Supporting Sparse Inference in XNNPACK with RISC-V Vector Extension

Yi-Hung Chen, Hung-Yuan Chang, and Quey-Liang Kao

Andes Technology

Abstract

Leveraging sparsity in neural network weights can significantly enhance efficiency when deploying models on mobile and edge devices. However, within the RISC-V ecosystem, a complete solution for sparse inference remains unavailable. This proposal identifies the challenges in enabling sparse inference for the RISC-V Vector Extension (RVV) and presents preliminary experimental findings that highlight existing gaps. While implementation and optimization efforts are ongoing, our goal is to contribute to both the RISC-V community and the XNNPACK project.

Introduction

Neural network inference presents distinct challenges from training. While inference eliminates gradient calculations, substantial opportunities for optimization remain in model weight storage and computational patterns.

Pruning is a key technique for optimizing neural networks, reducing redundant parameters, nodes, or even entire channels. Exploiting sparsity provides several advantages: reduced storage and memory requirements, lower computation demands, and consequently, faster execution.

The widely adopted AI framework stack—including TensorFlow Lite [1] and XNNPACK [2]—offers only limited support for sparse inference. These frameworks primarily target mobile and edge deployments and conventionally use NHWC tensor formats. However, handling sparse tensors effectively requires NCHW microkernels. While this approach introduces constraints, it remains flexible enough to support most convolutional neural network applications.

Bringing this feature to RISC-V presents two key challenges. First, converting dense NHWC models into sparse NCHW representations is non-trivial. Second, optimizing XNNPACK microkernels for RISC-V platforms with RVV is an open problem.

For model conversion, TensorFlow provides official pruning tools, but our work leverages the Network Optimizer in the AndesAIRE™ NNPILOT toolkit. This tool allows users to specify target sparsity levels while continuously monitoring accuracy, following the Automated Gradual Pruning methodology [3]. Table 1 presents inference results using MobileNetV1 on a Pixel 7 Pro, demonstrating the trade-off between sparsity and performance. While full details of the tool are outside the scope of this proposal, it serves as a foundation for our exploration.

Table 1: Results of Different Sparsities on Pixel 7 Pro

	Original	AndesAIRE 75%	Official 90%
Inference Time	18415µs	15222µs	7981µs
Speedup	1x	1.21x	2.31x
Accurarcy	73.40%	72.16%	68.4%

Table 2: MobileNetV2 Profile on RVV

Original		75% Sparsity	
Operation	Time	Operation	Time
Convolution NHWC, F32 GEMM	940ms	Convolution NCHW, F32 Conv2D HWC2CHW	515ms
		Convolution NCHW, F32 SPMM	1324ms
		Fully Connected NC, F32 GEMM	6ms
Convolution NHWC, F32 DWConv	285ms	Convolution NCHW, F32 DWConv	1241ms

Our primary focus is addressing the second challenge: optimizing XNNPACK microkernels for RVV. At present, XNNPACK includes a general sparse inference framework that allows inference on RISC-V CPUs using scalar microkernels, leaving the full potential of RVV untapped. As a preliminary reference, Table 2 shows the comparison between the original MobileNetV2 model and a pruned one with 75% sparsity. Both experiments run on the same platform, a FPGA at 40MHz with an engineering AndesCore™ AX45MPV loaded. The platform supports

RVV 1.0 spec and various Andes Custom Vector extensions. The results will be further discussed in the Methodology section.

Methodologies

Microkernel Implementation

To enable sparse inference on RISC-V with the Vector Extension (RVV), we focus on key microkernels that significantly impact performance. At this stage, our work is ongoing, and most of the experimental results in Table 2 reflect the use of scalar microkernels, with the exception of our initial SPMM implementation.

Our long-term goal is to contribute optimized RVV implementations to upstream XNNPACK. Through this process, we aim to share insights from implementation, profiling, and optimization to help establish best practices for RVV-based sparse inference development.

SPMM (Sparse Matrix Multiplication)

Previous work [4] has provided an initial RVV port for sparse inference, but it largely adheres to a traditional SIMD design with fixed vector dimensions. Our approach builds on this by exploring RVV's variable-length vector capabilities while aligning with existing microkernel structures in XNNPACK, thus simplifing residual handling.

The profiling results in Table 2 show that, in the 75% sparsity configuration, SPMM accounts for 46% of the total execution time and is invoked 34 times. This aligns closely with the GEMM microkernel's role in the original dense model, where GEMM is invoked 36 times.

Our ongoing implementation is guided by a reference trial on the Pixel 7 Pro, which serves as a target benchmark. Further profiling and optimizations are in progress.

DWConv (Depthwise Convolution)

Depthwise convolutions contribute 38% of total execution time in the pruned model. We are currently developing an NCHW-variant RVV microkernel that primarily utilizes vslide and vfmacc operations as the baseline. While this proof-of-concept is still evolving, early observations suggest that RVV can provide notable efficiency improvements over scalar implementations.

Conv2D HWC2CHW

This format conversion microkernel accounts for 16% of total execution time. Given our prior experiences with mobile inference optimizations (e.g., Pixel 7 Pro experiments), we believe it is possible to further reduce its overhead to a negligible level.

Fine-grained Profiling and Optimization

The execution profiling data in Table 2 was obtained using TensorFlow Lite's benchmark_model tool. While this tool provides execution time measurements, deeper optimization requires additional performance metrics, such as cache misses and stall cycles.

Currently, benchmark_model does not provide built-in support for these advanced metrics, nor is there a widely available RISC-V profiling framework that integrates such features. Although developing a comprehensive solution is beyond the scope of this work, we are actively monitoring ongoing efforts in RISC-V performance monitoring standardization.

Internally, we have developed a manual profiling wrapper that extends benchmark_model by interfacing with perf_event system calls on Andes platforms. This enables direct collection of hardware event counters, providing deeper insights into execution behavior. While this is an interim solution, it serves as a foundation for further optimization efforts in the broader RISC-V community

References

[1] Google. TensorFlow Lite. *TensorFlow*, 2025. Available at: https://www.tensorflow.org/lite. [Accessed 2025-02-07].

[2] Google. XNNPACK. *GitHub*, 2025. Available at: https://github.com/google/XNNPACK. [Accessed 2025-02-07].

[3] Zhu, Michael, and Suyog Gupta. "To prune, or not to prune: exploring the efficacy of pruning for model compression." *arXiv preprint arXiv:1710.01878* (2017).

[4] Mihai Olinovici, XNNPACK Pull Request #7116. *GitHub*, 2025. Available at: https://github.com/google/XNNPACK/pull/7116. Accessed: [2024-02-07].