Lattice sensAI Neural Networks Training Environment

User Guide

FPGA-UG-02226-1.0

November 2024

Disclaimers

Lattice makes no warranty, representation, or guarantee regarding the accuracy of information contained in this document or the suitability of its products for any particular purpose. All information herein is provided AS IS, with all faults, and all associated risk is the responsibility entirely of the Buyer. The information provided herein is for informational purposes only and may contain technical inaccuracies or omissions, and may be otherwise rendered inaccurate for many reasons, and Lattice assumes no obligation to update or otherwise correct or revise this information. Products sold by Lattice have been subject to limited testing and it is the Buyer's responsibility to independently determine the suitability of any products and to test and verify the same. LATTICE PRODUCTS AND SERVICES ARE NOT DESIGNED, MANUFACTURED, OR TESTED FOR USE IN LIFE OR SAFETY CRITICAL SYSTEMS, HAZARDOUS ENVIRONMENTS, OR ANY OTHER ENVIRONMENTS REQUIRING FAIL-SAFE PERFORMANCE, INCLUDING ANY APPLICATION IN WHICH THE FAILURE OF THE PRODUCT OR SERVICE COULD LEAD TO DEATH, PERSONAL INJURY, SEVERE PROPERTY DAMAGE OR ENVIRONMENTAL HARM (COLLECTIVELY, "HIGH-RISK USES"). FURTHER, BUYER MUST TAKE PRUDENT STEPS TO PROTECT AGAINST PRODUCT AND SERVICE FAILURES, INCLUDING PROVIDING APPROPRIATE REDUNDANCIES, FAIL-SAFE FEATURES, AND/OR SHUT-DOWN MECHANISMS. LATTICE EXPRESSLY DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY OF FITNESS OF THE PRODUCTS OR SERVICES FOR HIGH-RISK USES. The information provided in this document is proprietary to Lattice Semiconductor, and Lattice reserves the right to make any changes to the information in this document or to any products at any time without notice.

Inclusive Language

This document was created consistent with Lattice Semiconductor’s inclusive language policy. In some cases, the language in underlying tools and other items may not yet have been updated. Please refer to Lattice’s inclusive language FAQ 6878 for a cross reference of terms. Note in some cases such as register names and state names it has been necessary to continue to utilize older terminology for compatibility.

Contents

Abbreviations in This Document

A list of abbreviations used in this document.

Abbreviation	Definition
LATTE	Lattice sensAI Neural Networks Training Environment
LSCQuant	Lattice sensAI Neural Networks Quantizer
DNN	Deep Neural Network
GPU	Graphics Processing Unit
WSL2	Windows Subsystem for Linux version 2
FPGA	Field Programmable Gate Arrays
SIGINT	Interrupt signal, usually sent to a command line process by hitting the keyboard keys CTRL-C
SIGTERM	Termination signal, usually sent to a process through an operating system task manager or the kill process command
CLI	Command Line Interface
YAML	Yet Another Markup Language
JSON	JavaScript Object Notation
WandB	Weights & Biases, a platform used for managing machine learning experiments and models
NumPy	Numerical Python
HDF5	Hierarchical Data Format version 5
PB	Protocol Buffer
ONNX	Open Neural Network Exchange
TFLite	TensorFlow Lite

1. Introduction

This document serves as an introductory guide for Lattice sensAI™ Neural Networks Training Environment (LATTE), covering its core features, concepts, workflows, installation procedures, and basic usage instructions. LATTE has been designed to standardize DNN training code to eliminate the repetitive task of writing *boilerplate* code whenever a new DNN model is developed. By providing a unified framework, LATTE streamlines the development process, ensuring consistency and efficiency across various projects.

1.1. Concepts

LATTE is first and foremost a Python package providing a set of Python classes and utility functions that simplifies the development of the training process of DNN models. It is also a command line program *latte* that will help perform the three most common tasks that need to be done when it comes to DNN models: training, testing, and converting models.

The mains concepts are as follows:

- The main interaction with LATTE is through the command line program; the program has two main inputs:
  - A path to a configuration file (written in YAML or JSON), or a path to a folder containing a file named: `config.yaml, config.yml or config.json.`
  - A path to a Python module a `.py` file or to a Python package, a folder that contains a file `__init__.py`, which contains the code or import some other code to provide to LATTE.
- The configuration file contains a few main sections which contain, among other things:
  - The DNN model to be trained, tested or converted, is specified by the name of the Python class or the function that should be used to instantiate the model, along with the required parameters.
  - The datasets for training and testing are specified by the names of the Python classes implemented by the user for loading the datasets and generating batches of data (dataset handlers).
  - The losses and metrics to be computed during training and testing can be standard Keras losses or metrics, or user-defined ones.
  - The hyper-parameters related for training and testing the model include batch size, number of epochs, optimizer, the learning rate and more.
  - The format to which the trained DNN model should be converted to.
- The configuration file contains all the parameters that describe a specific experiment, such as training, testing or converting and keeps them separate from the Python code.
  - This allows for easier experiment tracking, as creating a new experiment can be accomplished by copying an existing configuration file, giving the file a new name, and changing some of the parameters in the new file.
  - If LATTE is provided with a path to a configuration file, it will create a folder which has the same name as the file (minus the file extension) to store all the artefacts related to running the training, the testing, and the conversion, such as the train checkpoints, the best model checkpoints, the logs, the converted model files, etc.
  - If LATTE is provided with a path to folder that contains configuration file named `config.yaml`, `config.yml` or `config.json`, then the artifacts will be stored alongside the configuration file in that folder.
- The code in the Python module or package minimally consists of:
  - The model implementation as a function that returns a Keras Model, or as a subclass of the Keras Model class.
  - The implementation of the dataset handler class(es).
  - Some calls to LATTE's functions that register the user-defined classes and functions that implement the model, the dataset handlers, the custom losses and metrics if any.

1.2. Features

The important features of LATTE include:

- Separation of hyper-parameters and Python code.
- Automatic checkpoint saving occurs after each pass over the training dataset after each epoch during training.
- Automatic checkpoint restoration when restarting training, if existing checkpoints are detected.
- Ability to stop training at any time with the keyboard keys combination CTRL-C (SIGINT) in the CLI or by sending an interrupt signal (SIGTERM) with a task manager or the kill command.
- Automatic logging files generation when training, testing, and converting models.
- Automatic creation of Tensorboard summaries for all losses and metrics at each epoch, for training, validation, and testing
  - The learning rate is also logged at each epoch when using a learning rate scheduler.
- Support for Weights and Biases (WandB)
  - The losses, metrics, learning rate are logged at each epoch, the training checkpoints and converted models are uploaded as artifacts.
- Support for standard Keras loss and metrics that takes the targets and the predictions as inputs
  - Support custom losses and metrics that can also take the model's inputs and some per-sample metadata as inputs along with the targets and the predictions.
- Base Python classes to derive new classes from to create dataset handlers
  - Easily write dataset handlers by overloading only a few methods.
  - Base classes for writing dataset handler using TensorFlow operations or just plain Python or NumPy operations.
- Powerful callback system that allows for custom code to be executed at every major step during training, testing, and conversion of models.
- Support models conversion to HDF5 (Keras .h5 files), frozen PB, ONNX, and TFLite.
- Powerful configuration substitution system
  - A parameter value in the configuration file can refer to other parts of the configuration to avoid repetition and share some common values.
  - A parameter value can be set to the content of other YAML, JSON, or text files.
  - Can define parameters containing paths relative to the configuration file itself.
  - Can set a parameter value to some runtime value such as the number of steps per epoch.
- Default LATTE's behaviors can be altered by setting some environment variables.

Any parameter in the configuration file provided can be overwritten by providing a new value as an argument to the *LATTE* command.

2. Installation

2.1. Prerequisites

This section lists all the prerequisites and system requirements for using the LATTE package.

- Python version 3.8 or later
- TensorFlow version 2.7 to 2.15 (2.16 or later is not supported)

Note:

TensorFlow 2.10 is the last release that supports native GPU training on Windows. When training on Windows with a GPU, ensure the installed TensorFlow version does not exceed 2.10. There is also the option to perform training through WSL2. Please see Install TensorFlow with pip for more information.

2.2. Python Environment

LATTE is a Python package that needs to be installed in the Python environment used for training DNN. If an environment has already been created for other SensAI tools such as, the Lattice sensAI Neural Networks Quantizer (LSCQuant), activate that environment and then install LSCQuant in it. If there is no such environment, then it should be created. To create and manage Python environments more easily, especially for dependencies such as CUDA and CuDNN it is recommended to use Miniconda or Anaconda. Miniconda can be downloaded from this link: Installing Miniconda. After installation, you can create a new Python environment by running the following commands in a CLI:

```
conda create -n fpga python~=3.10.0
conda activate fpga
```

In this example, the environment is named *fpga*, but you can choose any name. If you plan to use a GPU for training DNNs, install CUDA and CuDNN in the environment using the *conda* command:

```
conda install -c conda-forge cudatoolkit=11.2 cudnn=8.1.0
```

From this point on, any other Python packages should be installed using the pip command in the environment. The TensorFlow package should be installed as follows:

```
pip install tensorflow~=2.10.0 numpy~=1.24.0
```

2.3. Package Installation

To install LATTE in the Python environment that is currently activated, the following command must be run from the folder containing the LATTE Wheel (.whl) file:

```
pip install -U ./latte-x.y.z-py3-none-any.whl
```

where x.y.z is dependent on the exact version of the package. On Windows, replace the character / by \ in the command above.

Running the latte command below should successfully display the installed version of LATTE:

```
latte -V
```

3. Command Line Usage

Installing the LATTE package in a Python environment will also install a command line program named latte in that environment; it will be available only when the Python environment is activated. This program is the main way of interacting with LATTE.

Running the command without specifying a *mode* (see below) can be used to get some useful information about LATTE. These options and their documentation can be seen by executing the command latte --help**.**

The latte command is used as follows:

```
latte MODE [OPTIONS] CONFIGURATION CODE
```

where

- *MODE* is either train, test, or convert; this specifies what will be performed by LATTE, i.e., perform model training, perform model testing, or perform model conversion.
- *OPTIONS* is an optional list of different command line options to use for the given mode; running LATTE MODE --help will display a list of the different options available for the given mode.
- *CONFIGURATION* is either a path to a YAML or JSON file, or a path to a folder containing config.yaml, config.yml or config.json.
- *CODE* is either a path to a Python file (.py), or a path to a Python package, which is a folder containing at the very least a file named __init__.py).

For more information about the content of the configuration file or the Python code, refer to the documentation provided alongside the LATTE Python package.

4. Workflow

An example of a typical DNN model development workflow with LATTE is presented in the following sections.

4.1. Python Code

The first step is to write the necessary Python code. This code should have the following:

- Model Creation:
  - Develop your model either as a function that returns a tf.keras.Model, functional or sequential or as a subclass of tf.keras.Model. Avoid hardcoding hyperparameters unless they are unlikely to change. Instead, make hyperparameters configurable through the function or class __init__ method, allowing easy adjustments via the configuration file.
- Dataset Handler:
  - Create a dataset handler to read and format the inputs and targets for training the model. Like for the model, include as many parameters as possible in the __init__ method.
- Custom Losses and Metrics:
  - If the required losses or metrics are not available in Keras, implement them yourself. A loss can be a function or a subclass of tf.keras.losses.Loss, while a metric should be a subclass of tf.keras.metrics.Metric, in most cases a subclass of tf.keras.metrics.Mean should be defined. The losses or metrics parameters should be arguments in the function or class __init__ method.
- Create a Python file to import and register the model, the dataset handler, and any custom losses or metrics using the appropriate function or decorator from the latte Python package.

The code below shows how a simple image classification model would look like. Here, a small Keras functional model is defined in a function my_model, which is registered in LATTE using the register_model function. It takes as input the shape of the input image, and the number of categories. For the dataset handler, a subclass MyDatasetHandler is derived from one of the available base classes in LATTE (here NumpyDatasetHandler). The only parameter passed to the dataset handler is the path to the dataset. The first mandatory method implemented is finding all the PNG images in images folder in the provided dataset path, and all text files in labels folder. The category ID of each image file is contained in a text which name corresponds to the image file. The second mandatory method loads the image and the corresponding label, then returns them.

```
from pathlib import Path
import cv2
from tensorflow.keras import layers, models
from latte.dataset import register_dataset_handler, NumpyDatasetHandler
from latte import register_model

@register_model
def my_model(input_shape, num_classes):
    input = layers.Input(shape=input_shape, name="input")
    x = layers.Conv2D(32, (3, 3), use_bias=False)(inputs)
    x = layers.BatchNormalization()(x)
    x = layers.ReLU()(x)
    x = layers.MaxPooling2D((2, 2))(x)
    x = layers.Conv2D(64, (3, 3), use_bias=False)(x)
    x = layers.BatchNormalization()(x)
    x = layers.ReLU()(x)
    x = layers.Flatten()(x)
    x = layers.Dense(num_classes, activation=None)(x)
    model = models.Model(inputs=[input], outputs=[x])
```

```
    return model

@register_dataset_handler
class MyDatasetHandler(NumpyDatasetHandler):
    def __init__(self, data_path, **kwargs):
        super().__init__(**kwargs)
        self._data_path = Path(data_path)

    def _get_input_data(self):
        image_files = sorted(list(self._data_path.glob("images/*.png")))
        label_files = sorted(list(self._data_path.glob("labels/*.txt")))
        return image_files, label_files

    def _process_sample(self, inputs, targets, metadata):
        image = cv2.imread(inputs, cv2.IMREAD_GRAYSCALE)
        image = image.astype(np.float32) / 255.0
        with open(targets, mode="rt") as label_file:
            label = np.int32(label_file.read())
        return image, label, metadata
```

For a more complex and working code example, refer to the example code and configuration that is provided alongside the LATTE Python package.

4.2. Configuration File

The next step is to write an appropriate configuration file to perform an experiment. This file refers to the registered code objects above and defines some hyper-parameters. The YAML configuration below shows how a configuration file would look for the example code presented in the previous section.

```
model:
    class_name: "my_model"
    config:
        input_shape: [32, 32, 3]
        num_classes: 10
train:
    lr: 0.0001
    epochs: 10
    optimizer:
        class_name: "Adam"
    train_dataset_handler:
        class_name: "MyDatasetHandler"
        config:
            data_path: "mydataset/train"
            batch_size: 32
    losses:
        - class_name: "SparseCategoricalCrossentropy"
          name: "crossentropy"
          config:
              from_logits: true
    metrics:
        - class_name: "SparseCategoricalAccuracy"
          name: "accuracy"
```

```
    best_metric: "accuracy"
    best_metric_higher_is_better: true
test:
    dataset_handler:
        class_name: "MyDatasetHandler"
        config:
            data_path: "mydataset/test"
            batch_size: 32
convert:
    converters:
        - class_name: "h5"
        - class_name: "onnx"
          config:
              opset: 13
```

The configuration file is divided into four sections: *model, train, test,* and *convert*. The model used by LATTE is defined in the *model* section: it specifies the class name of the model that is registered in the code and the value of its parameters. The other three main sections define configuration for each of the three modes that can be used in LATTE.

The *train* section defines the learning rate, the optimizer, the name and parameters of the dataset handler to use the one that is registered in the code, and a loss and a metric. Here the loss and metric used are standard Keras loss and metric, and as such, there is no need to register them in the code.

The *test* section only defines the dataset handler to use for the test dataset. Here the same dataset handler is used, only the path to the dataset changes.

The *convert* section defines which conversion format should be used, and their parameters, if any.

For a more complex and working configuration example, refer to the example code and configuration that is provided alongside the LATTE Python package.

4.3. Train

To perform training with the example code and configuration presented above, the following command can be executed in the folder where the code and the configuration file reside:

```
latte train configuration.yaml code.py
```

Running this command for the first time will create a folder named configuration, which will store the log files, the Tensorboard summaries, the train checkpoints, and the best model checkpoint. At any point, the training can be stopped by performing the keyboard combination CTRL+C. Then, if the same command above is rerun, the training will be resumed at the epoch where the execution was stopped. It is possible with this command to use the option -r, which will remove the checkpoints and the Tensorboard logs from the configuration folder, thus forcing the model to be trained from scratch. Note that the checkpoints and logs are simply moved to the trash bin and can be restored if this option is used by mistake.

Once the model training is completed, the usual iterative development process begins. For instance, one might want to train again with different parameters; to do so, a copy of the file configuration.yaml could be made and named configuration-2.yaml, and the parameters changed. A folder named configuration-2 would be created by LATTE when running with this configuration file, thus preserving the data in the folder of the first experiment. When the code needs to be changed between two experiments, one way to ensure that older experiments can be rerun is to write the code in a backward-compatible way, or simply use a code versioning tool such as Git and commit the code changes before running the experiment.

4.4. Test

The trained model can be tested by running the following command:

```
latte test configuration.yaml code.py
```

The test losses and metrics will be displayed after the test dataset has been processed by LATTE.

4.5. Conversion

When the model is ready to be deployed, it can be converted into the specified format by running the following command:

```
latte convert configuration.yaml code.py
```

Here, the converted model files are generated into the folder configuration/convert.

Reference

- LATTE web page (variation 2.8.0)
- Lattice Insights for Lattice Semiconductor training courses and learning plans

Technical Support Assistance

Submit a technical support case through www.latticesemi.com/techsupport.

For frequently asked questions, refer to the Lattice Answer Database at www.latticesemi.com/Support/AnswerDatabase.

Revision History

Revision 1.0, November 2024

Section	Change Summary
All	Initial release.

www.latticesemi.com