IRONSmith: A Visual Dataflow Design Environment for AMD Ryzen AI NPUs

Brock Sorenson, Samer Ali, Curt Bansil, Aman Arora
Arizona State University
{btsorens, swali6, cbansil, aman.kbm}@asu.edu

Abstract

Machine learning inference increasingly relies on specialized hardware accelerators for throughput and power efficiency. Neural Processing Units (NPUs), such as the AMD Ryzen AI NPU, offer significant ML advantages over CPUs and GPUs, but programming them requires expertise in specialized frameworks. We present IRONSmith, the first visual dataflow design environment for programming AMD Ryzen AI NPUs. IRONSmith provides an interactive canvas displaying the AI Engine tile grid as visually connected blocks, allowing users to design ML dataflow applications by connecting tiles with wires representing FIFOs, split/join patterns, broadcast connections, and DDR transfers without writing any code. Compute kernels are assigned from a pre-built library, and worker functions are configured through property panels. IRONSmith's backend pipeline automatically translates the visual design into executable IRON Python, handling structural completion, import resolution, and dependency management automatically. Generated code executes directly on the AMD Ryzen AI NPU. We demonstrate IRONSmith across ML designs of increasing complexity, from a single-tile vector passthrough to multi-tile matrix operations to a complete Multi-Layer Perceptron, all designed visually and successfully executed on the AMD Ryzen AI NPU. IRONSmith serves educators, students, ML researchers, and engineers by bridging the gap between ML knowledge and NPU programming expertise, widening access to hardware that is rapidly becoming standard across consumer and enterprise devices.

1 Introduction

Neural Processing Units (NPUs) have emerged as a critical hardware platform for efficient ML inference at the edge, offering significant throughput-per-watt advantages over CPUs and GPUs. The AMD Ryzen AI NPU is rapidly becoming standard across consumer laptops - yet programming it requires deep expertise in IRON Python and mlir-aie, a tile-level spatial programming model requiring manual specification of inter-tile FIFO buffers, runtime data sequences, and compute tile worker functions. We present IRONSmith, the first visual design environment for AMD Ryzen AI NPU programming. IRONSmith provides an interactive canvas where users wire tiles together with dataflow connections and configure each component through properties panels - all without writing code. Structural verification catches design errors before code generation, and the IRONSmith backend automatically produces complete, executable IRON Python. IRONSmith serves as both a learning tool for students in NPU architecture courses and an onboarding accelerator for ML developers beginning work with the Ryzen AI NPU.

2 Related Work

The mlir-aie project (1) provides the open-source compiler infrastructure and the AMD IRON tutorial (2), which teach the programming model rather than lowering its barrier. AMD's Riallto (3) was the closest prior accessibility effort, offering Jupyter notebook tutorials for the spatial programming model, but was discontinued and never provided a visual canvas, design verification, or automated code generation. IRONSmith directly addresses the gap Riallto left.

Graphical programming has well-established precedent for lowering hardware entry barriers: Balid and Abdulwahed (4) showed LabVIEW-based dataflow programming reduced FPGA development

lifecycles by up to $5\times$, while Kuon et al. (5) and Winzker and Schwandt (6) demonstrated measurable student learning improvements with visual hardware tools. Commercial tools such as Matlab Simulink (7), VisualApplets (8), and AMD Vivado Block Design (9) apply this paradigm to FPGAs. For GPUs, NVIDIA Nsight (10) provides execution visualization but no design entry canvas. AMD Vitis (11) visualizes AI Engine mapping for Versal devices but requires C++ kernel authoring. No existing tool provides a visual design canvas for AMD Ryzen AI NPU programming. Ragan-Kelley et al.'s Halide (12) established the principle of separating algorithm description from lowering and scheduling - a principle IRONSmith extends to the NPU domain, with the IRONSmith backend pipeline handling all scope, ordering, and dependency management automatically.

3 Background

The AMD Ryzen AI NPU is built on the XDNA architecture: a spatial array of Shim tiles (DDR interface), Memory tiles (on-chip buffer storage), and Compute tiles (VLIW vector processors), connected exclusively through statically configured ObjectFifo buffers whose depths, types, and DMA channel counts must be resolved at design time. IRON Python (2) programs this hardware by declaring ObjectFifos between tiles, defining runtime sequences that fill and drain DDR tensors through shim DMA engines, and implementing Compute tile Worker functions whose acquire/kernel-call/release loops must exactly match the declared ObjectFifo element types - meaning even a moderately complex design requires manually coordinating dozens of interdependent FIFOs, split/join patterns, and worker configurations.

The fundamental challenge is that IRON Python expresses an inherently spatial programming model - a graph of tiles, FIFOs, and data movements - as a sequential text file, requiring the programmer to mentally track tile connectivity, hardware requirements, and dataflow across hundreds of lines of code. Structural errors such as disconnected FIFOs, DMA channel oversubscription, or kernel argument mismatches produce compiler errors that are difficult to diagnose without a visual representation of the design. IRONSmith eliminates this mismatch by making the spatial programming model directly visual, and catches the most common structural errors before code generation.

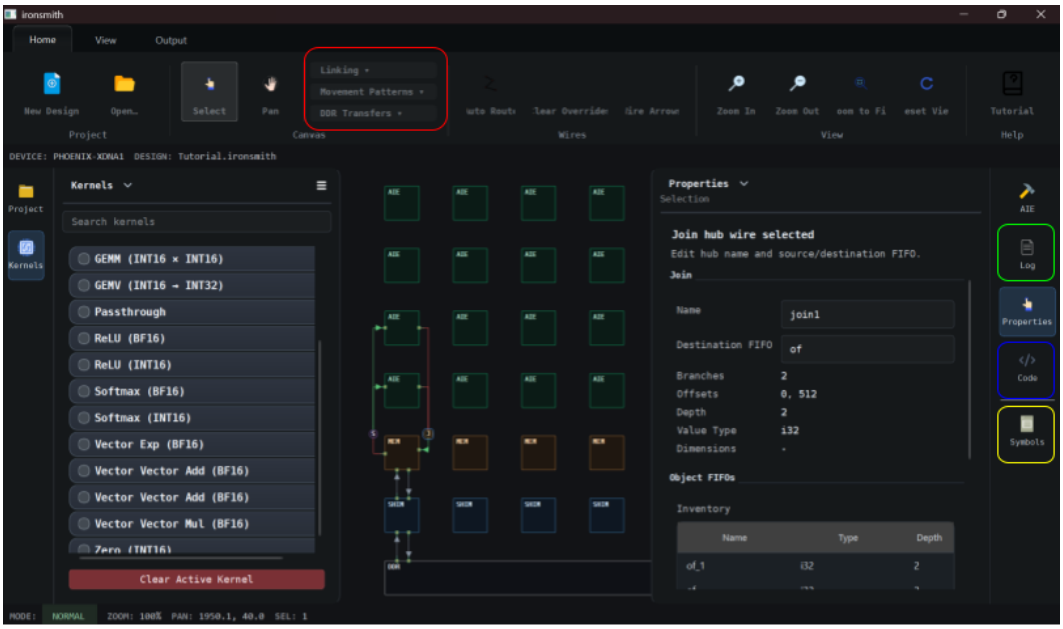

Figure 1: Annotated IRONSmith GUI overview showing the AI Engine tile grid canvas (center), FIFO wire connected tiles, kernel library (left), properties panel (right), data movement patterns (red), generated code panel (blue), symbol table (yellow), and output log (green).

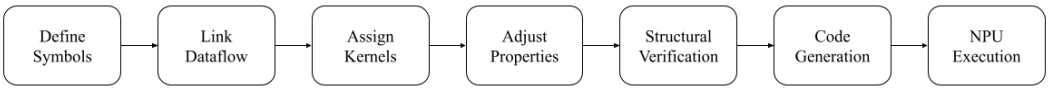

Figure 2: IRONSmith design workflow

4 IRONSmith

IRONSmith addresses the NPU programming barrier by replacing the text-based IRON Python workflow with a visual, interactive canvas (Figure 1). Users design NPU dataflow applications visually on a canvas and then generate executable IRON Python without writing a single line of code. IRONSmith is built on a microkernel plugin architecture in which a lightweight Core System manages the application lifecycle while modular plugins - Canvas, Project Explorer, Code Editor, etc. - contribute functionality through a central ExtensionSystem service registry, allowing new visual features or backend targets to be added without modifying the core. IRONSmith is architected with scalability in mind - there is a frontend (the visual user-facing canvas) and a backend (the compiler that lowers the visual design into code).

Building a design in IRONSmith follows a fixed seven-step workflow (Figure 2): users *define symbols* in the Symbol Table, *link dataflow* by wiring runtime, FIFOs, and other patterns between the canvas tiles, *assign kernels* to each compute tile from the Kernel Selection Library, and *adjust properties* - FIFO parameters, runtime sequences, worker functions, etc. - through the properties panel. Once the design is complete, users trigger *structural verification*, then *code generation*, producing executable IRON Python ready for *NPU execution* on the AMD Ryzen AI NPU.

4.1 Canvas and Visual Design Layer

The canvas presents the AMD AI Engine tile grid as an interactive layout where users connect tiles with typed wires representing the full set of dataflow patterns: ObjectFifo (tile-to-tile), Split/Join (sub-FIFO memory tile access), Broadcast (one-to-many), and Forward (pass-through). A context-sensitive Properties Panel exposes all underlying parameters - FIFO depth, element type, dimensions, DDR sizes, Tensor Access Patterns, Kernel assignments, and Worker acquire/call/release sequences - through structured form fields. A Symbol Table registers named constants and type abstractions for consistent reuse across the design. A searchable kernel library sourced from mlir-aie (1) lets users assign pre-built ML kernels (GEMM, ReLU, CONV2D, Softmax, etc.) per compute tile. Custom user-defined kernels can also be assigned to tiles through the GUI. Designs are persisted automatically to a `document.json` bundle on every change.

4.2 Design Verification and Code Generation

Before generating code, IRONSmith runs structural checks catching the most common IRON Python error classes: invalid runtime I/O configuration, disconnected dataflow, DMA channel limit violations, and kernel argument mismatches - each producing an actionable output log message rather than a cryptic compiler failure.

The IRONSmith backend takes the `document.json` (the auto-saved persistent canvas representation) and instantiates a High-Level Intermediate Representation (HLIR): an in-memory builder API that constructs typed component objects for each tile, FIFO, kernel assignment, and runtime sequence in the design. The HLIR is then lowered through a staged pipeline - to a `.xml` file representation, then a semantically directed graph, and finally a complete, executable IRON Python file - with each stage resolving interdependencies, inferring missing values, enforcing IRON coding conventions, and determining the correct import set automatically.

5 Demonstration

5.1 Benchmark Suite

We evaluate IRONSmith across three dimensions: correctness (proper execution of generated IRON Python), expressiveness (IRON dataflow patterns represented), and productivity (efficiency of design implementation). We developed seven benchmark designs of increasing complexity, each built visually in IRONSmith and executed successfully on the AMD Ryzen AI NPU (Table 1), progressing from a single-tile passthrough through element-wise vector operations to dense linear algebra with split/join and broadcast patterns. All seven benchmarks generated correct IRON Python that executed on the hardware NPU without modification. The benchmark suite exercises every major ObjectFifo pattern supported by the IRON Python model.

5.2 MLP Neural Network: Complex Use Case

To demonstrate IRONSmith's capability for complex, multi-layer ML designs, we implemented a complete Multi-Layer Perceptron (MLP) using the visual canvas. The MLP consists of 4 layers: 3 hidden layers each performing a 64×64 matrix multiply followed by a ReLU activation, and one output layer performing a 64×64 matrix multiply followed by a Softmax activation. All weight

Table 1: IRONSmith benchmark suite executed on AMD Ryzen AI NPU.

Benchmark	Shim	Mem	Compute	FIFOs	Capabilities Demonstrated
Passthrough	1	0	1	2	ObjectFifo, linear fill/drain
Vector Exp	1	1	1	2	Kernel selection, single worker
Vec-Vec Add	1	1	4	10	Split/Join mem tile, multiple workers
Vec-Vec Mul	2	2	8	20	Multi-shim, multi-column runtime
GEMV	2	2	8	24	Broadcast, matrix/vector tiling
GEAM	4	4	16	60	Dual matrix inputs, 16 tiles
GEMM	4	4	16	60	Accum. in-place, custom core body, TAPs
MLP	4	4	16	42	Tensor manipulation (`dims_from/to_stream`)

matrices are 64×64 BFloat16 values (4096 elements per matrix), and the design maps onto 16 AMD AI Engine Compute tiles. The network is designed entirely within IRONSmith's canvas, with layers connected sequentially by dataflow wires and kernels assigned from the pre-built library.

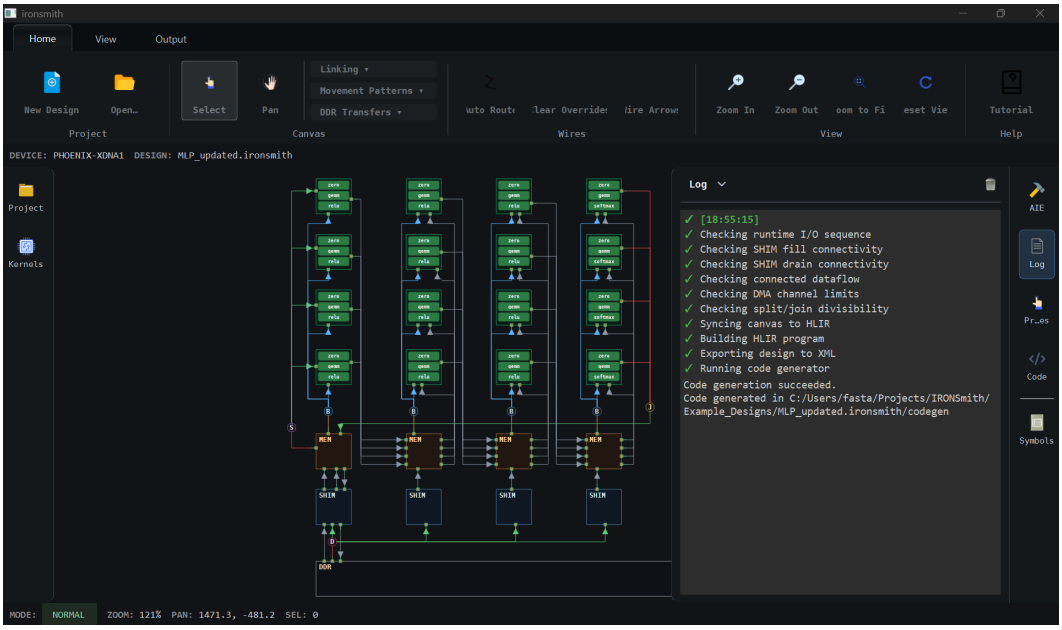

Figure 3: IRONSmith canvas view of the MLP design, showing 3 ReLU activation layers and one output layer with 4 broadcast weight matrices.

Figure 3 shows the complete MLP canvas: 16 Compute tiles with 42 inter-tile FIFOs, 4 weight broadcast connections fanning out from DDR to each tile column, and inter-layer activation connections passing through memory tiles for tensor manipulation. The design passes all structural verification checks and generates correct IRON Python that executes on the AMD Ryzen AI NPU - a design that would require approximately 200 lines of manually coordinated IRON Python code, produced here without writing any.

6 Conclusion

IRONSmith is the first visual dataflow design environment for AMD Ryzen AI NPU programming, providing a complete pipeline from graphical canvas design to automatically generated, directly executable IRON Python. We demonstrated IRONSmith across seven benchmark designs of increasing complexity - from a single-tile passthrough to GEMM - and a complete four-layer MLP, all successfully executed on the AMD Ryzen AI NPU. The benchmark suite covers every major IRON dataflow pattern, and the MLP demonstrates that IRONSmith scales to real ML applications without requiring users to write any code.

Future work includes expanding the pre-built kernel library, adding in-canvas performance profiling, and a quantitative user study measuring onboarding time against direct IRON Python programming. IRONSmith is open source (GPL v3.0) at `https://github.com/advent-lab/IRONSmith`, with user documentation and a getting-started guide available in the repository.

References

[1] Xilinx/AMD, "mlir-aie: Open-source compiler infrastructure for amd aie arrays," https://github.com/Xilinx/mlir-aie, 2024, provides IRON Python API and programming infrastructure for AMD AI Engine.

[2] AMD, "Leveraging the iron ai engine api to program the ryzen ai npu," Advanced Micro Devices, Tutorial, 2025, iSCA 2025 Tutorial. [Online]. Available: https://www.amd.com/content/dam/amd/en/documents/products/processors/ryzen/ai/iron-for-ryzen-ai-tutorial-isca-2025.pdf

[3] ——, "Riallto exploration framework," https://riallto.ai/, 2023, discontinued open-source educational framework for AMD Ryzen AI NPU programming.

[4] W. Balid and M. Abdulwahed, "A novel FPGA educational paradigm using the next generation programming languages," in *Proceedings of the IEEE Global Engineering Education Conference (EDUCON)*. IEEE, 2013, pp. 1–6, demonstrates LabVIEW graphical dataflow programming reduces FPGA development lifecycle up to 5× compared to textual HDL. [Online]. Available: https://ieeexplore.ieee.org/document/6530082

[5] I. Kuon, J. Rose, S. S. Chin, and J. Leitner, "Making fpgas accessible to scientists and engineers as domain expert software programmers with LabVIEW," 2014, graphical programming makes reconfigurable hardware accessible to domain experts without HDL background. [Online]. Available: https://arxiv.org/abs/1408.4715

[6] M. Winzker and A. Schwandt, "Open education teaching unit for low-power design and FPGA image processing," in *Proceedings of the IEEE Global Engineering Education Conference (EDUCON)*. IEEE, 2019, pp. 1–6, visual hardware instruction combined with FPGA lab access improves student learning outcomes. [Online]. Available: https://ieeexplore.ieee.org/document/9028694

[7] The MathWorks, Inc., "Simulink," https://www.mathworks.com/products/simulink.html, Natick, Massachusetts, 2024.

[8] Basler AG, "VisualApplets: Graphical FPGA programming tool," https://www.baslerweb.com/en-us/software/visualapplets/, graphical environment for FPGA programming in machine vision and embedded systems without VHDL/Verilog.

[9] AMD, "Vivado design suite," https://docs.amd.com/r/en-US/ug994-vivado-ip-subsystems/Getting-Started-with-Vivado-IP-Integrator, 2024, includes Block Design visual IP integration environment for FPGA development.

[10] NVIDIA Corporation, "NVIDIA nsight graphics," https://developer.nvidia.com/nsight-graphics, 2024, gPU kernel profiling and execution visualization tool for CUDA applications.

[11] AMD, "Vitis unified software platform," https://www.amd.com/en/products/software/adaptive-socs-and-fpgas/vitis.html, 2024, unified development environment for Versal AI Core and adaptive SoC devices.

[12] J. Ragan-Kelley, C. Barnes, A. Adams, S. Paris, F. Durand, and S. Amarasinghe, "Halide: A language and compiler for optimizing parallelism, locality, and recomputation in image processing pipelines," in *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*. ACM, 2013, pp. 519–530, separates algorithm from schedule for high-performance code generation. [Online]. Available: https://dl.acm.org/doi/10.1145/2491956.2462176