protobuf_interface.jar --evaluate ./tests/json/bvlc_alexnet_LB.json -lb
{ "id": 0, "execution_time": 1.2997074085E10, "energy": 2.70885261651, "memory": 2.77690192E8, "processors": 0 }
```

### Example 2

*Command:*

```
./espam --evaluate ./tests/json/lenet_NB.json -nb
```

*Description:*

Evaluate ./tests/json/lenet\_NB.json DNN model in terms of power/performance in -nb mode.

*Expected output:*

```
{ "id": 0, "execution_time": 4361469.0, "energy": 1.93671545212, "memory": 278280.0, "processors": 0 }
```

### Example 3

*Command:*

```
./espam --evaluate ./tests/onnx/mnist.onnx -bb -100 --split-step 4
```

*Description:*

Evaluate ./tests/onnx/mnist.onnx DNN model in terms of power/performance. Split model until it reaches <=100 blocks with --split-step = 4 (For more information about layers splitting see Cooments, block-based models).

*Expected output:*

```
{ "id": 0, "execution_time": 9932160.0, "energy": 1.47208333333, "memory": 834240.0, "processors": 0 }
```

## Comments

### 1. Output files

Output files are generated in output models directory (output\_models in protobuf\_interface.jar root by default). For every model created a directory, named after the model. E.g. for DNN called "LeNet", the LeNet directory will be created.

Inside the folder there are files generated in accordance with the file generation flags. For all possible files the folder will have the following structure

The files structure:

<Model\_name>

- app //sesame application
- dot //DNN model in .dot format
- json //DNN model in .json format
- sdfg //corresponding CSDF graph directory
  - dot //CSDF model in .dot format
  - json //CSDF model in .json format for DARTS\
  - xml //CSDF model in .xml format for SDF3

*NOTE: if directory with this name already exist it will be overwritten!*

## 2. Block-based models

Block-based model takes on input layer-based DNN model and transforms it according to the following algorithm:

```
While (number of blocks<expected && split_up flag = true):  
    Layer bottleneck = find_bottleneck(); // bottleneck node search is preformed by DARTS  
    if (bottleneck can be split up)  
        evaluate number of layers after bottleneck layer splitting;  
        if (number of layers after splitting>expected)  
            return;  
        Split bottleneck into –split-step child Layers;  
        Split all Dependent layers (nonlinear/maxpool ones), following the bottleneck layer  
    else  
        return;
```

As transformation is performed over DNN models,files (.json / .dot etc) generated from the model in -bb mode will be different from ones for input (layer-based) DNN model. If .json for block-based model is already generated, it can be reused without additional splitting in -lb mode.

## 3. Recommendations and possible issues.

3.1. It is recommended to run tests after the tool is build.

**TODO provide run tests after tool installation**

3.2. Tool is not accounted on huge graph models, so it is not recommended to use -nb mode for large DNNs. Otherwise, tool may work really long or even fall with memory (JavaHeapSpace/Python) errors.

For the same reason, it is not recommended to use –generation option with too many flags and multiple-models processing (--evaluation or ---generation) for heavy models, especially in -nb mode

3.3. Tool may complain on ‘unsafe’ libraries such as protobuf-java.jar.

3.4. If tool is frozen, kill the corresponding command line process and try to use the same command with –verbose (v) option. It will be shown on which step (model reading, model conversion, model evaluation etc.) the problem occurred and more details will be given.

3.5. If CSDF model is sent on input (with flag `-in-csdf`) , no specific DNN features (such as weights and proper data formats) will be taken into account. Thus, for processing DNN models it is recommended to use DNN .json/.onnx files. JSON files might be even preferable due to their small size.