STRIDE: Generating FPGA Design Datasets for Surrogate Models from Hierarchical Mathematical Kernels

Angelos Ioannou[1] **Fabien Chaix**[2] **Doru Thom Popovici**[1]
Mario Vega[1] **Tan Nguyen**[1] **Giuseppe Di Guglielmo**[4]
Benjamin Hawks[4] **Aaron Young**[3] **Xiaokun Yang**[5] **Dania Mosuli**[5]
Nhan Tran[4] **John Shalf**[1]

[1]Lawrence Berkeley National Laboratory [2]Foundation for Research and Technology Hellas
[3]Oak Ridge National Laboratory [4]Fermi National Accelerator Laboratory
[5]University of Houston Clear Lake

Abstract

Efficient FPGA accelerators for scientific kernels are still difficult to design because each algorithm, tool, device, and synthesis strategy exposes a large and irregular implementation space. We present STRIDE, an automated workflow for turning hierarchical mathematical kernels into structured FPGA implementation datasets for machine-learning-based hardware design. The STRIDE flow composes kernels such as GEMM, streaming FFTs, and QR factorization from reusable primitives, enumerates implementation parameters at multiple levels of the implemantation process, drives AMD Vivado synthesis and implementation at scale, and records resources, timing, strategies, target-devices, and failure metadata for each generated design. Unlike a flat benchmark collection, each STRIDE datapoint retains the algorithmic and structural provenance needed to train surrogate models for resource prediction, timing estimation, design-space pruning, and active exploration. Our current dataset contains approximately 100,000 fully implemented designs, with data collection ongoing across kernels, FPGA targets, and Vivado strategy pairs, while more kernels can be easily ingested.

1 Introduction

Scientific applications increasingly depend on specialized accelerators for kernels such as Fourier transforms, matrix multiplication and matrix factorizations. These kernels occur in molecular dynamics, materials modeling, beam and plasma simulations, quantum chemistry, and machine-learning workloads, yet building high-quality FPGA implementations still requires substantial hardware expertise. High-level synthesis (HLS) and hardware construction languages improve productivity by raising the programming abstraction [1, 6], but they do not remove the central design-space problem: the same mathematical operation can map to many datapaths, memory organizations, clock targets, FPGA families, and tool strategies.

This problem is becoming a dataset problem. Surrogate models can reduce synthesis-time feedback from hours to seconds, enabling rapid design-space pruning, hardware-aware algorithm search, and agentic design loops. Recent work has shown the value of learned predictors for HLS-generated ML accelerators [3, 5], but similar public-scale datasets are scarce for mathematically structured FPGA kernels. A useful dataset must contain more than final resource counts; it must preserve the hierarchy that explains why a design consumes resources or fails timing.

Submitted to 7th Fast Machine Learning for Science Conference (FastML 2026). Do not distribute.

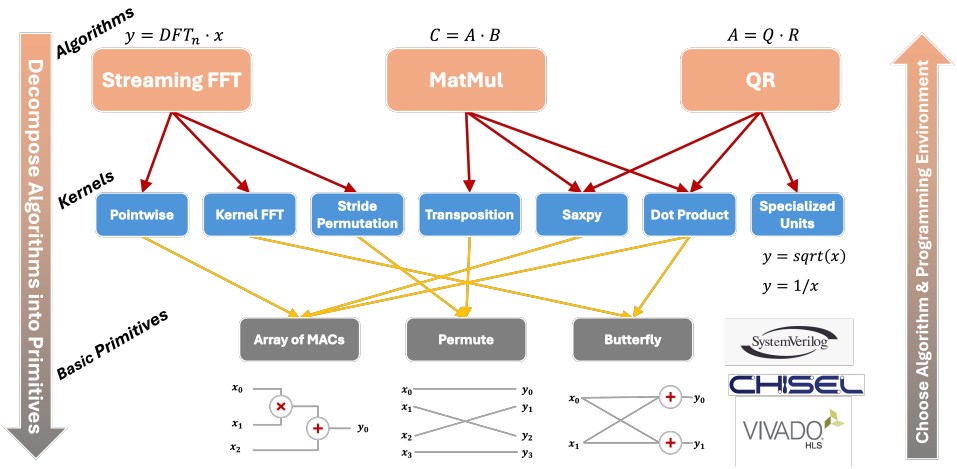


Figure 1: STRIDE builds on a hierarchical methodology in which kernels are decomposed into primitives, implemented through hardware-design front ends, and evaluated by an automated FPGA toolflow. The same structure becomes metadata for surrogate-model training.

We introduce STRIDE, a workflow that converts a hierarchical hardware design methodology [4] into a dataset-generation pipeline for FPGA surrogate modeling. STRIDE makes the following contributions:

- a hierarchical representation for kernels built from reusable primitives, exposing structure that can become model features;
- an automated Vivado-based synthesis and implementation loop that sweeps kernel, representation languages, FPGA target devices, and strategy parameters at cluster scale;
- a metadata schema that records design provenance, implementation settings, numeric parameters, outputs, and structured failure modes;
- a growing dataset of approximately 100,000 implemented designs across GEMM, FFT, QR factorization, and constituent primitives.

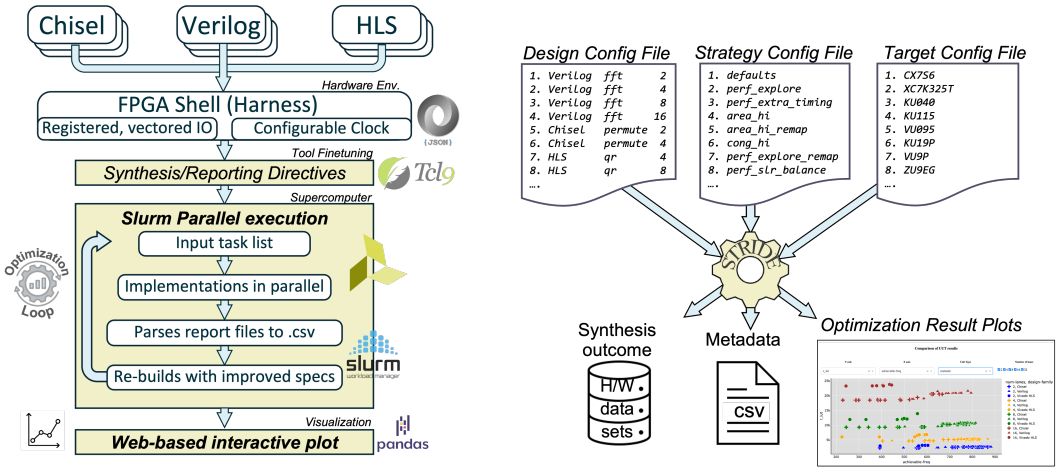
2 Hierarchical Kernel Representation

STRIDE starts from the observation that many scientific kernels can be expressed as compositions of a small number of reusable hardware primitives. As seen in Figure 1, in the architectural side of the workflow, complex algorithms are decomposed top-down into primitive blocks and then composed bottom-up into complete accelerators [4]. For the FFT, Cooley–Tukey decompositions [2] yield a streaming pipeline of permutation stages, butterfly units, and point-wise twiddle-factor multiplication. GEMM can be expressed using arrays of multiply-accumulate units, while QR factorization uses dot products, AXPY-like updates, divisions, and square roots.

This representation is important for machine learning because it makes the design space compositional. A design is not only a row in a CSV file, it is a graph or tree of known operations with typed parameters. For example, a streaming FFT datapoint can record that it contains three permutation units, two butterfly units, and one point-wise multiplication stage, along with streaming width, precision, buffering implementation, twiddle-factor specialization, front-end language, target device, and tool strategy. A surrogate model can therefore learn from both primitive-level behavior and full-kernel interactions, rather than inferring structure indirectly from names or aggregate metrics.

3 Automated FPGA Dataset Construction

For each sampled configuration, as captured in Figure 2, STRIDE generates a complete RTL design that wraps the unit under test, emits tool scripts, invokes AMD Vivado, and parses synthesis and implementation reports. The workflow records resource utilization, timing summaries, achieved frequency, and implementation status, along with implementation stratedy and targeted physical



(a) The framework launches hardware generation jobs on a supercomputer, iteratively evaluating results, across varying paradigms.

(b) Kernels and parameters are ingested to produce fully implemeted designs. Metadata to assist AI training are also delivered, along with human-friendly interactive plots.

Figure 2: STRIDE automation and dataset creation flows.

device. Achievable frequency is derived from the target frequency and worst negative slack (WNS), following the heuristic used during the exploration process:

$$f_a = \frac{f}{1 - \mathrm{WNS} \cdot f}. \tag{1}$$

This gives a consistent scalar target for regression while retaining the raw timing fields for alternative labels. Iterative timing exploration can also be enabled, in which case further and more aggressive target frequencies are explored, offereing a two-folded benefit: the actual maximum frequency is achieved in real hardware, and new useful data points are created, as the implementation process re-optimizes the low level structures to reach timing closure.

The STRIDE implementation leverages on a single-device exploration process and expands it in four directions. First, it treats the synthesis front end as a feature axis, including SystemVerilog, Chisel [1], and Vivado HLS where available. Second, it sweeps design-level parameters such as streaming width, precision, pipeline depth, unroll factor, interface flavor, and primitive-specific choices. Third, it spans multiple AMD FPGA families and device sizes. Fourth, it treats Vivado synthesis and implementation strategy pairs as a dataset dimension, enabling models to learn interactions between the design and the tool optimizer.

The workflow is designed for high-throughput execution on shared HPC systems. We are running it on NERSC's Perlmutter, where STRIDE launches multiple Vivado runs per Slurm job array, reuses synthesized netlists across compatible implementation strategies, and checkpoints intermediate state so that scheduler time limits do not discard completed work. Failed runs are retained rather than filtered out: the post-processing stage records whether a design failed because it exceeded device capacity, failed routing, violated timing, encountered a tool error, or was otherwise incomplete. These labels are useful for classification models and for active-learning policies that should avoid wasting synthesis cycles.

4 Dataset and Metadata

The resulting dataset is intended to serve both as a benchmark and as training material. Each datapoint receives a stable identifier and is accompanied by machine-readable metadata files plus aggregate CSV tables. Table 1 summarizes the main metadata groups.

This metadata allows several supervised-learning tasks. The most direct task is to predict resources and timing from a design specification before implementation. A second task is feasibility prediction: given a target FPGA and clock, classify whether the design is likely to fit and close timing. A

Table 1: Core metadata retained for each STRIDE design point.

Group	Examples
Design identity	kernel, primitive, hierarchy path, parent/child blocks, source generator, configuration hash
Algorithm parameters	matrix or transform size, streaming width, precision, unroll factor, pipeline depth, buffering mode
Tool and target	Vivado version, FPGA family, part number, board, clock target, synthesis and implementation strategies
Measured outputs	LUTs, FFs, DSPs, BRAM/URAM, WNS/TNS, achieved frequency, latency or initiation interval when available
Run status	completed, timing failure, routing failure, device overuse, synthesis failure, parser failure, timeout or resumed state

third task is design-space recommendation, where a model proposes promising parameter regions or Vivado strategies under resource and frequency constraints. Importantly, the metadata does not force these models to treat each design as a flat vector of parameters. Since the hierarchy is explicit, a streaming FFT can be represented as an ordered sequence of pipeline stages, such as permutations, butterfly units, and twiddle-factor multiplications, with parameters attached to each stage. This is a natural input for sequence models that can learn how local stage choices affect end-to-end timing and resource use. The same design can also be represented as a directed graph, where nodes are primitive instances and edges describe dataflow between them. Graph neural networks can then operate directly on the primitive composition rather than on a hand-flattened summary. Finally, primitive-level and full-kernel results can be used in a multi-task learning setup, where a shared model learns from both isolated primitive implementations and complete composed kernels. This does not assume that full-kernel behavior is a simple sum of primitive costs; instead, it gives the model access to both local measurements and whole-design labels, allowing it to learn effects introduced by composition, routing, target device, and tool strategy.

The current working dataset contains approximately 100,000 fully implemented designs across GEMM, streaming FFT, QR factorization, and their constituent primitives, including permutation engines, arrays of MACs, and butterflies. Because data collection is ongoing, we treat this version as a growing corpus rather than as a closed benchmark. For each completed run, STRIDE preserves the raw tool reports, normalized tabular summaries, and hierarchy metadata needed to reconstruct the design point and its implementation outcome. This organization makes it possible to define evaluation splits that ask specific hardware-design questions, such as interpolation within a known kernel and FPGA target, transfer from primitive blocks to composed kernels, transfer across FPGA families, or extrapolation to larger streaming widths and unseen tool strategies. These splits are evaluation protocols, and are included to make the generalization limits of future models measurable.

5 Discussion and Conclusion

STRIDE is not meant to replace FPGA implementation tools; rather, it exposes their behavior as data. The hierarchical decomposition provides a natural bridge between compiler and hardware-design knowledge on one side and machine-learning representations on the other. This creates practical opportunities for the FastML community: learned cost models can guide accelerator search, identify tool-strategy failures early, and provide uncertainty estimates for expensive synthesis campaigns.

These use cases require careful curation. Vivado implementation outcomes can change with constraints and strategies, resource counts must be interpreted in the context of the target FPGA, and failed runs are informative rather than simply disposable. STRIDE therefore retains raw tool settings, target-device context, and structured failure labels alongside normalized summaries, making it possible to study both successful and unsuccessful regions of the hardware design space.

Overall, STRIDE turns hierarchical FPGA design-space exploration into structured training data for surrogate models. The current 100,000-design corpus is a step toward reproducible ML benchmarks for FPGA accelerator design, while new scientific kernels can be introduced, explored, and transformed into training data that preserves the algorithmic provenance needed to learn across kernels, devices, and tool strategies.

References

[1] Jonathan Bachrach, Huy Vo, Brian Richards, Yunsup Lee, Andrew Waterman, Rimas Avižienis, John Wawrzynek, and Krste Asanović. Chisel: Constructing hardware in a scala embedded language. In *DAC Design Automation Conference 2012*, pages 1212–1221, 2012. doi: 10.1145/2228360.2228584.

[2] James Cooley and John Tukey. An algorithm for the machine calculation of complex fourier series. *Mathematics of Computation*, 19(90):297–301, 1965.

[3] B. Hawks et al. wa-hls4ml: A Benchmark and Surrogate Models for hls4ml Resource and Latency Estimation, 2025. URL `https://arxiv.org/abs/2511.05615`.

[4] Doru Thom Popovici, Mario Vega, Angelos Ioannou, Fabien Chaix, Dania Mosuli, Blair Reasoner, Tan Nguyen, Xiaokun Yang, and John Shalf. Towards An Approach to Identify Divergences in Hardware Designs for HPC Workloads, 2025. URL `https://arxiv.org/abs/2509.09774`. arXiv:2509.09774.

[5] M. M. Rahimifar et al. rule4ml: An Open-Source Tool for Resource Utilization and Latency Estimation for ML Models on FPGA, 2024.

[6] Hanchen Ye, Cong Hao, Jianyi Cheng, Hyunmin Jeong, Jack Huang, Stephen Neuendorffer, and Deming Chen. Scalehls: A new scalable high-level synthesis framework on multi-level intermediate representation. In *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 741–755, 2022. doi: 10.1109/HPCA53966.2022.00060.