Myrtle.ai

VOLLO

Product Description

Version 1.0, December 2022

Machine Learning Inference Acceleration for the Finance Industry

Unrivaled low latencies for LSTM-based neural network models

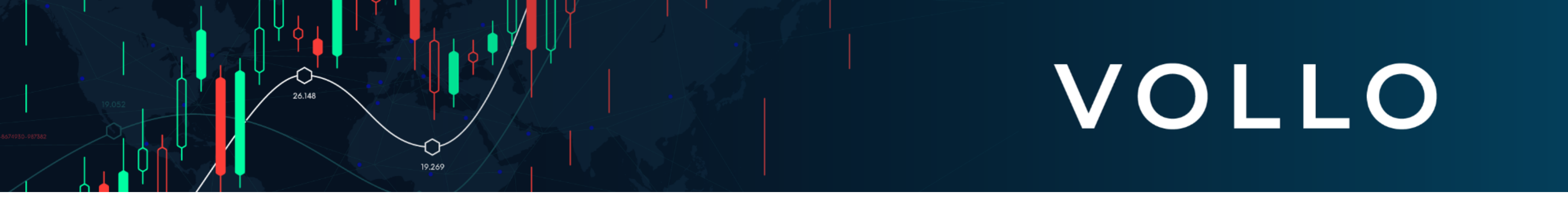
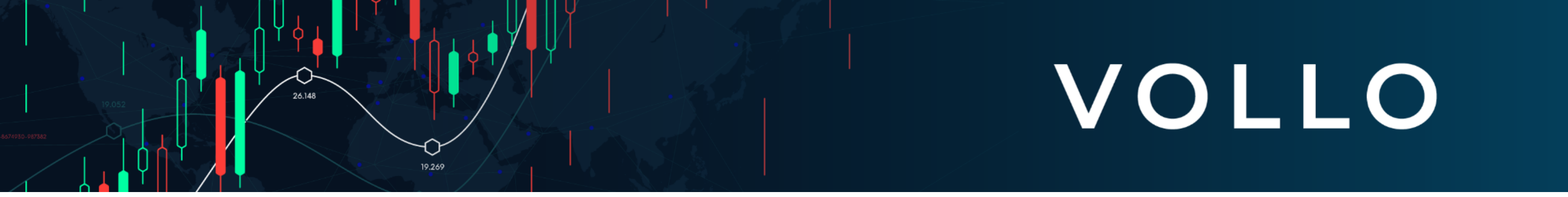


Table of Contents


VOLLO

1 Introduction

This document outlines VOLLO(™), an inference library for low latency acceleration of LSTM-based models.

The document outlines the following:

- Key features of VOLLO
- An overview of VOLLO, its architecture and different customer interfaces
- The hardware requirements for CPU and accelerator hardware
- An outline of the VOLLO APIs
- The Machine Learning user flow
- An overview of the product performance in various test scenarios
- A set of FAQ to further explain the product use by customers

2 Key Features

VOLLO accelerates machine learning inference for small LSTM models typically found in financial trading or fraud detection systems. It is able to process a range of LSTM neural networks with very low latency, in order to provide enhanced decision-making in a very short processing latency window. The system runs on a server CPU with Intel Agilex AGF027 FPGA accelerator cards and currently supports the BittWare IA-840f card.
Key characteristics of VOLLO are

- Ultra low latency LSTM inference on a set of LSTM model architectures across a range of model sizes from 262K to 67M parameters
- Suitable for tick to trade applications
- High accuracy inference through use of Brain Floating Point 16 (bfloat16) numerical format
- High density processing in a 1U server form factor suitable for co-located server deployment
- Supporting inference of multiple models simultaneously on one platform
- Machine Learning training flow supports user model training and deployment using the PyTorch environment.

The product is designed for use by financial firms to achieve lower latency processing and reduce rack space in co-located server settings.
VOLLO can be configured with customer-trained models, utilizing model architectures from the LSTM model zoo, enabling users to deploy a range of workloads specific to their application requirements via a simple export process from standard Machine Learning tool flows.

3 Product Overview

The product architecture is shown in the diagram below:

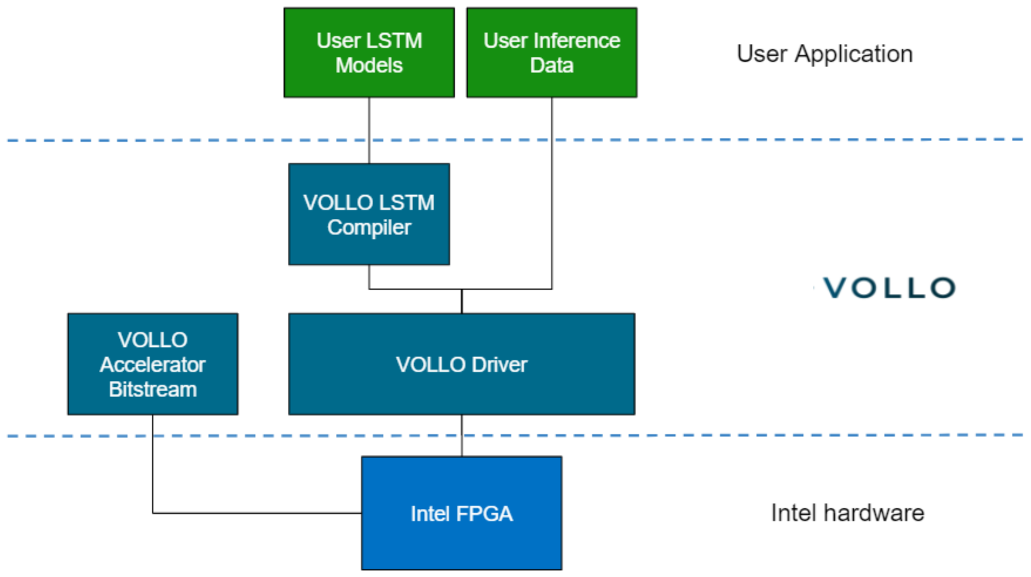


Figure 1 : VOLLO Product Architecture

VOLLO provides a C API to the user, running on the system host CPU.   An Intel FPGA is used by VOLLO to provide low latency inference. The VOLLO Accelerator Bitstream is included in the product; no FPGA configuration or programming is required by the user.

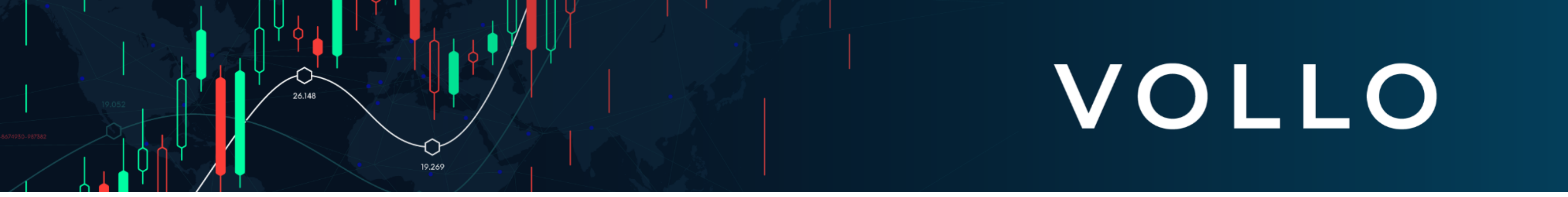
VOLLO consists of the following elements

- VOLLO Compiler - receives input LSTM models in ONNX format and configures the system for inference. It can accept multiple models and controls which models are active on the accelerator
- VOLLO Driver - provides the runtime inference interface for VOLLO. It handles data requests from multiple data streams accessing the models that are deployed on the system
- VOLLO Accelerator Bitstream - programming file for the Intel FPGA on the IA-840f accelerator card.

3.1 VOLLO Compiler

3.1.1 ONNX Model Compilation

The VOLLO LSTM Compiler accepts a range of ONNX models for inference. See section 6 for details of the model configurations that are supported by VOLLO.



The compiler will convert the ONNX models provided ready for inference, including conversion to Brain Float 16 numerical precision for faster inference.
Conversion to Brain Float 16 typically results in negligible loss of accuracy, see section 8 for accuracy results.
See section 6 for detailed information on preparing ONNX models for inference.

3.1.2 Configuring VOLLO with Multiple Models

VOLLO can support inference of multiple models in parallel. It can support up to 12 parallel models per FPGA accelerator card installed in the system, enabling a maximum of 48 parallel models for a system with 4 cards installed. On loading the ONNX configuration models, the system can accept multiple configuration files, to support a number of inference streams in parallel. This creates a number of active inference streams on the system.
The following constraints apply to using multiple models:

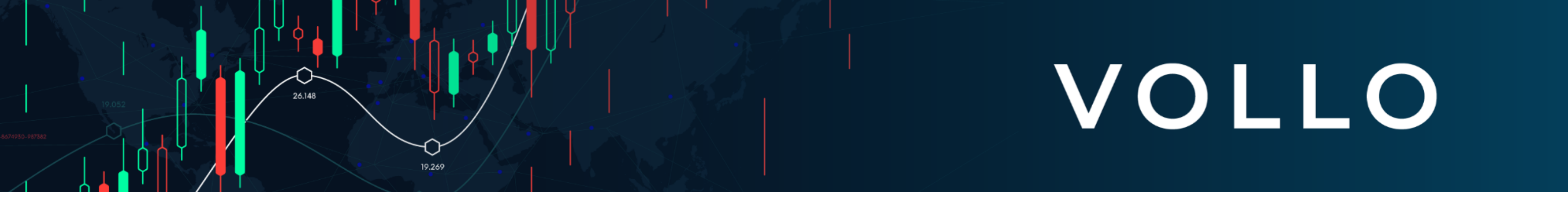
- The number of parallel models supported is dependent on the model architecture selected. See table 1 for the supported configurations.
- All models to be inferred in parallel must have the same base LSTM model from the model zoo
- Processing time for the LSTM models is dependent on the number of models that are active, for fastest inference times only one model should be computed. See table 3 and table 4 for latency measurements in different configurations.
- Changing the number of models that are inferred on the system requires all inference to be paused until VOLLO is configured with new parameters.

3.2 VOLLO Driver

The VOLLO Driver provides a low latency inference API for timing critical inference requests. It accepts inference requests independently for each active inference stream, i.e. one stream per active model.
At each inference request, the VOLLO Driver accepts data for a number of timesteps, performs inference over those timesteps and returns a single computed result, giving the state of the LSTM for the last computed timestep.

3.3 VOLLO Accelerator Bitstream

All LSTM model inference is performed on the FPGA using the VOLLO Accelerator Bitstream. Computation is performed in bfloat16 numerical format, with the API receiving data in bfloat16 format.
VOLLO handles all use of the FPGA, presenting a transparent acceleration capability to the user.



3.4 Model Preparation

Customer ONNX models can be prepared for inference using the Myrtle.ai model zoo and training on their own systems.

To prepare an ONNX model for inference the customer will

- Select an LSTM model from the Model Zoo to satisfy their accuracy and latency requirements.
- Train that model in PyTorch on their training platform using bfloat16 format.
- Evaluate the model performance at bfloat16.
- Convert to ONNX with the conversion scripts provided for use with the configuration API.

The LSTM Model Zoo models, conversion and evaluation scripts are included in the VOLLO inference library installation, but can be copied onto other systems for training on appropriate hardware.

4 Hardware Specification

4.1 CPU Requirements

The minimum CPU specification for the system is shown below.

- Single Socket 6 core Intel Xeon CPU at 2.0 GHz, or better.
- 32 GB RAM
- 10GbE
- 1-4 FHFL PCIe Accelerator slot

For co-located server scenarios, where compute density is a primary concern, up to 4 accelerator cards can be deployed in the following system
https://www.bittware.com/fpga/servers-systems/terabox1400b/

4.2 Accelerator Card Requirements

VOLLO supports the PCIe accelerator card from BittWare https://www.bittware.com/fpga/ia-840f/. Between one and four accelerator cards must be fitted into the server to provide the inference acceleration.
The accelerator cards can be sourced directly from BittWare. For large scale production, Intel partners will be able to supply complete hardware server systems with integrated accelerator cards. Adding more cards increases the number of models that can be inferred in parallel.

4.3 Operating System Requirements

VOLLO is compatible with Ubuntu v20.04 and later.

5 Getting Started

5.1 BIOS Settings

Enable the following parameters from the BIOS:

1. KVM
2. VT-d (or AMD-V for AMD processors)
3. SRIOV Enable
4. ARI (Alternative Routing ID Interpretation)

5.2 Configure boot parameters

1. Edit the **/etc/default/grub** file by adding the following to the **GRUB_CMDLINE_LINUX** field:
   ```
   GRUB_CMDLINE_LINUX="default_hugepagesz=1G hugepagesz=1G hugepages=16 panic=1 iommu=pt"
   ```
2. Generate GRUB configuration files
   ```
   > sudo grub-mkconfig -o /boot/grub/grub.cfg
   ```
3. Reboot the system
4. Verify the changes above:
   ```
   > cat /proc/cmdline
   ```

5.3 Install the SDK

1. Install the kernel driver. In the `vollo-sdk` directory run:
   ```
   > sudo ./load-kernel-driver.sh
   ```
2. Source the setup script:
   ```
   > source setup.sh
   ```

5.4 Run the example program

1. Generate a model following the instructions in section 6.
2. Run the model using the example script in ***examples***

5.5 Program the FPGA

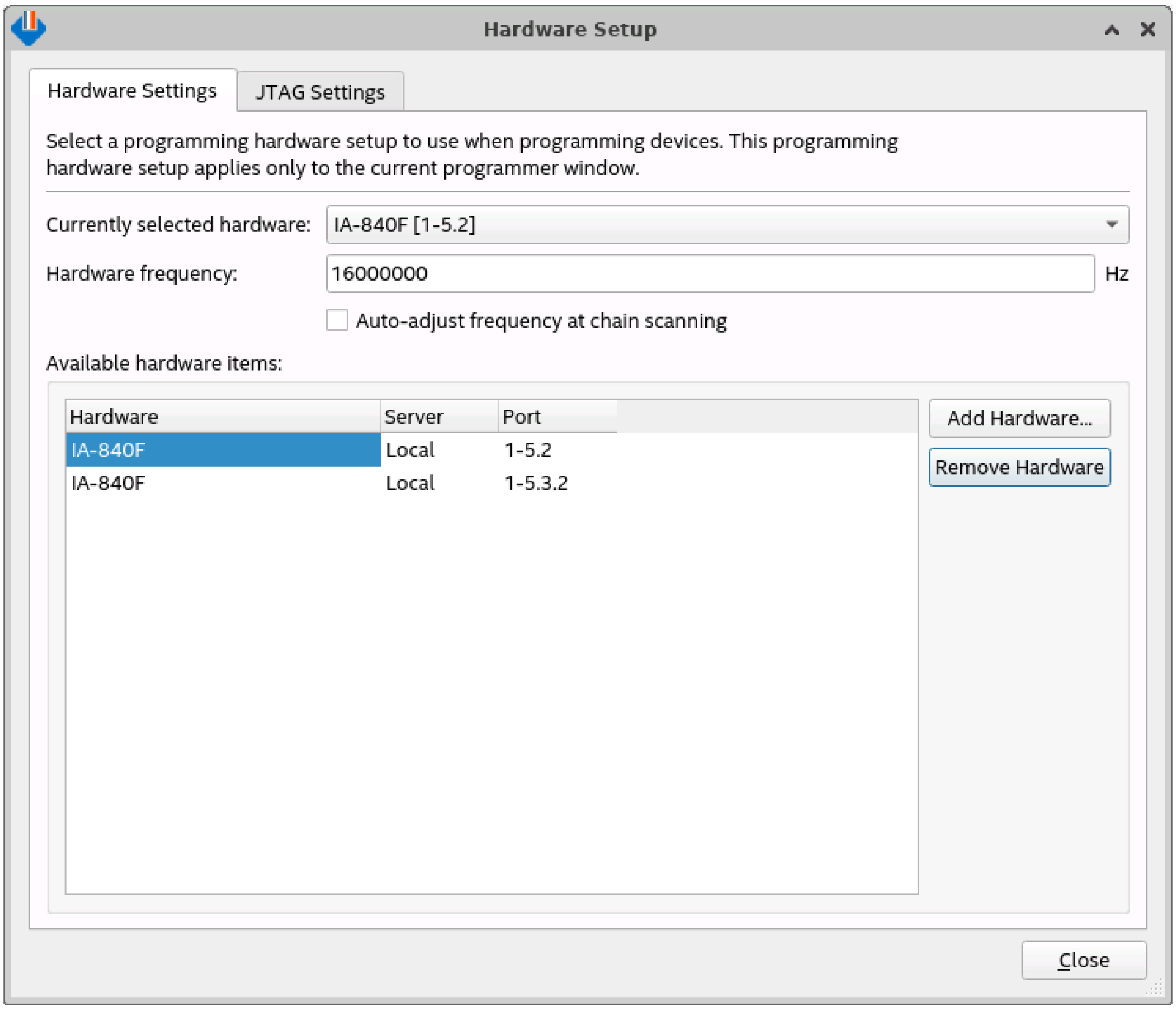
If your FPGA is not already programmed with the VOLLO accelerator then please follow these instructions to load the programming file to the FPGA flash.

1. Install Quartus 22.2 with Agilex board support

2. Run ***jtagconfig,*** you should see the device(s):

```
> jtagconfig
1) IA-840F [1-5.2]
  0341B0DD   AGFB027R25A(.|R0)
```

3. Open ***Quartus*** and then ***File -> Open*** and choose ***vollo-sdk/programming_file/vollo.jic*** the Quartus Programmer should open.
4. Click **Hardware Settings**, select the device you want to program and set the Hardware frequency to 16MHz:



Hardware Setup

Hardware Settings | JTAG Settings

Select a programming hardware setup to use when programming devices. This programming hardware setup applies only to the current programmer window.

Currently selected hardware: IA-840F [1-5.2]

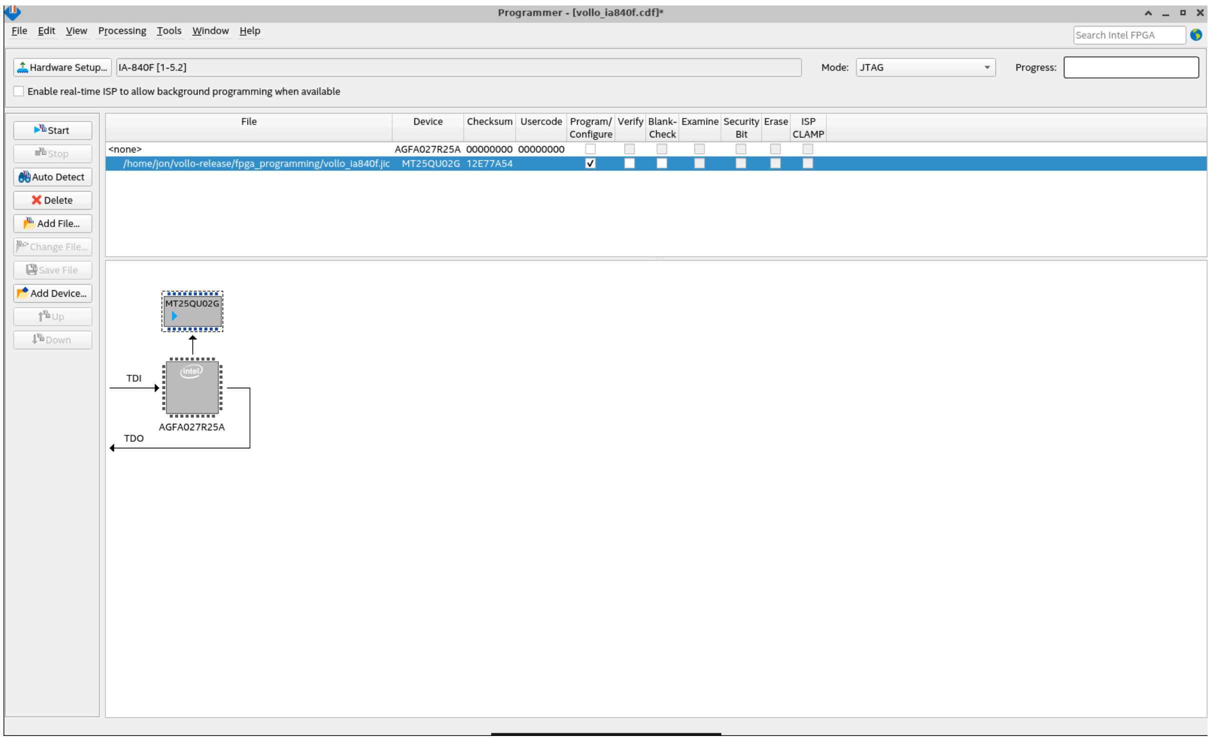
Hardware frequency: 16000000 Hz

Auto-adjust frequency at chain scanning

Available hardware items:

Hardware	Server	Port
IA-840F	Local	1-5.2
IA-840F	Local	1-5.3.2

5. In the Quartus Programmer select Program/Configure checkbox for the `jic` file:

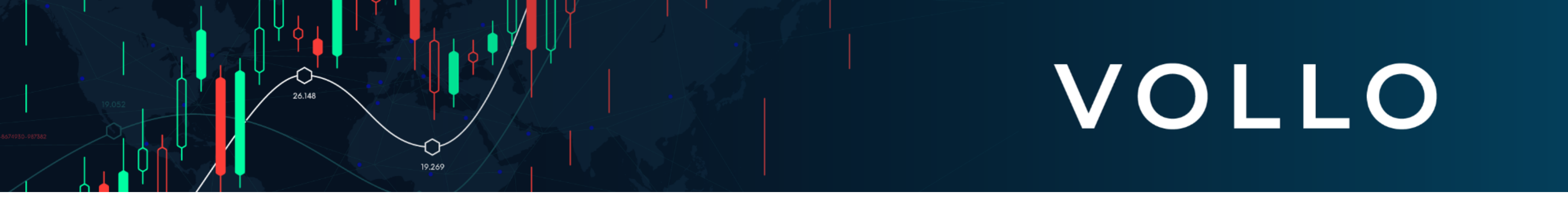

Programmer - [vollo_ia840f.cdf]*

Hardware Setup... IA-840F [1-5.2]   Mode: JTAG   Progress:

☐ Enable real-time ISP to allow background programming when available

File	Device	Checksum	Usercode	Program/ Configure	Verify	Blank-Check	Examine	Security Bit	Erase	ISP CLAMP
<none>	AGFA027R25A	00000000	00000000	☐	☐	☐	☐	☐	☐	☐
/home/jon/vollo-release/fpga_programming/vollo_ia840f.jic	MT25QU02G	12E77A54		✓	☐	☐	☐	☐	☐	☐

6. Press start and wait for flash to program. This takes around 20 minutes.
7. Go back to 3 and program any other devices.
8. Power off the system and start it back up. The bitstream will now be loaded onto the FPGA.



6 Inference API

6.1 C API

6.1.1 Configuration

The user loads ONNX models into the system using the commands below. Once all models are loaded the system is initialized ready for inference.

```
// Create context for VOLLO
vollo_new_ctx(vollo_ctx_t* ctx);
// Add new ONNX models to the configuration. Returns an error if ONNX description is not supported
vollo_stream add_model(vollo_ctx_t ctx, const char* model_path, vollo_kernel_t* kernel);
// Initializes the system to run, using the loaded models. Returns an error if the system cannot support the
given configuration, either non identical ONNX model architectures, or unsupported number of models.
vollo_prepare(vollo_ctx_t ctx);
```

6.1.2 Running inference

Each model loaded creates an inference stream that can be used for inference. The interface offers an asynchronous result function so that input inference requests can be made as fast as the system can support, prior to the return of output data.

```
// Provide some input data to the stream. Provide the number of timesteps worth of input data of the size the
model is expecting. A single timestep of output data will be written to the output once the output is completed
via a stream_poll. The user data will be given back in stream_poll.
const char* vollo_add_job_bf16(
  vollo_context_t vollo,
  vollo_kernel_t kernel,
  uint64_t user_ctx,
  uintptr_t num_timesteps,
  uintptr_t num_features,
  const bf16* input_data,
  bf16* outputs);

// Poll a stream for completed inferences. On a successful poll, returns the number of completed items and if
this is positive,
// set the user array to the list of user data of completed items from  stream_poll
const char* vollo_poll(
  vollo_context_t vollo, uintptr_t* num_completed, const uint64_t** returned_user_ctx);
```

7 LSTM Model Zoo & Machine Learning Flow

7.1 LSTM Model Zoo

The accelerator supports LSTM models as parameterized by a number of layers and layer dimension size. Customers should select a model that most closely matches their accuracy and performance objectives. Further model configurations can be added to the Model Zoo on request.
VOLLO supports LSTM models with the following parameterizable structure:

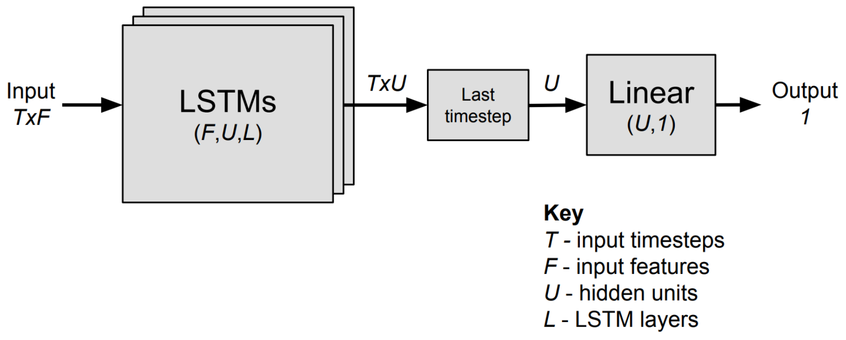


Figure 2 : Supported LSTM model structure

Where :

- L is the number of LSTM layers in the model, not including the final linear layer
- U = Number of units in each of the LSTM layers
- F = Number of input features at the input layer
- T is a number of timesteps that the model is executed for (note this does not affect model architecture)

The table below shows the supported model configurations as available in the model zoo at /installdir/modelzoo

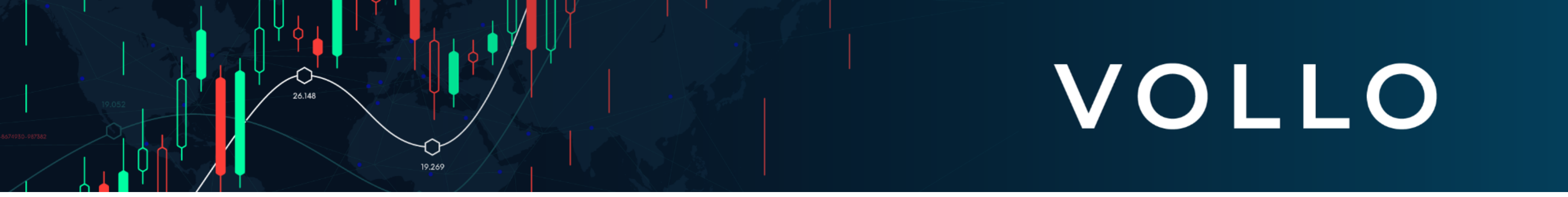
Table 1 : LSTM Model Zoo

Model	Parameters	Layers (L)	Units/Layer (U)	Input Features (F)
LSTM_2L_128U_128F	262K	2	128	128
LSTM_6L_320U_320F	4.9M	6	320	320
LSTM_8L_1024U_1024F	67M	8	1024	1024

Further models are available to members of STAC via the STAC-ML Markets (inference) STAC Pack for VOLLO.

Table 2 below shows the supported number of parallel model instances per model configuration

Table 2 : LSTM Model parallel instance support



Model	**Max models per Accelerator Card**
LSTM_2L_128U_128F	12
LSTM_6L_320U_320F	1
LSTM_8L_1024U_1024F	1

7.2 Training a Model from the Model Zoo

The Model Zoo reference models can be used to train and export ONNX models using PyTorch as follows in the example below. The example indicates how to train with bfloat16 numerical format. This format is recommended for training as it improves training time with no loss of accuracy in the trained model. No further conversion will then be required for inference, resulting in zero loss of accuracy between training and inference.

```
import torch
from vollo.models import VolloModel
config = VolloModel.all_configs()["lstm_2l_100u_100f"]
model = VolloModel.from_config(config)
# Train and evaluate your model on your dataset
num_epochs = ...
dataset = ...
dataloader = ...
for _ in range(num_epochs):
    for x, y in dataloader:
        with torch.autocast(dtype=torch.bfloat16):
            ...

# Export model to onnx
model.export_to_onnx(“model.onnx”)
```

8 Frequently Asked Questions

8.1 How can I run inference on my own LSTM model?

Select a reference model from the LSTM model zoo and train that architecture in PyTorch on your own environment. Export to ONNX using our export scripts and then load into the inference server during the configuration phase.
If multiple models are being used, then create these separately and load into the inference server.

8.2 Can I use more than one model on the system?

Yes, the system can provide model inference of up to 12 models in parallel at any moment in time. Configuration of the system to run multiple models in parallel affects overall latency of the system.

8.3 Can I use TensorFlow to train my model?

Yes, you can use TensorFlow to train the models. Myrtle.ai does not provide reference implementations in TensorFlow by default, but can do so on request. It is possible to use any framework provided you can generate an ONNX file format with the same output format as generated by the Myrtle.ai conversion scripts.

8.4 Can I modify LSTM models from the Model Zoo?

No, the architecture of the inference models in the Model Zoo is fixed. Further model architectures can be added on request from customers at no additional charge, provided they conform to the basic LSTM structure described in section 6. Myrtle.ai only need the LSTM model parameters as described in Section 6.1 to add new models, they do not need access to confidential model parameters. Additional models are available to STAC members in the STAC-ML Markets (Inference) STAC PACK for VOLLO.

8.5 How is latency affected by running more model instances?

The lowest latency for a model inference is achieved by running one model instance per accelerator card. Running a single model on multiple accelerator cards does not increase the latency achieved and so is the easiest way to increase the number of models that can be run in parallel in the lowest latency configuration. Configuring the application to run more than one model instance per accelerator card does increase model latency as the accelerator card resources are shared between models. The relationship between model instances and latency per card is not linear, but is illustrated in the results.

8.6 How can I maximize throughput of the system?

Throughput of the system is proportional to the number of accelerator cards used in the system. For maximum density of throughput, populating up to 4 cards in a system provides the highest overall throughput. Configuring the system to use more model instances also improves system throughput at the expense of latency. If throughput on a single model is of concern the system can be configured to duplicate models by loading identical ONNX models into different model slots. These can then be used in parallel to provide higher throughput overall for that model inference.

8.7 What is bfloat16 and why should I use it?

The VOLLO inference library uses bfloat16 format as championed by Google[1][2] and supported by major AI hardware providers. The bfloat16 format has the same dynamic range (exponent size) as fp32, making it a drop in replacement for single precision training flows. Training models at bfloat16 is simply achieved by setting a configuration parameter in PyTorch. Multiple works show that deep learning training using bfloat16 tensors achieves the same results as fp32 across multiple use cases, in the same number of iterations and with no changes to hyper-parameters [3]. Much faster training times can be achieved on standard GPU training hardware by using bfloat16, for example Nvidia A100 supports 16x higher bfloat compute than fp32. VOLLO inference using bfloat16 uses the same format as the trained model to ensure that all the accuracy of the trained model is preserved, as no conversion of the model is required.

References:

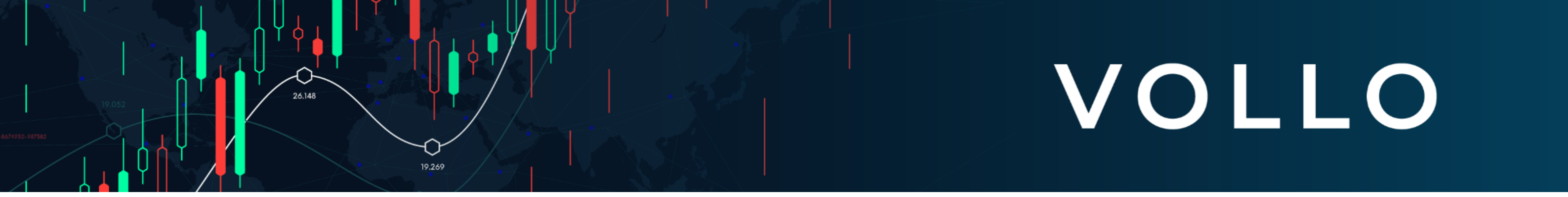
1. https://cloud.google.com/blog/products/ai-machine-learning/bfloat16-the-secret-to-high-performance-on-cloud-tpus
2. https://cloud.google.com/tpu/docs/bfloat16
3. Meta, Intel *A Study of BFLOAT16 for Deep Learning* https://arxiv.org/pdf/1905.12322.pdf

8.8 Can the hardware do other things?

Yes, Myrtle.ai can provide a range of AI inference products for this card. Those products include Automatic Speech Recognition, Natural Language Processing, Speech Synthesis and Vision Processing workloads.

8.9 Do I have to write RTL and create FPGA bitstreams to use VOLLO?

No, a pre optimised bitstream is included in the VOLLO inference library, so no specialist RTL or FPGA development is required.



8.10 Can the VOLLO inference library be run on a SmartNIC FPGA?

In its current format the VOLLO bitstream must run on a PCIe accelerator card accepting data from a host CPU. Contact Myrtle.ai for customization services to extend VOLLO to other platforms.