Low Latency Financial LSTM Inference for STAC-ML™ Markets (Inference) Benchmark using VOLLO® and Intel® Agilex® FPGA

Jon Fowler, Christopher Chalmers, Mike Ashby, Thomas Athorne, Liz Corrigan,
Myrtle.ai

***Abstract*—In this paper we outline VOLLO®, a highly performant inference library for low latency LSTM acceleration. Building on Intel ® Agilex® FPGA technology we highlight the technology advantages for running small LSTM models as presented in the STAC-ML™ Markets (inference) benchmark. We show that VOLLO can achieve a latency as low as 24 microseconds on small LSTM models. We present a highly dense 1U server configuration using Agilex FPGA accelerator cards to demonstrate a system that can maintain low latency operation when running up to 48 separate inference models simultaneously.**

I. Introduction

Machine Learning (ML) has found many applications in the financial services industry, including trading, pricing and risk analysis [1]. Algorithmic trading, which accounts for around two thirds [2] of all US equity trading, has extended from simple arbitrage models to more complex machine learning models. Growth in the use of ML in algorithmic trading has been accompanied by demand for reductions in latency in order for traders to be more competitive. This enables traders to react faster to changes in the market and to make more intelligent decisions within a given latency window.

Financial firms use a variety of machine learning techniques, including LSTM based neural networks [3], which are well-suited for time series prediction. The STAC Benchmark Council™, an organisation comprising over 400 financial institutions and 50 leading technology vendors, has recently established the STAC-ML Markets (Inference) Benchmark (henceforth simply STAC-ML) [4] to enable financial institutions to compare performance of LSTM based deep-learning models on different platforms in a controlled manner. Myrtle.ai has taken on the challenge of demonstrating excellent results against those benchmarks. This paper reviews the approach adopted and the results obtained.

II. STAC-ML Description

STAC-ML has been developed by financial firms to represent a key AI workload in finance. We give a summary of the benchmark here, describing the key components which are relevant for the performance analysis. For a detailed account of the benchmark, refer to the STAC-ML at https://STACresearch.com/ml.

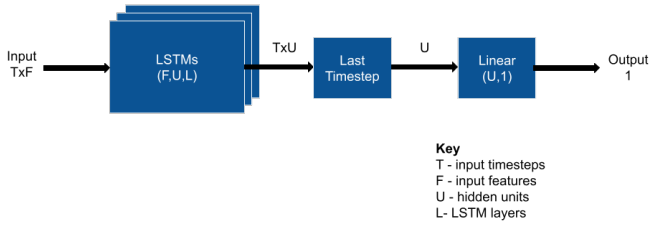

Fig. 1. Architecture of STAC-ML Markets (Inference) Benchmark LSTM models

A. Benchmark Models

The benchmark describes three stacked LSTM models with similar architectures but different sizes. These are referred to as LSTM_A (smallest), LSTM_B and LSTM_C (largest).

The structure of the LSTM models used in the benchmark is shown in Figure 1. The three models consist of a stack of LSTMs with a final linear layer to reduce the output to a single result. The models are summarized in Table I, with exact parameterization available in the STAC-ML specification:

Model	Model Size (fp32)	FLOPS	Relative Size
LSTM_A	640KB	16 MFLOP	1
LSTM_B	4MB	200 MFLOP	6
LSTM_C	120MB	6 GFLOP	187

TABLE I
STAC-ML Markets (Inference) Benchmark Models

STAC-ML presents a distinct inference challenge, due to the range of size of models that must perform well on one platform. Compared to MLCommons benchmark models [5], which are typically on order of GFLOPS per input, LSTM_A in particular is a distinctly small model. LSTM_C, at over 180 times larger than LSTM_A, requires the inference platform to be effective across two orders of magnitude of model compute.

B. STAC-ML Performance Metrics

STAC-ML primarily reports performance for latency and throughput. In addition, it also highlights system performance, normalizing results for power consumption and physical rack space required.

Given a target deployment in algorithmic trading platforms, latency is a key metric of interest to financial firms using LSTM models. Low latency processing requires high compute

utilization at low batch sizes which is difficult to achieve on traditional processor architectures, such as CPUs and GPUs, due to challenges with low latency core-to-core communication and lack of explicit memory control. As such this benchmark is anticipated to show promising results from alternative AI processing platforms with different architectures, such as AI inference ASICs and FPGAs.

A further differentiator of STAC-ML to other inference benchmark challenges is the inclusion of the number of parallel model instances (NMI) as a parameter of interest in the benchmark. Number of Model Instances is defined as the number of independent model instances that can be inferred on a system at the same time. This scenario highlights systems that can be segmented into different inference engines to run independently. An example of this would be running independent inferences on each of a CPU core, or using Multi-Instance GPU (MIG) mode on a GPU. One example of how this might be used in a financial model setting is where different trained models are used for different stock purchase decisions and systems are inferring on common data input, with different model parameters.

STAC-ML is targeted to applications that typically run in exchange co-located data centers, where physical equipment space incurs a higher premium rental price than other scenarios. STAC-ML challenges solution vendors to consider the density of the solutions that they can provide. This is reflected in physical density metrics for inferences per second per cubic foot, highlighting systems that make the most efficient use of physical space.

C. Benchmark Model Accuracy

STAC-ML includes accuracy results, rather than requiring a particular accuracy threshold to be met. This enables a range of inference solutions using different numerical precision to be submitted and compared. STAC-ML rules enable solutions to be compared only if the system with higher performance also achieves better or equal accuracy. Accuracy measurements are quantified as an absolute variation from a fp64 reference implementation, which provides a very high degree of resolution for understanding even small changes in accuracy from the fp64 baseline.

Accuracy of deep-learning model inference is influenced most heavily by the choice of numerical precision adopted in the solution. Quantization is a well understood technique improving inference performance, but typically trades off against model accuracy. Quantizing a model consists of using a different numerical format for the parameters and/or activations. This lowers the power consumption and latency, as both the memory requirements are reduced and the operations can be more efficiently executed in hardware. Converting parameters and activations to a different number format may require not only a reduction in the number of bits used to represent each value (e.g. 16 bits instead of 32 bits), but also a change to the arithmetic used when executing operations with these values (e.g. integer arithmetic instead of floating point).

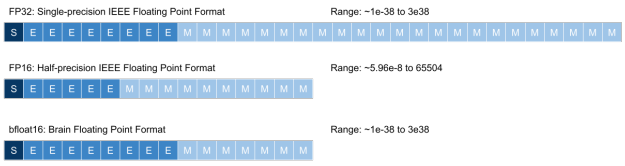

Fig. 2. bfloat16 Floating Point Format as compared to Single and Half precision IEEE Floating Point Formats

Brain Floating Point 16 format (bfloat16), as illustrated in Figure 2, is a popular format with common support in AI hardware including Google's TPU [6], Nvidia GPU [7], Intel CPU [8] and Intel FPGA [9]. The format was designed to provide efficient training performance. It keeps the same dynamic range of fp32, to simplify model training [10]; avoiding some of the complications encountered in training using fp16, which can require additional techniques [11]. This format has been shown to give near negligible accuracy drop over models trained at fp32 when applied to a range of neural network applications [12] and can be computed with smaller silicon area than fp32 format, resulting in both faster training and inference [6].

The VOLLO inference library uses bfloat16, so that models can be quickly and easily trained in bfloat16 on standard GPU hardware, and then run in that same format for inference, ensuring that all the accuracy of the trained model is preserved, as no conversion of the model is required between training and inference. Training in bfloat16 in PyTorch is simply achieved by specifying the data type as bfloat16, with no further pre or post modification steps required.

III. Agilex Inference Solution

A. Agilex FPGA features for high performance inference

The Agilex FPGA family has recently been launched by Intel. The family includes a range of FPGAs targeted to different end applications. Focusing on the F-series devices, built on 10nm SuperFin technology, the FPGA device contains advanced DSP blocks that support a range of fixed and floating point multipliers, central to the performance of AI computation on FPGA. Table II shows the numerical support in the Agilex DSP blocks compared to the previous Stratix 10 FPGA family. The addition of fp16 and bfloat16 formats in this family offers a good trade in accuracy and performance for Deep-learning models [13].

Format	Stratix 10 Mults per DSP	Agilex Mults per DSP
fp32	1	1
fp16	-	2
bfloat16	-	2
INT8	2	4

TABLE II
Multiplication support in Agilex and Stratix 10 FPGA DSPs

In addition to compute resource, memory availability and architecture are critical for efficient inference. The F-series Ag-

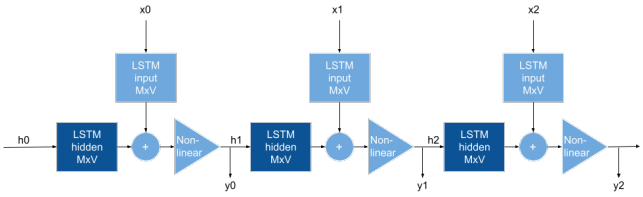

Fig. 3. Evaluation of three timesteps of an LSTM layer. The sequential evaluation path is highlighted and the weights for the LSTM hidden MxV need to be in a high-bandwidth cache for maximum performance.

ilex devices contain hardened memory controllers for DDR4 as well as 259Mb of onchip SRAM.

A further innovation in the Agilex devices is the second generation Intel hyperflex FPGA architecture. This allows designs to achieve higher timing closure via additional pipeline registers within the device routing networks. The use of these registers enables deeper pipelines on timing critical signals, without consuming registers in logic elements.

B. Design considerations for low latency LSTM inference

Evaluating an LSTM model at low-latency is difficult due to the recursive nature of the LSTM layer. This introduces two design challenges: high memory intensity driven by the matrix-vector computation; and the need for low latency core-to-core communication to enable parallelism within a layer.

The LSTM layer, illustrated in Figure 3, is recursive, that is, the output of evaluating a timestep feeds into the input of the LSTM for the next timestep. This creates a long sequential path which includes the hidden unit matrix-vector multiplication.

To maximise performance of the inference solution, it is important that the weights of matrix-vector multiplication fit into a local cache with very high memory bandwidth, as this computation has an arithmetic intensity of 0.5 at fp32 and otherwise will be limited by memory bandwidth.

The sequential evaluation path in the LSTM layer also creates a second challenge. In order to scale efficiently across different processing units, the activation, h_t, must be synchronized at every timestep.

Both CPUs and GPUs cannot efficiently synchronize data over processing units and so can be inefficient when parallelizing small models. For bigger models, such as LSTM_C, it is challenging to fit the hidden matrix within a high bandwidth cache.

The FPGA can overcome both these challenges to provide an efficient inference platform for low latency LSTM models. Synchronizing different processing units is straightforward in an FPGA fabric, and the Agilex FPGA has sufficient on-chip SRAM to hold the hidden matrix in high bandwidth memory.

C. VOLLO Inference Library using Agilex AGF027

To realise low latency inference of the STAC-ML LSTM models on FPGA, Myrtle.ai provide VOLLO, a highly optimized

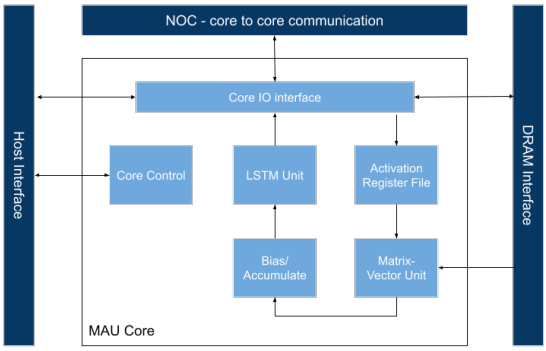

Fig. 4. VOLLO compute core architecture on Intel Agilex FPGA

inference library that uses Intel Agilex AGF027 for inference computation. The VOLLO inference library includes a software application for Intel CPU and an accelerator bitstream for the Intel Agilex FPGA. This bitstream implements a VLIW processor core architecture with a sequential execution path consisting of a matrix-vector unit, an addition unit and a LSTM unit. The Agilex AGF027 contains 12 VOLLO compute cores which are connected by a high-bandwidth network-on-chip (NOC). The FPGA uses a core processing frequency of 280 MHz in this design.

Each VOLLO compute core connects to the host CPU via PCIe, to DDR4 memory and to the other cores via a soft Network on Chip (NOC) routing network. The routing network between each pair of neighbouring cores carries 8 bfloat16 values on each clock cycle, resulting in an internal bus bandwidth of 4.5GB/s.

The architecture of the VOLLO compute core, optimized for Intel Agilex FPGA, is shown in Figure 4. This has all the functionality required to implement a range of LSTM models.

External DDR4 memory is used to store model weights and cell state activations for the larger LSTM_C model, with the hidden layer matrix being stored in on chip SRAM, to provide a very high bandwidth for this critical component of the inference solution.

Being designed as a native batch 1 computation engine means that the solution can make efficient use of the compute and memory resources, when targeting latency critical workloads, such as STAC-ML.

D. Programming Model and Software API

To enable the user to independently generate and deploy LSTM models for inference, the VOLLO inference library provides a PyTorch based model zoo of supported LSTM configurations. These can be trained in the customer's own ML environment and then exported as an ONNX model file, using export scripts provided. The CPU software API provided with the FPGA inference solution accepts ONNX model inputs at runtime, enabling customers to use the solution with their own trained models, as for other inference products, and without any knowledge or use of FPGA toolchains.

The VOLLO compute cores are configured by the VOLLO inference library, depending on the models selected for use by the user. The cores can be partitioned to support multiple model instances per FPGA accelerator, or can work together to provide lowest latency processing on a single LSTM model.

IV. Results

A. Methodology

We present results of the performance of the VOLLO inference library running on Agilex AGF027 FPGA. All results presented here were recorded during a STAC Audit of STAC-ML. These independently validated results were published by STAC and are available at https://STACresearch.com/MRTL221125.

The inference solution that was audited used a BittWare TeraBox 1402B server, comprising an Intel Xeon® Platinum 8351N CPU with 36 cores running at 2.40GHz and four BittWare IA-840F PCIe Accelerator cards containing Intel Agilex AGF027 devices. This 1U server was selected in order to demonstrate high system density suitable for co-located server environments.

To make use of four accelerator cards the VOLLO bitstream is duplicated on each card, offering a higher supported number of model instances (NMI) for the system, therefore increasing throughput and system density metrics. Model latency is not affected by use of more than one card.

B. Model Latency

Table III shows the latency achieved for the inference solution for 1 model instance per card, representing the fastest possible latency. We present figures for 99%-ile latency for VOLLO.

Model	VOLLO NMI per card	VOLLO 99%-ile Latency (ms)
LSTM_A	1	0.0241
LSTM_B	1	0.0648
LSTM_C	1	1.35

TABLE III
LOWEST LATENCY RESULTS FOR STAC-ML MARKETS (INFERENCE) BENCHMARK - SUT ID MRTL221125 STAC-ML.MARKETS.INF.S.LSTM_[A,B,C].4.LAT.V1

Figures 5 and 6 show the behaviour of latency for VOLLO as the number of model instances per accelerator card increase. In the case of LSTM_A the latency increases only marginally from one model instance to 3 model instances per card as the smaller LSTM_A does not use the entire capacity of the FPGA. Once the FPGA capacity is used, then latency is increased with more model instances as the accelerator uses less resource per model instance.

C. Model Throughput

Table IV shows the model throughput achieved for the inference solution for the largest achievable throughput configuration. When using VOLLO, highest throughput is achieved with the highest number of model instances (NMI). The table includes the latency achieved in that scenario, demonstrating that high throughput can be achieved while maintaining low latency.

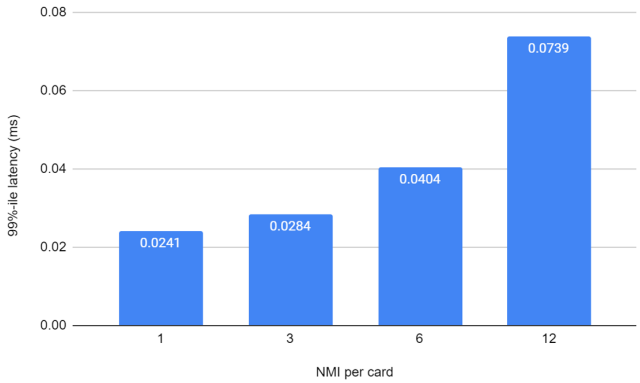

Fig. 5. LSTM_A 99%-ile latency with multiple model instances STAC-ML Markets (Inference) - SUT ID MRTL221125 STAC-ML.Markets.Inf.S.LSTM_A.[4,12,24,48].LAT.v1

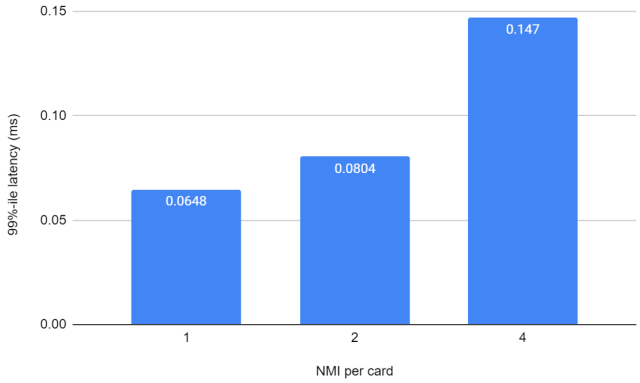

Fig. 6. LSTM_B 99%-ile latency with multiple model instances STAC-ML Markets (Inference) - SUT ID MRTL221125 STAC-ML.Markets.Inf.S.LSTM_B.[4,8,16].LAT.v1

D. System Power and Physical Density

The VOLLO System Under Test used in the STAC Audit consumes 1 rack unit, as a 1U full width, full length server, with a total volume of 1.0065 cubic feet. The system power ranges from 487W to 593W for the different measurement scenarios at approximately 20°C ambient inlet temperature.

Table V shows the achieved system power density for the system. The table below shows power efficiency for the highest throughput configuration (highest NMI per card) for each LSTM model, where the system is most highly utilized.

V. Conclusion

The audited results for VOLLO in STAC-ML highlight the effectiveness of an FPGA solution for low latency LSTM inference, achieving latency as low as 24 microseconds. The solution achieves low latency processing across a range of LSTM model sizes. Furthermore the VOLLO inference library enables users to maintain low latency processing while deploying multiple independent model instances for parallel inference on one system, with a latency increase of only 3.1x

Model	VOLLO NMI	VOLLO Throughput (inf/s)	VOLLO 99%-ile Latency (ms)
LSTM_A	48	650,803	0.0739
LSTM_B	16	109,211	0.147
LSTM_C	4	2,979	1.35

TABLE IV
HIGHEST THROUGHPUT RESULTS FOR STAC-ML MARKETS (INFERENCE)
BENCHMARK - SUT ID MRTL221125
STAC-ML.MARKETS.INF.S.LSTM_A.48[TPUT,LAT].V1
STAC-ML.MARKETS.INF.S.LSTM_B.16[TPUT,LAT].V1
STAC-ML.MARKETS.INF.S.LSTM_C.4[TPUT,LAT].V1

Model	NMI	VOLLO (inf/s/kW)
LSTM_A	48	1,183,279
LSTM_B	16	186,489
LSTM_C	4	5,023

TABLE V
POWER EFFICIENCY RESULTS FOR STAC-ML MARKETS (INFERENCE)
BENCHMARK - SUT ID MRTL221125
STAC-ML.MARKETS.INF.S.LSTM_A.48.ENERG_EFF.V1
STAC-ML.MARKETS.INF.S.LSTM_B.16.ENERG_EFF.V1
STAC-ML.MARKETS.INF.S.LSTM_C.4.ENERG_EFF.V1

for LSTM_A and 2.3x for LSTM_B between a single model instance per card and the maximum supported model instances per card. For LSTM_A, up to 48 independent model instances can be computed on the system with a 99%-ile latency of 72.9 microseconds.

The space and power efficiency of the system are highlighted by providing results for a 1U system containing 4 Agilex FPGAs and consuming a maximum power of under 600W in all test scenarios.

Using the Agilex FPGA in conjunction with VOLLO enables customers to access this technology without specialist knowledge of FPGA programming languages or toolchains. The use of bfloat16 in the implementation has enabled a fast inference solution that can be trained at equivalent performance to an fp32 solution, with models inferred without any further conversion losses between training and inference.

REFERENCES

[1] Mordor Intelligence, "Algorithmic trading market - growth, trends, covid-19 impact, and forecasts 2022 - 2027," https://www.mordorintelligence.com/industry-reports/algorithmic-trading-market, 2022.

[2] Bloomberg, "Global algorithmic trading market to surpass us$ 21,685.53 million by 2026," https://www.bloomberg.com/press-releases/2019-02-05/global-algorithmic-trading-market-to-surpass-us-21-685-53-million-by-2026, 2019.

[3] T. B. Shahi, A. Shrestha, A. Neupane, and W. Guo, "Stock price forecasting with deep learning: A comparative study," *Mathematics*, vol. 8, no. 9, 2020. [Online]. Available: https://www.mdpi.com/2227-7390/8/9/1441

[4] https://stacresearch.com/ml.

[5] V. J. Reddi, C. Cheng, D. Kanter, P. Mattson, G. Schmuelling, C. Wu, B. Anderson, M. Breughe, M. Charlebois, W. Chou, R. Chukka, C. Coleman, S. Davis, P. Deng, G. Diamos, J. Duke, D. Fick, J. S. Gardner, I. Hubara, S. Idgunji, T. B. Jablin, J. Jiao, T. S. John, P. Kanwar, D. Lee, J. Liao, A. Lokhmotov, F. Massa, P. Meng, P. Micikevicius, C. Osborne, G. Pekhimenko, A. T. R. Rajan, D. Sequeira, A. Sirasao, F. Sun, H. Tang, M. Thomson, F. Wei, E. Wu, L. Xu, K. Yamada, B. Yu, G. Yuan, A. Zhong, P. Zhang, and Y. Zhou, "Mlperf inference benchmark," *CoRR*, vol. abs/1911.02549, 2019. [Online]. Available: http://arxiv.org/abs/1911.02549

[6] https://cloud.google.com/tpu/docs/bfloat16.

[7] https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/a100/pdf/nvidia-a100-datasheet-us-nvidia-1758950-r4-web.pdf.

[8] https://www.intel.com/content/www/us/en/developer/articles/technical/intel-deep-learning-boost-new-instruction-bfloat16.html.

[9] https://www.intel.co.uk/content/www/uk/en/products/details/fpga/agilex/f-series.html.

[10] G. Henry, P. T. P. Tang, and A. Heinecke, "Leveraging the bfloat16 artificial intelligence datatype for higher-precision computations," *CoRR*, vol. abs/1904.06376, 2019. [Online]. Available: http://arxiv.org/abs/1904.06376

[11] P. Micikevicius, S. Narang, J. Alben, G. F. Diamos, E. Elsen, D. García, B. Ginsburg, M. Houston, O. Kuchaiev, G. Venkatesh, and H. Wu, "Mixed precision training," *CoRR*, vol. abs/1710.03740, 2017. [Online]. Available: http://arxiv.org/abs/1710.03740

[12] D. D. Kalamkar, D. Mudigere, N. Mellempudi, D. Das, K. Banerjee, S. Avancha, D. T. Vooturi, N. Jammalamadaka, J. Huang, H. Yuen, J. Yang, J. Park, A. Heinecke, E. Georganas, S. Srinivasan, A. Kundu, M. Smelyanskiy, B. Kaul, and P. Dubey, "A study of BFLOAT16 for deep learning training," *CoRR*, vol. abs/1905.12322, 2019. [Online]. Available: http://arxiv.org/abs/1905.12322

[13] J. Chromczak, M. Wheeler, C. Chiasson, D. How, M. Langhammer, T. Vanderhoek, G. Zgheib, and I. Ganusov, "Architectural enhancements in intel® agilex™ fpgas," in *Proceedings of the 2020 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 140–149. [Online]. Available: https://doi.org/10.1145/3373087.3375308