Fast Online Digital Twinning on FPGA for Mission Critical Applications

Bin Xu
School of ECEE
Arizona State University
Tempe, USA
binxu4@asu.edu

Ayan Banerjee
School of Computing and AI
Arizona State University
Tempe, USA
Ayan.Banerjee@asu.edu

Sandeep K. S. Gupta
School of Computing and AI
Arizona State University
Tempe, USA
Sandeep.Gupta@asu.edu

***Abstract*—Digital twinning enables real-time simulation and predictive modeling by maintaining a continuously updated virtual representation of a physical system. In mission-critical applications—such as mid-air collision avoidance—these models must operate online with extremely low latency to ensure safety. However, executing complex Model Recovery (MR) pipelines on edge devices is limited by computational and memory bandwidth constraints. This paper introduces a fast, FPGA-accelerated digital twinning framework that offloads key neural components—including gated recurrent units (GRU) and dense layers—to reconfigurable hardware for efficient parallel execution. Our system achieves real-time responsiveness, operating five times faster than typical human reaction time, and demonstrates the practical viability of deploying digital twins on edge platforms for time-sensitive, safety-critical environments.**

***Index Terms*—Model recovery, FPGAs, Mobile GPUs, hardware acceleration.**

I. Introduction

Digital twins are virtual models that mirror physical systems in real time using live data streams and simulations, enabling analysis, prediction, and control. According to Tao et al. [24], digital twins are increasingly deployed across domains such as manufacturing, aerospace, healthcare, and smart infrastructure, offering transformative capabilities in predictive maintenance, optimization, and decision support.

At the core of digital twinning is data-driven predictive inference, which enables virtual models to simulate, monitor, and forecast system dynamics in real time. A key approach within this framework is physics-guided model recovery (MR) [2], where the underlying governing equations of system behavior are learned from real-world data—despite challenges such as sparse sampling, latent or unmonitored dynamics, and human-induced noise. Broadly, data-driven inference engines can be categorized into: a) Generalized model learning (ML), where a high-dimensional neural network is trained directly to predict outputs from input data, and b) Model recovery (MR), which aims to extract interpretable physical laws—e.g., through frameworks such as Physics-informed Neural Ordinary Differential Equations (PiNODE) [22], sparse-regression PINNs (PINN+SR) [7], or Extracting sparse Model from ImpLicit dYnamics (EMILY) [1], [2] (Fig. 1).

The strength of MR lies in its ability to detect subtle deviations from expected system behavior—such as actuator faults, sensor spoofing, or external disturbances—without relying on pre-defined fault categories. This makes MR particularly valuable in mission-critical autonomous systems (MCAS) environments, where rapid response and reliability are paramount. However, the deployment of MR algorithms in real-time settings remains computationally intensive, as many frameworks rely on solving ODEs, often through neural ODE-based architectures that are not optimized for low-latency inference.

To investigate this challenge in a practical safety-critical setting, we focus on the F8 Crusader aircraft model [12]—a nonlinear dynamical system widely used in aerospace control and verification studies. Our objective is to apply MR techniques to detect collision-course anomalies and enable timely avoidance maneuvers. Prior studies have shown that human pilots typically require five seconds to respond to mid-air collision threats [9]. Autonomous systems must react significantly faster, often in sub-second windows, to ensure safety in constrained airspace.

Achieving such rapid inference requires compute-efficient deployment of the digital twin's predictive components. In MCAS scenarios, systems face tight constraints on power, compute, and memory—particularly for onboard or edge devices. As a result, there is growing interest in accelerating inference models using reconfigurable hardware such as Field-Programmable Gate Arrays (FPGAs). These devices provide high-throughput, low-latency, and energy-efficient execution, making them ideal for edge-based AI systems [13], [29]. Prior work has made significant progress in accelerating conventional ML models on FPGAs [11], [17]. Existing FPGA-based MR frameworks [27], [28], however, primarily target low-dimensional inputs in medical devices, with relatively little attention given to model scalability. This motivates our work on FPGA-based acceleration strategies specifically designed for MR-based digital twins operating on high-dimensional inputs.

To contextualize the challenge of accelerating model recovery (MR) architectures, it is helpful to contrast them with conventional time-series machine learning models. The widespread adoption of FPGA accelerators for time-series ML predictors can be attributed to their highly regular and well-understood architecture. These models typically consist of an input embedding layer followed by a stack of recurrent neural network (RNN) layers that compute discretized temporal dynamics. Their forward pass reduces to structured matrix multiplication operations, while nonlinear functions such as sine or cosine can be efficiently computed using specialized

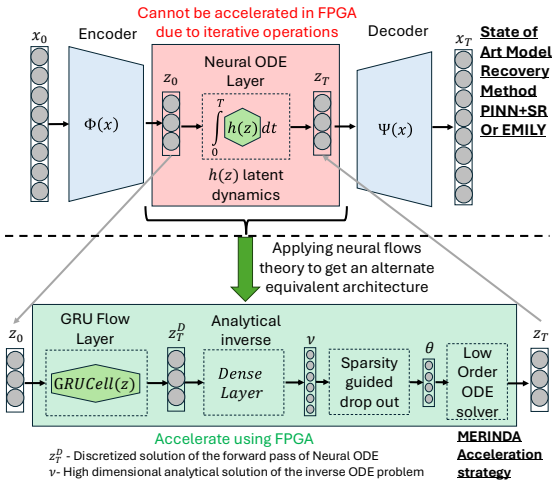

Fig. 1. FPGA acceleration strategy using neural flow based equivalent architecture to neural ODEs.

FPGA components like CORDIC [5]. The backward pass follows standard gradient-based pipelines, further simplifying their suitability for hardware acceleration.

In contrast, neural ODE-based architectures such as ODENet [26] and NODE [4] present greater challenges for acceleration. These works have primarily focused on fixed-depth, static architectures, which limit their flexibility and applicability to general MR frameworks. While they offer valuable insights into hardware-aware ODE acceleration, they fall short of supporting dynamic architectures like PiNODE, PINN+SR, or EMILY, where the number of NODE layers may vary depending on the application or system complexity.

As shown in Fig. 1, a significant portion of EMILY (a baseline comparator in our study) is composed of NODE cells [6]. The forward pass of a NODE cell involves solving a high-dimensional ODE using numerical solvers with tight precision requirements. One of the fundamental challenges in accelerating MR frameworks lies in the inherently iterative and adaptive nature of this computation. Recent studies have attempted to accelerate ODE solvers for standalone, fixed-coefficient equations on FPGA [10], [23], but these designs are not suitable for PiNODE-like settings, which require solving a large set of ODEs with varying, input-dependent model coefficients.

In this paper, we utilize the theory of neural flows [3] to obtain an alternative neural structure called MERINDA, model recovery in dynamic architectures, that is equivalent to the NODE layers used in EMILY, PiNODE or PINN+SR and more amenable for acceleration in an FPGA. MERINDA replaces the NODE layer with a layer of invertible functions designed using a combination of gated recurrent units (GRU) and a dense layer of neurons with nonlinear activation functions. GRU [21] are a type of recurrent neural network (RNN) architecture that introduces gating mechanisms to control the flow of information over time. Compared to traditional RNNs or LSTMs, GRUs are computationally efficient and require fewer parameters [8], making them favorable for deployment on resource-constrained platforms such as FPGAs.

The primary contributions of this paper are as follows:
1. An empirical evaluation of MERINDA against state-of-the-art model recovery (MR) approaches—specifically EMILY and PINN+SR—across four benchmark nonlinear dynamical systems, including two real-world and two simulated datasets, demonstrating its MR accuracy.
2. A detailed analysis of FPGA resource utilization and throughput across varying dimensional configurations of the F8 Crusader system, along with performance metrics after hardware-specific optimization.
3. A comparative study between MERINDA and GPU-based implementations (e.g., CUDA) to quantify the acceleration benefits.

II. Theoretical Background

This section presents the basics of MR and establishes approximate equivalence of neural flow architecture with NODE.

A. Basics of Model Recovery

The main goal of MR is akin to an auto-encoder (Fig. 1), where given a multivariate time series signal $X(t)$, the aim is to find a latent space representation that can be used to reconstruct an estimation $\tilde{X}(t)$ with low error. It has the traditional encoder $\phi(t)$ and decoder $(\Psi(t))$ of an autoencoder architecture. MR represents the measurements X of dimension n and N samples, as a set of nonlinear ordinary differential equation model in (1).

$$\dot{X} = h(X, U, \theta), \tag{1}$$

where h is a parameterized nonlinear function, U is the m dimensional external input, and θ is the p dimensional coefficient set of the nonlinear ODE model.
Sparsity: An n-dimensional model with M^{th} order nonlinearity can utilize $\binom{M+n}{n}$ nonlinear terms. A sparse model only includes a few nonlinear terms $p << \binom{M+n}{n}$. Sparsity structure of a model is the set of nonlinear terms used by it.
Identifiable model: A model in (1) is identifiable [25], if $\exists$ time $t_I > 0$, such that $\forall \theta, \tilde{\theta} \in \mathcal{R}^p$:

$$\forall t \in [0, t_I], f(X(t), U(t), \theta) = f(X(t), U(t), \tilde{\theta}) \implies \theta = \tilde{\theta}. \tag{2}$$

Eqn. 2 effectively means that a model is identifiable if two different model coefficients do not result in identical measurement X. In simpler terms this means $\forall \theta_i \in \theta$, $\frac{dX}{d\theta_i} \neq 0$. In this paper, we assume that the underlying model is identifiable.

Problem 1 (Sparse Model Recovery): Given N samples of measurements X and inputs U, obtained from a sparse model in Eqn. 1 such that θ is identifiable, recover $\tilde{\theta}$ such that for $\tilde{X}$ generated from $f(X, U, \tilde{\theta})$, we have $||X - \tilde{X}|| \leq \epsilon$, where ϵ is the maximum tolerable error.
Role of NODE: Both EMILY [1] and PINN+SR [7] utilize a layer of NODE cells in order to integrate the underlying nonlinear ODE dynamics. NODE cell's forward pass is by design the integration of the function h over time horizon T with N samples (Fig. 1). This effectively requires an ODE solver in each cell of the NODE layer:

$$z(t) = \int_0^T h(z, u, \theta) dt, \tag{3}$$

where $z \in Z$ and $u \in U$ are each cells output and input.

The results are then used further in the EMILY or PINN+SR pipeline to extract the accurate underlying nonlinear ODE model.

B. Neural flows and equivalent architectures to NODE

According to the theory of neural flows [3], the node layer can be replaced by an approximate solution to $F(t) \approx Z(t)$ in discretized form using recurrent nerual network architectures such as GRU provided that following conditions are satisfied:

$$F(0,u) = Z(0,u), \text{(initial condition), and } F(t,u) \text{ is invertible.} \quad (4)$$

Bilovs et al. [3] show that $F(t,u)$ can be achieved by replacing the original NODE layer by a GRU layer. However, the GRU layer does not statisfy by the invertible condition. The authors in [3] suggest the usage of a dense layer since it acts as a universal approximator of nonlinear functions and hence can also act as the inversion of the function $F(t,u)$.

MERINDA further enhances the equivalent architecture proposed in [3] by further pruning the dense layer as shown in Fig. 2. The main idea is to further reduce the dense layer structure by utilizing the inherent sparsity in the data. Given the definitions of identifiability and sparsity, we now discuss our full architecture that is equivalent to PiNODE

III. MERINDA Architecture

A. GRU NN-based MR architecture

In our approach (Fig. 2), we extend gated recurrent unit neural network (GRU-NN) to obtain advanced neural structure MERINDA that can solve the model recovery problem. The forward pass of GRU-NN structure expresses the coefficients of the model as a nonlinear function of the outputs Y and inputs U of the model. The measurements of Y, can be used to convert the set of implicit dynamics to an overdetermined system of equations that are nonlinear in terms of the model coefficients. As such an over-determined system of equation may have no solution unless either some equations are rejected or are expressed as linear superposition of other equations. To search for a set of consistent equations to estimate model coefficient, a dense layer is utilized. The search process of the dense layer is guided by a loss function (*ODE loss*) that computes the mean square error between the estimated Y_{est} using an ODE solver $\mathbf{SOLVE}(Y(0), \Theta, U)$ and the ground truth measurements Y.

The advanced neural architectures for model recovery in Fig. 2 is implemented by extending the base code available in [14]. We extract the training data consisting of temporal traces of Y, and U . Y is sampled at least at the Nyquist rate for the application, and U has the same sampling rate as Y. The resulting training data is then divided into batches of size S_B. This forms a 3D tensor of size $S_B \times |Y| + m \times k$.

Each batch is passed through the *GRU-NN* network with V nodes, resulting in V hidden states. A dense layer is then employed to transform these V hidden states into $p = |\Theta|$ model coefficient estimates and q input shift values. The dense layer is a multi-layer perceptron with ReLU activation function for the model coefficient estimate nodes, whose outputs are the estimated model coefficients. The dense layer converts the V dimensional hidden layer outputs to $\binom{M+|X|}{|X|}$ which is the number of nonlinear terms that can be used for an Mth order polynomial. A dropout rate of $|\Theta|$ is used so that the final number of output layers with non-zero activation is $|\Theta|$. The model coefficient estimates and the initial value $Y(0)$ is passed through an ODE solver to solve the nonlinear dynamical equations with the coefficients Θ_{est}, initial conditions $Y(0)$ and inputs U. The Runge Kutta integration method is used in the ODE solver, which gives Y_{est}. In the backpropagation phase the network loss is appended with ODE loss, which is the mean square error between the original trace Y and the estimated trace Y_{est}.

B. FPGA Architecture and Optimization

An FPGA (Field-Programmable Gate Array) is a reconfigurable semiconductor device that enables developers to implement custom digital circuits directly in hardware [18]. Unlike fixed-function processors, FPGAs consist of an array of Configurable Logic Blocks (CLBs), Look-Up Tables (LUTs) for implementing combinational logic, flip-flops and registers for sequential logic, and programmable interconnects [16]. FPGAs also incorporate on-chip memory resources, such as Block RAM (BRAM) and UltraRAM (URAM), as well as Digital Signal Processing (DSP) slices optimized for arithmetic-intensive operations.

One of the primary challenges in FPGA design lies in efficiently mapping high-level algorithms onto limited hardware resources while maximizing performance. Loop-carried dependencies—such as Read-After-Write (RAW), Write-After-Read (WAR)—can inhibit effective pipelining, and reduce throughput. In addition to control hazards, memory access patterns pose a significant design challenge. FPGAs feature a hierarchical memory system including block RAM (BRAM), Look-Up Tables (LUTs), and Flip-Flops (FF), all of which must be judiciously partitioned and scheduled to avoid access bottlenecks and ensure data locality.

In our design, illustrated in Fig. 3, we address these challenges through two key techniques: *array partitioning* and *loop pipelining*, both guided by high-level synthesis (HLS) directives. The FPGA kernel interfaces with the processor using an AXI4-Lite protocol, after which input data is transferred to on-chip memory. We apply full array partitioning using the directive `#pragma HLS ARRAY_PARTITION complete`, which instructs the HLS compiler to map each element of the input array to an independent storage resource—such as a dedicated register or BRAM segment. This partitioning strategy eliminates inter-element memory conflicts and enables parallel access to the data elements.

As shown in Fig.3, individual elements like `array[0][0]` through `array[0][3]` are independently accessible and routed to distinct computational units, allowing multiple operations to proceed concurrently. This fine-grained parallelism directly mitigates the memory bottlenecks described earlier, as each operation now operates on separate physical storage units. Additionally, it maximizes utilization of on-chip memory resources by spreading the access load across registers, BRAM, and LUTs.

We then construct a fully parallelized model recovery pipeline on the FPGA. All major computational stages—including the forward pass, backpropagation, and

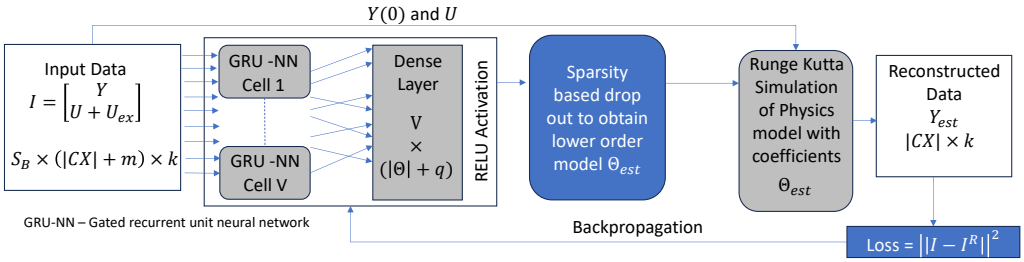

Fig. 2. MERINDA: Gated recurrent unit (GRU) NN-based MR architecture.

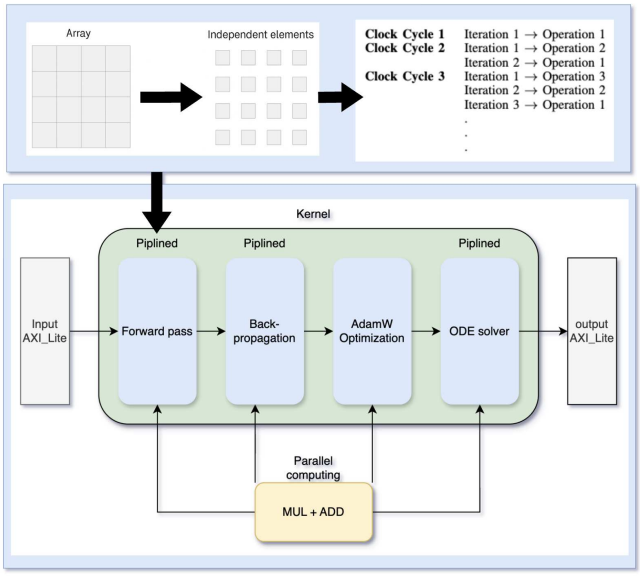

Fig. 3. After the input data is partitioned, then pipeline can be applied to compute.

loss computation—are pipelined using `#pragma HLS PIPELINE II=1`. Once the inputs are partitioned and loop-carried dependencies are removed, this setup achieves an initiation interval (II) of 1, allowing a new iteration to begin every clock cycle. This significantly boosts throughput and latency performance. There can be the violation of loop dependency in the simulation. In order to eliminate RAW and WAR hazards, we need to test `#pragma HLS PIPELINE II=2` or `#pragma HLS PIPELINE II=3`, which means a new iteration begins in every 2 cycles and 3 cycles. If there is no time violation in the simulation, it means there are no RAW and WAR hazards. However, more cycles mean more latency in the pipeline of computation.

In the presence of data dependencies, such as in the sequentially linked operations of the GRU forward pass (Operations 1 to 3) and backpropagation, pipelining allows for overlapping execution. Specifically, operations 1–3—outlined in the forward pass code—exhibit loop-carried dependencies, such as between `z[i]` and `rz_concat[i]`. With proper pipelining and no write-read hazards, the next operation can begin in the following cycle, thereby enabling deep pipelining across dependent computations.

Operation 1: Compute Reset and Update Gates

```
for i in 0 to H-1:
    z_sum = bias_z[i]
    for j in 0 to H + V - 1:
        z_sum += Wz[i][j] * concat[j]
    z[i] = sigmoid(z_sum)
```

Operation 2: Apply reset gate to previous hidden state

```
for i in 0 to H-1:
    rz_concat[i] = r[i] * a_prev[i]
```

Operation 3: Compute Candidate Activation

```
for i in 0 to H-1:
    cc_sum = bias_c[i]
    for j in 0 to H + V - 1:
        cc_sum += Wa[i][j] * rz_concat[j]
    c_t[i] = tanh(cc_sum)
```

IV. Evaluation and Results

A. Implementation Details

To evaluate the performance of the FPGA, we perform experiments on the mobile GPU and FPGA with the same hidden layer size, epoch size and model dimension size. The performance of the mobile GPU is set as a baseline.

a) Mobile GPU Platform: Our experiments are conducted on the NVIDIA Jetson Orin Nano Developer Kit, which features a 6-core Arm Cortex-A78AE CPU and 8 GB of LPDDR5 memory. The integrated GPU is based on the NVIDIA Ampere architecture, equipped with 1024 CUDA cores and 32 Tensor Cores.

We implemented our GPU simulation using CUDA C++, which offers near-zero runtime overhead compared to Python's interpreted execution model. Unlike Python, CUDA C++ enables fine-grained control over thread management, shared memory reuse, and global memory access patterns, allowing for highly optimized low-level execution [19]. In our evaluations, the CUDA C++ implementation achieved over 10× speedup relative to its Python version.

b) FPGA Platform: For the FPGA platform, the experiments were performed on Zynq UltraScale+ MPSoC with Quad ARM Cortex-A53, which includes 252K LUTs and 504K Flip-Flops. The GRU cell was built from scratch in C++ using High-Level Synthesis (HLS) on AMD's Vitis tool. The whole model recovery is simulated on Vitis. The execution time is based on **latency (cycles)** multiplied by the **clock period (ns)**.

TABLE I
COMPARISON BETWEEN MERINDA AND SOTA MR TECHNIQUES EMILY AND PINN+SR USING RECONSTRUCTION MSE. ERRORS ARE ABSOLUTE VALUES; NUMBERS IN PARENTHESES INDICATE STANDARD DEVIATION.

System	EMILY	PINN+SR	MERINDA
Lotka Volterra	0.03 (0.02)	0.05 (0.03)	0.03 (0.018)
Chaotic Lorenz	1.7 (0.6)	2.11 (1.4)	1.68 (0.4)
F8 Crusader	4.2 (2.1)	6.9 (4.4)	5.1 (2.2)
Pathogenic Attack	14.3 (12.1)	21.4 (5.4)	15.1 (10.2)

B. Results

a) Accuracy Comparison of MERINDA with EMILY and PINN+SR: We evaluate the accuracy of MERINDA using the mean squared error (MSE) metric across standard benchmark examples from [15]. Table I reports the reconstruction errors for MERINDA, along with published results for EMILY [1] and PINN+SR [20]. The comparison shows that MERINDA achieves comparable or improved accuracy relative to these state-of-the-art model recovery techniques while offering additional benefits in efficiency and hardware acceleration.

b) Performance of MERINDA on F8 Crusader: The Fig. 4 compares the model recovery time on FPGA for varying model dimensions, with and without hardware-level optimization. The x-axis denotes the dimensionality of the F8 Crusader model (Eqs. 7, 8 and 9 from [12]), and the y-axis shows the total execution time in seconds (log-scaled).

The blue curve represents the baseline implementation with no optimization, which exhibits a steep and nonlinear increase in runtime as the model dimension grows. This is primarily due to increased loop latency and limited parallelism in the unoptimized pipeline. In contrast, the red curve shows the performance of the optimized implementation, which employs both loop pipelining and unrolling. These optimizations reduce latency by allowing concurrent computation and minimizing memory access bottlenecks. Notably, at model dimension 150, the optimized design achieves more than a 10× speedup compared to the unoptimized version (1.04s vs. 11.84s), highlighting the effectiveness of hardware optimization in scaling up real-time model recovery.

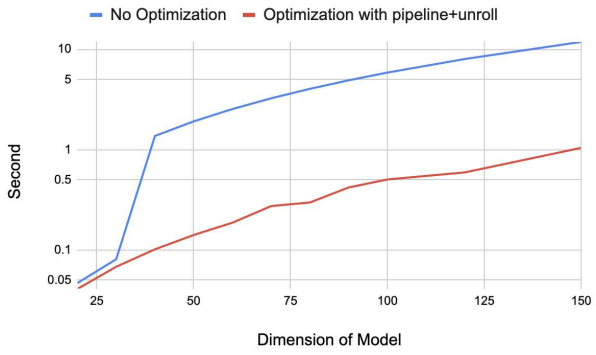

Fig. 4. Impact of Optimization on Model Recovery Execution Time.

Table II presents the resource utilization and execution time of the model recovery process on FPGA and GPU as the F8 Crusader model dimension increases. As expected, both execution time and hardware resource consumption (LUTs, DSPs, BRAM) grow with model complexity. Compared to

TABLE II
COMPARISON OF RESOURCE USAGE AND EXECUTION TIME BETWEEN FPGA AND GPU FOR INCREASING F8 CRUSADER MODEL DIMENSION. BRAM(KB), FPGA(S), GPU(S)

Size	Cycles	LUT	DSP	BRAM	FPGA	GPU
20	17,019	314,433	2,419	95	0.0408	0.323
30	28,336	463,953	3,745	135	0.0680	0.387
40	42,255	616,316	5,003	247	0.1014	0.400
50	58,754	745,437	6,119	297	0.1410	0.428
60	77,875	891,062	7,466	347	0.1869	0.494
70	114,082	806,340	7,276	397	0.2738	0.535
80	123,915	1,194,348	9,826	447	0.2974	0.619
90	174,862	1,038,178	9,271	497	0.4197	0.990
100	210,052	1,128,589	10,139	547	0.5041	1.051
120	247,195	1,779,201	14,831	647	0.5933	1.244
150	434,003	1,690,814	15,270	797	1.0416	1.541

GPU implementation, the FPGA implementation shows lower latency because more weights are loaded onto registers such as LUT for computation. With the pipeline optimization, the computation on FPGA can reuse data on shared memory.

Table III compares three FPGA configurations using dimension 30 as a reference point: a baseline design with no optimization, a partially optimized version with only unrolling in the loop, and a fully optimized version combining both loop pipelining and unrolling. The fully optimized design achieves the lowest latency (0.0680 seconds), a 1.4× speedup over the unoptimized version. It also exhibits more efficient resource utilization across DSPs and BRAM, validating the effectiveness of '`#pragma HLS PIPELINE`' and '`ARRAY_PARTITION`' in achieving high throughput on FPGAs. From the resource usage shown in the TableIII, fully optimization store more data and compute in the LUT and FF.

TABLE III
FPGA RESOURCE UTILIZATION AND EXECUTION TIME FOR DIFFERENT OPTIMIZATION STRATEGIES (DIMENSION = 30).

Configuration	Cycles	LUT	DSP	BRAM(KB)	Time(s)
No Optimization	40,390	219,238	372	136	0.0966
Unroll	34,991	366,741	1,847	154	0.0840
Pipeline + Unroll	28,336	463,953	3,745	135	0.0680

V. DISCUSSION

In Table II, we compare the timing of our FPGA implementation with a 13W mobile GPU, selected for its relevance in edge AI. For low-dimensional F8 Crusader models, the FPGA achieves superior speed due to its efficient parallelism and low-power design. However, beyond dimension 150, performance degrades as limited on-chip memory forces frequent off-chip access, introducing latency. In contrast, the mobile GPU handles high-dimensional models more effectively, thanks to greater on-chip memory and higher memory bandwidth. This reveals a key trade-off: FPGAs are optimal for compact, parallel workloads, while GPUs excel in memory-intensive scenarios.

During synthesis, we observed that Vitis HLS can report hardware resource utilization exceeding the physical limits of the target FPGA board. For example, at higher model dimensions (e.g., dimension 150), the estimated usage of LUTs and BRAMs surpassed the available resources on the device. It appears that Vitis sometimes overestimates utilization due to

conservative analysis during high-level synthesis, particularly when aggressive unrolling or partitioning is used. In practice, actual post-implementation resource usage is often lower after placement, routing, and optimization steps are completed.

Since we could not fully identify the root cause of this discrepancy within the scope of this study, we include this disclaimer: Resource utilization values reported by Vitis HLS for high-dimensional configurations may exceed the target device limits, so resource estimates should be interpreted cautiously, and actual deployability should be confirmed through on-board testing. This highlights a broader need for more reliable resource estimation in HLS workflows and calls for careful validation when designing edge-deployable FPGA systems for mission-critical applications.

VI. Conclusions

This work addresses the urgent need for fast, reliable, and interpretable model recovery (MR) in mission-critical autonomous systems (MCAS), where real-time anomaly detection and response are essential for safety. We proposed MERINDA, an FPGA-accelerated model recovery framework capable of recovering governing dynamics from data using architectures equivalent to PiNODE and EMILY.

By targeting the F8 Crusader benchmark and simulating the model on FPGA, we demonstrated MERINDA's capacity to support sub-second reaction times—far outperforming the 5-second human pilot baseline cited in prior literature [9]. Unlike traditional CPU or GPU implementations, MERINDA leverages HLS-based pipeline and parallelism strategies to significantly reduce execution latency, while maintaining accuracy and interpretability.

Acknowledgement

This work was partially supported by DARPA (AMP, N6600120C4020; FIRE, P000050426), the NSF (FDT-Biotech, 2436801), and the Helmsley Charitable Trust (2-SRA-2017-503-M-B).

References

[1] A. Banerjee and S. Gupta. EMILY: Extracting sparse Model from ImpLicit dYnamics. In C. Coelho, B. Zimmering, M. F. P. Costa, L. L. Ferrás, and O. Niggemann, editors, *Proceedings of the 1st ECAI Workshop on "Machine Learning Meets Differential Equations: From Theory to Applications"*, volume 255 of *Proceedings of Machine Learning Research*, pages 1–11. PMLR, 20 Oct 2024.

[2] A. Banerjee and S. K. Gupta. Recovering implicit physics model under real-world constraints. In *Proceedings of the 26th European Conference on Artificial Intelligence (ECAI 2024)*, volume 392 of *Frontiers in Artificial Intelligence and Applications*, pages 737–744, Hague, Netherlands, 2024. IOS Press.

[3] M. Biloš, J. Sommer, S. S. Rangapuram, T. Januschowski, and S. Günnemann. Neural flows: Efficient alternative to neural ODEs. *Advances in neural information processing systems*, 34:21325–21337, 2021.

[4] L. Cai, J. Wang, L. Yu, B. Yan, Y. Tao, and Y. Yang. Accelerating neural-ode inference on fpgas with two-stage structured pruning and history-based stepsize search. In *Proceedings of the 2023 ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, page 177–183, 2023.

[5] Y. Cao, W. Xiao, J. Jia, D. Wu, and W. Zhou. Cordic-based softmax acceleration method of convolution neural network on FPGA. In *2020 IEEE International Conference on Artificial Intelligence and Information Systems (ICAIIS)*, pages 66–70, 2020.

[6] R. T. Chen, Y. Rubanova, J. Bettencourt, and D. K. Duvenaud. Neural ordinary differential equations. *Advances in neural information processing systems*, 31, 2018.

[7] Z. Chen, Y. Liu, and H. Sun. Physics-informed learning of governing equations from scarce data. *Nature communications*, 12(1):6136, 2021.

[8] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

[9] V. Cutler and S. Revell. Development of a model of 'see and avoid'in parachuting. In *19th International Symposium on Aviation Psychology*, page 101, 2017.

[10] A. Ebrahimi and M. Zandsalimy. Evaluation of FPGA hardware as a new approach for accelerating the numerical solution of CFD problems. *IEEE Access*, 5:9717–9727, 2017.

[11] A. El Bouazzaoui, A. Hadjoudja, O. Mouhib, and N. Cherkaoui. FPGA-based ml adaptive accelerator: A partial reconfiguration approach for optimized ml accelerator utilization. *Array*, 21:100337, 2024.

[12] C. Fan, X. Qin, Y. Xia, A. Zutshi, and J. Deshmukh. Statistical verification of autonomous systems using surrogate models and conformal inference. *arXiv preprint arXiv:2004.00279*, 2020.

[13] T. Gunter, Z. Wang, C. Wang, R. Pang, A. Narayanan, A. Zhang, B. Zhang, C. Chen, C.-C. Chiu, D. Qiu, et al. Apple intelligence foundation language models. *arXiv preprint arXiv:2407.21075*, 2024.

[14] R. Hasani. Liquid Time Constant Networks. https://github.com/raminmh/liquid_time_constant_networks, 2024.

[15] E. Kaiser, J. N. Kutz, and S. L. Brunton. Sparse identification of nonlinear dynamics for model predictive control in the low-data limit. *Proceedings of the Royal Society A*, 474(2219):20180335, 2018.

[16] I. Kuon. *Measuring and navigating the gap between FPGAs and ASICs*. PhD thesis, 2007.

[17] Y. Ma, Y. Cao, S. Vrudhula, and J.-s. Seo. Optimizing the convolution operation to accelerate deep neural networks on FPGA. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 26(7):1354–1367, 2018.

[18] C. Maxfield. *The design warrior's guide to FPGAs: devices, tools and flows*. Elsevier, 2004.

[19] L. Oden. Lessons learned from comparing C-CUDA and Python-Numba for GPU-Computing. In *2020 28th Euromicro international conference on parallel, distributed and network-based processing (PDP)*, pages 216–223. IEEE, 2020.

[20] H. Robinson, S. Pawar, A. Rasheed, and O. San. Physics guided neural networks for modelling of nonlinear dynamics. *Neural Networks*, 154:333–345, 2022.

[21] F. M. Shiri, T. Perumal, N. Mustapha, and R. Mohamed. A comprehensive overview and comparative analysis on deep learning models: CNN, RNN, LSTM, GRU. *arXiv preprint arXiv:2305.17473*, 2023.

[22] A. Sholokhov, Y. Liu, H. Mansour, and S. Nabi. Physics-informed neural ODE (PINODE): embedding physics into models using collocation points. *Scientific Reports*, 13(1):10166, 2023.

[23] I. Stamoulias, M. Möller, R. Miedema, C. Strydis, C. Kachris, and D. Soudris. High-performance hardware accelerators for solving ordinary differential equations. In *Proceedings of the 8th International Symposium on Highly Efficient Accelerators and Reconfigurable Technologies*, pages 1–6, 2017.

[24] F. Tao, H. Zhang, A. Liu, and A. Y. Nee. Digital twin in industry: State-Of-The-Art. *IEEE Transactions on industrial informatics*, 15(4):2405–2415, 2018.

[25] N. Verdière and S. Orange. A systematic approach for doing an a priori identifiability study of dynamical nonlinear models. *Mathematical biosciences*, 308:105–113, 2019.

[26] H. Watanabe and H. Matsutani. Accelerating ODE-Based Neural Networks on Low-Cost FPGAs. In *2021 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 88–95, 2021.

[27] B. Xu, A. Banerjee, M. Urooj, and S. K. S. Gupta. Accelerated digital twin learning for edge ai: A comparison of FPGA and mobile GPU. In *Proceedings of the 68th IEEE International Midwest Symposium on Circuits and Systems (MWSCAS)*, Lansing, MI, USA, 2025. IEEE.

[28] B. Xu, A. Banerjee, M. Urooj, and S. K. S. Gupta. Model Recovery at the Edge under Resource Constraints for Physical AI. In *Proceedings of the 28th European Conference on Artificial Intelligence (ECAI 2025)*, Bologna, Italy, 2025. IOS Press.

[29] J. Zhu, C. Hu, E. Khezri, and M. M. M. Ghazali. Edge intelligence-assisted animation design with large models: a survey. *Journal of Cloud Computing*, 13(1):48, 2024.