RAG Accelerator on Xilinx Versal VCK5000

We built a dense vector similarity accelerator on Versal AIE tiles to speed up Retrieval-Augmented Generation (RAG)

Team Synapse

2023122005 – **Damini Chandi Priya**
2022101090 – **Divya Ravipati**
2023102032 – **RadheShyam M**

Hardware for AI — IIIT Hyderabad

Problem & Motivation

Core Problem:

- Dense retrieval computes Dot-Product between Queries and Databases.
- On CPUs and GPUs, this step is **Memory-Bound**: the full database is repeatedly streamed from DRAM.For large indices, exact search can account for up to **97% of end-to-end inference time**.

- **Why it matters:**
  - LLMs are Compute- and Memory-Intensive to retrain.
  - RAG reduces hallucinations and improves factuality by retrieving external knowledge.
  - High-quality (Exact) Retrieval enables **smaller context** and **faster generation**.

Project Objective

To design and analyse a RAG similarity accelerator on the Xilinx Versal VCK5000 using AI Engine tiles for high-throughput dense vector search.

Target Architecture & Platform

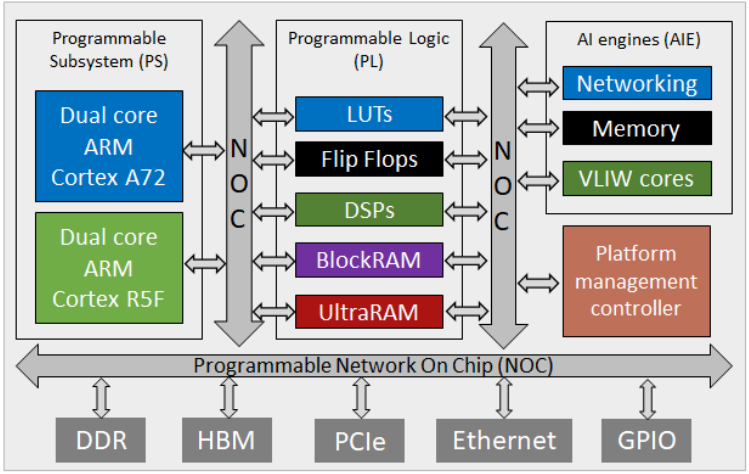


Figure: Architecture

(Source)

Target Architecture & Platform

Hardware: Xilinx Versal VCK5000

- **AI Engine array:** 400 AIE tiles
  $\Rightarrow$ 7-way VLIW cores with SIMD vector units @ 1.25 GHz.
- **Programmable Logic (PL):** Streaming, buffering, packet routing.
- **Processing System (PS):** ARM Cortex-A72 for control.
- High-bandwidth NoC + DDR with **102 GB/s** DRAM bandwidth.

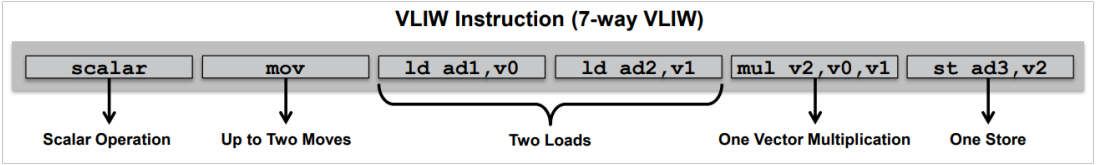


Figure: VLIW instruction **(Source)**

Software Stack

- Vitis / ADF graph-based programming for AIE kernels.
- AI Engine C++ kernels + packet streaming for multi-tile compute.

Formulation & Single-Tile Design

Core Operation: Dense Vector Similarity

$$V \in \mathbb{R}^{n \times d}, \quad Q \in \mathbb{R}^{d \times q}, \quad R = VQ \in \mathbb{R}^{n \times q}$$

$$R_{ij} = \langle V_{i,:}, Q_{:,j} \rangle$$

- **Matrix–Vector (MV):** Single query ($q = 1$) against a vector database.
- **Matrix–Matrix (MM):** Batched queries ($q > 1$) to improve data reuse.

Single-Tile Kernel

- Implemented MV/MM kernels on one AIE tile using VLIW + SIMD MACs.
- Used tiling to map matrices (e.g., $A = 128 \times 32$, $B = 32 \times 32$) onto local tile memory.
- Handled Large Databases by Blocking Vector Database

Handling Large Vector Databases

$$\underbrace{\begin{bmatrix} v_{11} & \cdots & v_{1d} \\ v_{21} & \cdots & v_{2d} \\ \hline v_{31} & \cdots & v_{3d} \\ \vdots & \ddots & \vdots \\ \hline v_{n-1,1} & \cdots & v_{n-1,d} \\ v_{n1} & \cdots & v_{nd} \end{bmatrix}}_{V} \times \underbrace{\begin{bmatrix} q_{11} & \cdots & q_{1q} \\ q_{21} & \cdots & q_{2q} \\ \vdots & \ddots & \vdots \\ q_{d1} & \cdots & q_{nq} \end{bmatrix}}_{Q} = \underbrace{\begin{bmatrix} r_{11} & \cdots & r_{1q} \\ r_{21} & \cdots & r_{2q} \\ \hline r_{31} & \cdots & r_{3q} \\ \vdots & \ddots & \vdots \\ \hline r_{n-1,1} & \cdots & r_{n-1,q} \\ r_{n1} & \cdots & r_{nq} \end{bmatrix}}_{R}$$

$$V \in \mathbb{R}^{n \times d}, \quad Q \in \mathbb{R}^{d \times q}, \quad R = VQ \in \mathbb{R}^{n \times q},$$

Experimental Analysis & Results

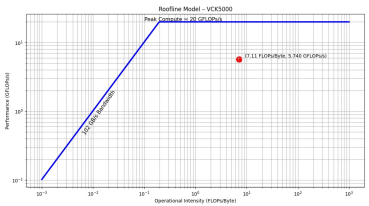
Peak Compute per AIE Tile (FP32)

$$\text{FP32}_{\text{peak}} = f_{\text{AIE}} \times 8 \times 2$$

$$= 1.25\,\text{GHz} \times 8 \times 2 \approx \mathbf{20\ GFLOP/s}$$

Measured for MM (128×32)·(32×32)

- FLOPs = 262,144
- Cycles = 56,993 (tiling 4×2×4)
- At 1.25 GHz: ≈ **5.74 GFLOP/s**
- Operational intensity ≈ **7.11** FLOPs/Byte



Roofline Analysis (Single Tile)

Memory System

- DRAM bandwidth: **102 GB/s**

(Source: Performance Analysis of GEMM Workloads on the AMD Versal Platform, ISPASS 2025)

Novelty

Novelty is!!!!!!

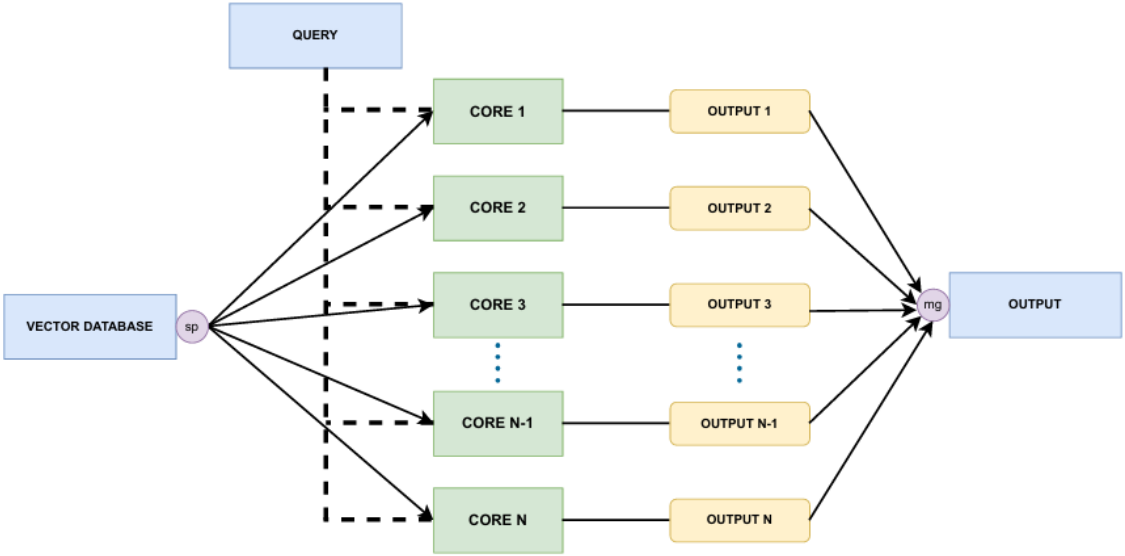
Packet-streamed Vector database delivery with broadcasted query flow for dense retrieval on the AIE array

Key Novel Points

- Treating RAG retrieval as a **GEMM** problem (MV/MM) tailored to AIE architecture.
- **Block-partitioned vector databases** across tiles with query broadcast and reduction.
- Queries are broadcast to all AIE tiles simultaneously, enabling parallel similarity computation without redundant memory transfers.
- Packet streaming enables high-throughput continuous dataflow and eliminates buffering overhead.
- Laying groundwork for a **Versal-based dense RAG accelerator** instead of approximate CPU-only ANN.

Novelty: Multiple Tile Parallelism

Packet Stream Vector Database and Broadcast Query



Inner and Outer Product

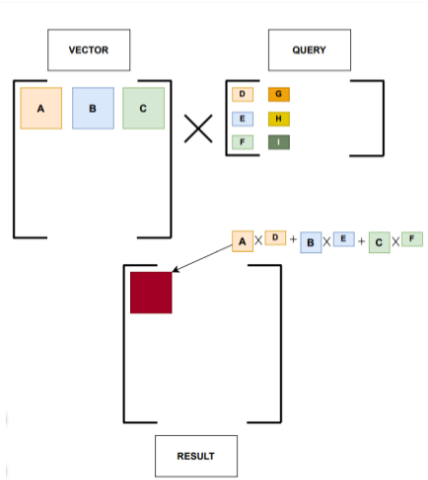


Fig. 5: Inner Product Flow

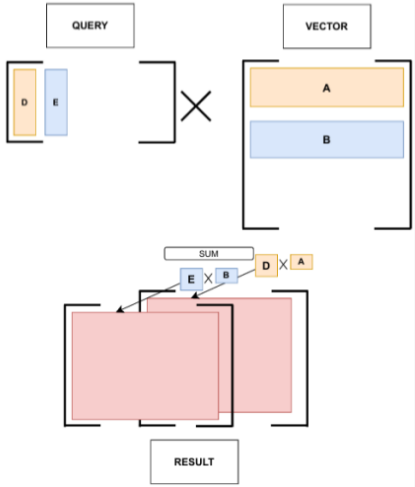


Fig. 6: Outer Product Flow

What Felt Cool

- **Roofline thinking is powerful:** we learned to read a roofline plot and understand whether we are limited by compute or memory.
- **Batching is king:** moving from matrix–vector to matrix–matrix made it obvious how much
- Increased from 5GFlops/s to 8.06GFlops/s using Multi-Tile Parallelism. **data reuse** we were previously missing.
- **Outer-Product is amazing and has a lot of scope:** since no data streaming is needed again and again
- **Working with AIEs:**
  - Got hands-on with VLIW + SIMD scheduling and tiling.
  - Saw how "paper peak" (20 GFLOP/s) translates to realistic performance.
- Cool moment: Realizing that performance gains came not from tiny micro-optimizations, but from fundamentally rethinking how data flows through the architecture.

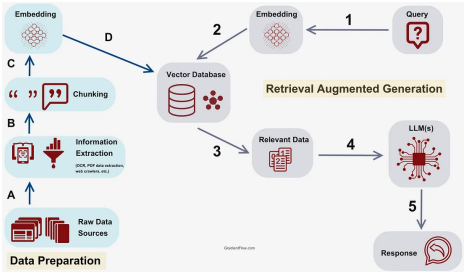
Team Synapse Signing off!

Thank you!!

Team Synapse
RAG Accelerator on Xilinx Versal VCK5000

Github Link
Paper Link

Backup: RAG Background



figureRAG Pipeline (Source)

Why RAG?

- Access external knowledge without retraining.
- Reduces hallucinations and improves factual accuracy.
- Used in modern systems like ChatGPT plugins, Copilot, Gemini, etc.

Challenges

- Large vector indices (tens of GB).
- Trade-off between exact vs. approximate nearest neighbour search.

Backup: References

- [1] Product Quantization for Nearest Neighbor Search, IEEE TPAMI, 2011.
- [2] Dense Passage Retrieval for Open-Domain Question Answering, EMNLP 2020.
- [3] Accelerating Retrieval-Augmented Generation, ASPLOS 2025 (preprint).
- [4] Performance Analysis of GEMM Workloads on the AMD Versal Platform, ISPASS 2025.

Meet the Team!

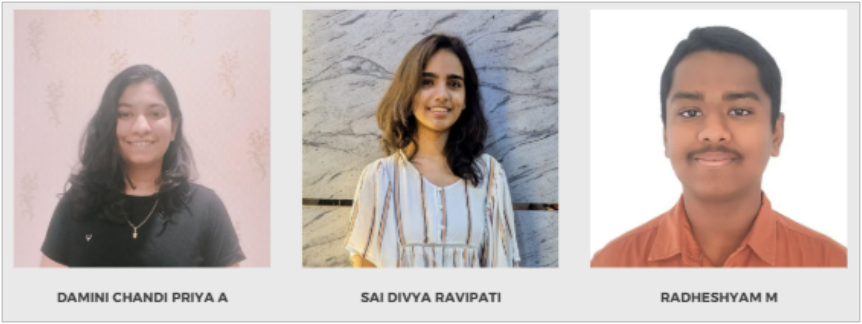


Figure: Team Synapse