

HLS4LFADS: A Framework for Real-Time and Parallel Neural Decoding on FPGAs

Chi-Jui Chen^{*§}, Yan-Lun Huang[†], Xiao-Han Liu[‡], Atharva Mattam[‡], Hao Fang[‡], Ling-Chi Yang^{*}, Chia-En Chang^{*}, Elham E. Khoda[‡], Eli Shlizerman[‡], Amy L. Orsborn[‡], Scott Hauck[‡], Shih-Chieh Hsu[‡], and Bo-Cheng Lai^{*}

^{*}National Yang Ming Chiao Tung University, Hsinchu, Taiwan

[†]The University of Texas at Austin, Austin, TX, USA

[‡]University of Washington, Seattle, WA, USA

[§]Corresponding author: Chi-Jui Chen (silencekugel.ee11@nycu.edu.tw)

Abstract—Latent Factor Analysis via Dynamical Systems (LFADS) is widely used to infer low-dimensional latent dynamics from population neural recordings and has been applied across neural decoding tasks. Closed-loop systems require decoding to complete within each temporal bin. CPUs and GPUs can meet this deadline for moderate workloads, but longer input sequences or multiple concurrent models increase latency. Field-programmable gate arrays (FPGAs) offer model-specific parallelism and spatial replication, yet limited resources and model-specific implementation effort hinder deployment. We present HLS4LFADS, an end-to-end FPGA framework that combines layer-wise calibrated fixed-point quantization with quantization-aware training, extends `hls4ml` to automatically generate accelerators for the complete LFADS inference graph, and supports concurrent inference through multi-compute-unit (multi-CU) deployment. On a Xilinx Alveo U55C, three concurrent CUs achieve mean end-to-end latencies of 0.367 ms and 0.696 ms for the monkey-reaching and synthetic Lorenz tasks, respectively. These values represent increases of only 11.2% and 6.9% over the corresponding single-CU latencies and remain below their respective 10-ms and 6-ms bin intervals.

Index Terms—FPGA, high-level synthesis, recurrent neural networks, quantization, LFADS, real-time neural decoding

I. INTRODUCTION

Recent advances in neural recording technologies, such as Neuropixels probes, enable simultaneous measurement of hundreds of individual neurons, allowing population-level analysis with unprecedented resolution [1]. Although these data are high-dimensional, neural population activity is often well-described by low-dimensional latent dynamics [2]. Recovering these dynamics within single trials is essential for linking neural activity with behavior and for advancing therapeutic technologies [3]. Latent Factor Analysis via Dynamical Systems (LFADS) [3], [4] is a widely used deep learning method that employs a recurrent variational autoencoder (VAE) to infer these dynamics while mitigating trial-to-trial variability. LFADS has been applied to neural decoding tasks, including kinematic brain-computer interface (BCI) control [5], long-term neural interface stabilization [6], and both speech and motor neuroprosthetic systems [7], [8].

This work was supported in part by the National Science and Technology Council (NSTC), Taiwan, under Grant 115-2221-E-A49-146.

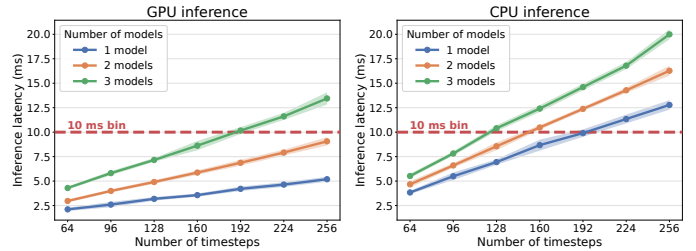


Fig. 1. LFADS inference latency on the evaluated GPU (left) and CPU (right) versus sequence length and concurrent model count. The profiled model has 30 input channels, 64 hidden units in each gated recurrent unit (GRU), and a latent-factor dimension of 4. Lines show the mean latency, and shaded regions indicate ± 1 standard deviation over 1,000 batch-size-one runs. The dashed line indicates a representative 10-ms update interval.

Real-time inference is critical in closed-loop BCI applications, which involve sequential stages of data acquisition, feature extraction, neural decoding, and effector control [10]. Because a new neural sample becomes available in each temporal bin, decoding must complete within the bin interval to avoid data queuing and timing jitter. CPUs and GPUs can often meet this requirement for moderate LFADS workloads, but the deadline becomes more difficult to maintain as the workload scales. First, longer input sequences arise when a wider temporal context is needed or when a fixed temporal window is represented using a smaller bin size to increase temporal resolution. Because recurrent updates are sequential, latency generally increases with sequence length. Second, several decoding scenarios involve the concurrent execution of multiple models. For example, MRAE [11] uses separate LFADS-like models to process complementary frequency bands of continuous neural waveforms to overcome time-bandwidth limitations. Multiple models may also process different brain regions before joint analysis [12] or form an ensemble [13]. On shared computing platforms, concurrent execution can cause resource contention and memory-bandwidth limitations, increasing inference latency [14]. Figure 1 illustrates both effects on the evaluated CPU and GPU, motivating hardware that can maintain low per-sequence latency while scaling to concurrent inference.

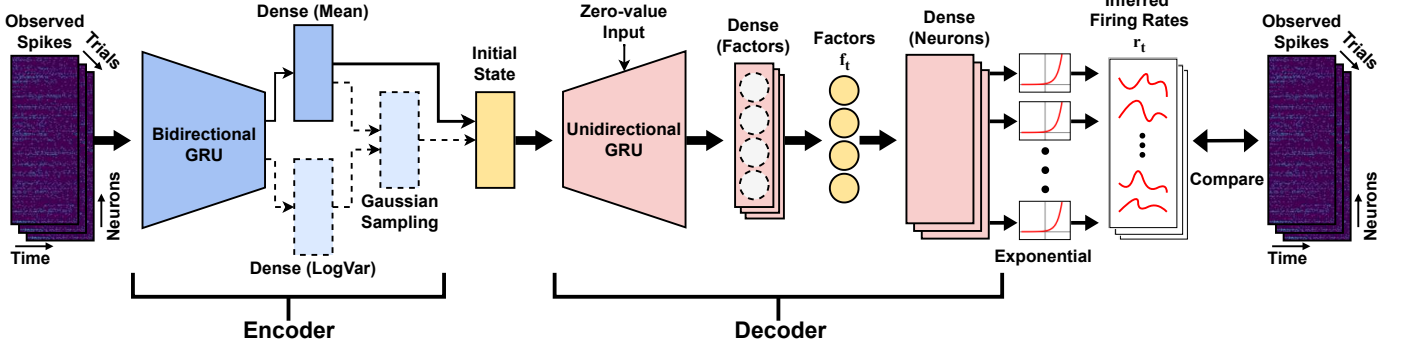


Fig. 2. LFADS architecture adopted in this work, based on [9]. Dashed components are omitted during deterministic inference.

Field-programmable gate arrays (FPGAs) are promising because both the compute datapath and memory organization can be tailored to LFADS inference. While recurrent timestep dependencies remain sequential, the matrix–vector operations within each step can run in parallel, reducing the latency of each update and thus the entire input sequence. Model parameters can remain on chip and be reused across timesteps, while recurrent states can be stored and updated locally between successive steps, avoiding repeated high-latency off-chip accesses. For concurrent workloads, full accelerators can be spatially replicated so each model has dedicated compute and local memory, reducing contention. However, these benefits require one or more full accelerators to fit within the FPGA’s limited compute and on-chip memory, making per-model hardware footprint a key constraint. Moreover, designing and adapting model-specific accelerators with conventional hardware flows demands substantial engineering effort [15], hindering rapid iteration as neuroscience experiments vary across datasets, subjects, and model configurations.

To address these challenges, we present HLS4LFADS, an end-to-end framework for deploying multiple LFADS models on FPGAs. Our main contributions are as follows:

- We develop a fixed-point quantization procedure for LFADS that combines layer-wise range calibration with quantization-aware training (QAT) to reduce the per-model hardware footprint while preserving model performance.
- We extend the recurrent backend and model-conversion flow of `hls4ml` [16] with the operators required for LFADS inference, enabling automatic generation of configurable LFADS accelerators.
- We construct a multi-compute-unit (multi-CU) deployment that spatially replicates complete LFADS accelerators to support concurrent inference on a single FPGA.
- We evaluate HLS4LFADS on a monkey-reaching task and a synthetic Lorenz task across different quantization configurations, hardware parallelism settings, and CU counts. On a Xilinx Alveo U55C, three concurrent CUs achieve mean end-to-end latencies of 0.367 ms and 0.696 ms for the monkey-reaching and Lorenz tasks, respectively, below their corresponding 10-ms and 6-ms bin intervals.

II. BACKGROUND AND MOTIVATION

A. Latent Factor Analysis via Dynamical Systems (LFADS)

LFADS has several variants for modeling different neural data and dynamical structures. We focus on the autonomous LFADS variant used in [9], whose structure is shown in Fig. 2. LFADS infers low-dimensional latent dynamics from observed spike trains and reconstructs firing rates over time. The observed spike counts are modeled as Poisson-distributed variables parameterized by the reconstructed firing rates.

LFADS consists of encoder and decoder stages. The encoder stage first processes the observed spike sequence using a bidirectional gated recurrent unit (GRU). The final forward and backward hidden states are concatenated and mapped through dense layers to the mean and log-variance of an approximate posterior distribution over the decoder GRU initial state. Following [9], the posterior mean is used to initialize the decoder GRU during inference rather than drawing a sample from this distribution.

The decoder stage begins with a unidirectional GRU initialized by this posterior mean. In the autonomous configuration, no time-varying input is provided to the decoder GRU, so its input is set to zero at each time step. The decoder GRU generates a sequence of hidden states, which are projected to low-dimensional latent-factor vectors $\mathbf{f}_t$. These factors are then mapped to firing-rate vectors $\mathbf{r}_t$ through a dense layer and an exponential activation.

For the reset-after GRU formulation used in this work, the state update at time step t is expressed as

$$\mathbf{z}_t = \sigma(\mathbf{W}_z \mathbf{x}_t + \mathbf{b}_z^x + \mathbf{U}_z \mathbf{h}_{t-1} + \mathbf{b}_z^h), \quad (1)$$

$$\mathbf{q}_t = \sigma(\mathbf{W}_q \mathbf{x}_t + \mathbf{b}_q^x + \mathbf{U}_q \mathbf{h}_{t-1} + \mathbf{b}_q^h), \quad (2)$$

$$\tilde{\mathbf{h}}_t = \tanh(\mathbf{W}_h \mathbf{x}_t + \mathbf{b}_h^x + \mathbf{q}_t \odot (\mathbf{U}_h \mathbf{h}_{t-1} + \mathbf{b}_h^h)), \quad (3)$$

$$\mathbf{h}_t = \mathbf{z}_t \odot \mathbf{h}_{t-1} + (1 - \mathbf{z}_t) \odot \tilde{\mathbf{h}}_t, \quad (4)$$

where $\mathbf{x}_t$ denotes the input at time step t , and $\mathbf{h}_{t-1}$ and $\mathbf{h}_t$ denote the previous and updated hidden states, respectively; $\mathbf{W}$ and $\mathbf{U}$ are the input and recurrent weight matrices; $\mathbf{b}^x$ and $\mathbf{b}^h$ are their corresponding biases; and $\odot$ denotes element-wise multiplication. The update gate $\mathbf{z}_t$ and reset gate $\mathbf{q}_t$ use the sigmoid function σ , while the candidate hidden state $\tilde{\mathbf{h}}_t$ uses the hyperbolic tangent $\tanh$.

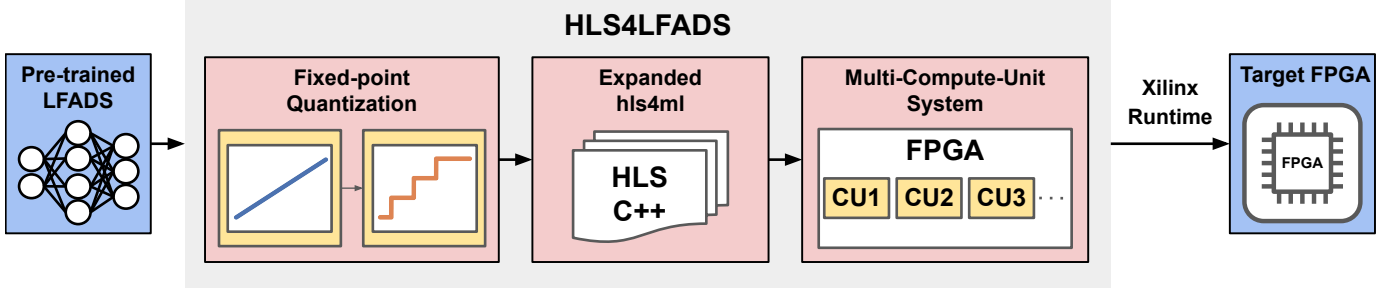


Fig. 3. Overview of the proposed HLS4LFADS framework.

These operations contain input and recurrent matrix–vector products, accumulation, nonlinear activation, and recurrent state updates. Their numerical formats therefore determine the required weight, accumulator, activation, and hidden-state precisions in the fixed-point implementation described in Section III-A.

B. RNN Acceleration with hls4ml

Developing FPGA accelerators directly in hardware description languages requires substantial hardware expertise and engineering effort. High-Level Synthesis (HLS) reduces this burden by generating hardware from C/C++ descriptions [17]. Built on HLS, `hls4ml` converts trained neural networks into configurable FPGA implementations and has been widely used for low-latency inference [16], [18], [19].

`hls4ml` generates fully on-chip dataflow designs in which model parameters and intermediate results remain on chip, and each layer is implemented as a dedicated hardware block. Successive layer blocks form a pipeline and can operate concurrently where data dependencies permit. Configurable design options, such as numerical precision, I/O architecture, and reuse factor (RF), enable resource–latency trade-offs. In particular, RF controls multiplier sharing within matrix multiplications. A higher RF increases multiplier sharing, reducing arithmetic parallelism and resource usage at the cost of increased latency.

Prior work extended `hls4ml` to support recurrent layers, including GRU and long short-term memory (LSTM) implementations, and demonstrated low-latency FPGA inference using post-training fixed-point quantization [18]. At the time this work was conducted, however, `hls4ml` did not support the bidirectional GRU processing required by the LFADS encoder considered here. We extend its recurrent backend and model-conversion flow to support the complete LFADS inference graph while retaining the configurable options of `hls4ml`. The resulting flow automatically generates LFADS accelerators with adjustable resource–latency trade-offs.

III. THE HLS4LFADS FRAMEWORK

Figure 3 presents an overview of HLS4LFADS. Starting from a pretrained floating-point LFADS model, we determine fixed-point formats through layer-wise calibration and fine-tune the model using QAT. The resulting quantized model is converted into an HLS accelerator through the extended

`hls4ml` flow. Multiple generated accelerators are then instantiated as independent compute units (CUs) and executed through Xilinx Runtime (XRT) [20].

A. Fixed-Point Quantization for LFADS

Fixed-point arithmetic maps efficiently to FPGA hardware and eliminates floating-point operations during deployed inference. We quantize the inference path of the LFADS model evaluated in this work using uniform fixed-point quantization with QKeras [21]. A signed fixed-point format is denoted by $\langle W, I \rangle$, where W is the total bit width and I is the integer width, including the sign bit. The fractional width is $F = W - I$, giving a quantization step

$$\Delta = 2^{-F}, \quad (5)$$

with representable range

$$[-2^{I-1}, 2^{I-1} - \Delta]. \quad (6)$$

Values are rounded to the nearest quantization level and clipped to this range. For a fixed W , I should be as small as possible while covering the tensor range, maximizing F and quantization resolution.

We assign a fixed-point format to each quantized tensor along the inference path, including weights, biases, layer inputs, and recurrent hidden states. Notably, the decoder GRU initial state and latent factors are also quantized. Within each GRU update, the gate outputs $\mathbf{z}_t$ and $\mathbf{q}_t$, candidate hidden state $\tilde{\mathbf{h}}_t$, and updated hidden state $\mathbf{h}_t$ are requantized to their assigned formats before subsequent use. Total bit widths are specified before range calibration. For model parameters, the bit width B_p is user-configurable; in our evaluated configurations, a common B_p is used for all weights and biases. We use 16 bits for all activations and recurrent states because the decoder GRU initial state and latent factors can have wider dynamic ranges than the bounded GRU nonlinearities. At smaller total bit widths, covering these ranges leaves fewer fractional bits, increases Δ , and produces greater quantization error in downstream computations.

Given these total bit widths, we determine each tensor’s integer width through layer-wise range calibration on the training set. Integer widths for weights and biases are derived from their pretrained floating-point tensor ranges. Activation calibration follows the inference order from the bidirectional

GRU through the posterior-mean projection, decoder GRU, and factor and firing-rate readouts. At each stage, we measure the output range using the already-quantized upstream subgraph, select the smallest integer width that covers this range, and assign the remaining bits to the fractional part. For sigmoid and tanh outputs, integer widths follow directly from their known bounded ranges.

The encoder GRU hidden states require no range calibration because they are initialized to zero and remain bounded within $[-1, 1]$. We therefore assign an integer width of $I = 1$. In contrast, the decoder GRU is initialized by an inferred initial state whose range is not restricted. We therefore run the complete decoder GRU sequence over the calibration data and collect the hidden-state range across all samples and time steps using the quantized upstream subgraph and decoder GRU operations. After assigning all fixed-point formats, we initialize the quantized model from its corresponding floating-point checkpoint and fine-tune it using QAT.

B. LFADS Mapping and hls4ml Extensions

The quantized LFADS inference graph is converted into an HLS implementation through the extended `hls4ml` model-conversion flow. Operations already supported by `hls4ml`, including dense layers, the unidirectional GRU, and nonlinear activations, use existing primitives. To support the complete LFADS inference graph, we extend the recurrent backend with bidirectional GRU processing and propagate the assigned fixed-point formats to the generated HLS implementation.

The bidirectional GRU consists of independent forward and backward engines. Because the backward GRU processes the sequence in reverse temporal order, the complete input sequence is first stored in an on-chip buffer. The forward engine reads the buffer chronologically, whereas the backward engine reads it in reverse. Their final hidden states are concatenated to form the encoder representation.

During conversion, each tensor’s fixed-point format in the quantized model is mapped to the corresponding HLS type. For each matrix multiplication in the GRU and dense layers, the operand types determine an accumulator type with sufficient precision for the multiply-accumulate (MAC) reduction, preventing overflow. Sigmoid and tanh functions are implemented using the existing lookup-table-based `hls4ml` primitives [19], and their outputs use the assigned 16-bit activation formats.

C. Multi-Compute-Unit FPGA Deployment

Each generated LFADS accelerator is packaged as an FPGA kernel, with each kernel instance corresponding to one CU. The deployment configuration determines the number of CUs instantiated in the FPGA design, and XRT invokes them independently for concurrent inference.

Figure 4 illustrates an example three-CU deployment and its high-bandwidth memory (HBM) connectivity. Each CU exposes three AXI4 memory-mapped master interfaces for the encoder input, decoder input, and model output. These interfaces access buffers stored in the FPGA HBM. The

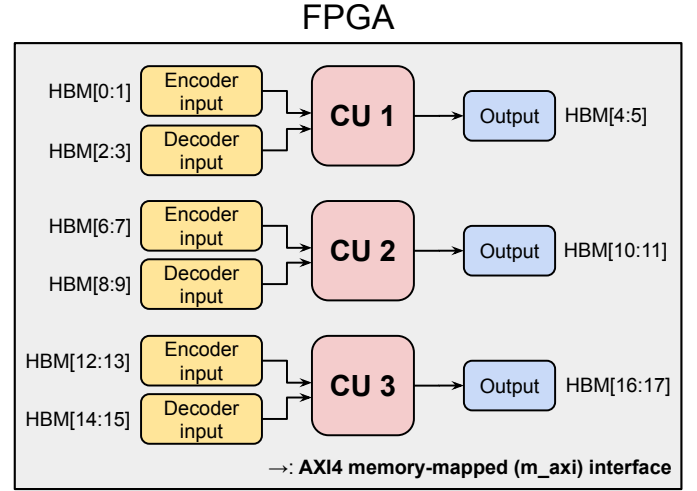


Fig. 4. Example three-CU LFADS deployment and high-bandwidth memory (HBM) connectivity. Each CU exposes three AXI4 memory-mapped master interfaces for the encoder input, decoder input, and model output. Labels indicate the HBM pseudo-channel pairs assigned to each interface.

autonomous LFADS model evaluated in this work uses zero-valued decoder inputs, but the decoder-input interface is retained to support LFADS variants that accept external decoder inputs. To reduce memory contention during concurrent execution, each CU interface is assigned to a non-overlapping group of two HBM pseudo-channels [22].

IV. EXPERIMENTAL RESULTS

We evaluate HLS4LFADS from three perspectives: quantization performance, single-CU FPGA implementation trade-offs, and multi-CU scaling for concurrent inference.

A. Datasets and Evaluation Metrics

Datasets. We evaluate two datasets with different temporal resolutions and sequence lengths. The monkey-reaching dataset (Reach) used in [9] contains 170 trials recorded from 70 neurons, with each trial represented by 73 time steps using a 10-ms bin size. The synthetic Lorenz dataset (Lorenz), generated following [9], [23], contains 1,600 trials from 30 simulated neurons. Each one-second trial uses a 6-ms bin size, resulting in 167 time steps. For both datasets, 80% of the trials are used for training and 20% for testing.

Evaluation Metrics. We evaluate model performance using Poisson log-likelihood (PLL), for which larger (less negative) values indicate a better fit to the observed spike counts. In addition, we report task-specific coefficient-of-determination (R^2) metrics. For Reach, R^2 measures hand-position prediction from the inferred latent factors using ridge regression. For Lorenz, R^2 measures firing-rate reconstruction against the ground-truth rates.

B. Experimental Setup

Model Configuration and Training. We follow the LFADS architecture and floating-point training procedure in [9], with 64 hidden units in each GRU. The latent-factor dimension is

TABLE I
LFADS PERFORMANCE ACROSS LATENT-FACTOR DIMENSIONS AND
QUANTIZATION CONFIGURATIONS.

Factor d_f	Config.	Reach		Lorenz	
		PLL	R^2 (%)	PLL	R^2 (%)
8	FP32	-1774.78	88.36	-862.10	95.62
	W16A16	-1780.12	85.18	-860.95	96.44
	W8A16	-1779.42	85.05	-860.60	96.78
	W6A16	-1777.36	87.04	-860.79	96.69
	W4A16	-1784.23	87.51	-864.88	92.82
4	FP32	-1783.83	87.54	-863.07	94.90
	W16A16	-1782.53	87.99	-862.63	94.67
	W8A16	-1782.80	87.91	-862.80	94.66
	W6A16	-1783.44	88.46	-863.23	94.05
	W4A16	-1792.28	85.33	-886.75	72.66
2	FP32	-1797.06	43.80	-879.41	85.44
	W16A16	-1797.27	41.27	-879.08	85.99
	W8A16	-1797.32	41.72	-879.19	85.86
	W6A16	-1798.00	42.99	-879.42	85.49
	W4A16	-1804.70	45.02	-884.47	80.59

specified for each experiment. QAT uses cosine annealing with an initial learning rate of 10^{-3} and the same number of epochs as the floating-point model.

Hardware Implementation. The FPGA accelerators are implemented on an Xilinx Alveo U55C targeting a clock frequency of 200 MHz. The board is hosted by an AMD EPYC 7742 processor through PCIe Gen4 $\times 16$ and controlled using XRT 2022.2. HLS synthesis and FPGA implementation are performed using Vitis 2023.2. We configure `hls4ml` with the `io_stream` I/O architecture and the static recurrent implementation from [18]. CPU and GPU latency baselines are evaluated on an Intel Core i5-11400 and an NVIDIA RTX A4000, respectively.

Latency Measurement. Unless otherwise specified, all latency experiments use a batch size of one. Each reported latency is the mean and standard deviation over 1,000 measured runs. For the CPU and GPU, 1,000 untimed warm-up runs are performed before measurement. The timed components included in each latency measurement are specified in the corresponding experiment.

C. Quantization Performance

We evaluate the joint effects of latent-factor dimension and fixed-point quantization on LFADS performance. Models with latent-factor dimensions $d_f \in \{2, 4, 8\}$ are evaluated using parameter widths $B_p \in \{4, 6, 8, 16\}$ bits, while activations and recurrent states remain at 16 bits. We denote each fixed-point configuration as WB_pA16 , indicating B_p -bit weights and biases with 16-bit activations and recurrent states; FP32 denotes the floating-point baseline. Table I reports the results.

The FP32 results isolate the effect of latent-factor dimension. Reducing d_f from 8 to 4 causes only minor performance changes on both datasets. In contrast, at $d_f = 2$, the Reach behavioral R^2 decreases from 88.36% to 43.80%, while the Lorenz reconstruction R^2 decreases from 95.62% to 85.44%.

TABLE II
SINGLE-CU POST-IMPLEMENTATION RESOURCE UTILIZATION AND
KERNEL LATENCY FOR THE W6A16 LORENZ MODEL ($d_f = 4$) ACROSS
RF SETTINGS.

RF	Resource utilization (%)				Kernel latency (μs)
	LUT	FF	BRAM	DSP	
32	13.97	7.49	16.06	35.64	235.3 ± 5.4
64	14.12	7.60	13.48	23.48	261.4 ± 5.1
128	16.57	7.64	12.10	17.44	371.6 ± 9.2
256	17.58	7.71	11.33	14.39	596.9 ± 8.7

TABLE III
SINGLE-CU POST-IMPLEMENTATION RESOURCE UTILIZATION AND
KERNEL LATENCY FOR THE REACH MODEL ($d_f = 4$, RF = 256) ACROSS
QUANTIZATION CONFIGURATIONS.

Config.	Resource utilization (%)				Kernel latency (μs)
	LUT	FF	BRAM	DSP	
W16A16	24.05	9.93	14.14	15.14	279.9 ± 7.0
W8A16	19.59	8.65	12.21	15.12	278.2 ± 1.6
W6A16	19.29	8.43	11.72	15.09	280.8 ± 2.8
W4A16	18.00	7.84	11.06	9.82	281.5 ± 10.2

Thus, the four-dimensional models retain performance close to the eight-dimensional models, whereas the two-dimensional models impose a substantially stronger representational constraint. This trend is consistent with the results reported in [9].

Across the evaluated latent-factor dimensions, W16A16, W8A16, and W6A16 generally remain close to their FP32 counterparts. In particular, the R^2 values of W6A16 differ from FP32 by at most 1.32 percentage points across all six dataset–dimension combinations. The PLL values also show only limited variation, with no consistent degradation as the parameter width is reduced from 16 to 6 bits. These results indicate that parameter precision can be reduced to 6 bits without substantial performance loss across the evaluated latent-factor dimensions.

The effect of 4-bit quantization depends more strongly on the latent-factor dimension and dataset. For the eight-dimensional models, W4A16 reduces R^2 by only 0.85 percentage points on Reach and 2.80 percentage points on Lorenz. However, for the four-dimensional Lorenz model, reconstruction R^2 decreases from 94.90% to 72.66%. With two latent factors, the Lorenz degradation is more moderate, from 85.44% to 80.59%. The interaction is therefore not strictly monotonic with latent-factor dimension, indicating that the suitability of 4-bit parameters depends on the specific model configuration.

D. Single-CU Resource–Latency Trade-off

At $d_f = 4$, we select W4A16 for Reach and W6A16 for Lorenz as the narrowest configurations that retain R^2 close to their FP32 baselines. We characterize the single-CU design along two axes: Table II sweeps RF for the Lorenz model, while Table III sweeps B_p for the Reach model at RF = 256.

TABLE IV
BATCH-SIZE-ONE LATENCY FOR ONE TO THREE LFADS MODEL
INSTANCES ON GPU, CPU, AND FPGA.

# Models	Latency (ms)		
	GPU (FP32)	CPU (FP32)	FPGA
Reach (10-ms bin; FPGA W4A16)			
1	2.226 ± 0.228	4.551 ± 0.223	0.330 ± 0.014
2	3.633 ± 0.233	5.614 ± 0.330	0.340 ± 0.012
3	5.462 ± 0.466	6.723 ± 0.268	0.367 ± 0.015
Lorenz (6-ms bin; FPGA W6A16)			
1	3.491 ± 0.217	8.706 ± 0.838	0.651 ± 0.019
2	6.115 ± 0.261	11.867 ± 1.266	0.683 ± 0.017
3	8.840 ± 0.326	13.684 ± 0.508	0.696 ± 0.017

Post-implementation utilization of lookup tables (LUTs), flip-flops (FFs), block RAMs (BRAMs), and digital signal processing (DSP) blocks is reported for the CU kernel alone as a percentage of available U55C resources. Latency covers kernel execution only and excludes host-device transfers. Because the two models have different sequence lengths, we interpret trends within each sweep rather than comparing their absolute latencies.

Table II shows that RF primarily trades DSP utilization for latency. At $RF = 32$, the Lorenz accelerator achieves the lowest mean latency of $235.3 \mu s$ using 35.64% of DSPs. Increasing RF to 256 reduces DSP utilization to 14.39% but increases latency to $596.9 \mu s$, a $2.54\times$ slowdown. Other resources vary less: FF utilization remains nearly constant, BRAM decreases from 16.06% to 11.33%, and LUT utilization increases from 13.97% to 17.58%.

At fixed RF, reducing parameter width has little impact on latency (Table III). From W16A16 to W4A16, latency remains within 278.2–281.5 μs because the recurrent schedule and RF are unchanged. However, LUT, FF, BRAM, and DSP utilization decrease by 6.05, 2.09, 3.08, and 5.32 percentage points, respectively. The larger DSP reduction at W4A16 reflects the HLS tool mapping narrow multipliers to LUTs instead of DSP blocks [19]. Unlike increasing RF, which reduces DSP utilization at the cost of latency, quantization reduces the overall per-CU footprint with negligible latency impact, increasing the potential to instantiate additional CUs for concurrent multi-model inference.

E. Multi-CU Scaling for Multi-Model Inference

We evaluate multi-model scaling on the CPU, GPU, and FPGA using one to three identical copies of each LFADS model. Each copy has a separate encoder input, decoder input, and output. All model instances use $d_f = 4$. The FPGA implementations use $RF = 256$, with W4A16 for Reach and W6A16 for Lorenz, while the CPU and GPU baselines use the corresponding FP32 models.

For the CPU and GPU baselines, we construct a single Keras [24] graph containing the corresponding number of identical LFADS branches. The complete graph is evaluated

in one inference call as a proxy for concurrent multi-model execution on a shared processor. For the CPU and GPU, latency is measured from prepared input tensors through host-visible outputs. FPGA latency is measured end to end, including host-to-device transfer, kernel execution, and device-to-host transfer.

For Reach, all evaluated platforms remain below the 10-ms bin interval for up to three model instances. From one to three models, GPU latency increases from 2.226 to 5.462 ms, or 145.4%, while CPU latency increases from 4.551 to 6.723 ms, or 47.7%. In comparison, FPGA end-to-end latency increases by only 11.2%, from 0.330 to 0.367 ms, and remains below one millisecond. This leaves substantial timing headroom within the 10-ms interval for other closed-loop processing stages or smaller temporal bin sizes.

The shorter 6-ms Lorenz bin imposes a more restrictive latency requirement. The single-model GPU completes in 3.491 ms, but its two- and three-model latencies increase to 6.115 and 8.840 ms, respectively, exceeding the bin interval. CPU latency exceeds 6 ms even for one model. In contrast, FPGA end-to-end latency remains below one millisecond, increasing from 0.651 ms for one CU to 0.696 ms for three CUs. This represents an increase of only 6.9%, compared with 153.2% on the GPU and 57.2% on the CPU. Across the Reach and Lorenz workloads, three-CU execution requires only $1.11\times$ and $1.07\times$ the corresponding single-CU latencies, respectively, indicating that scaling from one to three concurrent models introduces only limited end-to-end latency overhead on the FPGA.

V. CONCLUSION AND FUTURE WORK

This work presented HLS4LFADS, an end-to-end framework for low-latency, concurrent LFADS inference on FPGAs. HLS4LFADS combines calibrated fixed-point quantization and QAT with an extended `hls4m1` flow for automatic accelerator generation and multi-CU deployment. Across the Reach and Lorenz tasks, W6A16 maintained R^2 within 1.32 percentage points of FP32, while quantization reduced the per-CU resource footprint with negligible kernel-latency impact. Scaling from one to three CUs increased end-to-end latency by only 11.2% and 6.9% for Reach and Lorenz, respectively, remaining below their corresponding bin intervals. Future work will explore finer-grained precision configurations and integration into a complete closed-loop system.

ACKNOWLEDGMENT

OpenAI ChatGPT was used during manuscript preparation to assist with drafting, language editing, restructuring, and presentation of the authors' technical content across the Abstract and Sections I–V. All research design, implementation, experiments, data analysis, technical conclusions, and final manuscript content were reviewed and determined by the authors.

REFERENCES

- [1] N. A. Steinmetz, C. Koch, K. D. Harris, and M. Carandini, "Challenges and opportunities for large-scale electrophysiology with neuropixels probes," *Current Opinion in Neurobiology*, vol. 50, pp. 92–100, 2018, neurotechnologies. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0959438817303161>
- [2] M. M. Churchland, J. P. Cunningham, M. T. Kaufman, J. D. Foster, P. Nuyujukian, S. I. Ryu, and K. V. Shenoy, "Neural population dynamics during reaching," *Nature*, vol. 487, no. 7405, pp. 51–56, 2012.
- [3] C. Pandarinath, D. J. O'Shea, J. Collins, R. Jozefowicz, S. D. Stavisky, J. C. Kao, E. M. Trautmann, M. T. Kaufman, S. I. Ryu, L. R. Hochberg *et al.*, "Inferring single-trial neural population dynamics using sequential auto-encoders," *Nature methods*, vol. 15, no. 10, pp. 805–815, 2018.
- [4] D. Sussillo, R. Jozefowicz, L. F. Abbott, and C. Pandarinath, "Lfads - latent factor analysis via dynamical systems," *arXiv*, Aug 2016, accessed: Aug. 03, 2023. [Online]. Available: <http://arxiv.org/abs/1608.06315>
- [5] R. D. Flint, M. C. Tate, K. Li, J. W. Templer, J. M. Rosenow, C. Pandarinath, and M. W. Sutzky, "The representation of finger movement and force in human motor and premotor cortices," *Eneuro*, vol. 7, no. 4, 2020.
- [6] B. M. Karpowicz, Y. H. Ali, L. N. Wimalasena, A. R. Sedler, M. R. Keshkaran, K. Bodkin, X. Ma, D. B. Rubin, Z. M. Williams, S. S. Cash *et al.*, "Stabilizing brain-computer interfaces through alignment of latent dynamics," *Nature Communications*, vol. 16, no. 1, p. 4662, 2025.
- [7] J. Huang, P. Tostado-Marcos, S. M. Narasimha, A. N. Patel, E. M. Arneodo, T. Q. Gentner, G. Mishne, and V. Gilja, "Guiding brain-to-vocalization decoder design using structured generalization error," in *2024 46th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, 2024, pp. 1–4.
- [8] C. Li, Y. Zhang, T. Wang, X. Xu, Q. Wang, B. Xu, and H. Cui, "Generative decoding of intracortical neuronal signals for online control of robotic arm to intercept moving objects," in *Journal of Physics: Conference Series*, vol. 1576, no. 1. IOP Publishing, 2020, p. 012057.
- [9] C. Hurwitz, A. Srivastava, K. Xu, J. Jude, M. Perich, L. Miller, and M. Hennig, "Targeted neural dynamical modeling," *Advances in Neural Information Processing Systems*, vol. 34, pp. 29 379–29 392, 2021.
- [10] Y. H. Ali, K. Bodkin, M. Rigotti-Thompson, K. Patel, N. S. Card, B. Bhaduri, S. R. Nason-Tomaszewski, D. M. Mifsud, X. Hou, C. Nicolas *et al.*, "Brand: a platform for closed-loop experiments with deep network models," *Journal of Neural Engineering*, vol. 21, no. 2, p. 026046, 2024.
- [11] M. Nolan, B. Pesaran, E. Shlizerman, and A. Orsborn, "Multi-block rnn autoencoders enable broadband ecog signal reconstruction," *bioRxiv*, 2022. [Online]. Available: <https://www.biorxiv.org/content/early/2022/09/07/2022.09.07.507004>
- [12] M. A. Lebedev and M. A. Nicolelis, "Brain-machine interfaces: past, present and future," *TRENDS in Neurosciences*, vol. 29, no. 9, pp. 536–546, 2006.
- [13] Y. Qi, B. Liu, Y. Wang, and G. Pan, *Dynamic ensemble modeling approach to nonstationary neural decoding in brain-computer interfaces*. Red Hook, NY, USA: Curran Associates Inc., 2019.
- [14] F. Yu, D. Wang, L. Shangguan, M. Zhang, X. Tang, C. Liu, and X. Chen, "A survey of large-scale deep learning serving system optimization: Challenges and opportunities," 2022. [Online]. Available: <https://arxiv.org/abs/2111.14247>
- [15] Y.-H. Lai, E. Ustun, S. Xiang, Z. Fang, H. Rong, and Z. Zhang, "Programming and synthesis for software-defined fpga acceleration: status and future prospects," *ACM Transactions on Reconfigurable Technology and Systems (TRETs)*, vol. 14, no. 4, pp. 1–39, 2021.
- [16] F. Fahim, B. Hawks, C. Herwig, J. Hirschauer, S. Jindariani, N. Tran, L. P. Carloni, G. Di Guglielmo, P. Harris, J. Krupa *et al.*, "hls4ml: An open-source codesign workflow to empower scientific low-power machine learning devices," *arXiv preprint arXiv:2103.05579*, 2021.
- [17] J. Cong, J. Lau, G. Liu, S. Neuendorffer, P. Pan, K. Vissers, and Z. Zhang, "Fpga hls today: successes, challenges, and opportunities," *ACM Transactions on Reconfigurable Technology and Systems (TRETs)*, vol. 15, no. 4, pp. 1–42, 2022.
- [18] E. E. Khoda, D. Rankin, R. T. de Lima, P. Harris, S. Hauck, S.-C. Hsu, M. Kagan, V. Loncar, C. Paikara, R. Rao *et al.*, "Ultra-low latency recurrent neural network inference on fpgas for physics applications with hls4ml," *Machine Learning: Science and Technology*, vol. 4, no. 2, p. 025004, 2023.
- [19] J. Duarte, S. Han, P. Harris, S. Jindariani, E. Kreinar, B. Kreis, J. Ngadiuba, M. Pierini, R. Rivera, N. Tran, and Z. Wu, "Fast inference of deep neural networks in fpgas for particle physics," *Journal of Instrumentation*, vol. 13, no. 07, p. P07027–P07027, Jul. 2018. [Online]. Available: <http://dx.doi.org/10.1088/1748-0221/13/07/P07027>
- [20] Xilinx, *Xilinx Runtime (XRT) Documentation*, AMD Xilinx, 2023, accessed: February 10, 2026. [Online]. Available: <https://xilinx.github.io/XRT/>
- [21] C. N. Coelho, A. Kuusela, S. Li, H. Zhuang, J. Ngadiuba, T. K. Aarrestad, V. Loncar, M. Pierini, A. A. Pol, and S. Summers, "Automatic heterogeneous quantization of deep neural networks for low-latency inference on the edge for particle detectors," *Nature Machine Intelligence*, vol. 3, no. 8, p. 675–686, Jun. 2021. [Online]. Available: <http://dx.doi.org/10.1038/s42256-021-00356-5>
- [22] AXI High Bandwidth Memory Controller LogiCORE IP Product Guide, Advanced Micro Devices, Inc., 2025, version 1.0. [Online]. Available: <https://docs.amd.com/t/en-US/pg276-axi-hbm>
- [23] J. Ye and C. Pandarinath, "Representation learning for neural population activity with neural data transformers," *arXiv preprint arXiv:2108.01210*, 2021.
- [24] F. Chollet *et al.*, "Keras," <https://keras.io>, 2015.