Analyzing inference workloads for spatiotemporal modeling

Milan Jain [*], Nicolas Bohm Agostini, Sayan Ghosh, Antonino Tumeo

Pacific Northwest National Laboratory, Richland, WA, USA

A R T I C L E I N F O

Keywords:
Inference
Spatiotemporal modeling
Performance analysis and profiling
GPU
High-level synthesis

A B S T R A C T

Ensuring power grid resiliency, forecasting climate conditions, and optimization of transportation infrastructure are some of the many application areas where data is collected in both space and time. Spatiotemporal modeling is about modeling those patterns for forecasting future trends and carrying out critical decision-making by leveraging machine learning/deep learning. Once trained offline, field deployment of trained models for near real-time inference could be challenging because performance can vary significantly depending on the environment, available compute resources and tolerance to ambiguity in results. Users deploying spatiotemporal models for solving complex problems can benefit from analytical studies considering a plethora of system adaptations to understand the associated performance-quality trade-offs.

To facilitate the co-design of next-generation hardware architectures for field deployment of trained models, it is critical to characterize the workloads of these deep learning (DL) applications during inference and assess their computational patterns at different levels of the execution stack. In this paper, we develop several variants of deep learning applications that use spatiotemporal data from dynamical systems. We study the associated computational patterns for inference workloads at different levels, considering relevant models (Long short-term Memory, Convolutional Neural Network and Spatio-Temporal Graph Convolution Network), DL frameworks (Tensorflow and PyTorch), precision (FP16, FP32, AMP, INT16 and INT8), inference runtime (ONNX and AI Template), post-training quantization (TensorRT) and platforms (Nvidia DGX A100 and Sambanova SN10 RDU).

Overall, our findings indicate that although there is potential in mixed-precision models and post-training quantization for spatiotemporal modeling, extracting efficiency from contemporary GPU systems might be challenging. Instead, co-designing custom accelerators by leveraging optimized High Level Synthesis frameworks (such as SODA High-Level Synthesizer for customized FPGA/ASIC targets) can make workload-specific adjustments to enhance the efficiency.

1. Introduction

Graphics Processing Units (GPUs) are well positioned to be the *de facto* training architecture, owing to mature vendor-supported software ecosystems to tackle the computational challenges for supporting evolving machine learning models. As such, the training performance of contemporary foundational models at scale generally depends on the capacity of the GPU computing infrastructure [1–4]. However, while training performance is important to scale models to extraordinarily large number of parameters, once these are deployed, their effectiveness depends only on inference.

Inference takes precedence over training because once a model is deployed, it continuously processes new, unseen data to make predictions or decisions. Inference, especially in recurrent or autoregressive networks (such as predicting the next word), often requires multiple iterations, resulting in more computation than traditional DL models. These models are deployed across a variety of applications, from edge devices of all sizes and types to single-machine servers and large data centers, where performance requirements vary based on the specific use case. Achieving high performance in inference involves optimizing for latency, accuracy, energy efficiency, area efficiency, or a combination of these metrics. To address the evolving machine learning workloads and diverse inference use cases, vendors and researchers are developing new specialized hardware and software solutions [5–7].

Contemporary inference workloads for a wide range of application scenarios currently utilize very diverse hardware platforms — from general-purpose Central Processing Units (CPUs) and GPUs to domain-specific Artificial Intelligence Accelerators (AIA) with various levels of specialization to the models and use cases, leveraging reconfigurable

[*] Corresponding author.
E-mail addresses: milan.jain@pnnl.gov (M. Jain), nicolas.agostini@pnnl.gov (N.B. Agostini), sayan.ghosh@pnnl.gov (S. Ghosh), antonino.tumeo@pnnl.gov (A. Tumeo).

Table 1
Current spatiotemporal modeling applications — models, platforms and precision considerations. The underlined models are studied as part of this paper.

Domain	Models used	Numerical precision			Hardware platforms			Post training optimization & quantization		
		FP16/AMP	FP32	FP64	CPU	GPU	AIA	ONNX	INT8	FP16
Weather Forecasting	LSTM, CNN,	[10–12]	[10,11,13]	[12,14]	[10–12]	[10]	[15,16]	[17]	[18,19]	[18,19]
Power Systems	Random Forest, GNN, GRU,	[20]	[21–23]	[24–26]	[22,26,27]	[25,28]	[20,29,30]	[29]	[28,30]	[30,31]
Traffic Forecasting	Transformer	[32,33]	[32]		[33]	[34]	[32,35,36]	[32]	[32,36]	[35]

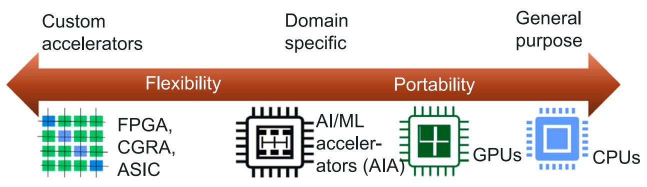

Fig. 1. Platforms for contemporary inference workloads.

architectures such as coarse-grained reconfigurable arrays (CGRAs) or field programmable gate arrays (FPGAs) or even application specific integrated circuits (ASICs), as shown in Fig. 1.

On the other hand, despite the investments around foundational models, practitioners must adapt existing deep learning models according to specific application scenarios. One important area where there are no standard models to represent the domain is *spatiotemporal modeling*, used to capture temporal relationships in data, which is particularly relevant due to the pervasiveness of time series prediction.

In time series prediction, historical sequences from a set of spatially correlated sensors are used to train a model, and, given a history of observations, the trained model is used to predict the evolution of the dynamical system in time. When a system consists of several interrelated nodes (not necessarily interconnected), the problem becomes a multivariate time series prediction that typically requires spatiotemporal data for model training and inference. Fig. 2 depicts the data flow in a typical multivariate time series prediction application.

Unlike cross-sectional data (*e.g.*, images), time-series data ingestion is usually interleaved to maintain the original sequence, posing challenges to parallelization in the context of machine learning prediction. Consequently, spatiotemporal modeling applications can spend relatively longer times for data transfer, transformation, and memory accesses compared to floating-point computation-intensive DL applications, as mentioned in [8,9] in the context of model training. Therefore, execution characteristics of spatiotemporal modeling cannot be extrapolated from existing machine learning benchmarks. For training and inference of arbitrary machine learning models, data is typically divided into mini-batches. For spatiotemporal data, a mini-batch consists of multiple overlapping windows, which can significantly increase the memory requirements for maintaining halos. Running inference on large time series data, preprocessed in near real-time while ensuring data quality and consistency across temporal and spatial dimensions, necessitates a profound understanding of spatiotemporal models for dynamic systems, especially given hardware constraints on memory, runtime, and communication.

While these factors impact both training and inference in general, inference in spatiotemporal modeling depends on the granularity of the prediction, and as such, can leverage specific optimizations in the underlying model architecture, floating-point precision, hardware architecture, and underlying software methodologies. In this work, we discuss various aspects of spatiotemporal inference workloads, such as, characterizing the underlying computational motifs for co-design, applying various software optimizations to leverage mixed-precision, analyze the performance on diverse platforms, and ultimately devising specialized and efficient hardware solutions. Specifically, the contributions of this work are as follows:

- Capture unique characteristics of the inference workloads for spatiotemporal modeling using suitable CNN, LSTM, and STGCN architectures on Power Grid and Climate datasets.
- Describe the underlying computational motifs of prevalent spatiotemporal models and quantify performances for standalone transformation/mixed-precision model adaptations at different levels.
- Explore methodologies to optimize spatiotemporal workloads — evaluating contemporary AI/ML frameworks (*e.g.*, PyTorch and Tensorflow), numerical precisions (AMP, INT8, INT16, FP16, FP32, and FP64), hardware platforms (NVIDIA A100™ and Sambanova RDU™), and quantization methodologies (via ONNX, AI Template and NVIDIA™ TensorRT).
- Employ the SODA HLS toolchain to develop highly specialized inference accelerators (targeting both FPGA and ASIC) for spatiotemporal models and discuss relevant optimizations by exploiting reduced precision and the inherent parallelism in the model layers.

The rest of the paper is organized as follows. We motivate the work and provide a brief background on spatiotemporal modeling in Section 2. We discuss related work in Section 3. Section 4 discusses the main principles behind our proxy application and outlines our methodology. Section 5 analyzes the baseline performance/scalability, quantifies the impact of the computational motifs, and discusses custom accelerators via HLS. Section 7 concludes the paper.

2. Motivation and background

Spatiotemporal modeling is still an emerging area for Machine Learning/Deep Learning (DL) space with potential to impact power systems [37], climate analysis [38,39], and transportation [40], among many other critical real-world applications [41]. For instance, the 2023 Urban Mobility Report by Texas A&M [42] revealed that spatiotemporal models can optimize the traffic flow, mitigating congestion by up to 20%. Similarly, the National Oceanic and Atmospheric Administration's 2023 Science Report [43] highlighted a 15% improvement in hurricane forecasting accuracy, enabling disaster preparedness and diminishing the severity of extreme weather events, thereby potentially saving lives and reducing economic losses. Furthermore, in power grid management, Pacific Gas & Electric (PG&E, California) reported a 10% reduction in peak demand and a 5% reduction in energy waste, attributable to the deployment of spatiotemporal models for optimizing energy storage integration [44]. Additionally, in the agriculture context, the implementation of these models has yielded a 15% increase in crop yields and a 20% reduction in water consumption [45].

Since these models are applicable across a wide range of use cases, their real-world implementation and operational environments differ significantly between applications. The deployment environments exhibit unique hardware configurations, with constraints on computing power, memory, and communication bandwidth. Table 1 categorizes some studies from three such application domains and indicates their corresponding configurations, including variety in model architectures, floating-point precision, hardware architectures, and post-training optimizations and quantization.

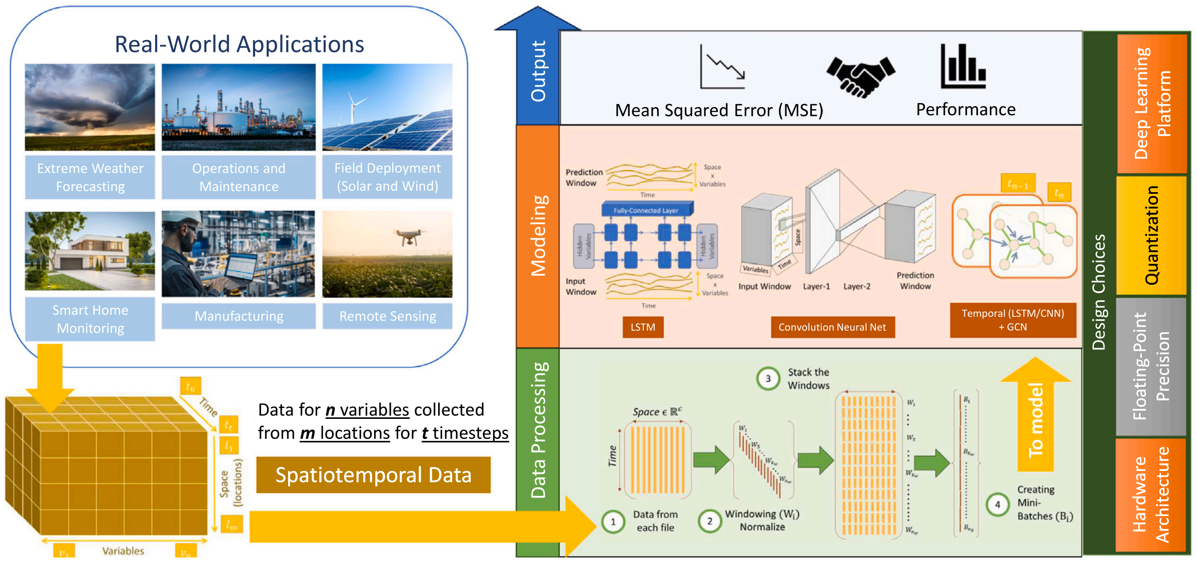

Fig. 2. Spatiotemporal models, crucial for applications such as extreme weather forecasting and assessing cascading power grid failures, face unique challenges due to overlapped time-series data (*i.e.*, windows). Our study is a quantitative assessment of the impact of data, model architecture, ML frameworks and software/hardware optimizations on spatiotemporal inference workloads.

2.1. Model architecture

Prior to Deep Learning (DL), statistics and numerical methods were commonplace for spatiotemporal modeling. DL offers major advantages in considering larger datasets and learning complex relationships, thereby enhancing the accuracy and efficiency of spatiotemporal models. DL models typically used in spatiotemporal modeling belong to the fraternity of sequence-to-sequence modeling like Recurrent Neural Network (RNN) [46], Long Short-Term Memory (LSTM) [47], Graph Neural Networks (GNN) [48], and Transformer [49]. Recently, pre-trained foundational models based on language model architecture (LLMs) have been used for time series forecasting [50]. Often, the DL models designed for spatiotemporal modeling stack one or more of these layers together, which are then further customized (including the size of the historical window, prediction window, and the time scale) to suit the application needs.

Studying the underlying computational patterns for emerging spatiotemporal models can lead to new insights to improve the software and hardware stack design for their inference workloads, enabling the identification of new features and co-design appropriate solutions depending on the use case. We consider relatively established spatiotemporal modeling architectures – LSTM, CNN, and STGCN – across various execution models and implementation variants. Each architecture offers distinct advantages: LSTM enables recurrence, CNN minimizes computational complexity, and STGCN incorporates spatial relationships. Additionally, Transformers and LLM-based foundational models are also gaining traction as promising alternatives, but their effectiveness in long-term time series forecasting, particularly regarding prediction accuracy and computational cost, remains an open research question [51].

2.2. Floating-point precision

Since training and inference in DL models for spatiotemporal modeling are data-intensive, floating-point precision is a key parameter that can vary depending on the numerical accuracy requirements of the application. For instance, while ML/DL applications for environmental monitoring (temperature, humidity forecasting) can work with reduced floating-point precision, many power-system applications, such as power flow calculation and transient stability analysis, require double-precision to retain relatively high accuracy [24]. Due to the need for specialization, commodity hardware might be sub-optimal for the deployment of particular spatiotemporal models. For instance, weather/traffic forecasting stations might require optimizing for power efficiency and quicker turnaround times, whereas power system simulations can exploit data-center scale GPUs or AI/ML accelerators (AIA), supporting higher throughput for the associated computations.

2.3. Hardware specialization

The generation of highly specialized accelerators is critical, especially for inference. These can be implemented on FPGAs or even ASICs to reduce latency of inference from streaming inputs when attached directly to the data source (*e.g.*, at the edge, near the experimental instruments) or perhaps to the network where data are collected. However, designing a highly specialized accelerator requires knowledge of hardware description languages (HDLs) or the availability of tools that could generate the accelerators starting from the high-level specifications of the algorithms. These tools are commonly known as High-Level Synthesis (HLS) tools. Conventional HLS approaches start from general-purpose programming languages such as C or C++ and often need vendor-specific pragma and significant code restructuring to achieve performance. In this work, we employ the SODA Synthesizer [52], a solution composed of a high-level Optimizer (SODA-OPT [53]), developed with the MLIR compiler framework [54], and a state-of-the-art HLS tool (PandA-Bambu [55]), which allows generating specialized accelerators for FPGAs or ASICs starting from high-level models in PyTorch and TensorFlow with minimal developer interaction. Recent enhancements in the frontend compiler have enabled the generation of ML accelerators for INT8 precision to maximize the throughput. Since spatiotemporal computations are data-driven, non-traditional dataflow based architectures have emerged that can exploit parallelism and optimize data movements for sparse and dense computations associated with the model training and inference. The Reconfigurable Data Architecture (RDA) [56] from Sambanova AI™ is a tiled architecture consisting of reconfigurable units and fast on-chip data delivery networks for execution of dataflow graphs.

Ultimately, the choice of diverse inputs, ML frameworks, source-to-source transformations enhancing operator fusion, and, quantizations, lead to distinct computational workloads, which can be analyzed through low-level profiling, to identify the potential scenarios for further optimizations.

3. Related work

Benchmarking suites. Prior work on workload characterization of DL systems has primarily focused on architectures and models that dominate the industry [57,58]. In addition, several inference benchmarking suites, including MLPerf Inference [59], have been developed in the past for the performance characterization of these real-world deep learning models. However, these previous studies have rarely explored time-series prediction and modeling. Recent studies examine time-series prediction in the context of AI-based Internet-of-Things (IoT) applications on edge architectures [60,61]. However, they did not capture the full spectrum of floating-point precision, hardware configurations, and post-training quantization. Our work contributions bridge this gap for modernizing spatiotemporal modeling workloads' characterization.

Spatiotemporal modeling and characterizing inference workload. Huang et al. lay the direction for benchmarking deep learning systems in the context of time series [8] and highlight that most of the work in this area has been concentrated toward model accuracy/interpretability and not performance characterization. Existing studies on workload characterization of recurrent networks have focused on applications related to natural language processing, speech processing, and machine translation [62], and/or limited the analysis to performance characterization based on training time, accuracy, or energy consumption [63–65]. Gawande et al. [66] carried out performance and power scaling analysis of important CNN training workloads on two architectures: (a) NVIDIA DGX-1 (8 Pascal P100 GPUs interconnected with NVLink) and (b) a cluster with Intel Knights Landing (KNL) CPUs interconnected with Intel Omni-Path. Recent work analyzing spatiotemporal models focuses on characterizing the training workloads on GPGPU systems, which exhibit diverse computational patterns [9]. Specifically for inference, Liu et al. [67] proposed an adaptive DNN inference acceleration framework to accelerate DNN inference by fully utilizing the end–edge–cloud collaborative computing. Choudhary et al. [68] proposed pruning as an optimization problem to improve DNN run-time inference performance by pruning low-impacting parameters (filters) and their corresponding feature maps. Unlike current research on performance characterization of general AI/ML applications, our goals are to quantify the patterns of the underlying computational motifs on contemporary AI/ML platforms, such that more specialized accelerators can be developed for these applications.

High-level synthesis of inference workload. HLS tools are crucial in converting software algorithms into efficient hardware designs. Commercial solutions usually translate general-purpose programming languages (C/C++), heavily annotated with tool-specific pragmas that specify optimizations and hardware information, into hardware designs in Verilog or VHDL. While using these solutions does not need knowledge of HDLs, they still require expertise in hardware design and the ability to write and restructure algorithms in imperative languages. Several solutions propose translating models described in high-level programming frameworks into code that commercial HLS tools can ingest. PyLog [69] provides a custom high-level compilation infrastructure that transforms Python programs into annotated C/C++ code for Xilinx Vivado HLS. The hls4 ml [70] framework translates machine learning models by selecting operators from a library of C/C++ templates optimized for Vivado HLS or Siemens Catapult C. ScaleHLS [71] leverages the MLIR framework to perform high-level transformations (*e.g.*, computational graph and loop optimizations) and regenerate annotated C code for Vivado HLS. Jakšić et al. [72] proposed using FPGAs and OpenCL to accelerate the learning process of Conditional Restricted Boltzmann Machine (CRBM). Cadenelli et al. [73] proposed offloading throughput-oriented genomics workloads in using OpenCL on GPUs and FPGAs. These tools provide a bridge between high-level machine learning frameworks and hardware generation. Still, they have limited flexibility: they only support specific high-level frameworks and backend HLS tools, and they generate code at a different (higher) level of abstraction after applying hardware-related optimizations, potentially losing a considerable amount of semantic information in the process. The SODA Synthesizer [52] combines MLIR and HLS to build an integrated open-source toolchain, optimizing input models at appropriate levels of abstractions without the need to generate intermediate C/C++, and offering a wide choice of FPGA or ASIC targets in the backend.

4. Methodology

In this section, we cover the main components of our study on inference workloads for spatiotemporal modeling. First, we probe the time-series data loading process and identify fundamental computation patterns as motifs of interest. Next, we elaborate on the structure of the studied spatiotemporal models, followed by discussions on precision and post-training quantization via external inference runtimes. Finally, we discuss our HLS compiler-driven strategies for system design.

4.1. Data loading

In a time-series dataset, the sequence of the data points must be preserved to ensure correctness, adding concurrency constraints. Higher concurrency can be achieved by loading the data into batches. For instance, when the GPU is inferring on a specific batch of data, the CPU can prefetch the next batch to avoid the GPU contention for the next iteration. Our studies use both TensorFlow and PyTorch frameworks, which provide optimized data processing interfaces: `tf.data.Dataset` and `torch.utils.data.DataLoader`. These data processing interfaces can use multiple threads or processes to optimize the I/O performance. For the Graph Neural Network (GNN) model (discussed below), we use Deep Graph Library's (DGL) [74] GraphLoader, *i.e.*, `dgl.dataloading.GraphDataLoader` to load the graph in batches.

4.2. Computational motifs

To study the impact of the diverse computational patterns of the inference workloads associated with various spatiotemporal models, we organize them into high-level motifs. These motifs can be considered as the building blocks of the deep learning models and are multifarious in practice. We aim to generalize the motifs according to compute patterns based on the underlying tasks (*e.g.*, all variations of a particular operation are classified under the same motif).

We identify three common motifs— Matrix-Matrix Multiplication, Elementwise operations, and Data transformations, which are respectively denoted as *mmm*, *elem*, and *data* in the rest of the paper. Here, `mmm` captures operations related to tensor contractions or matrix-matrix multiplication (*e.g.*, different versions of *gemm*). Next, `elem` denotes elementwise operations, including the arithmetic and gradient operations applied to tensors. Finally, `data` depicts data transformations: layout conversion, bias, reduction, activation, and residual connections.

4.3. Spatiotemporal models

We now briefly discuss the spatiotemporal models implemented for this study. For each model, the corresponding computation graph captures the model's layers, the input/output dimensions, and the associated motifs used to implement a specific layer. The solid blue arrows in the graph represent transformations: permutations, reshaping, and transpose of the tensors.

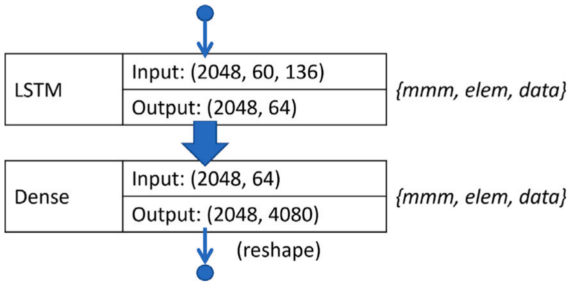

Fig. 3. LSTM layers.

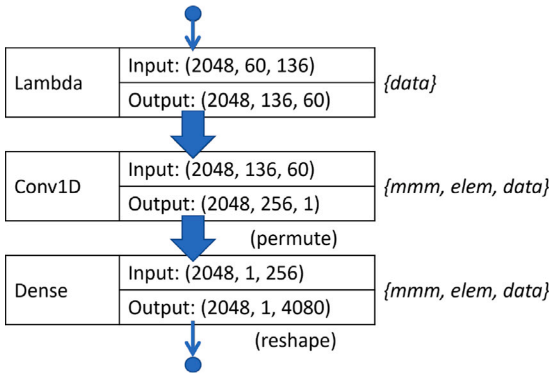

Fig. 4. CNN layers.

Long Short Term Memory (LSTM). LSTM [47] is an extension of recurrent neural network (RNN) [46], which was originally designed for sequential data. In RNN, the system's current state depends on the previous state. RNNs have the concept of "memory" that helps them retain the past to generate the next output of the sequence. However, RNNs are only effective when the history is small and typically perform poorly in practice when learning long-range dependencies in the data. Long Short-Term Memory (LSTM) networks were developed to address these limitations by improving the gradient flow within the network using multiple gates. Fig. 3 depicts the computation graph of the LSTM model, which includes a Lambda layer to select an LSTM layer with 64 nodes followed by a dense layer with 4080 nodes.

Convolutional Neural Network (CNN). The convolution layer is primarily designed to process visual data and fundamentally differs from an LSTM layer. Several studies in the past have used CNN for time-series prediction [75,76]. We developed a unidirectional Convolutional Neural Network (CNN) for the time-series prediction. Fig. 4 depicts the computation graph of the trained CNN model. It includes a Conv1D layer with 256 nodes followed by a dense layer with 4080 nodes.

Graph Neural Network (GNN). Graph Neural Network (GNN) is a powerful tool for spatiotemporal modeling because it can combine node feature information with the graph structure by recursively passing neural messages along the edges of the input graph. GNNs can be combined with recurrent networks, exploiting both *spatial* and *temporal* dimensions. Fig. 5 shows the computational graph of the STGCN model used in this study; three Conv2D layers are used to encode the temporal dimension and a Graph Convolution Network (GCN) is used to encode the relationship between the entities. We implemented STGCN using Amazon™ Deep Graph Library (DGL) [74].

4.4. Floating-point precision

Our TensorFlow/PyTorch variants were trained using three floating-point precision policies — Automated Mixed Precision (AMP), FP32,

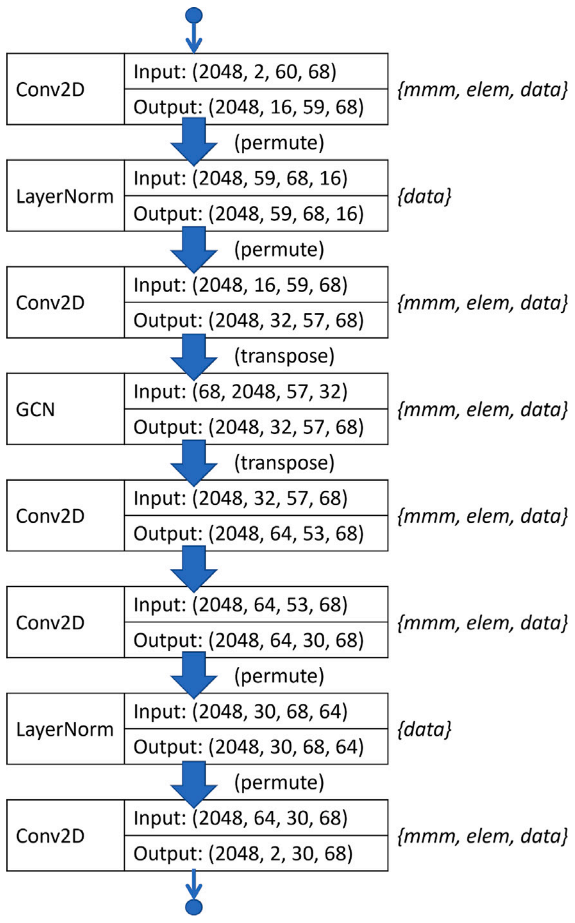
Conv2D
Input: (2048, 2, 60, 68)
Output: (2048, 16, 59, 68)
{mmm, elem, data}
(permute)
LayerNorm
Input: (2048, 59, 68, 16)
Output: (2048, 59, 68, 16)
{data}
(permute)
Conv2D
Input: (2048, 16, 59, 68)
Output: (2048, 32, 57, 68)
{mmm, elem, data}
(transpose)
GCN
Input: (68, 2048, 57, 32)
Output: (2048, 32, 57, 68)
{mmm, elem, data}
(transpose)
Conv2D
Input: (2048, 32, 57, 68)
Output: (2048, 64, 53, 68)
{mmm, elem, data}
Conv2D
Input: (2048, 64, 53, 68)
Output: (2048, 64, 30, 68)
{mmm, elem, data}
(permute)
LayerNorm
Input: (2048, 30, 68, 64)
Output: (2048, 30, 68, 64)
{data}
(permute)
Conv2D
Input: (2048, 64, 30, 68)
Output: (2048, 2, 30, 68)
{mmm, elem, data}

Fig. 5. STGCN layers.

and FP64. In both TensorFlow and PyTorch, enabling the AMP policy causes the model network layers to use FP16 for weights and FP32 for the data. Even without enabling mixed-precision, the runtime might use TensorFloat −32 or TF32 mode for matrix operations on NVIDIA™ A100 GPUs. The TF32 type uses the same numerical precision as FP16 (10 bits) but has the range of FP32 (8 bits).

4.5. Inference runtimes for post-training graph optimization

Computational graphs of Deep Neural Network (DNN) architectures comprise of numerous compute-intensive tasks, and the degree of the complexity and size of the DNNs ultimately affect the performance of inference. The computational graphs can be optimized without changing the model itself (post-training quantization) via runtimes that can automatically intercept and convert the tasks in the model to high-performance implementations. We discuss two such post-training inference runtimes for the spatiotemporal models considered.

4.5.1. ONNX

Open Neural Network eXchange (ONNX) [77] is a format specification that defines a standard set of operators — the building blocks of machine learning and deep learning models, and a standard file format to enable practitioners to use the models with a variety of frameworks, tools, runtimes, and compilers. We exported PyTorch/Tensorflow FP32 models to the ONNX format using `opset` version 13 with constant folding enabled and later used it for inference using the `onnxruntime`. We were unable to evaluate STGCN with ONNX because the DGL

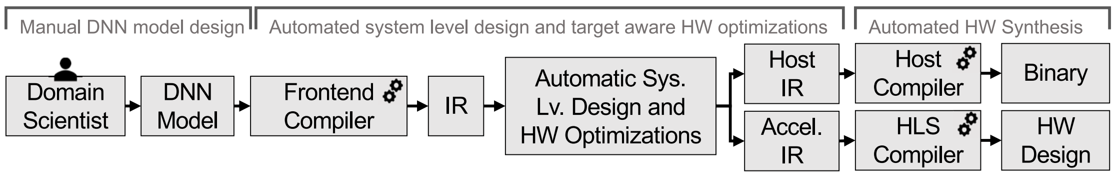

Fig. 6. Taxonomy of high-level synthesis frameworks.

GraphLoader is currently unsupported.[1,2,3] PyTorch provides an in-built API: `torch.onnx.export` for model transformation. Likewise, TensorFlow offers `tf2onnx` routine to convert TensorFlow models to ONNX format. The ONNX runtime provides several graph optimizations to improve the performance, ranging from node eliminations to more complex node fusions and layout optimizations.

4.5.2. AI template

AI Template (AIT) [78] is a Python framework for transforming models into high-performance templatized C++ GPU code for accelerating inference. AIT consists of two layers— a front-end layer for various graph transformations to optimize the computational graph and a back-end layer to generate C++ templates for the GPU target. In addition, AIT maintains a minimal dependency on external libraries. For example, the generated runtime library for inference is self-contained and only requires GPU vendor libraries (*i.e.*, NVIDIA CUDA runtime). Currently, AIT does not support LSTM and GCN models, and the TensorFlow framework. Therefore, we could only convert the PyTorch-based CNN model using AIT.

4.6. Post-training quantization

Most deep learning applications use FP32 for inference [8]. However, lower precision types, especially INT8, can offer a significant efficiency (performance per energy unit) advantage. Post-training or model quantization is a conversion technique that can reduce the model size (*i.e.*, decreasing the memory consumption) while improving the runtime latencies, with minor degradation in the model accuracy. PyTorch, TensorFlow, and ONNX support quantization in their model formats and frameworks, which can be leveraged by GPU devices using the ONNX plugin in the NVIDIA TensorRT library. NVIDIA™ TensorRT™ [79] is an asynchronous interface for enhancing deep learning inference via customizable optimization profiles.

PyTorch, TensorFlow, and ONNX support TensorRT through Torch-TensorRT, TF-TRT, and ONNX-TensorRT, respectively. Post-training INT8 quantization could either be *dynamic* or *static*. Dynamic quantization calculates the quantization parameters (scale and zero point) for activations during the model execution, trading off performance with accuracy compared to the static option. Static quantization, on the other hand, first runs the model using a portion of the training dataset called the calibration data. During these runs, the quantization parameters are computed for each activation and written as constants to the quantized model, which are finally used for all the inputs. We used static post-training quantization due to a better performance prospect.

4.7. High-level synthesis

Fig. 6 shows an overview of the SODA Synthesizer. The SODA toolchain has three main components: a *compiler frontend* [53] to interface with high-level programming frameworks that automatically search, outline, tile, and pre-optimize relevant code regions to generate high-quality accelerators through HLS; A *compiler backend* [55], to perform HLS, generate Verilog code and interface with external tools that compile the final design to be deployed into a FPGA; and a *Design Space Exploration* (DSE) engine that enables an optimized selection of compiler parameters, resulting in the generation of faster accelerators. As shown in the picture, SODA can also generate the glue code needed to orchestrate the execution of the outlined accelerators.

This flow allows domain scientists to easily move from model to hardware implementation of accelerators and explore hardware design parameters. Leveraging the trained TensorFlow models (both FP32 and Post-Training Quantized), we outlined the key layers of the computational motifs and applied the high-level compiler optimizations provided by SODA-OPT. These include the decision on the size of each outlined operator (while tiling the operations to different magnitudes) and HLS-specific optimizations of the intermediate representation (IR), such as loop unrolling, common sub-expression and dead-code elimination, and memory optimization (early alias analysis, scalar replacement of aggregates, temporary buffer allocation, alloca buffer promotion), and the number of memory channels for the accelerator. The optimizations happen automatically, and the generated accelerator designs are then synthesized with the backend logic synthesis tools. In this paper, we target a Xilinx Virtex 7 FPGA (XC7VX690). Hence, our HLS tool (Bambu) generates Verilog optimized for FPGAs and scripts for Xilinx Vivado.

5. Experimental evaluation

We discuss the evaluation of the inference workloads on diverse platforms in this section, explained in Section 4. First, we discuss the platform details in Section 5.1. Next, we mention the input datasets used in the experiments and batching information in Section 5.2. NVIDIA A100 GPU is our baseline experimental platform, and we consider various combinations of mixed-precision models and external runtimes as discussed in Section 5.3, listed in Table 2. Next, we discuss the experiments associated with external inference runtimes to exploit mixed-precision models in Section 5.4. Post-training quantization is discussed in Section 5.5, to leverage operations with integer precision. Apart from NVIDIA A100 GPU, we use an AI Accelerator (AIA), namely the SambaNova Reconfigurable Dataflow architecture, discussed in Section 5.6. Finally, we discuss provisioning custom accelerators via the SODA toolchain in Section 5.7.

5.1. Platforms, software and definitions

Platform details. We use three platforms in our evaluation, resembling the inference hardware spectrum in Fig. 1. The code used in this study is available at https://github.com/pnnl/ProxyTSPRD. Relevant details regarding software/hardware are listed below.

- **GPU**: Single Nvidia™ DGX-2 "Ampere" A100 GPU (108 SMs) with 40 GB HBM2 memory/GPU and two-way 128-core AMD EPYC™ 7742 CPUs at 2.25 GHz, 256MB L3 cache, 8 memory channels, and 1TB DDR4 memory. We use CUDA 11.6, Tensorflow 2.12 [80] (TF), PyTorch 2.0.1+cu117 [81] (PT) and Distributed Graph Library (DGL v1.1.0+cu116) [74].

[1] https://github.com/dmlc/dgl/issues/3418
[2] https://github.com/dmlc/dgl/issues/4442
[3] https://github.com/dmlc/dgl/issues/5379

Table 2
Inference variants considered (for NVIDIA A100).

Categories	Variants	LSTM		CNN		STGCN	
		PT	TF	PT	TF	PT	TF
Mixed Precision	AMP	✓	✓	✓	✓	✓	✗
	FP32	✓	✓	✓	✓	✓	✓
	FP64	✓	✓	✓	✓	✗	✗
Inference runtime	ONNX	✓	✓	✓	✓	✗	✗
	AI Template	✗	✗	✓	✗	✗	✗
Post Training Quantization	ONNX-TRT-INT8	✓	✓	✓	✓	✗	✗
	ONNX-TRT-FP16	✓	✓	✓	✓	✗	✗
	TF-TRT-INT8	✗	✓	✗	✓	✗	✗
	TF-TRT-FP16	✗	✓	✗	✓	✗	✗

- **AIA**: SambaNova SN10™ Reconfigurable Dataflow Unit (RDU) system [56], 8× SN10 RDUs with 6 channels/RDU and 4 tiles/RDU, 256 GB DDR4 memory/channel (total, 12TB memory). We use a single RDU, comprising of 4 tiles - a tile consists of discrete compute and memory units interconnected into a mesh topology via switches. We used SambaFlow 1.16.2 which at the time of study supported Python 3.7.6, and PyTorch 1.10.2+cpu.
- **HLS**: SODA Synthesizer V15.07, DNN model implemented and synthesized with TensorFlow 2.9, MLIR 15 (LLVM 15.0.7), Bambu 0.98, targeting a Xilinx Virtex 7 (XC7VX690) FPGA. Simulation performed with Verilator, area obtained with Xilinx Vivado 2020.2 post place-and-route.

We compute the *inference time* as the total time over batches of the data, excluding the time to compute the loss function (we use this metric as execution time performance). We evaluate model accuracy using Mean Square Error (MSE), where lower values indicate better performance.

GPU workloads. For the GPU variants, we use Nvidia™ Nsight™ Profiler to examine the underlying workloads and quantify the percentage of the overall time spent by specific CUDA Runtime API and kernel motifs (categorized in Section 4.2). We follow this format to identify a specific GPU variant:

`{TF|PT},{Runtime-Precision}`.

For example, `TF,AMP` refers to the Tensorflow model with AMP (`Runtime-Precision` to be substituted from Table 2). We use the following notations to depict the share of the overall time taken by a group of arithmetic kernels and the CUDA Runtime API functions for inference on the GPU platforms.

CUDA Runtime API (modules) grouping:

xfer Host↔device data transfers.
mem Memory (de)allocation and initialization.
event Event management (*e.g.*, synchronizations).
stream Stream management (enabling task concurrency).
mod Module management (*e.g.*, loading external kernels).
exec Execution control (*e.g.*, launching kernels).
dev Device synchronization.

Above classification is based on the NVIDIA™ CUDA™ Runtime API modules[4]; we aggregate the %-time of the individual routines belonging to a specific group (a *group* corresponds to particular CUDA™ Runtime API module). Apart from the `xfer` and `mem` groups (both are part of the CUDA™ Runtime Memory Management module), every other group is associated with a distinct CUDA™ Runtime API module.

GPU kernels:

mmm Half/Single/Double precision matrix-matrix multiplication/tensor contractions on GPU SMs and tensor cores.
elem Elementwise operations on tensors, direct copying data to kernels, applying bias, etc.

[4] https://docs.nvidia.com/cuda/cuda-runtime-api/modules.html#modules

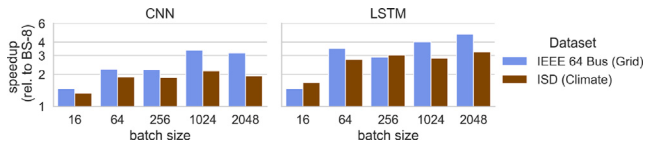

Fig. 7. Speedup by increasing the batch size relative to batch size of 8.

data Data transformation functions, *e.g.*, splitting/slicing tensors, FFT, etc.

We combine similar operations into representative motifs, excluding %-contributions less than 1% of the overall time.

5.2. Data and batching

The raw time-series from each of these scenarios is transformed into sliding windows (Fig. 2), which are then further divided into batches. Fig. 7 shows the impact of batch size on inference runtime. While the larger batch sizes can converge faster, they are hard to fit into the memory. The batch size is 2048 because that was the largest batch size we could fit into the memory of NVIDIA™ A100 GPU for the STGCN model. We noticed a speedup of 3× for CNN and 4× for LSTM with a batch size of 2048 relative to the batch size of 8. On the SambaNova SN10 RDU, we used a batch size of 64 (our version of SambaFlow resulted in compilation errors for larger batch sizes). For HLS system design, focusing on real-time processing of the workloads, we consider a batch size of 1 (also amenable to FPGA devices with limited memory).

IEEE 64-bus system (power systems). We used simulated data from an IEEE68-bus system [82], where 68 denotes the number of nodes, and for each node, the data consists of two columns — frequency and voltage. The data is generated over multiple scenarios where each *scenario* corresponds to a unique configuration (*e.g.*, load changes) of the power network and data. We use 140 scenarios to analyze the inference workload in this study.

Integrated surface dataset (climate). Historical temperature and humidity data sampled at an hourly basis for the last 20 years from 162 stations across the United States were extracted from the Integrated Surface Database (ISD) from National Centers for Environmental Information (NCEI) of National Oceanic and Atmosphere Administration (NOAA) [83]. The data was transformed into monthly files prior to processing.

5.3. Baseline performance on NVIDIA A100 GPU

Fig. 8 compares the performance of the three models implemented in PyTorch and Tensorflow for both power systems and climate datasets. Fig. 9 depicts the profiles (demonstrating the overall percentage of time taken by a particular workload) for the power grid data (increasing magnitude of data does not affect the time distribution). DGL has limited support for Tensorflow and only supports distributed graphs

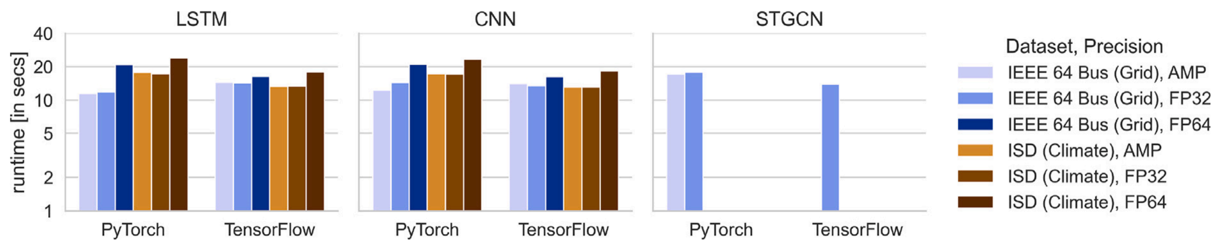

Fig. 8. Baseline inference performance on NVIDIA A100 GPU using Grid and Climate data.

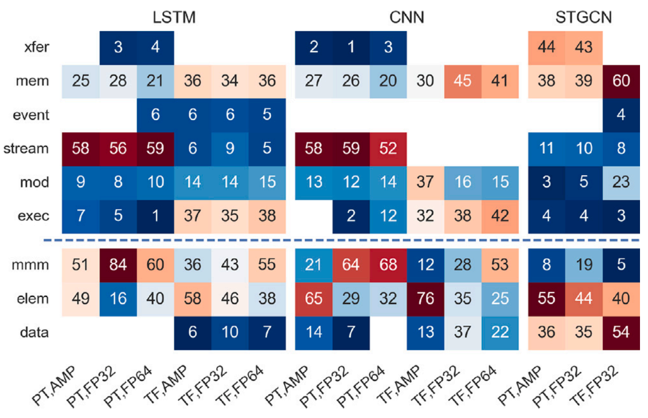

Fig. 9. Baseline profiles of LSTM, CNN and STGCN on A100 using Grid data (X-axis: {ML framework, Precision}). Top of the broken line: CUDA API, Bottom: Computational motifs.

when using `dgl.dataloader` for PyTorch. Therefore, only FP32 results were collected for the STGCN model implemented in Tensorflow. STGCN with the designated batch size of 2048 with FP64 led to out-of-memory errors for multiple Tensorflow/PyTorch versions. The key takeaways from this section are the following:

- FP64 Tensorflow is about 25% faster than PyTorch.
- AMP results are close to FP32 (FP32 in A100 may internally use TF32[5]) but depicts 1.6× benefit over FP64.
- For PyTorch variants, ∼50% of the time in CUDA is spent in *stream* management operations (associated with asynchronous data transfers), divided between querying the computation graph and synchronizing operations.
- For Tensorflow variants, about 25%–70% of CUDA time is spent in kernel launches.
- For PyTorch models, > 50% of the overall time is spent in tensor contractions, whereas for Tensorflow, elementwise and data transformation operations take >70% of the time.
- The motifs categorized as *data* correspond to data manipulation/transformation between model layers. Since STGCN is the largest among the three models, the time spent in *data* is relatively higher.

In terms of accuracy, the average MSE for FP32 implementations of LSTM, CNN, and STGCN are $2.2e^{-2}$, $1.1e^{-4}$, and $1.8e^{-3}$ respectively, with a corresponding standard deviation of $2e^{-3}$, $5.1e^{-5}$, and $1.2e^{-2}$ across different runs over the dataset collected from IEEE 64 bus system.

5.4. Unified and interoperable inference runtimes

This section analyzes the performance and constituent workloads of different variants using ONNX and AI Template (AIT) runtimes for the power grid dataset.

[5] https://discuss.pytorch.org/t/torch-cuda-amp-cannot-speed-up-on-a100/120525

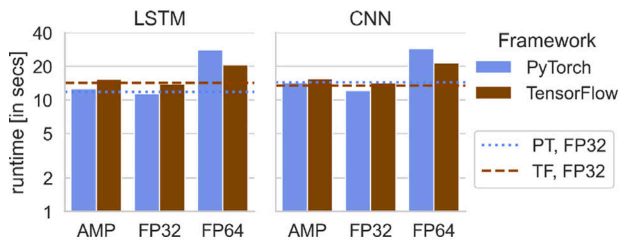

Fig. 10. Inference performance on NVIDIA A100 using ONNX runtime (X-axis: model precision).

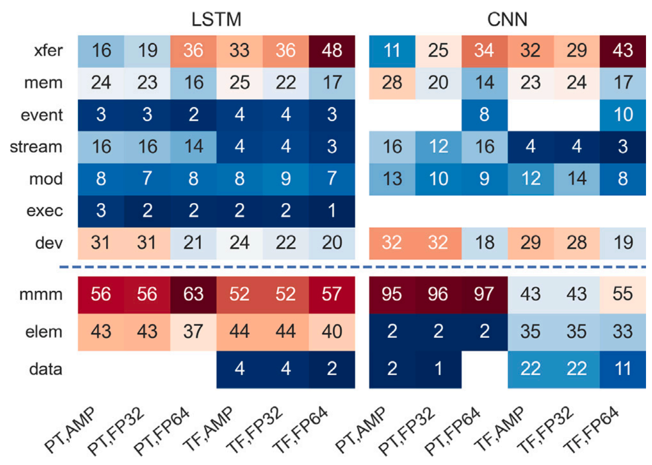

Fig. 11. Profiles of LSTM and CNN on A100 using ONNX on Grid data (X-axis: {ML framework, Precision}).

Open Neural Network eXchange (ONNX). Figs. 10 and 11 demonstrate the performance and profile of GPU ONNX variants for LSTM and CNN models.

The key takeaways from this section are:

- Performance of ONNX models is comparable to GPU baseline results, except FP64.
- Data transfer and synchronizations are dominant compared with the GPU baseline.
- Like GPU baseline, stream operations are significant for PyTorch-converted ONNX models.
- Tensor contractions and elementwise operations are the dominant workloads, except for CNN PyTorch (nearly 100% of time in tensor contractions).
- 80% of the overall CUDA runtime API is in data transfers, memory management and synchronizations.

In terms of accuracy, the average MSE for ONNX models varied by $2.2e^{-4}$ for the LSTM and $1.2e^{-5}$ for the CNN.

AI template. AI Template supports multiple batch sizes: 1, 8, 16, 64, and 256. For a batch size of 256, the inference time for PyTorch-based CNN through AIT is 47 s. Like ONNX, AIT spends significant time in tensor contractions, as shown in Fig. 12. About 70% of the time in

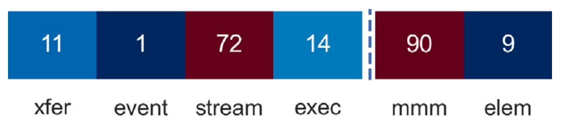

Fig. 12. AIT profile for CNN PyTorch implementation using grid data.

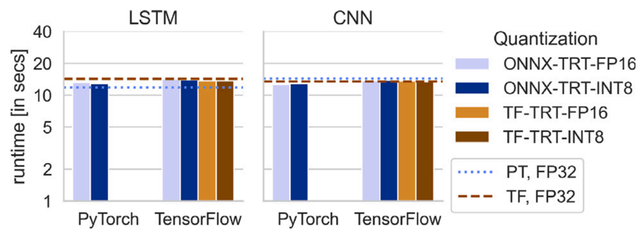

Fig. 13. Post training quantization inference on A100 using TensorRT on grid data.

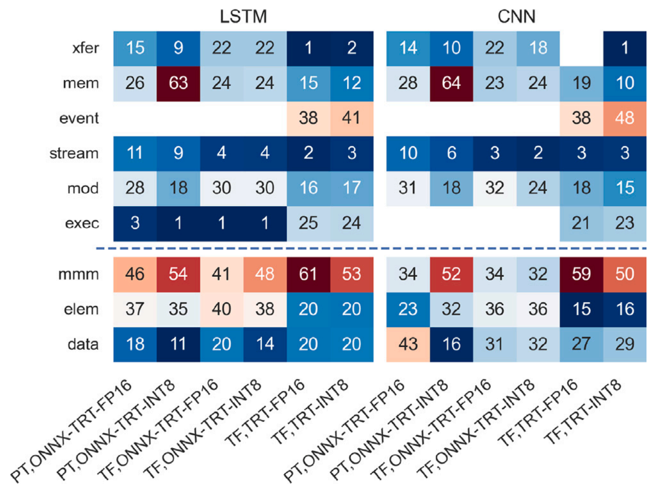

Fig. 14. Profiles for post-training quantization using TensorRT on A100 using grid data (X-axis: {ML framework, Precision}).

CUDA API is spent performing stream operations. Regarding accuracy, the average MSE for AIT is $9e^{-3}$ for the CNN model.

5.5. Post-training quantization

PyTorch, TensorFlow, and ONNX support quantization on NVIDIA GPUs through TensorRT. We were unable to evaluate Torch-TensorRT due to a driver compatibility issue on our NVIDIA DGX-2 based evaluation platform.[6] Therefore, for post-training quantization, we currently consider Tensorflow-TensorRT (TF-TRT) and ONNX-TensorRT (ONNX-TRT) variants. A portion of the training data was used to calibrate the model for INT8 quantization for both TF-TRT and ONNX-TRT. Figs. 13 and 14 demonstrate the execution-time performance and workload characterization. The takeaways from this section are:

- CUDA Event management operations are high for TF-TRT, as consistent with the GPU and ONNX variants, when compared with ONNX-TRT. We speculate this is due to the overhead of registering events on streams, corroborated by equally high kernel launch overheads.
- Loading quantization related modules during the inference takes about 30% of the time spent in CUDA API.

[6] https://forums.developer.nvidia.com/t/tensorrt-for-cuda-11-4-requested/185903

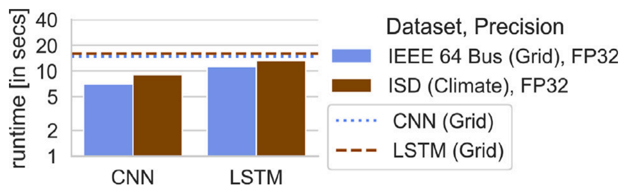

Fig. 15. Baseline performance on Sambanova SN10 RDU (4 tiles). Broken lines indicate A100 performance with batch size: 64.

- Relatively balanced spread of workloads for TF-ONNX, between tensor contractions and elementwise operations.

In terms of accuracy, we noticed a slightly increased MSE of $4.7e^{-2}$ and $1.2e^{-4}$ for the quantized LSTM and CNN models, respectively.

5.6. Performance on sambanova SN10 RDU

Fig. 15 demonstrates the performance of LSTM and CNN models compared to GPU baseline for IEEE 64 Bus (Grid) dataset, implemented in PyTorch, on a single Sambanova RDU (4 tiles). For SambaNova SN10, the SambaFlow software stack generates a compiled model from the original PyTorch model, optimizing the computational graph. Models execute with (default) BF16 precision on the SambaNova RDU system. BF16 (or bfloat16) maintains the same numerical range as FP32 (8 bits) but a significantly lower precision (7 bits, vs. 23 bits for FP32). Since DGL and PyTorch Graph have limited Sambanova RDU platform support, STGCN was not considered. Performance comparison indicates about 1.7× speedup compared to the corresponding AMP GPU baseline. Sambanova benchmarks used a batch size of 64 (higher batch sizes led to out-of-memory errors), therefore, GPU runs were also carried out on a batch size of 64. The results were compared to the AMP version because Sambanova uses BF16 for weights and biases to optimize the performance. The average MSE was observed to be 0.075 and 0.072 for CNN and LSTM, respectively, for the data collected for the IEEE 64-bus system. The relatively higher MSE is speculated due to the precision loss owing to default BF16 usage.

5.7. Custom accelerators via SODA HLS toolchain

HLS tools are crucial in converting software algorithms into efficient hardware designs. Custom hardware accelerators allow for optimized execution of spatiotemporal algorithms. They can be tailored to harness available parallelism and leverage custom datatypes while enhancing performance and energy efficiency.

We discuss the experimental approach used while synthesizing different accelerators with the SODA toolchain. First, we describe the CNN model of Fig. 4 in TensorFlow and subsequently translate it to the MLIR framework. Utilizing SODA-OPT, the compiler frontend of the SODA toolchain, we perform a series of experiments at varying granularities. These include synthesizing a single accelerator for the entire model, generating distinct accelerators for each layer, outlining reusable tiles of different sizes for the most computationally expensive layer, and applying SODA-OPT's optimization pipeline. Furthermore, we investigate the benefits of increasing the number of channels to alleviate memory bottlenecks, which become more pronounced after exposing parallelism through structural optimizations. In prior work [52,53], SODA has been used to synthesize accelerators for layers of image classification models. This is the first time SODA is employed to synthesize accelerators for spatiotemporal models. This is also the first time presenting results for accelerators synthesized with INT8 precision since support for non-FP32 models has just been added. Our experiments aim to showcase the versatility and effectiveness of SODA in custom accelerator generation for deep learning models.

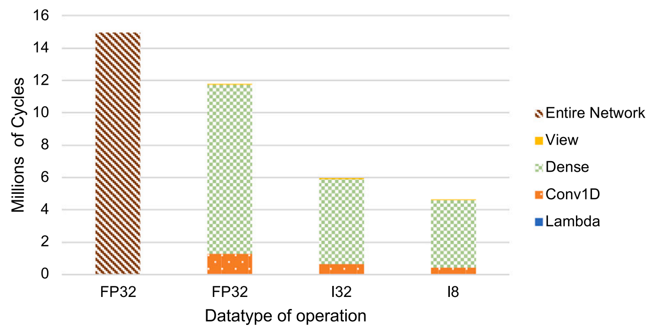

Fig. 16. Baseline execution delay of CNN Layers (lower is better).

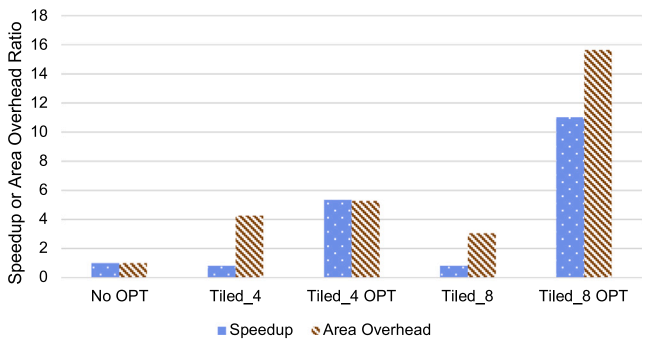

Fig. 17. Quality of results for baseline and optimized Dense accelerators.

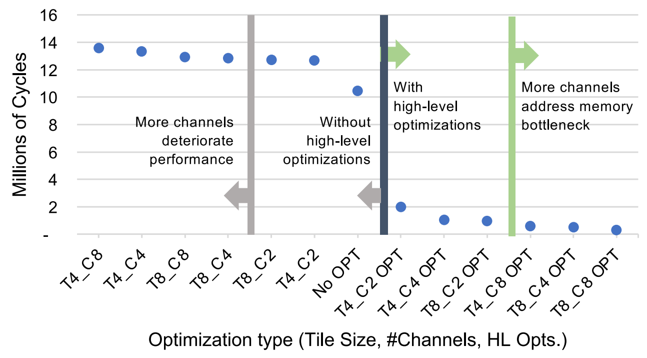

Fig. 18. Ablation study of SODA optimizations when varying tile size and number of memory channels with and without SODA-OPT optimizations.

5.7.1. Results

Fig. 16 presents execution latency (in million of clock cycles) of CNN layers at different precision: FP32, and post-training quantization in 32-bit (I32) and 8-bit (I8) integers. We also report the total execution latency for the end-to-end synthesis of the entire network in FP32. The results reported here consider the latency of memory access to the FPGA's internal BRAMs, which is a reasonable assumption since the synthesized models fit the FPGA's internal memory. Synthesis of individual accelerators in FP32 provides a speed-up of 1.2× over synthesizing the entire network. Moving from FP32 to I32 halves the execution latency, providing a speed-up of 2.5× over the entire network. Further, moving from I32 to I8 only increases performance (reduces execution latency) by 1.5× for the 4× data reduction. The reason is that while the synthesizer can reduce the number of states for the accelerators (due to simpler functional units that can be chained in a single operation), the overall execution latency remains limited by the memory operations, which still execute across two memory channels. This provides an overall speed-up of 3.2× with respect to the entire network. The Dense layer is the operator that contributes the most to the overall execution time, corresponding to 89%, 88%, and 90% of the total execution time, respectively, for FP32, I32, and I8 precision.

Since the Dense Layer is the most significant contributor to the execution delay, we used SODA-OPT to outline and optimize accelerators to speed up the computation of tiles of the Matrix Multiplication, the underlying algorithm of the Dense layer. We synthesize designs at FP32 precision with different parameters for tile sizes (a single element unoptimized accelerator, $4 \times 4 \times 4, 8 \times 8 \times 8$) and collect performance and area results post place-and-route. We plot the results in Fig. 17 discussing the Quality of Results (QoR), *i.e.*, the trade-off between performance and area. For each configuration, we provide speed-up and area overhead (increase in resource utilization) for just the tiled designs and the tiled designs with the full set of SODA-OPT optimizations. We can see an expected trend, *i.e.*, as optimizations are applied, the area increases, but so does the execution speed. It is interesting to note that for small tiles, the speed-up to area increase in area ratio is bigger than one (*i.e.*, the designs are more area efficient). In contrast, for larger tiles, the area overhead is slightly larger than the increase in performance. The reason is an increase in routing complexity on the FPGA (as we report results post place-and-route).

Our final experiment aims to discern the primary factors contributing to the achieved speed-up. Fig. 18 provides an ablation study of the optimizations applied through SODA to the FP32 accelerators for the Dense layer. We consider varying tile sizes (TN, where N is the size of a dimension of the tri-dimensional tensor) and varying numbers of memory channels (CN, where N is the number of memory channels) with and without SODA-OPT high-level IR optimizations. Surprisingly, we see that some structural modifications, such as increasing the number of memory channels, impact the execution latency in different ways, depending on the combined use of other high-level optimizations. Without high-level optimizations, we even see a reduction in performance when more memory channels are used, which should have increased memory parallelism (and thus improve execution latency). To effectively use more memory channels, additional optimizations are necessary.

When enabling the SODA-OPT high-level optimization pipeline [53] we obtain a substantial difference in performance, with speed ups of 5× over the unoptimized baseline (No Opt.). With SODA-OPT optimizations, increasing structural parameters (size of the tiles and number of memory channels), significantly improves the execution latency, achieving up to 37× speed up for the largest design (T8_C8 OPT) over our baseline.

6. Discussion

In this section, we highlight and summarize the key takeaways from the comprehensive analysis of runtimes and workload profiles.

Disparities in ML/DL platforms

Workload characterization reveals disparities in the implementation and execution of various ML frameworks and inference runtime engines. For example, PyTorch relies heavily on CUDA streams, while TensorFlow uses CUDA *mod* and *exec*. This distinction also applies to ONNX-based implementations (see Fig. 11). Inference runtime engines, like ONNX and AI Template, aim to optimize the graph further and transfer operations to well-optimized *mmm* motifs. The *mmm* motif reached as high as 95% for ONNX-based and 90% for AI Template-based implementations of PyTorch CNN models. This demonstrates the ability of inference runtime engines to reduce reliance on data and shift the workload to already optimized kernels. However, for LSTM, a recurrent network, runtime inference engines could not achieve such high *mmm* operation runtimes. Overall, while implementation disparities exist, these software-based optimizations can enhance performance for inference in spatiotemporal modeling.

Spatiotemporal models are data intensive

Performance analysis on the NVIDIA A100 GPU reveals that regardless of the model, deep learning (DL) framework, or precision, spatiotemporal models primarily spend about 80% of CUDA time on data movement and launch-related operations. This trend is similarly reflected in computational motifs, albeit with some exceptions. While the time spent on *elem* and *data* motifs exceeds 50%, except for some cases, it is significantly higher than that spent on matrix-matrix multiplication (*mmm*) operations. Specifically, for spatiotemporal graph convolutional network (STGCN), which involves extensive data transfer and manipulation, *mmm* operations are nearly negligible. Given that the data expands in the order of $O(T \times W)$, where T is the length of the time series and W depicts the size of the sliding window, spatiotemporal models are inherently data-intensive. Consequently, kernels optimizing *mmm* operations have minimal impact on the performance of spatiotemporal models. Custom accelerators tailored for spatiotemporal models should prioritize parallelizing data operations.

Inference runtime engines

Unified and interoperable inference runtime engines, such as ONNX and AI Template, employ various graph optimizations to enhance performance. These optimizations encompass graph-level transformations, including minor graph simplifications, node eliminations, and more complex node fusions and layout optimizations. Though, similar to GPU runs, ONNX and AI Template also spend roughly 80% of CUDA time on data movement (*xfer*, *mem*) and launch-related operations (*dev*), the optimized graph is allocating more time to matrix-matrix multiplication (*mmm*) motifs by optimizing *data* motifs. This adjustment enables the spatiotemporal models to benefit from kernel-level optimizations.

Speedup with post-training quantization on custom accelerator

Post-training quantization offers significant optimization for DL inference by reducing data size, particularly beneficial for data-intensive spatiotemporal modeling. We explored post-training quantization using ONNX-TensorRT and TensorFlow-TensorRT for INT8 and FP16 precision, as well as HLS-Synthesis for INT8 and INT32 precision. Quantization often involves sacrificing precision to achieve speed gains. While we did not observe a speedup on GPU (due to overhead) with quantization, we did notice a speedup of $2.5\times$ and $3.75\times$ with INT32 and INT8 quantization, respectively. On the GPU (as depicted in Fig. 14), the performance gains are offset by the increased data operations required to run the quantized model. In an attempt to achieve performance gain through quantization, we noticed an increase of 9% in the reported MSE for the CNN architecture. Additionally, we observed high memory consumption for PyTorch-based INT8 quantization, which was not seen in TensorFlow-based INT8 quantization. Despite these challenges, post-training quantization remains promising due to its data reduction benefits. Currently, the current data processing requirements outweigh the gains from quantization on GPU, but this could potentially be mitigated with custom accelerators.

Custom hardware showing performance gain over GPUs

We observed speedups of $1.7\times$ with Sambanova and $5\times$ with SODA-OPT. Sambanova's Reconfigurable Dataflow Unit (RDU) architecture, tailored for deep learning workloads, appears to offer potentially superior performance compared to general-purpose GPUs. While we were unable to verify this through profiling due to limited options in Sambanova, our HLS synthesis revealed that certain structural modifications, such as increasing the number of memory channels, affect execution latency differently depending on the combined use of other high-level optimizations. Interestingly, without these optimizations, we observed a decrease in performance when more memory channels were used, despite the expectation of increased memory parallelism. This indicates that additional optimizations are required to effectively utilize additional memory channels.

7. Conclusion

Organizing multivariate timeseries data into overlapping windows in spatiotemporal modeling introduces redundancies in data processing and substantially amplifies data movements and associated transformations. De facto optimizations targeted for image and text-based Deep Learning applications may not translate to efficient solutions for spatiotemporal inference workloads, which is a common application underpinning several critical infrastructures. Inference workloads for spatiotemporal modeling demonstrate significant optimization potential — as shown by our experiments and analysis on GPU, AIA, and reconfigurable platforms. Our multi-level analysis of the computational patterns underlying the inference workloads reveal essential gaps that must be bridged for achieving sustainable performance improvements of spatiotemporal models on next-generation software/hardware platforms. Analyzing the possible combinations of the spatiotemporal inference workloads can provide broad empirical justifications and trade-offs guiding real-world application deployments.

For instance, automatic mixed-precision and post-training quantization (to exploit integer precision) can offer spatiotemporal models some performance advantage over full precision, however, the benefits diminish for 32-bit model precision on GPU. Likewise, kernels optimizing matrix multiplication operations can have minimal impact on the overall performance of spatiotemporal models. Custom accelerators tailored for spatiotemporal models can prioritize reducing the memory access bottlenecks (*e.g.*, increased memory channels). Kernel-level optimizations in inference runtime engines can enhance GPU utilization and mitigate framework disparities, but their impact varies depending on architectural specifics.

Consequently, for successful real-world deployment, it is crucial to consider a broader range of performance metrics beyond just prediction accuracy when selecting models. We observe Sambanova's Reconfigurable Dataflow Unit (RDU) architecture improves the inference time by 1.7× (due to BF16 precision and efficient data parallelism) and synthesizing custom accelerators via the SODA compiler toolchain (aiming for higher resource utilization) demonstrated 5× improvement over GPUs. Future research directions can explore further kernel-level optimizations at different levels (high-level runtime API, device portability layer, compiler intermediate representation, etc.) to inform the design of emerging machine learning platforms optimized for efficient processing of the spatiotemporal inference workloads.

Funding

This work was supported by the Data-Model Convergence (DMC), a multi-disciplinary Pacific Northwest National Laboratory (PNNL) initiative that applies co-design to develop heterogeneous computing capabilities to tackle challenging scientific problems.

CRediT authorship contribution statement

Milan Jain: Writing – review & editing, Writing – original draft, Visualization, Validation, Software, Methodology, Investigation, Formal analysis, Data curation, Conceptualization. **Nicolas Bohm Agostini:** Writing – review & editing, Writing – original draft, Visualization, Validation, Software, Methodology, Investigation, Formal analysis, Conceptualization. **Sayan Ghosh:** Writing – review & editing, Writing – original draft, Validation, Supervision, Resources, Project administration, Methodology, Investigation, Funding acquisition, Conceptualization. **Antonino Tumeo:** Writing – review & editing, Validation, Supervision, Conceptualization.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Sayan Ghosh reports financial support was provided by US Department of Energy Office of Science. If there are other authors, they declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Acknowledgment

We would like to acknowledge the contributions of the Research Computing team at PNNL for their support.

References

[1] K. Hazelwood, S. Bird, D. Brooks, S. Chintala, U. Diril, D. Dzhulgakov, M. Fawzy, B. Jia, Y. Jia, A. Kalro, et al., Applied machine learning at facebook: A datacenter infrastructure perspective, in: 2018 IEEE International Symposium on High Performance Computer Architecture, HPCA, IEEE, 2018, pp. 620–629.

[2] J. Rasley, S. Rajbhandari, O. Ruwase, Y. He, Deepspeed: System optimizations enable training deep learning models with over 100 billion parameters, in: Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, 2020, pp. 3505–3506.

[3] T.L. Scao, T. Wang, D. Hesslow, L. Saulnier, S. Bekman, M.S. Bari, S. Bideman, H. Elsahar, N. Muennighoff, J. Phang, et al., What language model to train if you have one million GPU hours? 2022, arXiv preprint arXiv:2210.15424.

[4] S. Smith, M. Patwary, B. Norick, P. LeGresley, S. Rajbhandari, J. Casper, Z. Liu, S. Prabhumoye, G. Zerveas, V. Korthikanti, et al., Using deepspeed and megatron to train megatron-turing nlg 530b, a large-scale generative language model, 2022, arXiv preprint arXiv:2201.11990.

[5] D. Monroe, Chips for artificial intelligence, Commun. ACM 61 (4) (2018) 15–17.

[6] S. Krishna, R. Krishna, Accelerating recommender systems via hardware “scale-in”, 2020, arXiv preprint arXiv:2009.05230.

[7] S. Yu, H. Jiang, S. Huang, X. Peng, A. Lu, Compute-in-memory chips for deep learning: Recent trends and prospects, IEEE Circuits Syst. Mag. 21 (3) (2021) 31–56.

[8] X. Huang, G.C. Fox, S. Serebryakov, A. Mohan, P. Morkisz, D. Dutta, Benchmarking deep learning for time series: Challenges and directions, in: 2019 IEEE International Conference on Big Data (Big Data), IEEE, 2019, pp. 5679–5682.

[9] M. Jain, S. Ghosh, S.P. Nandanoori, Workload characterization of a time-series prediction system for spatio-temporal data, in: Proceedings of the 19th ACM International Conference on Computing Frontiers, 2022, pp. 159–168.

[10] T. Kurth, S. Treichler, J. Romero, M. Mudigonda, N. Luehr, E. Phillips, A. Mahesh, M. Matheson, J. Deslippe, M. Fatica, et al., Exascale deep learning for climate analytics, in: SC18: International Conference for High Performance Computing, Networking, Storage and Analysis, IEEE, 2018, pp. 649–660.

[11] T. Kurth, S. Subramanian, P. Harrington, J. Pathak, M. Mardani, D. Hall, A. Miele, K. Kashinath, A. Anandkumar, Fourcastnet: Accelerating global high-resolution weather forecasting using adaptive fourier neural operators, in: Proceedings of the Platform for Advanced Scientific Computing Conference, 2023, pp. 1–11.

[12] S. Abdulah, Q. Cao, Y. Pei, G. Bosilca, J. Dongarra, M.G. Genton, D.E. Keyes, H. Ltaief, Y. Sun, Accelerating geostatistical modeling and prediction with mixed-precision computations: A high-productivity approach with parsec, IEEE Trans. Parallel Distrib. Syst. 33 (4) (2021) 964–976.

[13] M. Jain, A. Singh, Combining multiple forecast for improved day ahead prediction of wind power generation, in: Proceedings of the 2015 ACM Sixth International Conference on Future Energy Systems, 2015, pp. 199–200.

[14] M. Burtscher, P. Ratanaworabhan, High throughput compression of double-precision floating-point data, in: 2007 Data Compression Conference, DCC'07, IEEE, 2007, pp. 293–302.

[15] M.A. Guillén, A. Llanes, B. Imbernón, R. Martínez-España, A. Bueno-Crespo, J.-C. Cano, J.M. Cecilia, Performance evaluation of edge-computing platforms for the prediction of low temperatures in agriculture using deep learning, J. Supercomput. 77 (2021) 818–840.

[16] M.A. Krinitskiy, V.M. Stepanenko, A.O. Malkhanov, M.E. Smorkalov, A general neural-networks-based method for identification of partial differential equations, implemented on a novel AI accelerator, Supercomput. Front. Innov. 9 (3) (2022) 19–50.

[17] N.-O. Skeie, H.N. Vahl, H. Viumdal, Modelling of snow depth and snow density based on capacitive measurements using machine learning methods., Scand. Simul. Soc. (2022) 84–91.

[18] D.H. Tran, H. Nguyen, Y.M. Jang, et al., Short-term solar power generation forecasting using edge AI, in: 2022 13th International Conference on Information and Communication Technology Convergence, ICTC, IEEE, 2022, pp. 341–343.

[19] I.N.K. Wardana, J.W. Gardner, S.A. Fahmy, Optimising deep learning at the edge for accurate hourly air quality prediction, Sensors 21 (4) (2021) 1064.

[20] H.-C. Chih, W.-C. Lin, W.-T. Huang, K.-C. Yao, Implementation of EDGE computing platform in feeder terminal unit for smart applications in distribution networks with distributed renewable energies, Sustainability 14 (20) (2022) 13042.

[21] T. Kato, S. Sugimura, M. Kurimoto, Y. Suzuoki, Y. Manabe, T. Funabashi, Short-run fluctuation of residual load by high penetration photovoltaic power generation system, in: 2014 International Conference on Probabilistic Methods Applied To Power Systems, PMAPS, IEEE, 2014, pp. 1–6.

[22] L. Ren, A. Osman, F. Lin, N.-y. Chiang, M. Sun, R. Melville, F. Koufogiannis, M. Moulin, T. Wagner, F.F. Li, et al., WISP: Watching grid Infrastructure Stealthily through Proxies, Tech. Rep., Raytheon Technologies Research Center, 2022.

[23] M. Jain, X. Sun, S. Datta, A. Somani, A machine learning framework to deconstruct the primary drivers for electricity market price events, in: 2023 IEEE Power & Energy Society General Meeting, PESGM, IEEE, 2023, pp. 1–5.

[24] M. Takatoo, S. Abe, T. Bando, K. Hirasawa, M. Goto, T. Kato, T. Kanke, Floating vector processor for power system simulation, IEEE Trans. Power Appar. Syst. (12) (1985) 3360–3366.

[25] S. Abhyankar, S. Peles, R. Rutherford, A. Mancinelli, Evaluation of ac optimal power flow on graphical processing units, in: 2021 IEEE Power & Energy Society General Meeting, PESGM, IEEE, 2021, pp. 01–05.

[26] W. Hatahet, L. Wang, Hybrid CPU-GPU-based electromagnetic transient simulation of modular multilevel converter for HVDC application, in: 2022 IEEE Electrical Power and Energy Conference, EPEC, IEEE, 2022, pp. 44–49.

[27] M. Jain, S. Kundu, A. Bhattacharya, S. Huang, V. Chandan, N. Radhakrishnan, V. Adetola, D. Vrabie, Occupancy-driven stochastic decision framework for ranking commercial building loads, in: 2021 American Control Conference, ACC, IEEE, 2021, pp. 4171–4177.

[28] G. Gürses-Tran, H. Flamme, A. Monti, Probabilistic load forecasting for day-ahead congestion mitigation, in: 2020 International Conference on Probabilistic Methods Applied To Power Systems, PMAPS, IEEE, 2020, pp. 1–6.

[29] B. Li, S. Wu, S. Wang, L. Zhang, Automatic inspection of power system operations based on lightweight neural network, in: 2022 7th Asia Conference on Power and Electrical Engineering, ACPEE, IEEE, 2022, pp. 2161–2165.

[30] Y. Zhang, J. Li, W. Sun, J. Zhang, Y. Peng, Z. Chen, X. Zheng, L. Peng, Intelligent patrol terminal of transmission line based on AI chip accelerated calculation, Energy Rep. 9 (2023) 190–198.

[31] I. Gudmundsson, G. Falco, Porting computer vision models to the edge for smart city applications: Enabling autonomous vision-based power line inspection at the smart grid edge for unmanned aerial vehicles (UAVs), in: HICSS, 2022, pp. 1–10.

[32] J. Strohbeck, V. Belagiannis, J. Müller, M. Schreiber, M. Herrmann, D. Wolf, M. Buchholz, Multiple trajectory prediction with deep temporal and spatial convolutional neural networks, in: 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS, IEEE, 2020, pp. 1992–1998.

[33] P.R. Ovi, E. Dey, N. Roy, A. Gangopadhyay, Aris: A real time edge computed accident risk inference system, in: 2021 IEEE International Conference on Smart Computing, SMARTCOMP, IEEE, 2021, pp. 47–54.

[34] A. Dutta, M. Jain, A. Khan, A. Sathanur, Deep reinforcement learning to maximize arterial usage during extreme congestion, 2023, arXiv preprint arXiv:2305.09600.

[35] O. Jayasinghe, S. Hemachandra, D. Anhettigama, S. Kariyawasam, T. Wickremasinghe, C. Ekanayake, R. Rodrigo, P. Jayasekara, Towards real-time traffic sign and traffic light detection on embedded systems, in: 2022 IEEE Intelligent Vehicles Symposium, IV, IEEE, 2022, pp. 723–728.

[36] D.-L. Dinh, H.-N. Nguyen, H.-T. Thai, K.-H. Le, Towards AI-based traffic counting system with edge computing, J. Adv. Transp. 2021 (2021) 1–15.

[37] Z. Han, J. Zhao, H. Leung, K.F. Ma, W. Wang, A review of deep learning models for time series prediction, IEEE Sens. J. 21 (6) (2019) 7833–7848.

[38] D. Ding, M. Zhang, X. Pan, M. Yang, X. He, Modeling extreme events in time series prediction, in: Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, 2019, pp. 1114–1122.

[39] M. Jain, N.M. Mohankumar, H. Wan, S. Ganguly, K.D. Wilson, D.M. Anderson, Training machine learning models to characterize temporal evolution of disadvantaged communities, 2023, arXiv preprint arXiv:2303.03677.

[40] R. Fu, Z. Zhang, L. Li, Using LSTM and GRU neural network methods for traffic flow prediction, in: Proceedings - 2016 31st Youth Academic Annual Conference of Chinese Association of Automation, YAC 2016, Institute of Electrical and Electronics Engineers Inc., 2017, pp. 324–328, http://dx.doi.org/10.1109/YAC.2016.7804912.

[41] M. Jain, V. Amatya, B. Harrison, K. Hazelwood, G. Panapitiya, W. Pellico, B. Schupbach, K. Seiya, J. St John, J. Strube, The L-CAPE Project at FNAL, Fermi National Accelerator Lab.(FNAL), Batavia, IL (United States); Pacific, 2022.

[42] P. Lasley, 2023 Urban mobility report, 2023.

[43] M. Deehan, I. Renta, S. Yaary, J. Fillingham, S. Sarkar, G. Matlock, L. Newcomb, E. Bayler, J. Ghirardelli, M. Grasso, et al., 2023 NOAA science report, 2024.

[44] Pacific Gas and Electric Company (PG&E), 2018 Energy Efficiency Annual Report, Research Report, PG&E, 2018.

[45] C. Zhang, J.M. Kovacs, The application of small unmanned aerial systems for precision agriculture: a review, Precis. Agric. 13 (2012) 693–712.

[46] J.J. Hopfield, Neurons with graded response have collective computational properties like those of two-state neurons, Proc. Natl. Acad. Sci. 81 (10) (1984) 3088–3092, http://dx.doi.org/10.1073/PNAS.81.10.3088, URL https://www.pnas.org/content/81/10/3088 https://www.pnas.org/content/81/10/3088.abstract.

[47] S. Hochreiter, J. Schmidhuber, Long short-term memory, Neural Comput. 9 (8) (1997) 1735–1780, http://dx.doi.org/10.1162/NECO.1997.9.8.1735.

[48] J. Zhou, G. Cui, S. Hu, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, M. Sun, Graph neural networks: A review of methods and applications, AI Open 1 (2020) 57–81.

[49] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A.N. Gomez, Ł. Kaiser, I. Polosukhin, Attention is all you need, in: Advances in Neural Information Processing Systems, 2017, pp. 5998–6008.

[50] A.F. Ansari, L. Stella, C. Turkmen, X. Zhang, P. Mercado, H. Shen, O. Shchur, S.S. Rangapuram, S.P. Arango, S. Kapoor, et al., Chronos: Learning the language of time series, 2024, arXiv preprint arXiv:2403.07815.

[51] A. Zeng, M. Chen, L. Zhang, Q. Xu, Are transformers effective for time series forecasting? in: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 37, (9) 2023, pp. 11121–11128.

[52] N. Bohm Agostini, S. Curzel, J. Zhang, A. Limaye, C. Tan, V. Amatya, M. Minutoli, V.G. Castellana, J. Manzano, D. Brooks, G.-Y. Wei, A. Tumeo, Bridging python to silicon: The SODA toolchain, IEEE Micro 42 (5) (2022) http://dx.doi.org/10.1109/MM.2022.3178580.

[53] N. Bohm Agostini, S. Curzel, V. Amatya, C. Tan, M. Minutoli, V.G. Castellana, J. Manzano, D. Kaeli, A. Tumeo, An MLIR-based compiler flow for system-level design and hardware acceleration, in: Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design, ICCAD'22, 2022, pp. 1–9, http://dx.doi.org/10.1145/3508352.3549424.

[54] C. Lattner, M. Amini, U. Bondhugula, A. Cohen, A. Davis, J.A. Pienaar, R. Riddle, T. Shpeisman, N. Vasilache, O. Zinenko, MLIR: Scaling compiler infrastructure for domain specific computation, in: J.W. Lee, M.L. Soffa, A. Zaks (Eds.), IEEE/ACM International Symposium on Code Generation and Optimization, CGO 2021, Seoul, South Korea, February 27 - March 3, 2021, IEEE, 2021, pp. 2–14, http://dx.doi.org/10.1109/CGO51591.2021.9370308.

[55] F. Ferrandi, V.G. Castellana, S. Curzel, P. Fezzardi, M. Fiorito, M. Lattuada, M. Minutoli, C. Pilato, A. Tumeo, Bambu: an Open-Source Research Framework for the High-Level Synthesis of Complex Applications, in: Proceedings of the ACM/IEEE Design Automation Conference, DAC '21, 2021, pp. 1327–1330, http://dx.doi.org/10.1109/DAC18074.2021.9586110.

[56] R. Prabhakar, S. Jairath, SambaNova SN10 RDU: Accelerating software 2.0 with dataflow, in: 2021 IEEE Hot Chips 33 Symposium, HCS, IEEE, 2021, pp. 1–37.

[57] O. Shafi, C. Rai, R. Sen, G. Ananthanarayanan, Demystifying TensorRT: Characterizing neural network inference engine on NVIDIA edge devices, in: 2021 IEEE International Symposium on Workload Characterization, IISWC, IEEE, 2021, pp. 226–237.

[58] H. Zhang, Y. Li, W. Xiao, Y. Huang, X. Di, J. Yin, S. See, Y. Luo, C.T. Lau, Y. You, MIGPerf: A comprehensive benchmark for deep learning training and inference workloads on multi-instance GPUs, 2023, arXiv preprint arXiv:2301.00407.

[59] V.J. Reddi, C. Cheng, D. Kanter, P. Mattson, G. Schmuelling, C.-J. Wu, B. Anderson, M. Breughe, M. Charlebois, W. Chou, et al., MLPerf inference benchmark, in: 2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture, ISCA, IEEE, 2020, pp. 446–459.

[60] Q. Liang, P. Shenoy, D. Irwin, AI on the edge: Characterizing AI-based IoT applications using specialized edge architectures, in: 2020 IEEE International Symposium on Workload Characterization, IISWC, IEEE, 2020, pp. 145–156.

[61] M. Suri, A. Gupta, V. Parmar, K.H. Lee, Performance enhancement of edge-AI-inference using commodity MRAM: IoT case study, in: 2019 IEEE 11th International Memory Workshop, IMW, IEEE, 2019, pp. 1–4.

[62] Y. Wang, G.-Y. Wei, D. Brooks, J.A. Paulson, Benchmarking TPU, GPU, and CPU platforms for deep learning, 2019, arXiv preprint arXiv:1907.10701.

[63] J. Appleyard, T. Kocisky, P. Blunsom, Optimizing performance of recurrent neural networks on GPUs, 2016, arXiv preprint arXiv:1604.01946.

[64] S. Braun, LSTM benchmarks for deep learning frameworks, 2018, arXiv preprint arXiv:1806.01818.

[65] D. Li, X. Chen, M. Becchi, Z. Zong, Evaluating the energy efficiency of deep convolutional neural networks on CPUs and GPUs, in: 2016 IEEE International Conferences on Big Data and Cloud Computing (BDCloud), Social Computing and Networking (SocialCom), Sustainable Computing and Communications (SustainCom)(BDCloud-SocialCom-SustainCom), IEEE, 2016, pp. 477–484.

[66] N.A. Gawande, J.A. Daily, C. Siegel, N.R. Tallent, A. Vishnu, Scaling deep learning workloads: Nvidia DGX-1/pascal and intel knights landing, Future Gener. Comput. Syst. 108 (2020) 1162–1172.

[67] G. Liu, F. Dai, X. Xu, X. Fu, W. Dou, N. Kumar, M. Bilal, An adaptive DNN inference acceleration framework with end–edge–cloud collaborative computing, Future Gener. Comput. Syst. 140 (2023) 422–435.

[68] T. Choudhary, V. Mishra, A. Goswami, J. Sarangapani, Inference-aware convolutional neural network pruning, Future Gener. Comput. Syst. 135 (2022) 44–56.

[69] S. Huang, K. Wu, H. Jeong, C. Wang, D. Chen, W. Hwu, PyLog: An algorithm-centric python-based FPGA programming and synthesis flow, IEEE Trans. Comput. 70 (12) (2021) 2015–2028.

[70] J. Ngadiuba, V. Loncar, M. Pierini, S. Summers, G. Di Guglielmo, J. Duarte, P. Harris, D. Rankin, S. Jindariani, M. Liu, et al., Compressing deep neural networks on FPGAs to binary and ternary precision with hls4ml, ML Sci. Technol. 2 (1) (2020) 015001.

[71] H. Ye, C. Hao, J. Cheng, H. Jeong, J. Huang, S. Neuendorffer, D. Chen, ScaleHLS: Scalable high-level synthesis through MLIR, 2021, pp. 1–13, arXiv preprint arXiv:2107.11673.

[72] Z. Jakšić, N. Cadenelli, D.B. Prats, J. Polo, J.L.B. Garcia, D.C. Perez, A highly parameterizable framework for conditional restricted Boltzmann machine based workloads accelerated with FPGAs and OpenCL, Future Gener. Comput. Syst. 104 (2020) 201–211.

[73] N. Cadenelli, Z. Jaksić, J. Polo, D. Carrera, Considerations in using OpenCL on GPUs and FPGAs for throughput-oriented genomics workloads, Future Gener. Comput. Syst. 94 (2019) 148–159.

[74] M. Wang, D. Zheng, Z. Ye, Q. Gan, M. Li, X. Song, J. Zhou, C. Ma, L. Yu, Y. Gai, T. Xiao, T. He, G. Karypis, J. Li, Z. Zhang, Deep graph library: A graph-centric, highly-performant package for graph neural networks, 2019, arXiv preprint arXiv:1909.01315.

[75] E. Hoseinzade, S. Haratizadeh, CNNpred: CNN-based stock market prediction using a diverse set of variables, Expert Syst. Appl. 129 (2019) 273–285, http://dx.doi.org/10.1016/J.ESWA.2019.03.029.

[76] K. Wang, K. Li, L. Zhou, Y. Hu, Z. Cheng, J. Liu, C. Chen, Multiple convolutional neural networks for multivariate time series prediction, Neurocomputing 360 (2019) 107–119, http://dx.doi.org/10.1016/J.NEUCOM.2019.05.023.

[77] J. Bai, F. Lu, K. Zhang, et al., ONNX: Open neural network exchange, 2019.

[78] B. Xu, Y. Zhang, H. Lu, Y. Chen, T. Chen, M. Iovine, M.-C. Lee, Z. Li, AITemplate, 2022, URL https://github.com/facebookincubator/AITemplate.

[79] H. Vanholder, Efficient inference with tensorrt, in: GPU Technology Conference, vol. 1, 2016.

[80] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G.S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, X. Zheng, TensorFlow: Large-scale machine learning on heterogeneous systems, 2015, Software available from tensorflow.org.

[81] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, S. Chintala, PyTorch: An imperative style, high-performance deep learning library, in: H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché Buc, E. Fox, R. Garnett (Eds.), in: Advances in Neural Information Processing Systems, vol. 32, Curran Associates, Inc., 2019, pp. 8024–8035.

[82] S.P. Nandanoori, S. Kundu, S. Pal, S. Choudhury, K. Agarwal, Nominal and adversarial synthetic PMU data for standard IEEE test systems, 2021, https://data.pnl.gov/publication/grid_prediction, (Accessed 1 July 2021).

[83] NO.A.A. National Centers For Environmental Information, Integrated surface dataset, 2023, https://www.ncdc.noaa.gov/isd, (Accessed 17 July 2023).

Milan Jain is a Data Scientist at the Pacific Northwest National Laboratory. His research interests lie at the intersection of Machine Learning, Sensor Networks, and Human. In his Ph.D., he chose residential buildings as his application domain and worked on designing low-cost, scalable, and ubiquitous thermostats to make residential air-conditioners smart, energy-efficient, comfortable, and reliable for the tenants. In his Ph.D., he collaborated with several well-known research groups around the world. In 2018, he was awarded prestigious IUSSTF BHAVAN Fellowship to visit the Optimization and Control Group at Pacific Northwest National Laboratory in the USA as a Research Intern. In 2018 only, he also interned with the Responsive Environments Group at MIT Media Lab to explore the influence of the thermal properties of the environment on human psychology. Apart from this, during his Ph.D., he also visited and collaborated with the University of Waterloo, IBM Research, and Zenatix on various projects.

Nicolas Bohm Agostini is a Computer Scientist at the Pacific Northwest National Laboratory. He did his Ph.D. from the Department of Electrical and Computer Engineering at Northeastern University (NEU, Boston, MA). In 2015, he completed his Bachelors's degree in Electrical Engineering at the Universidade Federal do Rio Grande do Sul (UFRGS, Brazil). In 2022, he received his Master of Science in Computer Engineering from NEU. His primary research interests include Computer Architecture and High-Performance Computing. He enjoys teaching and mentoring, which he exemplified by instructing courses such as Compilers, GPU Programming, and Embedded Robotics during his graduate career. Recently, he has been working on projects focused on accelerating machine learning and linear algebra applications by proposing compiler extensions for different targets or designing new computer architecture features. He joined PNNL in 2019, working as the lead developer of the SODA-OPT compiler, which performs automatic system-level partitioning of high-level applications and automatic code optimizations for better custom hardware generation outcomes.

Sayan Ghosh is a Computer Scientist at the Pacific Northwest National Laboratory. His research interests are broadly in the application of parallel programming models for HPC systems. Particularly, my interests are in analyzing and improving application communication patterns (such as bulk-synchronous, irregular one-sided, producer–consumer, implicit/explicit cartesian/graph process topology, etc.) for building scalable parallel codes on supercomputers. I also have a keen interest in performance analysis of parallel applications, and strongly believe that aiming for scalability without a thorough analysis often leads to costly mistakes.

Antonino Tumeo received the M.S degree in Informatic Engineering, in 2005, and the Ph.D degree in Computer Engineering, in 2009, from Politecnico di Milano in Italy. Since February 2011, he has been a research scientist in the PNNL's High Performance Computing group. He Joined PNNL in 2009 as a post doctoral research associate. Previously, he was a post doctoral researcher at Politecnico di Milano. His research interests are modeling and simulation of high performance architectures, hardware-software codesign, electronic design automation, high-level synthesis, FPGA prototyping and GPGPU computing. Antonino is currently the PI of SODALITE (Software Defined Accelerators from Learning Tools Environment) a project in the DARPA Real Time Machine Learning program related to the automatic generation of Machine Learning accelerators, and SO(DA)ˆ2 (Software Defined Architectures for Data Analytics), a project in the Data-Model Convergence (DMC) Initiative related to the development of a toolchain to efficiently accelerate emerging high-performance computing applications (integrating scientific simulation with machine learning and data analytics) on novel reconfigurable architectures.