Open Tools for NPU Development Lifecycle

NexNPU Tutorial @ ISCA'26

Yuqi Xue

Ph.D. Student, Electrical & Computer Engineering,

Systems Platform & Intelligence Lab @ University of Illinois Urbana-Champaign

yuqixue2@illinois.edu

The Open Tools for NPU Development

Architectural Modeling:
1. Quick (and often cheap) early exploration of design trade-offs
2. Verify the theoretical benefits of a new idea
3. Evaluate a design choice before full-stack implementation
4. Develop cost models for other downstream tasks

Hardware Prototypes:
1. Validate functional correctness of a design
2. Measure realized benefits with consideration of physical implementation constraints
3. Develop and validate software stack based on the hardware platform

Compiler Infrastructure:
1. Map real workloads to the hardware
2. Perform optimizations to best utilize hardware
3. Provide reusable optimizations across variations of hardware architectures
4. Integrate as a backend for popular ML frameworks (e.g., PyTorch) to support diverse ML workloads

Taxonomy of Architectural Simulators for NPUs

Simulation Scope: What should we simulate?
- Important components only vs. whole NPU core/chip
- NPU-only vs. whole system (host+NPU)
- Key workload behaviors vs. end-to-end workload

Modeling Methodology: How should we simulate?
- Analytical model
- Profile-based model
- Instruction-level (microarchitectural) simulation
- …

Target Architecture:
- GPU or GPU-like
- Spatial/dataflow architecture
- Systolic array-based NPU
- Others ...

Target Workload:
- Generic ML
- Training vs. Inference
- Single forward pass vs. whole workload
- LLM serving
- …

Output Statistics:
- Performance, utilization, area, energy, power, carbon, …

Major Use Cases:
- Chip parameter sweep
- Algorithm validation
- Pre-RTL design validation
- …

Examples of NPU Simulators

Tool	Modeling Methodology	Simulation Scope		Target Architecture	Target Workload	Major Use Cases
		HW	SW			
Timeloop+Accelergy, MAESTRO, ZigZag	Analytical	Whole chip	Dataflow mapping policies	Spatial/dataflow accelerators	Generic ML	Mapping/dataflow search, sparsity exploration
SCALE-Sim	Analytical	Systolic array, SRAM, DRAM	Input/output tensor shapes	Systolic-array-based NPUs	GEMM & Conv	Array-shape and dataflow sweeps, bandwidth analysis
ASTRA-sim	Analytical	Whole system (network-focused)	Collective communication patterns	Distributed GPU/TPU/NPU systems	Distributed ML training	Parallelism studies, network topology studies
LLMCompass	Analytical+ Profile-based	Whole chip	LLM operators	GPU-like and TPU-like accelerators	LLM inference	Prefill-vs-decode tradeoff studies, Chip parameter sweep.
NeuSim	Analytical	Whole chip	tensor operators	Systolic array-based NPUs	LLM, DLRM	LLM parallelism & chip parameter sweep, energy/carbon analysis.
PyTorchSim	Tile-level simulation	While chip	PyTorch-compiled graph	GPU-like and TPU-like accelerators	PyTorch programs	Microarchitectural studies, compiler-architecture co-design.
Accel-Sim	Microarchitectural simulation	Whole chip	CUDA kernel PTX/SASS	NVIDIA GPUs	CUDA kernels	GPU kernel performance evaluation, microarchitectural exploration
gem5-Aladdin + SMAUG	Microarchitectural simulation	Whole SoC + system stack	End-to-end system	Custom accelerator SoCs	Generic ML	SoC design for custom accelerators

1. No single simulator fits every research question.

Examples of NPU Simulators

Tool	Modeling Methodology	Simulation Scope		Target Architecture	Target Workload	Major Use Cases
		HW	SW			
Timeloop+Accelergy, MAESTRO, ZigZag	Analytical	Whole chip	Dataflow mapping policies	Spatial/dataflow accelerators	Generic ML	Mapping/dataflow search, sparsity exploration
SCALE-Sim	Analytical	Systolic array, SRAM, DRAM	Input/output tensor shapes	Systolic-array-based NPUs	GEMM & Conv	Array-shape and dataflow sweeps, bandwidth analysis
ASTRA-sim	Analytical	Whole system (network-focused)	Collective communication patterns	Distributed GPU/TPU/NPU systems	Distributed ML training	Parallelism studies, network topology studies
LLMCompass	Analytical+ Profile-based	Whole chip	LLM operators	GPU-like and TPU-like accelerators	LLM inference	Prefill-vs-decode tradeoff studies, Chip parameter sweep.
NeuSim	Analytical	Whole chip	tensor operators	Systolic array-based NPUs	LLM, DLRM	LLM parallelism & chip parameter sweep, energy/carbon analysis.
PyTorchSim	Tile-level simulation	While chip	PyTorch-compiled graph	GPU-like and TPU-like accelerators	PyTorch programs	Microarchitectural studies, compiler-architecture co-design.
Accel-Sim	Microarchitectural simulation	Whole chip	CUDA kernel PTX/SASS	NVIDIA GPUs	CUDA kernels	GPU kernel performance evaluation, microarchitectural exploration
gem5-Aladdin + SMAUG	Microarchitectural simulation	Whole SoC + system stack	End-to-end system	Custom accelerator SoCs	Generic ML	SoC design for custom accelerators

2. Different simulation details & scopes are required for different design stages.

Examples of NPU Simulators

Tool	Modeling Methodology	Simulation Scope		Target Architecture	Target Workload	Major Use Cases
		HW	SW			
Timeloop+Accelergy, MAESTRO, ZigZag	Analytical	Whole chip	Dataflow mapping policies	Spatial/dataflow accelerators	Generic ML	Mapping/dataflow search, sparsity exploration
SCALE-Sim	Analytical	Systolic array, SRAM, DRAM	Input/output tensor shapes	Systolic-array-based NPUs	GEMM & Conv	Array-shape and dataflow sweeps, bandwidth analysis
ASTRA-sim	Analytical	Whole system (network-focused)	Collective communication patterns	Distributed GPU/TPU/NPU systems	Distributed ML training	Parallelism studies, network topology studies
LLMCompass	Analytical+ Profile-based	Whole chip	LLM operators	GPU-like and TPU-like accelerators	LLM inference	Prefill-vs-decode tradeoff studies, Chip parameter sweep.
NeuSim	Analytical	Whole chip	tensor operators	Systolic array-based NPUs	LLM, DLRM	LLM parallelism & chip parameter sweep, energy/carbon analysis.
PyTorchSim	Tile-level simulation	While chip	PyTorch-compiled graph	GPU-like and TPU-like accelerators	PyTorch programs	Microarchitectural studies, compiler-architecture co-design.
Accel-Sim	Microarchitectural simulation	Whole chip	CUDA kernel PTX/SASS	NVIDIA GPUs	CUDA kernels	GPU kernel performance evaluation, microarchitectural exploration
gem5-Aladdin + SMAUG	Microarchitectural simulation	Whole SoC + system stack	End-to-end system	Custom accelerator SoCs	Generic ML	SoC design for custom accelerators

3. Architectural design space exploration often requires cross-stack simulation.

Examples of NPU Simulators

Tool	Modeling Methodology	Simulation Scope		Target Architecture	Target Workload	Major Use Cases
		HW	SW			
Timeloop+Accelergy, MAESTRO, ZigZag	Analytical	Whole chip	Dataflow mapping policies	Spatial/dataflow accelerators	Generic ML	Mapping/dataflow search, sparsity exploration
SCALE-Sim	Analytical	Systolic array, SRAM, DRAM	Input/output tensor shapes	Systolic-array-based NPUs	GEMM & Conv	Array-shape and dataflow sweeps, bandwidth analysis
ASTRA-sim	Analytical	Whole system (network-focused)	Collective communication patterns	Distributed GPU/TPU/NPU systems	Distributed ML training	Parallelism studies, network topology studies
LLMCompass	Analytical+ Profile-based	Whole chip	LLM operators	GPU-like and TPU-like accelerators	LLM inference	Prefill-vs-decode tradeoff studies, Chip parameter sweep.
NeuSim	Analytical	Whole chip	tensor operators	Systolic array-based NPUs	LLM, DLRM	LLM parallelism & chip parameter sweep, energy/carbon analysis.
PyTorchSim	Tile-level simulation	While chip	PyTorch-compiled graph	GPU-like and TPU-like accelerators	PyTorch programs	Microarchitectural studies, compiler-architecture co-design.
Accel-Sim	Microarchitectural simulation	Whole chip	CUDA kernel PTX/SASS	NVIDIA GPUs	CUDA kernels	GPU kernel performance evaluation, microarchitectural exploration
gem5-Aladdin + SMAUG	Microarchitectural simulation	Whole SoC + system stack	End-to-end system	Custom accelerator SoCs	Generic ML	SoC design for custom accelerators

4. Existing tools share many similarities but are fragmented.

Summary & Takeaways for NPU Architectural Simulators

1. No single simulator fits every research question.

2. Different simulation details & scopes are required for different design stages.

3. Architectural design space exploration often requires cross-stack simulation.

4. Existing tools share many similarities but are fragmented.

It is desirable to have an integrated simulator toolchain covering different simulation scopes and design requirements.

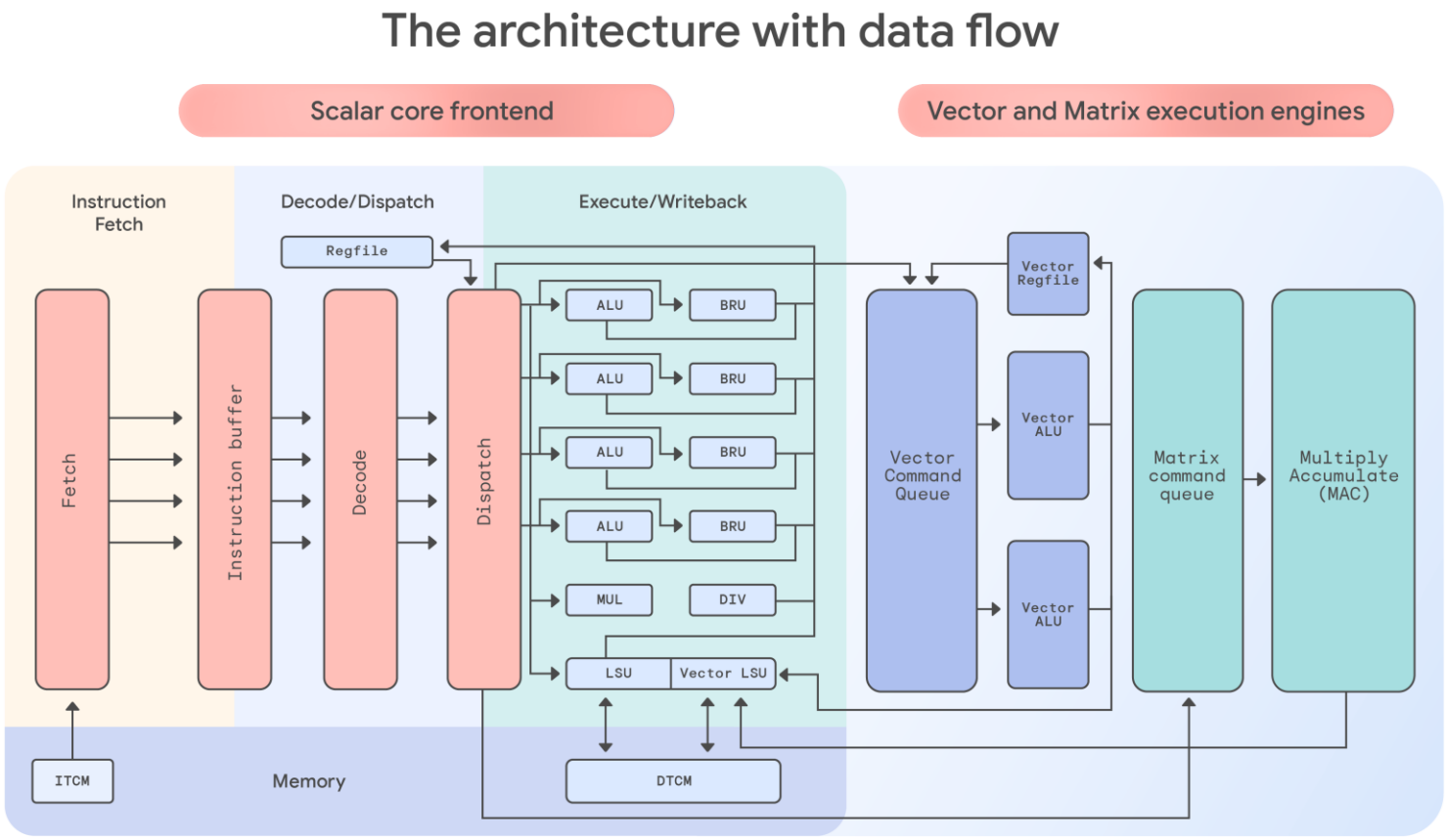
Open-Source NPU Hardware Prototypes

Type	Representative Tools	Scope	Software Support for NPU
Programmable/reusable NPU Core IP	CoralNPU, Gemmini, NVDLA, VTA, NEureka	RTL simulation, FPGA, SoC-integrable RTL	Bare-metal or runtime + C/assembly programming
Model-to-hardware generators	FINN, hls4ml	Model-specialized FPGA bitstream, HLS/RTL design	ML Framework export graph → HLS compiler
Instruction/operator co-design	CFU-Playground	FPGA soft SoC with new instructions	Runtime + C/assembly programming
Generic tapeout infrastructure	Chipyard/Hammer, OpenROAD, OpenLane	Physical implementation (GDS)	N/A

Existing NPU hardware prototypes lack end-to-end ML compiler & framework support; they diverge in ISAs and hardware implementations, even though the core architectures are similar.

The CoralNPU Core: An Example Open-Source NPU RTL Prototype

- RISC-V ISA with vector extension & custom systolic array extension
- Frontend: 4-stage, in-order dispatch, out-of-order retire
- Backend: 4-way scalar, 2-way vector dispatch
- Vector Unit: 128-bit/256-bit SIMD
- Systolic Array: N/A (not public)
- SRAM: 8 KB instruction, 32 KB data
- Standard RISC-V compiler support
- No ML compiler/framework integration



*https://github.com/google-coral/coralnpu

Open-Source ML Compiler Frameworks

Compiler Framework	Graph-Level	Operator-Level	Device-Level	Required work for supporting NPU backend
MLIR / LLVM	Partially supported	Supported	Partially supported	Graph-level optimizations, custom dialect in LLVM, device-specific optimizations, NPU driver/runtime
TVM	Supported	Supported	Partially supported	NPU codegen, device-specific optimizations, NPU driver/runtime
OpenXLA	Supported	Partially supported	Unsupported	Full NPU backend compiler, NPU driver/runtime
IREE	Supported	Supported	Partially supported	NPU codegen, device-specific optimizations, NPU driver/runtime
PyTorch / TorchInductor	Supported	Supported	Partially supported	Full NPU backend compiler, NPU driver/runtime

Support Matrix of Open-Source ML Compiler Frameworks for NPU Architecture

To What Extent Do Existing ML Compilers Support NPU Architecture?

Hardware-independent = directly reusable for NPUs.
Hardware-aware = functionally valid on any hardware, but optimality relies on hardware characteristics.
Hardware-specific = depends on explicit hardware architecture / specification / ISA.

	Graph-Level Passes	Operator-Level Passes	Device-Level Passes
Hardware-independent	shape / type inference, constant folding, CSE, DCE, canonicalization, simplification	algebraic simplification, op decomposition, loop simplification / CSE	dependence & liveness analysis, execution-order legality
Hardware-aware	fusion / fission decisions, layout propagation, quantization planning, sharding / partitioning	tiling / loop reorder, layout transformation, vectorization, autotuning	memory-space assignment engine placement overlap / software pipelining collective selection
Hardware-specific	backend-specific legalization, custom-op replacement, topology-specific shard rules	tensorization / intrinsic matching, DMA double buffering, custom microkernel selection	SRAM-bank allocation, reg alloc + VLIW / issue, ISA lowering / binary emission

Representative Optimization/Transformation/Analysis Passes in an ML Compiler

Summary & Takeaways for Open-Source ML Compiler Frameworks

1. Existing graph- and operator-level compilers are broadly reusable for NPUs.

2. However, these ML graph compilers still need new NPU backends, including codegen, device-specific passes, and runtime integration.

3. Graph/operator-level optimizations may still require NPU-specific strategies for optimality.

A shared NPU device abstraction that covers all major components in an NPU architecture (SA, VU, SRAM, HBM, ICI) is desirable to enable the reuse of NPU-specific optimization strategies across diverse hardware implementations.

References

Architectural Modeling & Simulation

[1] A. Parashar et al., “Timeloop: A Systematic Approach to DNN Accelerator Evaluation,” ISPASS 2019.

[2] Y. N. Wu et al., “Accelergy: An Architecture-Level Energy Estimation Methodology for Accelerator Designs,” ICCAD 2019.

[3] H. Kwon et al., “Understanding Reuse, Performance, and Hardware Cost of DNN Dataflows: MAESTRO,” MICRO 2019.

[4] L. Mei et al., “ZigZag: A Memory-Centric Rapid DNN Accelerator DSE Framework,” arXiv:2007.11360, 2020.

[5] R. Raj et al., “SCALE-Sim v3,” ISPASS 2025.

[6] W. Won et al., “ASTRA-sim 2.0,” arXiv:2303.14006, 2023.

[7] H. Zhang et al., “A Hardware Evaluation Framework for LLM Inference (LLMCompass),” arXiv:2312.03134, 2023.

[8] W. Yang et al., “PyTorchSim,” MICRO 2025.

[9] M. Khairy et al., “Accel-Sim,” ISCA 2020.

[10] Y. S. Shao et al., “Aladdin,” ISCA 2014; S. Xi et al., “SMAUG,” ACM TACO 2020.

Hardware Prototypes & Implementation

[11] Google, CoralNPU open RTL and software: github.com/google-coral/coralnpu.

[12] H. Genc et al., “Gemmini: Enabling Systematic Deep-Learning Architecture Evaluation via Full-Stack Integration,” DAC 2021.

[13] NVIDIA, NVDLA Open Hardware and Documentation: nvdla.org.

[14] T. Moreau et al., “A Hardware-Software Blueprint for Flexible Deep Learning Specialization (VTA),” arXiv:1807.04188.

[15] PULP Platform, NEureka: github.com/pulp-platform/neureka.

[16] Y. Umuroglu et al., “FINN: A Framework for Fast, Scalable Binarized Neural Network Inference,” FPGA 2017.

[17] J. Duarte et al., “Fast Inference of Deep Neural Networks in FPGAs for Particle Physics (hls4ml),” JINST 2018.

[18] Google, CFU-Playground: github.com/google/CFU-Playground.

[19] A. Amid et al., “Chipyard: Integrated Design, Simulation, and Implementation Framework for Custom SoCs,” IEEE Micro 2020; Hammer docs.

[20] OpenROAD and OpenLane 2 project documentation: openroad.readthedocs.io; openlane2.readthedocs.io.

Compiler Infrastructure

[21] C. Lattner and V. Adve, “LLVM: A Compilation Framework for Lifelong Program Analysis & Transformation,” CGO 2004.

[22] C. Lattner et al., “MLIR: Scaling Compiler Infrastructure for Domain-Specific Computation,” CGO 2021.

[23] T. Chen et al., “TVM: An Automated End-to-End Optimizing Compiler for Deep Learning,” OSDI 2018.

[24] OpenXLA Project documentation: openxla.org.

[25] IREE Project documentation: iree.dev.

[26] J. Ansel et al., “PyTorch 2: Faster ML Through Dynamic Python Bytecode Transformation and Graph Compilation,” ASPLOS 2024.

[27] PyTorch compiler / TorchInductor documentation: docs.pytorch.org.