Pacific Northwest
NATIONAL LABORATORY

PNNL-38967



ENCODE
End-to-end co-design for performance, energy efficiency, and security in AI-enabled computational science
@PNNL

End-to-end co-design for performance, energy efficiency, and security in AI-enabled computational science (ENCODE)

2025 Annual Report

January 2025

Mahantesh Halappanavar
Kevin Barker
Vidya Chhabri (ASU)
Andreas Olofsson (ZeroASIC)
Mark Raugas
Robert Rallo



U.S. DEPARTMENT of ENERGY

Prepared for the U.S. Department of Energy
under Contract DE-AC05-76RL01830

DISCLAIMER

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor Battelle Memorial Institute, nor any of their employees, makes **any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights**. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof, or Battelle Memorial Institute. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

PACIFIC NORTHWEST NATIONAL LABORATORY
operated by
BATTELLE
for the
UNITED STATES DEPARTMENT OF ENERGY
under Contract DE-AC05-76RL01830

Printed in the United States of America

Available to DOE and DOE contractors from
the Office of Scientific and Technical Information,
P.O. Box 62, Oak Ridge, TN 37831-0062
www.osti.gov
ph: (865) 576-8401
fox: (865) 576-5728
email: reports@osti.gov

Available to the public from the National Technical Information Service
5301 Shawnee Rd., Alexandria, VA 22312
ph: (800) 553-NTIS (6847)
or (703) 605-6000
email: info@ntis.gov
Online ordering: http://www.ntis.gov

PNNL-38967

End-to-end co-design for performance, energy efficiency, and security in AI-enabled computational science (ENCODE)

2025 Annual Report

January 2025

Mahantesh Halappanavar
Kevin Barker
Vidya Chhabri (ASU)
Andreas Olofsson (ZeroASIC)
Mark Raugas
Robert Rallo

Prepared for
the U.S. Department of Energy
under Contract DE-AC05-76RL01830

Pacific Northwest National Laboratory
Richland, Washington 99354

Executive Summary

ENCODE is building an open, end-to-end co-design workflow that reduces the time required to translate new algorithms into manufacturable hardware from years to days. The project's central premise is that major gains in performance, energy efficiency, and assurance cannot come from optimizing isolated stages of today's hardware design process. Instead, ENCODE tightly couples compilation, high-level synthesis, physical design, and verification so that decisions made early in the flow are informed by downstream realities like routing congestion, timing closure, memory behavior, and validation costs. In 2025, the first year of the project, the team advanced this vision through two coordinated thrusts: Intelligent and Scalable Hardware Design Automation and Next Generation Testbeds. Together they establish a pipeline in which high-level models can be transformed into optimized accelerators and layouts, while simulation and verification infrastructure ensures the resulting designs are trustworthy and measurable.

Within Thrust 1, ENCODE made concrete progress toward push-button, reproducible hardware generation. A major milestone was integrating core components of the SODA toolchain into ZeroASIC's SiliconCompiler framework to enable an automated path from PyTorch-level kernels to GDSII-eligible layouts with artifact tracking and provenance. This integration not only demonstrated feasibility on representative microkernels, but it also created a scalable foundation for systematic design-space exploration where optimization strategies, tool configurations, and quality-of-results can be compared consistently. In parallel, the project expanded the range of deployable accelerator structures on heterogeneous platforms. The DataFlex effort introduced a software-based MLIR dataflow scheduler that supports cyclic graphs, overcoming constraints that commonly block recurrent and feedback-driven workloads in conventional HLS dataflow models. Evaluations on an FPGA SoC demonstrated substantial throughput gains from graph pipelining with low scheduler overhead, while improving reliability by addressing memory synchronization failure modes.

ENCODE also addressed a persistent barrier to rapid hardware iteration: the loss of observability and intent as software-like descriptions are lowered into complex RTL and physical implementations. In this reporting period, the team developed a compiler-centric approach that embeds modular instrumentation and verification IP at the MLIR level and preserves that intent through LLVM lowering into HLS backends. The result is a proof-of-concept flow where synthesizable assertion checkers and performance counters can be carried through the toolchain as explicit design intent, enabling more systematic debugging, validation, and performance characterization without manual RTL surgery. Complementing this, ENCODE explored how large language models can reduce the expert effort required to craft effective compiler transformation schedules. An LLM-in-the-loop workflow was implemented to generate MLIR Transform-dialect schedules from kernels and hardware constraints, then evaluate outcomes through HLS and metric extraction. Early results on a streaming scientific workload showed multi-fold speedups across FPGA- and ASIC-targeted flows, while clarifying the need for stronger automated correctness checks and guardrails before these techniques can be widely deployed.

Year 2025 also produced a strong demonstration of ENCODE's co-design philosophy in the context of transformer inference. The ELiTeFormer effort paired algorithmic changes, including hybrid linear and sliding-window attention with ultra-low-precision ternary projections, with a tailored accelerator micro-architecture that maps ternary operations efficiently onto FPGA resources. The combined approach yielded large compressions in weights and KV cache requirements and showed meaningful latency and energy improvements in deployment experiments, reinforcing the project's thesis that joint optimization across model structure and

hardware implementation can unlock superior system-level trade-offs. To make these advances comparable and reproducible across tools and targets, ENCODE also advanced SODA-Benchmarks, an open, extensible suite that standardizes end-to-end evaluation from popular ML front ends through MLIR, HLS, and physical design. This suite has already served as shared infrastructure for multiple FY2025 efforts and will underpin broader benchmarking and regression testing going forward.

A defining feature of the first year was the project's movement from methodology toward silicon validation. ENCODE completed two MPW tapeouts: one in SkyWater 130 nm using an open-source backend flow and the SKY130 PDK, and another in GF180 using the open PDK combined with proprietary synthesis tools to enable cross-tool and cross-node comparisons. These tapeouts anchor ENCODE's next stage, where measured post-silicon results will be used to assess prediction fidelity and drive improvements in synthesis and physical optimization. Supporting that transition, ENCODE strengthened physical-awareness across the flow, including floorplan-informed interconnect delay estimation within Bambu scheduling, functional block recharacterization to improve modeling accuracy, LLM-assisted timing-optimization sequencing in OpenROAD, and a parallel global routing strategy aimed at scaling routing workloads while preserving solution quality.

Thrust 2 focused on building the evaluation backbone needed to make rapid hardware generation credible for mission workloads. The first-year activities established the early stages of a cycle-accurate memory co-simulation capability by integrating Ramulator2 into Bambu to replace fixed-time memory models, enabling more realistic estimation of memory-system impacts and supporting future exploration of memory technology trade-offs. The thrust also demonstrated an LLM-orchestrated, multi-agent RTL testing workflow centered on Verilator, establishing a foundation for more automated, coverage-driven verification. In addition, the team advanced OpenROAD-oriented verification directions and built an OpenROAD-integrated UCIe simulation capability to begin capturing interconnect performance and power-relevant metrics, setting up a path toward richer models and workload-traced simulations.

Our goal for next year focus on the transition from integrated prototypes to scalable, hardened workflows validated against real silicon. Planned work emphasizes broader configurability and cloud-enabled execution in the end-to-end flow, stronger automated validation of LLM-generated transformations, deeper physical-aware co-optimization that accounts for routing and timing pressures, and systematic correlation between pre-silicon predictions and post-silicon measurements. With expected chip deliveries in early 2026 and a planned next-generation tapeout incorporating new ENCODE capabilities, the project is set to close the loop between automated design, verification, and measured performance, accelerating the delivery of efficient, trustworthy accelerators for AI-enabled computational science.

Contents

Executive Summary ii
1.0 Project Overview and Introduction 1
2.0 Thrust 1: Intelligent and Scalable Hardware Design Automation 3
2.1 End-to-end application-specific chiplet compiler 3
2.1.1 Zero-ASIC's Silicon Compiler Integration 3
2.1.2 SW-based pipelining of generated accelerators 4
2.1.3 IP Integration in MLIR for instrumentation and verification 5
2.1.4 LLM-driven optimization schedules for better generated HW outcomes 6
2.1.5 Efficient transformer model and micro-architecture design 8
2.1.6 Open-source Software Defined Accelerators Benchmark 9
2.1.7 Physical-aware Logic Resynthesis (MLCAD 2025 Contest) 10
2.1.8 Physical awareness in Bambu High-level Synthesis (HLS) 11
2.1.9 Tapeouts 13
2.2 Optimization for Efficient Hardware Design 14
2.2.1 Hardware Software Partitioning 14
2.2.2 Global Routing 18
2.2.3 Timing Optimization using LLMs for the OpenROAD Physical Design Tool 21
2.2.4 Resource Binding Problem 22
3.0 Thrust 2: Next Generation Testbeds 23
3.1 Testbed Activities 23
3.1.1 Cycle-Accurate Memory Co-simulation for Synthesized Accelerators 23
3.1.2 LLM-Based Agentic Workflow for Hardware Testing and Verification 24
3.1.3 LLMs for OpenROAD Verification 25
3.1.4 UCIe Simulation for Thermal and Power 25

Figures

Figure 1. Flow diagram of floorplan-aware wirelength and interconnect delay estimation in Bambu 11
Figure 2. Simulated annealing-based floorplanning information is used to build a lookup table for interconnect delay estimation inside the Bambu scheduler. 12
Figure 3. Make span comparison across different heuristics and objective functions 15
Figure 4. A pathwidth 5 subgraph of the RESNET computational task graph constructed using SODA. Each node represents a neural network operation, such as

convolution, multiplication, or transposition. This subgraph, or one similar, appears multiple times in the RESNET task graph. 16
Figure 5. A comparison of running time in seconds between the dynamic programming (DP) and linear programming (LP) approaches between two sets of randomly generated graphs with bounded pathwidth under two different weighting schemes. The graphs are grouped by size (N) and pathwidth (k). We see that the dynamic programming approach is better on every instance, with an improvement of up to 200x for some graphs in group 2. A time limit of 1800s was set for both algorithms. 17
Figure 6. Scalability of LockRoute showing time speedup factor (y×) with increasing number of threads. The difficult benchmark newblue3 lies separated from the rest. 19
Figure 7. LLM-based flow to generate custom transformation sequences to optimize violating paths. 21
Figure 8. An overview of the flow of the UCIe simulation tool..... 26

Tables

Table 1. Improvement in post-route PPA metrics due to physical awareness..... 12
Table 2. Benchmark applications with details about the generated task graph..... 14
Table 3. Overflow results for various strategies. MO and TO are max and total overflows. Locks and No Locks strategies ran using 32 threads, whereas Sequential ran single-threaded. 20
Table 4. Quality and runtime results comparing LockRoute against NCTU-GR 2.0 on overflow-free cases. WL and # VIA give the wirelength and the number of vias in units of 105 CPU gives runtime in seconds. 20
Table 5. Delay (TNS/WNS) improvement using LLM-based sequence generation over the default sequence and simulated annealing-based sequence generation. 22

1.0 Project Overview and Introduction

ENCODE is an ASCR-funded collaborative effort led by Pacific Northwest National Laboratory (PNNL), with Arizona State University (ASU) and ZeroASIC Inc., to advance open, end-to-end co-design methods for AI-enabled computational science. The project is motivated by a persistent mismatch between the pace of algorithm innovation and the pace of hardware realization. In many DOE-relevant scientific workflows, algorithmic advances emerge rapidly, yet translating those advances into specialized, high-efficiency hardware implementations remains slow, expensive, and difficult to reproduce. Traditional design practices often require lengthy iteration cycles across disconnected tool stages, significant expert intervention, and repeated hand-tuning to achieve acceptable power, performance, and area (PPA). ENCODE addresses this gap by developing interoperable open-source methodologies and tools that reduce the time from algorithm to physical layout from years to days, while improving design quality and strengthening confidence in functional correctness.

The core strategy of ENCODE is to integrate compilation, synthesis, physical design, and verification into a tightly coupled optimization loop. Rather than treating hardware synthesis and layout as downstream steps that simply "consume" earlier design choices, ENCODE's approach makes physical and verification considerations first-class inputs to earlier compilation and high-level synthesis decisions. This integration is essential for modern accelerators, where achievable performance and energy efficiency are increasingly dominated by effects such as memory movement, interconnect delay, congestion, and timing closure. ENCODE therefore emphasizes new abstractions and introspection mechanisms that expose hardware-relevant information throughout the stack, enabling automated transformations, measurable trade-offs, and reproducible results. The project combines these abstractions with scalable co-optimization algorithms, including combinatorial optimization techniques and learning-assisted methods, to navigate large design spaces efficiently and to guide the toolchain toward high-quality implementations.

The project is organized into two complementary thrusts that together deliver a complete "design-to-evaluation" pipeline.

Thrust 1: Intelligent and Scalable Hardware Design Automation develops the automated tool flow that translates high-level program and model descriptions into hardware realizations suitable for fabrication or prototyping. This thrust focuses on compiler- and HLS-centered approaches that can generate accelerators from widely used ML and scientific programming environments, while enabling systematic optimization across multiple objectives. A key emphasis is end-to-end automation that remains modular and open, so that individual tools and passes can be composed, replaced, and evaluated without breaking reproducibility. Thrust 1 advances research in applied mathematics and computer science, including design-space exploration, hardware/software partitioning, routing and timing optimization, and physical-awareness techniques that bring layout realities into earlier stages of synthesis. The thrust also prioritizes practical integration paths that can produce manufacturable outputs, enabling real tapeouts and measurable outcomes rather than remaining solely at the level of simulated designs.

Thrust 2: Next Generation Testbeds develops the simulation, verification, and measurement infrastructure required to evaluate ENCODE-generated accelerators credibly and efficiently. The project's goal of compressing design cycles is only valuable if results are trustworthy, comparable, and grounded in quantitative evidence. Thrust 2 therefore builds capabilities that connect pre-silicon modeling and verification to post-silicon testing and validation. This includes cycle-accurate performance modeling for key bottlenecks such as memory behavior, automated test generation

and coverage-driven verification to reduce verification time, and integration of verification tooling into the same open design flow used by Thrust 1. As the project transitions from year-one prototype generation to year-two silicon evaluation, Thrust 2 expands from simulation and verification toward physical testbed installation and operation, including test harnesses and measurement workflows for packaged prototypes. A central objective is correlation: predictions made during compilation, synthesis, and simulation must be compared to measured hardware results, and discrepancies must feed back into improved models, constraints, and design decisions.

Across both thrusts, ENCODE places strong emphasis on openness, interoperability, and reproducibility. The project leverages and extends open-source infrastructure spanning MLIR/LLVM-based compilation, high-level synthesis, and RTL-to-GDSII physical design. It also develops shared benchmark and evaluation artifacts so that improvements can be measured consistently over time and compared across tools, targets, and optimization strategies. This benchmarking foundation supports rigorous assessment of PPA, throughput, latency, and efficiency trade-offs, and it accelerates progress by enabling rapid regression testing as tools evolve.

FY2025 represents the first year of ENCODE execution and focused on establishing integrated foundations and early demonstrations across the two thrusts. The project advanced an end-to-end pipeline capable of lowering representative high-level workloads toward manufacturable implementations, introduced new mechanisms for scheduling and deploying complex dataflow graphs on heterogeneous platforms, and developed instrumentation strategies that preserve observability through aggressive compilation and synthesis transformations. In parallel, ENCODE began building the testbed infrastructure needed to validate those designs with higher-fidelity simulation and more automated verification. The year also initiated a transition toward silicon-grounded evaluation through completed tapeouts, which will enable deeper validation and model calibration in FY2026.

The remainder of this report summarizes progress until the end of 2025 and outlines planned activities for FY2026. The narrative is organized by thrust and by key technical tasks within each thrust. For each task, we describe the motivation, the technical approach, key accomplishments and preliminary results from the reporting period, and the next steps that will move ENCODE toward scalable, reproducible, and physically grounded end-to-end co-design for AI-enabled computational science.

2.0 Thrust 1: Intelligent and Scalable Hardware Design Automation

2.1 End-to-end application-specific chiplet compiler

2.1.1 Zero-ASIC's Silicon Compiler Integration

The rapid evolution of algorithms for machine learning and scientific computing places increasing demand on application-specific accelerators that must meet strict performance, area, and power constraints. Manual hardware design flows are too slow to match this pace; consequently, automated, compiler-driven methodologies that translate high-level software descriptions into manufacturable ASIC implementations are essential. The work described here integrates the SODA toolchain — our open, modular hardware compiler — into the SiliconCompiler framework from Zero-ASIC, to enable reproducible, automated end-to-end flows from PyTorch-level descriptions to GDSII-eligible layouts, thereby lowering the barrier to ASIC generation for domain-specific accelerators.

During FY2025 the team completed the initial integration of core SODA components (notably SODA-OPT and Bambu) into SiliconCompiler, developing plugins and orchestration logic to permit these tools to be invoked within SiliconCompiler's schema-driven flows. Work focused on implementing a complete PyTorch-to-GDSII pipeline for representative microkernel examples, exposing configuration knobs to allow the existing soda-opt preconfigured optimization pipeline to run under SiliconCompiler's control, and validating artifact tracking and reproducibility. The integration also took advantage of SiliconCompiler's updated module search and schema features, simplifying support for tools with complex configuration spaces and enabling more robust flowgraph-based parallel execution and metric capture.

As a result of these efforts, the integrated flow now supports automated translation of selected PyTorch kernels through SODA-backed synthesis and downstream RTL-to-GDSII steps managed by SiliconCompiler, with reproducible artifact tracking and provenance. This demonstrates a working end-to-end path from high-level model descriptions to silicon-ready data, validates the use of plugin-based tool integration within SiliconCompiler for SODA components, and establishes baseline examples and metrics that can be used to compare optimization strategies and toolchain configurations.

Planned work for FY2026 emphasizes finer-grained optimization and broader configurability:

- updating plugins to leverage SiliconCompiler's new transform-dialect capabilities to support more precise optimization pipelines
- adding richer configuration surfaces for Bambu and soda-opt
- integrating Zero-ASIC's cloud execution capabilities and PDK-supported libraries to enable full ASIC generation on supported processes

These steps aim to move the integration from proof-of-concept microkernels toward a scalable, reproducible path for generating ASIC accelerators using the Zero-ASIC infrastructure.

2.1.2 SW-based pipelining of generated accelerators

Deploying neural networks in hybrid platforms with Systems-on-Chip (SoCs) and generated accelerators presents significant challenges. Commercial tools like Vitis High-Level Synthesis (HLS) impose restrictions on dataflow graphs, such as prohibiting cyclic structures, which limits the deployment of Recurrent Neural Networks (RNNs). Additionally, these tools face memory access constraints that can lead to synchronization failures and data corruption. These limitations hinder the performance and flexibility of generated accelerators, particularly for latency-sensitive applications like autonomous vehicles and edge anomaly detection.

To address these challenges, DataFlex introduces a novel software-based dataflow scheduler. Unlike hardware-based schedulers, DataFlex supports cyclic graphs, enabling RNN deployment, and overcomes memory constraints through innovative techniques like ping-pong buffering. By leveraging the Multi-Level Intermediate Representation (MLIR) framework, DataFlex provides a platform-agnostic solution that integrates seamlessly with OpenMP and OpenCL for flexible scheduling.

In FY2025, the development of DataFlex focused on creating a robust framework for software-based dataflow scheduling. A new MLIR dialect was designed to represent both cyclic and acyclic graphs. This dialect includes operations for synchronization, feedback, and memory management, enabling explicit modeling of dependencies and delays. The scheduler dynamically manages execution and synchronization of dataflow nodes, leveraging OpenMP for CPU tasks and OpenCL for FPGA tasks. This approach bypasses the limitations of Vitis HLS by implementing synchronization in software. DataFlex was evaluated on an AlphaData ADM-PA101 SoC, demonstrating its ability to achieve a 4.86× average speedup through graph pipelining across four neural network models. The scheduler overhead was measured to be less than 5.6%, becoming negligible for realistic workloads. Additionally, DataFlex extended MLIR with a dataflow dialect and runtime extensions, enabling seamless transformation of computational graphs into scheduled dataflow programs.

The activities in FY2025 resulted in several significant outcomes. DataFlex successfully demonstrated the ability to schedule and execute cyclic graphs on FPGA SoCs, a first in the field. The framework achieved substantial speedups through pipelined execution, with benchmarks showing up to 6.45× improvement in throughput for certain models. By decoupling scheduling from hardware, DataFlex enables deployment across heterogeneous platforms, including CPUs and FPGAs. The use of software-based synchronization mechanisms, such as ping-pong buffering, eliminated issues like external memory corruption, ensuring reliable execution.

Building on the successes of FY2025, the planned activities for FY2026 include extending DataFlex to support additional hardware targets, such as custom ASICs. Mechanisms for hybrid CPU-FPGA and ASIC systems with SoC+custom hardware co-execution will be developed to further enhance performance and flexibility. The integration of design space exploration (DSE) techniques will optimize individual node implementations and improve overall system efficiency. Efforts will also focus on improving the user interface and documentation to facilitate adoption by a broader audience. Finally, more extensive evaluations with diverse neural network architectures will be conducted to validate the framework's scalability and robustness.

DataFlex represents a significant advancement in the deployment of neural networks on SoCs with generated accelerators, addressing long-standing limitations and paving the way for more efficient and flexible hardware accelerators.

2.1.3 IP Integration in MLIR for instrumentation and verification

High-level synthesis (HLS) transforms software-like descriptions into deeply restructured hardware implementations, often obscuring source-level intent and making post-synthesis validation and performance analysis difficult. The resultant RTL can be orders of magnitude larger and more complex than the originating C/C++ or MLIR, which complicates debug, verification, and performance characterization workflows for both FPGA and ASIC targets. To mitigate this, the task integrates modular instrumentation IP directly into the MLIR-to-HLS flow so that verification hooks and performance monitors are carried as explicit intent through the compiler stack. By embedding instrumentation at the MLIR level and preserving that metadata through LLVM IR to the HLS backend, the flow restores a level of source-aware observability that remains robust under aggressive optimizations and backend transformations.

During FY2025 the team developed a compiler-centric approach that:

1. Defines a small set of instrumentation primitives and templated IP recipes.
2. Implements MLIR passes to mark and instantiate instrumentation.
3. Extends the LLVM lowering to export compact metadata that drives HLS tool integration.

The current work included new passes that annotate operations and regions with instrumentation intent (for example, assertion insertion points and region boundaries for counters). A proof-of-concept MLIR pass was implemented to annotate operations and to instantiate parameterized recipes that describe assertion checkers and latency/throughput counters. On the backend side, the LLVM lowering preserves this intent via well-structured metadata, and the HLS integration layer consumes those fragments to instantiate the corresponding hardware modules, add necessary interconnect and control signals, and update HLS constraint files or XML integration manifests. The team also designed a compact metadata protocol and a small library of hardware templates that support single-cycle control commands and multi-site monitoring and evaluated trade-offs between enabling instrumentation for simulation versus synthesis.

The project produced a working proof-of-concept demonstrating that instrumentation intent expressed at the MLIR level can be reliably propagated to an HLS backend and materialized as synthesizable IP. Implemented modules include transparent assertion checkers, which signal mismatches without affecting the functional data path timing, and configurable performance counters that measure visits to regions and enable simple latency and throughput calculations. The instrumentation recipes support single-cycle command interfaces so that control can be exerted from surrounding logic with minimal latency, and multi-site monitoring that aggregates events from many points in a design. Early resource and runtime analysis quantified the cost of these modules: assertion checkers can be made nearly zero-overhead in common cases by avoiding additional pipeline stages, whereas active performance counters impose synthesized area and potential timing overheads; these costs can be suppressed by selective enabling and by restricting counters to non-critical-region entry points. The team also established an integration pattern for HLS constraint and XML manifests, enabling handoff to vendor tools with minimal manual edits.

Planned work will expand the recipe library, harden the metadata protocol, and close several practical gaps observed during the proof-of-concept. First, the MLIR pass suite will be extended to provide more expressive instrumentation directives (conditional assertions, sampled traces, and hierarchical counters) and to provide automated placement heuristics that avoid critical-path

intervention. Second, the LLVM-to-HLS metadata emission will be made more robust and standardized, including richer constraint generation, naming conventions, and explicit hooks for both FPGA and ASIC HLS targets. Third, the IP templates will be optimized for area and timing through RTL-level improvements and optional lightweight trace compression for high-frequency events. The team will also develop a test harness and integration examples demonstrating how to use the instrumentation for in-situ verification, performance benchmarking, and post-silicon signal tracing. Finally, the project will produce guidance on selective enabling strategies, an assessment of common cost/benefit trade-offs, and a set of recommended practices for integrating instrumentation into production HLS flows.

The design emphasizes modularity and portability: instrumentation recipes are parameterized so that they can be instantiated with different widths, enable semantics, and aggregation strategies, and the metadata protocol is intentionally minimal to avoid fragility in the LLVM/HLS handoff. The flow discriminates between simulation-only instrumentation (which may rely on richer tracing that is impractical to synthesize) and synthesis-safe instrumentation (which must obey timing and resource constraints). Mapping higher-level functions to HLS finite-state machines is supported by attaching instrumentation to region entry/exit points and to function-level call boundaries; the lowering phase translates those annotations into FSM-state-anchored checks or counter hooks. Where appropriate, placeholders for integration figures and performance tables have been left so results and quantitative numbers may be added as experiments complete.

This task advances the goal of bridging software-level programmability with hardware-level observability. By carrying instrumentation intent through MLIR and LLVM to the HLS backend, the approach enables systematic, selective in-situ validation and profiling that works for both FPGA and ASIC-oriented flows, improving both the debugging experience and the fidelity of performance characterization for synthesized accelerators.

2.1.4 LLM-driven optimization schedules for better generated HW outcomes

Mapping high-performance computation kernels to hardware accelerators remains a time-consuming and expertise-intensive step in the development of low-latency, high-throughput scientific workflows. Many experiments that generate streaming or high-volume data—such as Electron Energy Loss Spectroscopy (EELS) microscopes—require near real-time analysis to inform experimental decisions. Software-only solutions on CPUs and GPUs can fail to meet this latency and throughput demands, motivating the use of custom accelerators implemented with High-Level Synthesis (HLS) on FPGAs or by targeting ASIC flows. However, producing high-quality HLS designs typically depends on expert-crafted transformation schedules and iterative performance tuning. This task investigates a complementary approach that leverages Large Language Models (LLMs) to generate MLIR Transform-dialect schedules that guide compiler transformations toward hardware-efficient implementations, thereby reducing manual effort and time-to-insight while preserving expert control over correctness and constraints.

During FY2025 the work focused on integrating LLM-driven schedule synthesis into an MLIR-centered compilation flow and validating the approach on realistic scientific workloads. The team developed an LLM-in-the-loop generator that ingests original MLIR kernels together with concise hardware specifications (memory sizes and ports, initiation intervals, and available parallelism) and user objectives (e.g., latency or area constraints) and emits Transform-dialect schedules for the compiler to apply. Emphasis was placed on engineering the LLM context and prompts: providing clear optimization objectives and constraints, curating example transforms and Transform-dialect primitives and enforcing a reason-first generation pattern, so the model produces structured, verifiable output. The implementation integrated with existing tools in the

SODA/MLIR ecosystem to apply generated schedules, run HLS flows, and collect performance and resource metrics. The team iteratively refined prompting strategies and the examples used as context to improve the syntactic validity and practical effectiveness of generated schedules. A primary evaluation case used an EELS autoencoder kernel representative of streaming scientific workloads; the pipeline captured and compared baseline MLIR-derived hardware kernels against kernels produced after applying LLM-suggested transformations, across both FPGA and ASIC-targeted HLS workflows.

The FY2025 activities demonstrated that LLM-driven Transform-dialect schedules can produce substantial performance improvements with modest human oversight. On the EELS autoencoder case study the approach achieved multi-fold runtime speedups relative to the baseline MLIR kernel: measured speedups were approximately 2.3x on an FPGA-targeted flow and 2.66x for an ASIC-targeted flow in the reported experiments, showing consistent gains across target technologies. Beyond raw performance, the work established a reproducible pattern for constructing LLM contexts that balance clarity, hardware constraints, and transform primitives, reducing the incidence of syntactic errors in generated schedules. The integration validated an end-to-end evaluation loop—generation, schedule application, HLS compilation, and metric extraction—enabling rapid iteration on prompts and example schedules. The team also identified critical limitations: LLMs sometimes produce transformations that are syntactically valid but suboptimal, and there remains a nontrivial need for human verification to ensure semantic correctness and alignment with target-specific constraints. These outcomes illustrate that LLMs can substantially lower the barrier to obtaining performant hardware kernels, while also clarifying the remaining role of human experts and verification tooling.

Planned work for FY2026 focuses on improving robustness, expanding evaluation breadth, and closing the verification loop. First, the effort will invest in automated validation techniques that check semantic correctness of Transform-dialect schedules and detect harmful transformations before HLS, including lightweight static analyses and regression tests against golden kernels. Second, the prompt and example corpus will be broadened to cover a wider variety of kernels and hardware constraint patterns, increasing generalization across scientific workloads and across FPGA and ASIC target tradeoffs. Third, the team will pursue tighter co-optimization with physical design considerations so that generated schedules account for resource fragmentation, memory banking, and routing pressures that impact ASIC quality-of-results. Fourth, a more systematic benchmarking effort will be conducted by expanding the set of evaluated models beyond the EELS encoder to include additional streaming and ML kernels, and by quantifying tradeoffs in latency, throughput, area, and power. Finally, the work will explore human-in-the-loop interfaces that let domain experts provide high-level guidance and verify candidate schedules more efficiently. Collectively, these activities aim to transition the prototype capability toward a robust, reusable toolchain that reduces time-to-hardware for domain scientists while ensuring correctness and predictable design tradeoffs.

LLM-driven Transform-dialect schedule synthesis has emerged as a promising approach to reduce manual effort in producing high-performance HLS designs for both FPGA and ASIC targets. FY2025 established a working integration and showed concrete performance improvements on a representative scientific workload. FY2026 will emphasize verification, broader benchmarking, and tighter hardware-aware co-optimization to make the approach practical for a wider set of kernels and to ensure reliable outcomes in production-like flows.

2.1.5 Efficient transformer model and micro-architecture design

This task addresses the challenge of making large transformer-based language models practical for latency- and energy-sensitive inference by co-designing algorithmic model changes together with a reconfigurable-hardware implementation. The work centers on ELiTeFormer, a Transformer architecture that integrates hybrid linear attention with ultra-low-precision (ternary) linear projections, and on a bespoke micro-architecture for accelerating the resulting ternary operations on accelerators generated with HLS, currently targeting reconfigurable devices such as FPGAs. The motivation is twofold: to reduce both the memory footprint and the arithmetic cost of generation-time inference (especially for long contexts), and to align algorithmic structure with operations that map efficiently to FPGA datapaths and on-chip memory. Where conventional efforts typically optimize attention or the feed-forward network (FFN) in isolation, this task demonstrates that simultaneous optimization yields substantially better system-level trade-offs in model compression, cache size, latency, and energy efficiency.

During FY2025 the team pursued a sequence of interlinked activities spanning model design, adaptation, training, simulation, and hardware synthesis. On the algorithmic side, we adapted an 8B LLaMA-3–derived BitNet b1.58 checkpoint and replaced classical attention with a hybrid linear-attention + sliding-window mechanism; the attention block was trained and distilled using an adapted LoLCATs workflow to preserve accuracy while enabling a static, a priori calculable inference profile. Simultaneously, we employed the BitNet b1.58 ternary quantization pattern in attention projections and FFNs so that most linear projections operate with weights in {−1, 0, 1}. On the hardware side, we defined a pipelined per-layer dataflow suitable for generation-only workloads, specified AXI-based memory interfaces (512-bit frames and 8-bit packed weight dataframes), and designed the ELTF PE micro-architecture that replaces ternary multiplications with indexed bitmasking and reduction trees. The implementation work included writing C-level kernels and HLS-compatible variants, simulating RTL via Bambu HLS and Verilator, synthesizing with Vitis HLS and Vivado, and deploying a scaled-down accelerator targeting a Xilinx VCK5000 board. The team also ran focused training experiments (sliding-window size sweeps, ablations on LoRA-style fine-tuning, and small-scale distillation runs) and bench-marked model quality on standard evaluation suites.

The key outcomes achieved in FY2025 validate the HW/SW co-design hypothesis and quantify practical benefits. ELiTeFormer attains approximately 10× model weight compression and 12.8× key-value (KV) cache compression relative to the LLaMA-3 baseline while maintaining competitive task accuracy (for example, 31.9% on MMLU, within ~3.0% of the BitNet b1.58 checkpoint from which it was derived). Block-level HLS simulations show substantial compute reductions: a typical BitNet-style ternary linear projection accelerated by the ELTF PE attains an average 9.6× cycle reduction for FFN-like projections and up to a 4.4× cycle reduction for the attention core compared to quadratic references. End-to-end experiments and deployments further substantiate the system-level impact: a scaled-down ELiTeFormer instance synthesized for a VCK5000 demonstrates up to 3.9× lower latency and roughly 3.2× better energy efficiency than a similarly scaled LLaMA-3 baseline on an NVIDIA A100 at long context lengths. Resource analyses confirm that the design avoids or greatly reduces DSP usage for ternary projections by substituting bitwise mask operations, and that the attention stage is the dominant consumer of LUT and BRAM resources in the pipelined design.

The report also documents practical engineering findings that shaped these results. Hybrid linear combined with sliding-window attention provides a static memory profile (suitable for FPGA static compilation) and, when combined with ternary projection compression, enables much smaller attention caches at large contexts. Distillation with smaller training windows than evaluation

windows improve reconstruction and stability; conversely, second-stage low-rank fine-tuning (LoRA) degraded quality for aggressively quantized models in our experiments. On the implementation side, packing weights into 8-bit dataframes and reading large bursts over 512-bit AXI frames improved off-chip bandwidth efficiency in HLS flows, while adding a mini-batch partitioning dimension (`mB`) to non-attention linear projections boosted weight reuse and throughput for multi-sample processing.

Planned FY2026 activities build on these results and address scaling, robustness, and broader applicability. First, the team intends to pursue fuller-scale deployments on HBM-equipped FPGA platforms (e.g., Alveo U280-class hardware) to realize the throughput potential indicated by DDR→HBM bandwidth scaling and to measure performance at the full 8B parameter scale. Second, we plan to increase the fidelity of pretraining / fine-tuning used for ternary checkpoints (more tokens and further compute) to close the small remaining quality gaps on hard benchmarks and to investigate hybrid quantization schemes that trade a modest accuracy improvement for tiny increases in weight storage. Third, we will refine the ELTF PE and memory banking strategies to increase head-level parallelism without overburdening off-chip bandwidth, explore more aggressive dataframe encodings (e.g., small LUT decoding schemes), and automate mapping flows through high-level HLS templates to ease retargeting. Finally, we will evaluate heterogeneous deployment strategies that delegate prefilling to a different accelerator while reserving the FPGA for generation, extend benchmark coverage, and prepare an open-source release and reproducibility package to allow independent verification and further community-driven optimization.

This task produced a concrete, reproducible demonstration that co-designing efficient attention mechanisms and ultra-low-precision linear projections yields favorable Pareto trade-offs for long-context LLM inference on reconfigurable hardware. The ELiTeFormer artifacts—model recipes, attention-distillation findings, ELTF PE micro-architecture, and an HLS-friendly accelerator implementation—provide a practical foundation for FY2026 efforts to scale, harden, and broaden the technology for production-grade LLM serving in constrained and energy-sensitive environments.

2.1.6 Open-source Software Defined Accelerators Benchmark

The SODA-Benchmarks project provides a curated, extensible benchmark suite designed to evaluate and compare tools across the SODA (Software-Defined Accelerators) toolchain. The benchmark repository consolidates representative machine learning workloads and microkernels and standardizes dataflow from high-level frameworks through intermediate representations to hardware implementation. By supporting inputs from PyTorch and TensorFlow and lowering models to MLIR and LLVM IR, SODA-Benchmarks enables consistent end-to-end evaluation of compiler optimizations, HLS flows, synthesis tools, and physical design toolchains, facilitating apples-to-apples comparisons between FPGA and ASIC generation flows.

During FY2025 the project prioritized making the benchmark suite broadly usable and reproducible within open-source toolchains. Key activities included open-sourcing the repository and integrating example flows that demonstrate a complete hardware generation pipeline: PyTorch models are lowered to MLIR, optimized with `soda-opt`, and passed to HLS tools such as Bambu; the resulting RTL is then taken through place-and-route with OpenROAD to generate GDSII-level layouts.

A set of canonical workloads was implemented to cover common model classes and deployment scenarios:

- For PyTorch, image-classification architectures (MobileNetV3-small, ResNet-18, SqueezeNet).
- For TensorFlow, an AutoEncoder tailored to EELS data.
- A collection of TFLite-quantized models for edge tasks including anomaly detection, image classification, keyword spotting, and visual-wake-words.

The repository also introduced a plugin mechanism for `soda-opt` that simplifies development and testing of custom compiler extensions and passes, enabling rapid experimentation with optimization and IP-integration strategies.

Work completed in FY2025 produced several notable outcomes:

- The benchmark flows were validated end-to-end for most of the models, demonstrating that the pipeline—from high-level model to synthesized RTL and GDSII—produces functional designs with measurable performance and area metrics.
- The suite has been adopted internally by multiple FY2025 efforts to generate hardware/software partitioning graphs and to exercise partitioning algorithms, providing empirical data for algorithm evaluation.
- The plugin architecture reduced development friction for custom passes and allowed teams to prototype IP-integration and instrumentation strategies that support both pre- and post-silicon verification workflows.
- The benchmarks established a set of reproducible testcases that facilitate cross-tool comparison and community contributions.

Planned activities for FY2026 build on this foundation to broaden coverage and increase utility for ASIC-focused flows. The team intends to translate the PolyBenchC microkernel suite into PyTorch front ends so that a richer set of computational kernels is available for compiler and HLS evaluation. Work will continue expanding the plugin ecosystem to enable automated IP insertion for performance and verification features and to support more tightly coupled hardware/software co-design experiments. Metrics collection and reporting will be standardized further (including placeholders for figures and tables in the report) to provide clear baselines for area, timing, power, and energy efficiency across both FPGA and ASIC targets. Anticipated challenges include ensuring toolchain portability across evolving MLIR/HLS backends and scaling the physical design steps to more complex designs; mitigation strategies include automated CI for benchmark validation, tighter test harnesses for end-to-end regression, and modularization of the design flow to isolate platform-specific stages.

2.1.7 Physical-aware Logic Resynthesis (MLCAD 2025 Contest)

We conducted the MLCAD25 Contest on ReSynthAI, which incorporates post-global-route physical awareness into logic resynthesis after floorplanning. The contest provides a complete infrastructure, including benchmarks in standard EDA and ML-friendly formats, and examples of "ML inside" EDA tools using Python APIs. It leverages OpenROAD and CircuitOps, enabling participants to build on accessible, open-source tools and data formats. The released benchmarks have been useful in evaluating OpenROAD and our LLM-driven timing optimization effort. The published paper for this effort can be found at:

https://ieeexplore.ieee.org/abstract/document/11189135

and the GitHub repository that holds the contest benchmarks and scripts operating with OpenROAD for the contest can be found at:

https://github.com/ASU-VDA-Lab/MLCAD25-Contest-Scripts-Benchmarks/tree/main.

Overall, SODA-Benchmarks delivers a practical, open framework for measuring progress in software-defined accelerator toolchains and provides actionable artifacts for both method development and comparative evaluation. The forthcoming FY2026 efforts aim to broaden the suite's kernel coverage, strengthen ASIC-relevant workflows, and continue improving reproducibility and extensibility for the community.

2.1.8 Physical awareness in Bambu High-level Synthesis (HLS)

The standard HLS to GDS flow is inherently sequential, making it difficult to account for information from later stages early in the design process. Many critical design decisions must be made up front to optimize Power, Performance, and Area (PPA). Current EDA tools rely heavily on heuristics and coarse modeling to inject limited physical awareness early in the flow, but these methods are often inaccurate, leading to suboptimal outcomes. The goal of introducing physical awareness in HLS is to enhance post-route PPA, which requires improvements across both the HLS and Physical Design stages. Our work focuses on two fronts: (1) integrating physical awareness into Bambu, and (2) enhancing physical design and timing optimization in OpenROAD.

2.1.8.1 Floorplan-aware interconnect delay estimation within Bambu

Scheduling is a key stage in Bambu that occurs early in the HLS flow. It assigns functional units in the module-allocated, unscheduled call graph to time slots within the target clock period, based on their execution times and interconnect delays. Currently, wirelength estimation relies on fixed templates rather than physical layout information, resulting in lower Quality of Results (QoR) in later stages of the HLS-to-GDS flow. To address this, we integrated a floorplanning engine into Bambu that places modules on a 2D die, providing approximate block locations and more realistic wirelength estimates for module-to-module connections. The overall flow is shown in Figure 1.

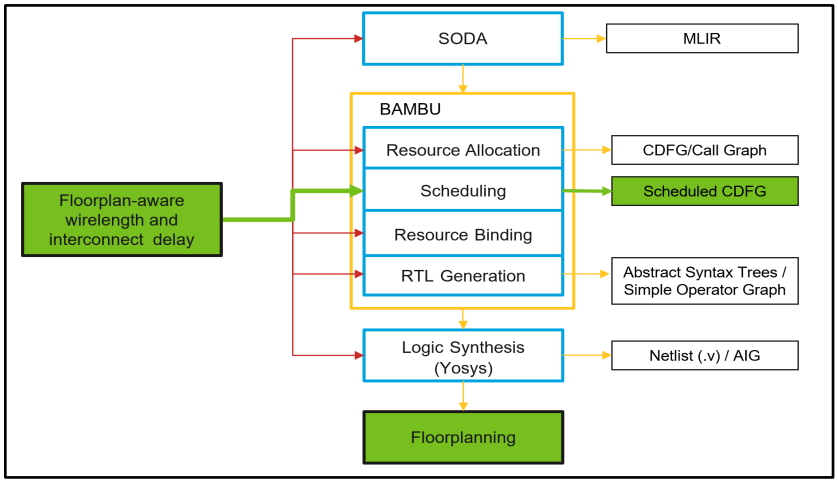


Figure 1. Flow diagram of floorplan-aware wirelength and interconnect delay estimation in Bambu

Based on the floorplan solution, we extract wirelengths between the blocks, use wire parasitics for the specific technology node to estimate interconnect delays. This directly feeds back into Bambu for physically aware scheduling.

Figure 2 shows implementation details of the interconnect delay estimation in detail.

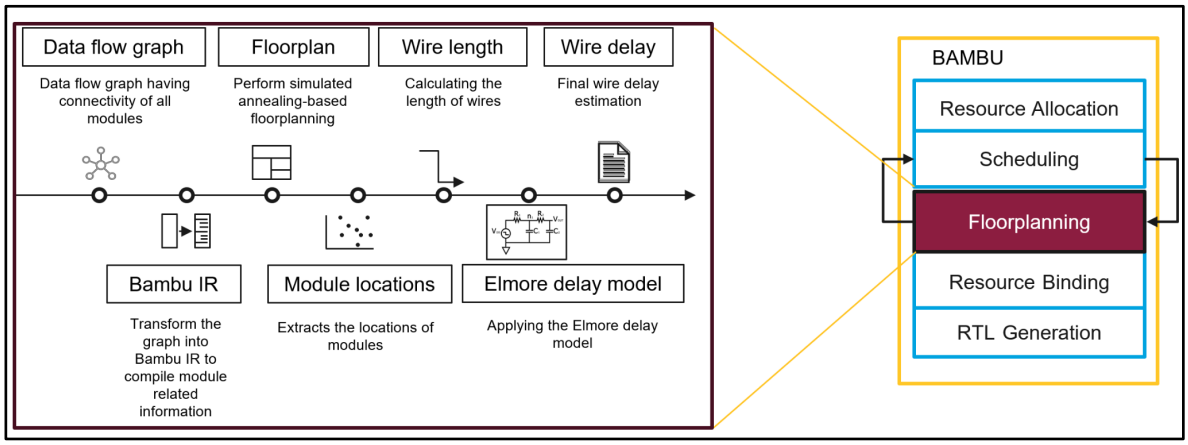


Figure 2. Simulated annealing-based floorplanning information is used to build a lookup table for interconnect delay estimation inside the Bambu scheduler.

Table 1 shows post-routed PPA results obtained from using physically aware Bambu vs. primitive (old) Bambu where physical awareness improves post routing delay significantly for designs synthesized in a 7nm technology. It comes with a slight increase in power consumption due to additional pipeline stages added by the scheduler.

Table 1. Improvement in post-route PPA metrics due to physical awareness

Design	# Insts.		Total negative slack (ps)		Worst negative slack (ps)		Area μm^2		Total power W	
	Old	Physical -aware	Old	Physical -aware	Old	Physical -aware	Old	Physical -aware	Old	Physical -aware
Gemm cp-500ps	34648	40301	-109913	-64172.7	-1412.76	-568.18	4446.7	5546.5	0.04	0.06
Gemver cp-500ps	50338	58411	-191957	-77641	-1751.84	-407.15	6452	7910.93	0.23	0.27
Gesummv cp-500ps	30055	36948	-57921.8	-47111	-576.5	-383.82	3653.19	4925.27	0.12	0.18
Symm cp-500ps	33843	41176	-73956.8	-46583.5	-877.03	-382.145	4187.24	5620.6	0.21	0.28

2.1.8.2 Recharacterization of functional blocks in Bambu

Bambu uses built-in libraries of functions to handle software-to-hardware mapping, scheduling, allocation, and related tasks. These libraries store delay, area, and other physical parameters for

each block—some of which are used during floorplanning. We discovered inconsistencies in this data and traced them back to errors in the original module characterization process. To address this, we developed a flow to correctly recharacterize these blocks. This improvement directly enhances PPA in Bambu, as the updated data is now used throughout the HLS.

In FY26, we plan to replace the simulated annealing-based floorplanning engine with an enhanced Graph Neural Network model to predict wire delays for quicker execution. We will also embed physical awareness in other stages of the SODA design flow.

2.1.9 Tapeouts

This task integrates the end-to-end chiplet synthesizer flows, demonstrating the capability of the ENCODE toolchain to deliver push-button hardware designs that can be directly prototyped in silicon. In Year 1, we completed two multi-project wafer (MPW) tapeouts: one using SkyWater 130 nm technology through Chip Foundry, and one using GlobalFoundries 180 nm technology through the academic MPW program supported by GlobalFoundries and Synopsys at Arizona State University (ASU).

The Chip Foundry tapeout was executed in place of the prior shuttle submitted through eFabless in September 2024. eFabless ceased operations before delivery; Chip Foundry subsequently acquired the eFabless assets and plans to deliver the earlier design (pending payment). We are actively coordinating retrieval of that tapeout.

The SkyWater 130 nm (Chip Foundry) tapeout uses a fully open-source backend flow (OpenLANE 2.0, which internally repackages tools from the OpenROAD flow) and the open SkyWater 130 nm PDK. The design targets the standard Caravel harness, which provides I/O pads and an integrated PicoRV32 RISC-V microcontroller for exercising the user design. For this shuttle, we taped out the unoptimized accelerator from the canonical SODA tutorial—a batched matrix-multiplication operator—integrated into the Caravel platform along with an SRAM used to store inputs and outputs. The 2025 MPW submission closely mirrors the 2024 eFabless tapeout.

The GlobalFoundries 180 nm (GF180) tapeout uses the open GF180 PDK but relies on Synopsys proprietary synthesis tools. This combination enables comparison of quality-of-results across different process nodes and toolchains. For this shuttle, we taped out the full encoder of the energy-loss electron spectroscopy (EELS) autoencoder model, one of the low-latency domain-specific workloads distributed with the SODA benchmarks repository. Because the I/O pads in the GF180 program do not include a harness platform and no memory compiler or pre-designed macros are provided, post-silicon evaluation will focus on logic-level functional testing rather than full system evaluation.

Chip delivery for both shuttles is expected around March 2026, with approximately 100 QFN-packaged parts per design. For the Chip Foundry submission, a prototyping board will also be provided to support bring-up and initial validation. Retrieval of the 2024 eFabless tapeout would further expand our evaluation set.

Overall, the 2026 workplan includes:

1. Evaluating results from all available tapeouts
2. Establishing and refining a systematic validation flow

3. Identifying backend toolchain limitations

4. Feeding back functional and PPA (power, performance, area) data to improve synthesis methodologies.

We are targeting a Spring 2026 tapeout with Chip Foundry for a next-generation design that incorporates new ENCODE-developed capabilities—such as physically-aware HLS, streaming architectures, and automated instrumentation—and are also exploring opportunities for MPW submissions in more advanced technology nodes.

2.2 Optimization for Efficient Hardware Design

2.2.1 Hardware Software Partitioning

Hardware/software (H/S) partitioning is a key step in embedded-system design, where each computational component must be assigned to hardware or software to meet performance, power, and cost goals. A common formulation casts the task as graph bipartitioning, where vertices (or nodes) represent components with implementation costs for hardware and software, and edges (or links) carry weights equal to the penalty of communicating across partitions. By augmenting the graph with two terminal nodes (HW and SW), connecting each component to these terminals with capacities equal to its implementation costs, and retaining intercomponent edges with communication weights, a polynomial-time minimum s-t cut (e.g., push–relabel/Goldberg–Tarjan) yields the globally optimal unconstrained partition that minimizes implementation plus split-communication cost. In practice, additional resource constraints—limited area, energy budgets, timing or throughput targets—couple decisions and render the problem NP-hard, so it is typically addressed with heuristics or relaxation-based search.

Table 2. Benchmark applications with details about the generated task graph

Framework	Model	#Operators	TOSA #Nodes	Linalg #Nodes	Model Size	Origin
PyTorch	SqueezeNetV1.1	86	119	149	9.5MB	Benchmarks
PyTorch	Resnet18	68	249	286	90MB	Benchmarks
PyTorch	MobileNetV3 (S)	209	589	594	20MB	Benchmarks
TF	EELs Encoder	29	37	45	140KB	ATSCALE
TFLITE	ResNetV1-10	16	96	137	99kB	TinyML
TFLITE	Anomaly Detect	17	48	47	280kB	TinyML
TFLITE	Audio Wake Words	15	90	125	53kB	TinyML
TFLITE	Visual Wake Words	33	161	196	334kB	TinyML

In FY2025, we automated HW/SW partitioning by extending the SODA-OPT compiler to generate solver-ready computation graphs. We implemented MLIR transformation passes that annotate each task with memory-access and arithmetic-operation counts, enabling a per-task cost model. These costed graphs feed directly into both min-cut relaxations and heuristic partitioners. Using this pipeline, we generated a benchmark suite of graphs derived from diverse, real-world ML models to evaluate the proposed approach. This suite of directed acyclic graphs (DAGs) represents task graphs composed of multiple operations which can be realized on hardware or

software. Table 2 shows details about the benchmark suite (SODA-Benchmarks) we used to generate task graphs on using TOSA operations. These benchmarks include varying number of nodes and operators internally to represent well known ML models for various scientific workloads.

2.2.1.1 Heuristic algorithms for Hardware/Software Partitioning

We developed heuristic-based and reinforcement learning approaches to solve the partitioning problem in these task graphs. We consider scheduling constraints in our optimization formulation to enable parallel execution of hardware operations, while software operations are sequentially executed on a single CPU core. The objective function is the minimum make span of the task graph. However, for the heuristic algorithms, we used multiple objective functions which approximates the make span of the task graph. We compared the heuristics with the mixed integer problem (MIP) formulation as a baseline.

Across three objective functions, the MIP baseline consistently outperforms all heuristic methods (Figure 3), demonstrating the advantage of joint optimization (partitioning and scheduling), albeit the longer computation time. However, when using a communication-aware objective function (bottom right), the GL25 heuristic achieves slightly better make span estimates than MIP, likely because we limit exploration of all possible scheduling sequences of software nodes to reduce the complexity of the MIP.

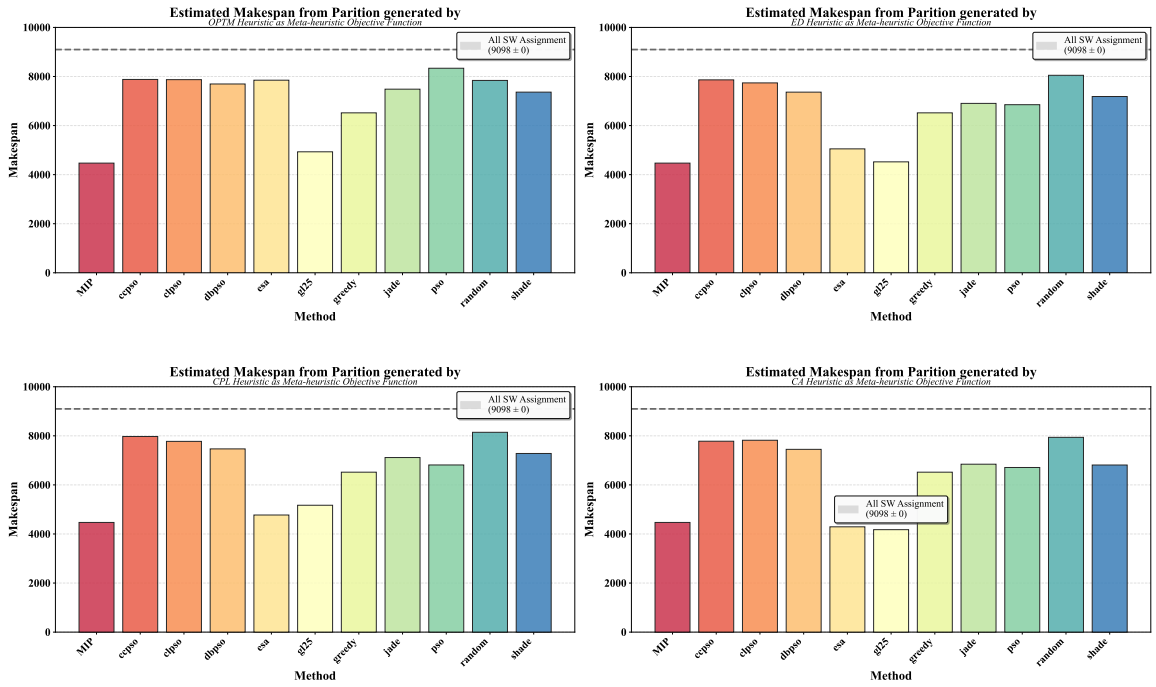


Figure 3. Make span comparison across different heuristics and objective functions

2.2.1.2 Parametrized Exact Hardware/Software Partitioning

Several important formulations of the H/S Partitioning problem are NP-hard when the number of available resources needed to construct hardware is in some way constrained. Two such notable formulations are the minimization of the span of the task graph or minimization of the execution time of the tasks evaluated sequentially subject to a constraint on the available hardware area. Consequently, it can be computationally expensive to get an exact solution for some generic problem instances while using these formulations. We show that constrained formulations are fixed parameter tractable to solve; that is, the cost required to compute a solution is determined primarily by the topological pathwidth of the computational task graph (illustrated in Figure 4). Topological pathwidth is itself a variant of the measure of graph complexity known as pathwidth, which has been studied extensively in the fields of graph theory and fixed parameter tractable algorithms.

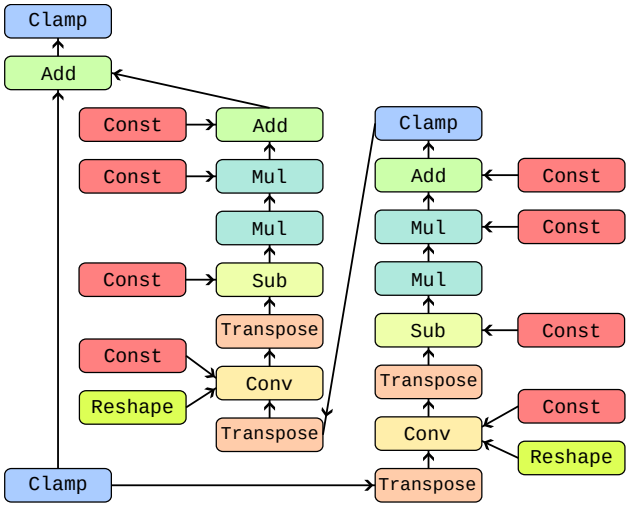


Figure 4. A pathwidth 5 subgraph of the RESNET computational task graph constructed using SODA. Each node represents a neural network operation, such as convolution, multiplication, or transposition. This subgraph, or one similar, appears multiple times in the RESNET task graph.

We developed a novel algorithm for solving the H/S Partitioning problem based on dynamic programming utilizing a topological path decomposition of the task graph. Using this algorithm, we observe speed up of more than **200**× relative to an equivalent linear programming approach. Furthermore, we also developed a novel random graph model for generating graphs with low topological pathwidth, which can be used for experimental analysis. We used the SODA toolchain to generate the task graph associated with the RESNET architecture and showed that it contains a major subgraph which follows this path-like structure, showing that low pathwidth tasks do appear in relevant applications (illustrated in Figure 4).

Our preliminary results directly support the broader goals of automated design exploration and hardware-software co-optimization by providing a structure-aware method that dramatically improves the computational performance using H/S partitioning, as summarized in Figure 5. By revealing tractable regimes in realistic computational task graphs and demonstrating practical speedups, our approach enables more scalable optimization pipelines for emerging heterogeneous architectures. These results have been submitted to DAC'26 conference.

In FY26, we plan to:

1. Extend this framework to richer cost models that incorporate additional architectural constraints, such as communication overhead or heterogeneous memory structures.

2. Explore realistic graph generators that learn from existing computationally inefficient H/S partitioning scenarios to provide valuable benchmark families and guide the design of more robust algorithms.

Such generators will allow systematic stress testing of partitioning strategies and help reveal structural bottlenecks present in real applications.

In the following years, a promising direction to explore is developing automated pathwidth-aware compilation pipelines that detect low–pathwidth regions in large task graphs and selectively apply fixed-parameter techniques to further accelerate real-world design flows.

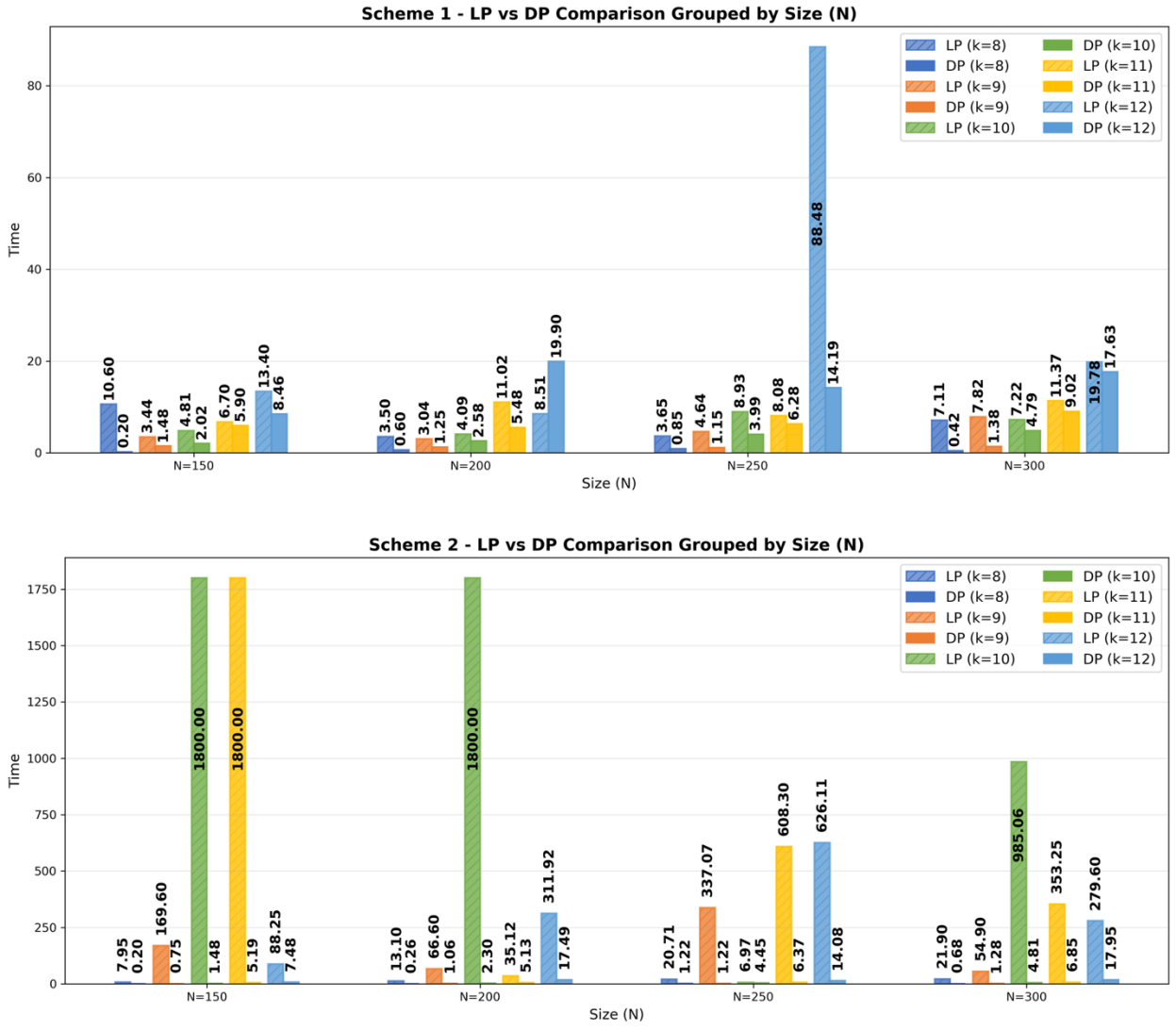


Figure 5. A comparison of running time in seconds between the dynamic programming (DP) and linear programming (LP) approaches between two sets of randomly generated graphs with bounded pathwidth under two different weighting schemes. The graphs are grouped by size (N) and pathwidth (k). We see that the dynamic programming approach is better on every instance, with an improvement of up to 200x for some graphs in group 2. A time limit of 1800s was set for both algorithms.

2.2.1.3 Graph Neural Network (GNN) based Hardware/Software Partitioning

In FY26, we will complement the pathwidth-aware exact and heuristic approaches with a learned solver based on the Graph Combinatorial Optimization Network (GCON) framework for combinatorial optimization on graphs. Each SODA-OPT–generated task graph will be treated as a binary labeling problem where the GNN predicts, for every task, the probability of assigning it to hardware or software. The input features will combine the existing cost annotations (latency, memory accesses, arithmetic intensity) with structural descriptors such as in-degree, out-degree, critical-path distance, and approximate pathwidth indicators. We will implement loss function for HW/SW by defining a differentiable objective that captures implementation cost plus communication overhead, with soft penalties for violating hardware area, energy, or memory-bandwidth budgets. A rule-based decoder will then round the probabilistic assignments into a feasible partition that satisfies resource constraints and is compatible with the SODA scheduling model.

We will also explore a separate GCON model which learns to solve the H/S partitioning problem jointly with the scheduling of operations on both resources. The resulting joint optimization problem can be cast as problem of scheduling task graph operations on either hardware or software, subject to a constraint on available hardware area. The non-differentiable makespan objective can then be minimized by training a reinforcement learning agent with Monte Carlo gradient estimation. This approach will allow us to draw upon the existing literature on reinforcement learning for complex scheduling problems, for which best practices in training, architectural design and feature engineering are well-established.

We will use the fixed-parameter dynamic programming (DP) solver and the existing heuristic and reinforcement learning methods as both baselines and oracles for training and evaluation. On low–pathwidth graphs where the DP algorithm is tractable, we will treat its solutions as ground truth and investigate supervised or distillation-style training, while on larger graphs we will rely on self-supervised objectives that approximate the makespan and area-constrained cost used in our current pipeline. This will enable a systematic comparison of generalization: the GNN-based solver will be evaluated on families of task graphs derived from diverse ML models (CNNs, transformers, GNNs) and a range of hardware budgets, measuring not only solution quality and makespan but also how quickly it adapts when the architectural constraints change. We will also study how structural statistics such as estimated pathwidth or depth influence the learned representations, and whether the GNN implicitly recovers the low–pathwidth regimes where DP is particularly effective.

Additionally, we will integrate the GNN-based partitioner as a first-class component in the SODA-OPT design-space exploration loop. The goal is to use GNN inference as a fast surrogate that proposes high-quality HW/SW partitions and schedules which can then be selectively refined by exact or fixed-parameter methods on the most promising candidates. Over FY’26, this effort will deliver a learning-augmented, structure-aware partitioning framework that combines the strengths of combinatorial optimization and graph neural networks, and that can scale to the large heterogeneous task graphs produced by emerging accelerator-rich architectures.

2.2.2 Global Routing

Global routing is a crucial step in VLSI design that has become increasingly more complex as chip sizes and design scales grow. The result of the global router (GR) directly influences the performance of detailed routing, improvements at this stage result in significant downstream impact on the overall design. Many global routers divide the process into several stages: two-pin

decomposition, congestion map generation, maze routing, and layer assignment. Each of these stages requires routing thousands to millions of nets providing opportunities for parallelization. While there has been considerable progress in accelerating GR algorithms, comparatively little attention has been given to exploiting parallel computing.

To address the current limitations, we have developed a parallel global routing algorithm, named LockRoute[1] that makes use of a spatial locking framework for scaling on multithreaded shared-memory systems by identifying net dependencies using an R-tree data structure. Our locking framework is flexible enough to be applied to every stage of the global routing pipeline. When evaluated on the ISPD 2008 and ISPD 2019 global routing contest benchmark suites, our approach achieves a significant speedup in runtime without a reduction in quality. Compared to OpenROAD's default GR, FastRoute, we achieved a reduction in runtime by 1035.15% with comparable quality of solution. On the benchmarks from the 2008 global routing contest held by ISPD, LockRoute reduces the runtime by 412.76% on average compared to NCTU-GR 2.0 with minimal degradation to quality. This performance speedup is further demonstrated on ISPD's 2019 detailed routing contest benchmarks in which LockRoute had a decrease in runtime of 1035.15% compared to FastRoute and 245.48**%** decrease in runtime compared to CuGR.

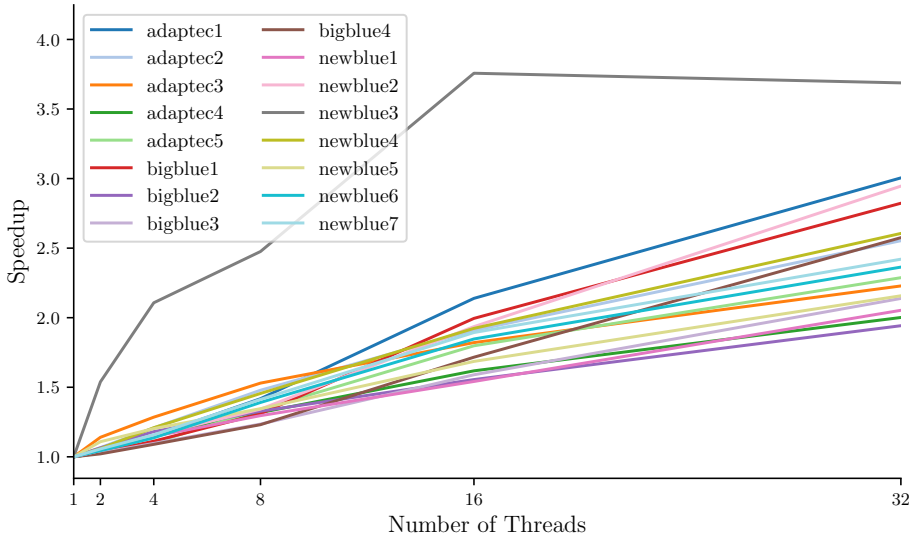


Figure 6. Scalability of LockRoute showing time speedup factor (y×) with increasing number of threads. The difficult benchmark newblue3 lies separated from the rest.

LockRoute's scalability was tested against ISPD'08 benchmarks and ran on an AMD EPYC 7502 32-core processor with 256 GB of DDR-3200 memory. To test parallel speedup, we tested each benchmark 10 times for various number of threads and took the average runtime. We also tracked the total overflow (TO) and max overflow (MO) to ensure there was no degradation in quality. Our results (Figure 6) show a significant speedup in all benchmarks as we increase the number of threads. LockRoute had the most significant speedup on newblue3, a difficult benchmark among the ISPD'08 benchmarks. Moreover, the total overflow and max overflow were consistent despite the change in thread count (see Table 3). Note that these results also demonstrate the necessity of a parallelization strategy as all benchmarks had an increase in overflow when ran without the use of a parallelization strategy. By using locks, we achieve significant speedup while avoiding

[1] Available at https://anonymous.4open.science/r/LockRoute-FD0D

the degradation in quality that occurs without a parallelization strategy. Our findings have been submitted to DAC'26 conference.

Table 3. Overflow results for various strategies. MO and TO are max and total overflows. Locks and No Locks strategies ran using 32 threads, whereas Sequential ran single-threaded.

Benchmark	Sequential		Locks		No Locks	
	MO	TO	MO	TO	MO	TO
adaptec1	0	0	0	0	1	24
adaptec2	0	0	0	0	2	37
adaptec3	0	0	0	0	1	32
adaptec4	0	0	0	0	1	18
adaptec5	0	0	0	0	2	32
bigblue1	0	0	0	0	2	34
bigblue2	0	0	0	0	2	19
bigblue3	0	0	0	0	2	27
bigblue4	0	0	0	0	2	16
newblue1	0	0	0	0	2	10
newblue2	0	0	0	0	1	19
newblue3	263	5705	235	5833	189	10426
newblue4	0	0	0	0	2	39
newblue5	0	0	0	0	2	28
newblue6	0	0	0	0	2	32
newblue7	0	0	0	0	2	60

Table 4. Quality and runtime results comparing LockRoute against NCTU-GR 2.0 on overflow-free cases. WL and # VIA give the wirelength and the number of vias in units of 105 CPU gives runtime in seconds.

Benchmark	LockRoute			NCTU-GR 2.0		
	WL	# VIA	CPU (s)	WL	# VIA	CPU (s)
adaptec1	33.91	27.14	2.98	37.83	20.38	11.54
adaptec2	32.13	27.01	1.71	34.04	20.04	7.82
adaptec3	93.56	50.33	4.49	97.78	38.49	19.12
adaptec4	89.02	44.63	4.05	90.28	38.82	16.12
adaptec5	98.12	77.19	5.07	107.87	58.33	40.55
bigblue1	34.34	29.67	7.01	41.57	23.43	23.69
bigblue2	46.32	52.87	3.68	49.93	45.04	27.34
bigblue3	77.06	70.97	5.54	80.44	53.28	20.19
newblue1	23.41	30.22	1.80	25.03	23.08	23.97
newblue2	46.17	37.79	2.40	46.83	29.80	9.94
newblue5	142.52	126.74	10.79	152.75	89.58	55.43
newblue6	97.83	106.54	7.01	110.48	82.77	37.21

In the upcoming fiscal year, we plan to optimize and expand LockRoute along several facets: optimal lock-distribution schemes, alternative spatial partitioning structures, and better implementation of Dijkstra's algorithm. We also plan to further analysis of the effects global router on the runtime of detailed routing.

2.2.3 Timing Optimization using LLMs for the OpenROAD Physical Design Tool

Every place-and-route tool includes a Static Timing Analysis (STA) engine that identifies timing violations at each step and applies repair transformations, such as buffering, VT swaps, and upsizing. Most tools rely on a fixed sequence of these transformations for timing optimization or require manual intervention by the user. OpenROAD follows a similar approach, using a resizer module that applies a default sequence of fixes to all paths across all designs. However, this static approach does not adapt to design-specific features or past results, limiting its effectiveness for specific violating paths.

We explored varying the sequence of these transformations and found that sequences tailored to design characteristics yield better outcomes. The main challenge lies in the vast solution space, the high computational demand, and the long runtimes required to evaluate different move sequences and their orders. To address this, we leverage large-language models (LLMs) to generate sequences by learning from conversation history. Figure 7 illustrates our LLM-based flow for generating custom transformation sequences to optimize violating paths and recover PPA metrics after global placement.

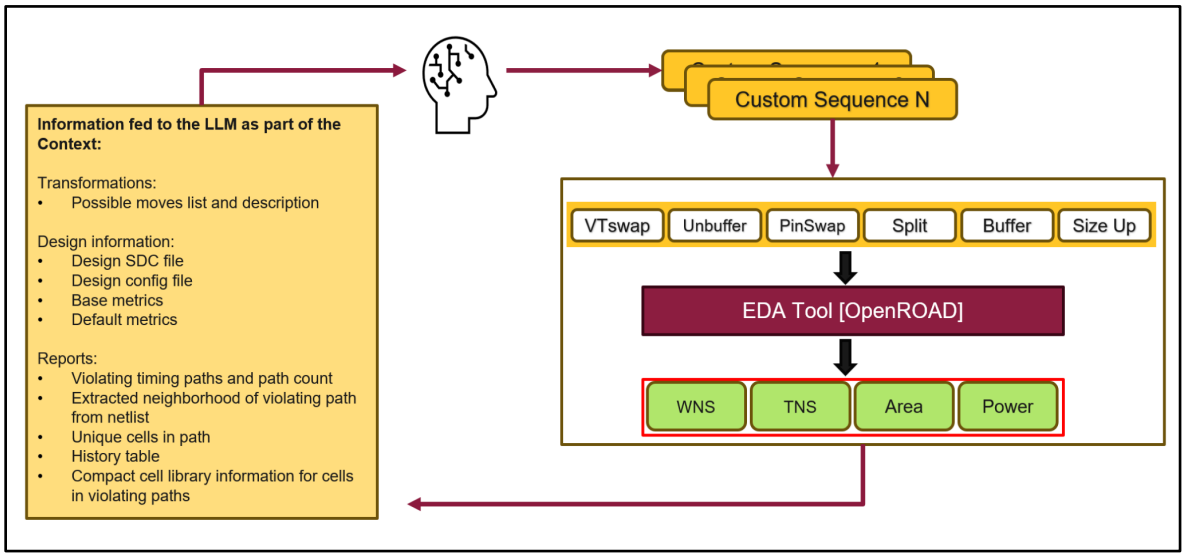


Figure 7. LLM-based flow to generate custom transformation sequences to optimize violating paths.

We have incorporated an LLM-based timing optimization into OpenROAD, which iteratively generates a new sequence of transformations tailored to the design/violating end point by building intelligent context and learning from previous sequences to identify patterns and achieve rapid convergence. We have also shared some initial results on some benchmarks in Table 5. It shows that the LLM-generated sequences were able to recover timing and meeting the timing constraint as against the default sequence (which is fast but low quality) and a simulated-annealing-based solution (which is slow and low quality). For large designs such as IBEX and JPEG the simulated annealing engine cannot finish in reasonable runtime.

Table 5. Delay (TNS/WNS) improvement using LLM-based sequence generation over the default sequence and simulated annealing-based sequence generation.

Design Name	Clock Period	Total Negative Slack			Worst Slack			Runtimes		
		D	L	S	D	L	S	D	L	S
	(ns)	(ns)			(ns)					
GCD	0.22	-0.02	0	0	-0.02	0.01	-0.003	1m02s	2m36s	2h08m13s
AES	0.35	-0.52	0	-0.07	-0.02	0.003	-0.003	2m06s	4m54s	4h17m
IBEX	1.16	-1109.8	-963.4	N/A	-1.07	-0.94	N/A	18m31s	2h12m41s	N/A
JPEG	0.7	-102.1	0	N/A	-0.2	0.02	N/A	9m36s	1h09m47s	N/A

D = Default flow, L = LLM based flow, S = SA based flow

In FY26, we plan to scale the LLM-based timing optimization engine to larger designs. Release the LLM-based timing optimization to the public. Apply LLM-based tuning of various hyperparameters in OpenROAD and Bambu.

2.2.4 Resource Binding Problem

In high-level synthesis (HLS), data path allocation (DPA) corresponds to binding variables to registers and operations to functional units. DPA typically takes a schedule as input and produces these bindings. During scheduling, the lifetimes of variables and the demand for functional units are determined across control steps (cycles). Given this information, the next step is to (i) allocate registers to variables that are live in each control step and (ii) allocate functional units to the operations executed in that control step. The goal is to incorporate interconnection cost (e.g., the number of multiplexers) into both the register and functional-unit allocation.

One of the approaches implemented in the Bambu framework for register allocation is as follows. For each control step, it first computes the set of live variables. It then constructs a complete weighted bipartite graph whose edge weights are defined by heuristics designed to minimize interconnection cost. A minimum-weight matching (assignment) on this bipartite graph yields a resource allocation that approximately optimizes the interconnection cost. Bambu uses an optimal implementation of the assignment algorithm for this step. This workflow, however, suffers from performance bottlenecks: storing the complete bipartite graph is memory-intensive, and computing an optimal matching is computationally expensive. In FY'26, we aim to alleviate the bottleneck challenges by leveraging approximate streaming matching and covering algorithms.

3.0 Thrust 2: Next Generation Testbeds

Thrust 2 develops capabilities centered around next-generation testbeds to test and evaluate AI-enabled computational science accelerators for DOE mission workloads developed under Thrust 1 above. In Year 1 of ENCODE, focus has been on developing simulation and modeling capabilities, AI-driven automated testbench generation, and integration of pre-silicon verification tools with the overall design tool flow. Using these methods, the aim is to enable a quantitative and accurate understanding of designs under test relative to key performance and efficiency metrics. As ENCODE moves into evaluation of silicon prototypes in Year 2, the Testbeds Thrust will ramp up activities related to physical test and verification. This will include completing the installation of PNNL's physical testbed and prototyping laboratory and conducting validation of prototype silicon. This laboratory will include test harnesses for connecting prototype chips to server systems and measurement capabilities for performance and efficiency. A major effort during this period will also include connecting the pre-silicon simulation and modeling capability to post-silicon measurement – predictions made prior to prototype implementation must be correlated with measurements taken from hardware samples. This will drive design decisions, such as pin-out allocation, making test a key contributor to overall design.

As a result of needing to develop expertise and an operating plan, the pipeline leading to publications is only now beginning to fill. Recognizing the importance of publications, not only to the project, but to connect PNNL's hardware design and prototyping capabilities to the wider community, we will be placing additional emphasis to drive publications on ENCODE's Testbed verification in FY26.

3.1 Testbed Activities

3.1.1 Cycle-Accurate Memory Co-simulation for Synthesized Accelerators

A cornerstone effort of ENCODE's simulation capability is developing tools to understand the interaction of novel architecture concepts with memory – developing architectural concepts in isolation may lead to sub-optimal system performance. To achieve this, in FY25, the ENCODE Testbeds thrust began implementing a cycle-accurate timing estimation for synthesized accelerators as they interact with external memory components (such as DRAM), replacing Bambu's current fixed-time memory model with Ramulator2 DRAM simulator integration. This will allow designers to develop a much more accurate estimation of the impact of memory traffic on overall workload performance. This work also accelerates the development of memory interfaces for chiplets with more accurate memory characteristics, and more specifically, to test the impact of different memory technologies in the generated chiplets. As a result, co-optimization decisions, e.g., regarding data placement and communication scheduling, can be made using quantitative information, providing designers greater levels of insight during the design process.

We have begun by integrating Ramulator2[1] directly into Bambu, replacing the existing memory model. We are using Bambu's Memory Direct Programming Interface (MDPI) to enable Verilog testbenches enhanced with memory-aware simulation and facilitate data exchange between hardware testbenches and models.

As the starting point of this effort, in 2025, we started with the initial integration of the enhanced Ramulator module in Bambu by inserting several functions calls in the testbench generation

[1] https://github.com/CMU-SAFARI/ramulator2/tree/main

procedure. We are working closely with the Bambu team to ensure the correctness of this approach and its possible adaptation in future toolchains.

In 2026, we are planning to complete and to test the integrated Ramulator/Bambu toolchain using common DRAM models as the base technology. We are planning to run the infrastructure with both synthetic and real-world memory traffic to test our initial assumptions and the fidelity of the toolchain. Finally, we are planning to expand the experiments to more memory technologies, such as HBM models (already available in Ramulator).

3.1.2 LLM-Based Agentic Workflow for Hardware Testing and Verification

This task is developing an intelligent testing framework for ASIC hardware designs that seamlessly integrates with PNNL's EDA workflow being developed under Thrust 1 and including opensource tools developed by the community, comprising Soda-Opt (Python to LLVM), Bambu HLS (LLVM to RTL Verilog), and OpenROAD (RTL to GDSII). We envision creating a framework where large language models (LLMs) orchestrate a multi-agent workflow to generate test scenarios for hardware designs automatically. The framework will reduce verification time and validate designs produced by the ENCODE design team through unit testing and coverage-driven verification.

The initial phase in FY25 focused on developing an LLM-based framework for RTL unit testing using a LangGraph (https://www.langchain.com/langgraph) workflow centered on the open-source Verilator (https://www.veripool.org/verilator/) for RTL simulation, targeting simple sequential designs as proof-of-concept demonstrations. The next phase in FY26 will extend the framework to support PNNL's EDA workflow integration with Bambu HLS, implementing random verification strategies for LLVM-to-RTL equivalence checking through fuzzed testing approaches, developing functional coverage analysis derived from input LLVM code, and creating iterative RTL code coverage for design sign-off to proceed with next steps in the EDA flow.

In 2025, we have successfully demonstrated a functional multi agent workflow, where the LangGraph-based system coordinates specialized agents, including a Verilog code parser, test vector generator, C++ Verilator-based testbench generator, and a simulator executor. We have also integrated unit testing capabilities (such as counters) in the workflow. This capability helps us to validate the generation of valid test vectors and test benches; thus, demonstrating the validity of the LLM-driven hardware verification. We also integrated an error handling routing (through retry mechanisms) and conditional execution paths to bypass the LLM inconsistencies.

For 2026, we are planning to add metric-based closure to enhance unit testing which includes systematic verification with quantifiable coverage goals and completion criteria. We are incorporating functional verification for comprehensive design validation. As part of the flow, we are planning a more detailed integration with the Bambu HLS tool to further integrate to the rest of ENCODE thrusts. Moreover, under this tool, we are planning to develop specialized agents for LLVM-RTL functional equivalence verification through random testing methodologies. Finally, we are further developing agents that will provide functional coverage extraction using LLVM IR code as its input and generate coverage-driven test stimuli as its output. In the long term, we plan to develop an end-to-end automated verification pipeline that is integrated with PNNL's EDA workflow.

3.1.3 LLMs for OpenROAD Verification

OpenROAD is an open-source toolchain aimed at automating and simplifying chip design processes, making them more accessible to researchers, students, and smaller companies. It automates the entire "RTL-to-GDSII" pipeline, transitioning designs from high-level digital descriptions to physical layouts ready for fabrication. This innovative approach eliminates the need for costly commercial tools and large teams of experts. In recent efforts, researchers have incorporated large language models (LLMs) like OpenROAD-Agent (developed by ENCODE collaborator Arizona State University - https://ieeexplore.ieee.org/document/11106006) to further improve accuracy and automation in tasks such as script generation, debugging, and verification.

OpenROAD-Agent uses LLMs to generate and self-correct flow scripts, streamlining tasks like Tcl scripting, error detection, and task correction. A comprehensive retrieval-augmented generation (RAG) database enhances its capabilities by providing OpenROAD-specific APIs and script templates. OpenROAD's ongoing advancements in incorporating AI aims to revolutionize chip design by improving efficiency, accuracy, and accessibility for various users.

Some of the highlights from 2025 include the following. Experiments with OpenROAD-Agent 7B were conducted to assess its accuracy in generating Python flow scripts, utilizing resources such as NVIDIA A100 GPUs on JetStream2. These experiments revealed discrepancies in reported and replicated accuracy due to semantic correctness not being considered in prior evaluations. RAG configurations were tested for script generation accuracy. The RAG database was expanded to include APIs for IR Drop Analysis, which examines voltage drops across power delivery networks. Initial flow scripts for IR Drop Analysis were generated but failed during execution due to missing dependencies, such as liberty libraries. Efforts are underway to deploy an MCP (Model Context Protocol) server. This server will enable an iterative process by routing error messages from OpenROAD back to OpenROAD-Agent for script corrections, ensuring smoother script execution.

In 2026, we are focusing on these key activities. Following initial success with IR Drop Analysis, additional OpenROAD verification modules will be integrated. This includes tasks like Design Rule Checking (DRC), Layout Versus Schematic (LVS), and Static Timing Analysis (STA). More sophisticated verification techniques, including those powered by third-party tools such as Bumbu-empowered floorplan verification, will be integrated to enhance reliability. Multiple agentic models, including Claude-Code, will be tested for their ability to handle extended verification tasks effectively. This evaluation aims to identify optimal models for integration. The primary goal is to ensure that flow scripts generated by OpenROAD-Agent are executed successfully within the OpenROAD framework without semantic or execution errors.

3.1.4 UCIe Simulation for Thermal and Power

As more chiplet designs are introduced, their communication with different computation components, such as other chiplets, external memory, and the host, should be considered. This aspect of computing is essential since these substrates can be the source of congestion that leads to inefficient use of the generated chiplets. Moreover, thermal and power issues are usually underrepresented in simulations. As a result, in FY25 we developed a simulation infrastructure as a plugin for the OpenROAD flow infrastructure. An overview can be seen in Figure 8.

To date, we have used synthetic traffic patterns based on bursty, random, or streaming traffic. In the next FY, we plan to expand this capability to used traced traffic patterns from simulated workloads running on architecture concepts developed in ENCODE. By developing this capability

as a plug-in for the OpenROAD tool flow, we integrate with other design and verification activities, developing an integrated design/test/validate toolchain.

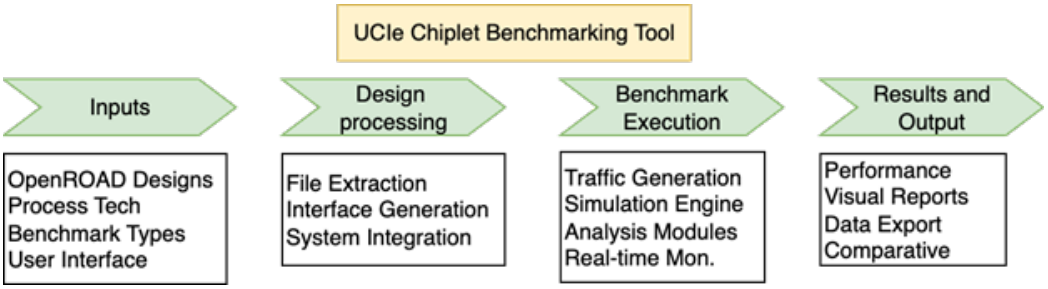


Figure 8. An overview of the flow of the UCIe simulation tool.

In 2025, we focus our efforts on creating the tool flow, with an emphasis on integration with existing state of the art tools, aka OpenROAD, to generate realistic designs (in Verilog which can be used by other open-source tools). Moreover, using these designs, we focused on the integration with simulation tools, like Verilator, to model and collect different metrics (bytes transfers, latency, bandwidth, power over different time windows, etc.) on the interconnect models that we integrated. Finally, we have created an easy-to-use interface to run, collect, and analyze results from the tool.

The main objectives of 2026 include the validation of the current simulation against published results, the expansion of power modeling capabilities to include dynamic models based in link utilization and data paths, and to support more chiplet types and configurations to create an interactive co-design refinement flow.

4.0 Publications

4.1 Journals, Conferences and Workshops

1. V. Agostinelli, N. B. Agostini and A. Tumeo, "UltraFormer: An Efficient Transformer for FPGAs," *2025 IEEE 33rd Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, Fayetteville, AR, USA, 2025, pp. 274-274, doi: 10.1109/FCCM62733.2025.00034.

2. Yin, J., Z. Song, N. Bohm Agostini, A. Tumeo, and C. Yu. (2025). HEC: equivalence verification checking for code transformation via equality saturation. Proceedings of the 2025 USENIX Conference on Usenix Annual Technical Conference. Boston, MA, USA, USENIX Association**:** Article 69.

3. Mandulak M., S. Ghosh, S. Ferdous, M.M. Halappanavar, and G. Slota. 2025. "Anonymized Network Sensing using C++26 std::execution on GPUs." In 29th Annual IEEE High Performance Extreme Computing Virtual Conference.

4. Manjunath D., N. Bohm Agostini, A. Tumeo, J. Zhang, and C. Chackrabarti. 2025. "ML4SODA: A Decision Tree Guided Design Space Exploration for Fast and High Quality MLIR-based HLS." In Great Lakes Symposium on VLSI (GLSVLSI 2025), June 30-July 2, 2025, New Orleans, LA, edited by L. Peng, et al, 758 - 763. New York, New York: Association for Computing Machinery. doi:10.1145/3716368.3735223

5. James Alfred Ang, Antonino Tumeo, Nícolas Bohm Agostini, and Ankur Limaye. 2025. The U.S. Microelectronics Research Programs and Where They Lower Barriers to Co-Design. In Proceedings of the 22nd ACM International Conference on Computing Frontiers: Workshops and Special Sessions (CF '25 Companion). Association for Computing Machinery, New York, NY, USA, 135–139. https://doi.org/10.1145/3706594.3732236.

6. Jiang H., J. Zhu, Y. Wan, B. Fang, H. Zhang, R. Jin, and Q. Guan. 2025. "Can Large Language Models Understand Intermediate Representations in Compilers?" In 42th International Conference on Machine Learning (ICML 2025, poster).

7. Cohavi E., N. Bohm Agostini, J. Haris, A. Tumeo, D. Kaeli, and J. Cano. "Defeat the Heap: Zero-Copy Data Movement in AXI4MLIR." 7th Compilers for Machine Learning Workshop, Sydney, Australia, Australia. (*accepted*)

8. Rodriguez Canal G., N. Brown, M. Jamieson, N. Bohm Agostini, A.M. Limaye, V.G. Castellana, and J.B. Manzano Franco, et al. 2026. "MLIR-DPR: A Compiler Framework for Automating Dynamic Partial Reconfiguration." In The 34th IEEE International Symposium on Field-Programmable Custom Computing Machines. (*submitted*)

9. Rodriguez Canal G., N. Bohm Agostini, A.M. Limaye, V.G. Castellana, J.B. Manzano Franco, N. Brown, and M. Jamieson, et al. 2026. "Towards scheduling of pipelined dataflow graphs in MLIR." 34th ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA 2026), Monterey, California. (*submitted*)

10. Rodriguez Canal G., N. Bohm Agostini, A.M. Limaye, V.G. Castellana, J.B. Manzano Franco, A. Tumeo, and M. Jamieson, et al. 2025. "DataFlex: Flexible scheduling of

pipelined dataflow graphs for ML on SoC." ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA 2026), Monterey, California. (*submitted*).

11. Bel O.M., D. Gajaria, J.S. Firoz, J.B. Manzano Franco, J.D. Suetterlein, A. Marquez, and K.J. Barker. 2025. "Embracing the Post-Moore Era: Hardware-Software Co-design Validation." ACM Transactions on Modeling and Computer Simulation. (*submitted*)

12. Varshika M.L., J. Hollenbach, N. Bohm Agostini, A.M. Limaye, R.M. Ramstad, V.G. Castellana, and J.B. Manzano Franco, et al. 2025. "Neuromorphic Computing for Autonomous Scientific Workflows: Real-Time Material Characterization at the Edge." Computer. PNNL-SA-219092. (*submitted*)

13. Ramstad R.M., N. Bohm Agostini, and A. Tumeo. "LLM-Driven Optimization for High-Level Synthesis." In Compiler Construction. PNNL-SA-218321. (*submitted*)

14. Thrall A.R., V. Chhabria, S. Ferdous, M.M. Halappanavar, and B. Krishnamoorthy. "LockRoute: A Spatial Locking Framework for Parallel Global Routing." DAC '26: Proceedings of the 63rd Annual ACM/IEEE Design Automation Conference. (*submitted*)

15. Ibrahim C.A., S. Ferdous, E. Mutlu, I. Safro, and M.M. Halappanavar. "Parametrized Exact Hardware/Software Partitioning." In DAC '26: Proceedings of the 63rd Annual ACM/IEEE Design Automation Conference. (*submitted*)

4.2 Invited Talks and Presentations

1. Tumeo A. "From Graphs to Design Automation: Custom Systems for Data Analytics." Presented by A. Tumeo at Dagstuhl Seminar on Holistic Graph-Processing Systems: Enabling Real-World Scale and Societal Impact, Wadern, Germany.

2. Ang J.A., A. Tumeo, N. Bohm Agostini, and A.M. Limaye. "The U.S. Microelectronics Research Programs and Where They Lower Barriers to Co-Design." Presented by J.A. Ang at 22nd ACM International Conference on Computing Frontiers, Cagliari, Italy.

3. Tumeo A., N. Bohm Agostini, A. Limaye, V.G. Castellana, M. Minutoli, J.B. Manzano Franco, and R.M. Ramstad, et al. 2025. "Bridging Python to Silicon: the SODA Toolchain." Presented by A. Tumeo at 1st Hawai'i Scientific Data Workshop, May 19-22, Wailea, Hawaii.

4. Marquez A., J.B. Manzano Franco, and A. Tumeo. 2025. "Advanced Memory to Support AI for Science." Presented by J.B. Manzano Franco at Red-C: Reclaiming Bus-based Systems During Compromise, Arlington, Virginia.

5. Ang J.A., and A. Tumeo. 2025. "The Democratization of Co-design." 12th Multicore World 2025, Christchurch, New Zealand.

6. Halappanavar M. 2025. "Scalable Combinatorial Optimization for Electronic Design Automation." Fusing Theory and Practice of Graph Algorithms, Providence, Ri, Rhode Island.

7. Tumeo A., N. Bohm Agostini, A.M. Limaye, M. Minutoli, V.G. Castellana, R.M. Ramstad, and J.B. Manzano Franco. 2025. "Bridging Python to Silicon: the SODA Toolchain."

Presented by A. Tumeo at DOE/MEXT collaboration monthly meeting, Online Conference, Washington.

8. Ramstad R.M., N. Bohm Agostini, and A. Tumeo. "Accelerating Scientific Workflows with LLM-Driven Compiler Optimizations for Generated High-Performance Hardware." Presented by R.M. Ramstad at the International Conference for High Performance Computing, Networking, Storage, and Analysis, Saint Louis, Missouri (poster).

9. Neff R.W., M. Minutoli, and M.M. Halappanavar. 2025. "DIMPLES: Distributed Influence Maximization for Pandemic pLanning on Exascale Systems." Presented by M.M. Halappanavar at The Chesapeake Large-Scale Analytics Conference (CLSAC), October 6-9, Annapolis, Maryland.

10. Bohm Agostini N. 2025. "SODA: Bridging Python to Silicon." The Chesapeake Large-Scale Analytics Conference (CLSAC), October 6-9, Annapolis, Maryland.

11. Thrall A.R. 2025. "Global Routing and the Steiner Tree Problem." Department of Mathematics and Statistics, Washington State University, Pullman, Washington.

12. Bohm Agostini N. 2025. "Invited Talk at Jacques Pienaar's Google Reading Group “SODA: SOftware Defined Accelerators Toolchain." Presented by N. Bohm Agostini at Jacques Pienaar's Google Reading Group, Online, Washington.

13. Tumeo A., N. Bohm Agostini, A.M. Limaye, V.G. Castellana, M. Minutoli, J.B. Manzano Franco, and R.M. Ramstad. 2025. "Bridging Python to Silicon: the SODA Toolchain." Presented by A. Tumeo at ARCHIDE Workshop, Berkeley, California.

14. Ang J.A. 2025. "Generative AI for Chip Design: Are We There Yet?" Presented by J.A. Ang at Supercomputing 2025 - Panel: Generative AI for Chip Design: Are We There Yet? Saint Louis, Missouri.

4.3 Tutorials

1. Tutorial on SODA Synthesizer at the International Symposium on Computer Architecture (ISCA) 2025: https://hpc.pnl.gov/SODA/tutorials/2025/ISCA25.html

2. Tutorial on SODA Synthesizer at the Fast Machine Learning for Science (FastML) 2025 conference: https://hpc.pnl.gov/SODA/tutorials/2025/fastml.html

3. Tutorial on Soda Synthesizer at the Symposium on Parallel Architectures and Compilation Techniques (PACT) 2025: https://hpc.pnl.gov/SODA/tutorials/2025/PACT25.html

Pacific Northwest National Laboratory

902 Battelle Boulevard
P.O. Box 999
Richland, WA 99354

1-888-375-PNNL (7665)

www.pnnl.gov