SoftwareX 31 (2025) 102243

Contents lists available at ScienceDirect

SoftwareX

journal homepage: www.elsevier.com/locate/softx

Original software publication

GEMM-ArchProfiler: A simulation framework for hardware-level profiling and performance analysis of General Matrix Multiplication in real CNN workloads on heterogeneous CPU architectures

Binu Ayyappan [a,b,*], G. Santhosh Kumar [a]

[a] Cyber Physical Systems Lab, Department of Computer Science, Cochin University of Science & Technology, Kochi, India
[b] Department of Artificial Intelligence and Data Science, Rajagiri School of Engineering & Technology, Kochi, India

ARTICLE INFO

Keywords:
Hardware profiling
CNN workload performance profiling
GEMM
GEM5
Performance analysis

ABSTRACT

In this paper, the authors present GEMM-ArchProfiler, a simulation framework for evaluating General Matrix Multiplication performance in convolutional neural networks. Targeted at resource-constrained edge and IoT systems, which rely on CPU-based architectures, the framework addresses hardware limitations through optimized workload profiling. Powered by the gem5 simulator, GEMM-ArchProfiler provides insights into memory usage, cache behavior, execution latency, and energy consumption. It integrates customized Darknet libraries to simulate realistic CNN workloads and includes a user-friendly CPU configuration mechanism and event analysis script. This tool bridges workload analysis and deployment, aiding efficient AI implementation on diverse CPU architectures.

Code metadata

Current code version	v 1.0
Permanent link to code/repository used for this code version	https://github.com/ElsevierSoftwareX/SOFTX-D-25-00041
Permanent link to Reproducible Capsule	None
Legal Code License	GNU General Public License (GPL)
Code versioning system used	git
Software code languages, tools, and services used	C, Make, python, openmp, gem5
Compilation requirements, operating environments & dependencies	gem5, gcc, python3
If available Link to developer documentation/manual	https://github.com/binooa/GEMM-ArchProfiler
Support email for questions	binua.cusat@gmail.com

1. Motivation and significance

General Matrix Multiplication (GEMM) is a pivotal operation, especially in the deep learning workloads [1]. Its efficiency directly impacts the performance of tasks such as image recognition, object detection, and real-time analytics. While GEMM optimization techniques are well-established for multi-core CPUs [2–4] and GPUs [5–7], adapting these techniques to resource-constrained edge computing devices presents new challenges. Identifying bottlenecks in these processors and exploring hardware-specific optimizations [8] are critical for achieving efficient GEMM execution on edge systems, which often rely on cost-effective CPU architectures instead of more powerful GPUs.

A significant challenge in GEMM optimization lies in managing computational efficiency and data movement through the memory hierarchy [9]. Inefficient data movement can severely limit algorithm performance, especially on devices with limited cache and memory resources [2]. Techniques such as loop tiling and permutation have proven effective in reducing data movement and enhancing computational throughput. However, finding the best configurations for these optimizations requires systematic analysis and modeling, particularly across diverse processor architectures with varying constraints [10].

Convolutional Neural Networks (CNN) workloads, such as those executed by models like YOLOv4 [11], increasingly target embedded systems for real-time applications like autonomous vehicles and surveillance. These systems often operate on resource-limited platforms, such

* Corresponding author at: Cyber Physical Systems Lab, Department of Computer Science, Cochin University of Science & Technology, Kochi, India.
E-mail address: binua.cusat@gmail.com (Binu Ayyappan).

https://doi.org/10.1016/j.softx.2025.102243
Received 19 January 2025; Received in revised form 23 May 2025; Accepted 18 June 2025
Available online 28 June 2025

as CPUs or FPGAs, where memory and computational resources are constrained. Optimized GEMM implementations have been shown to significantly enhance the resource utilization of such platforms [12], enabling faster processing and more accurate results. However, deploying CNNs on embedded systems remains challenging, highlighting the need for systematic evaluation tools to address computational bottlenecks and memory inefficiencies.

The development, optimization and evaluation of GEMM algorithms pose significant challenges due to the complexities involved in their implementation and performance testing across diverse hardware configurations. GEMM serves as a critical computational kernel in deep learning workloads, particularly in CNNs, where its efficiency directly impacts the overall performance of the system. However, designing GEMM algorithms optimized for specific hardware architectures, such as CPUs, multi-core processors, or embedded systems, requires meticulous testing and analysis under realistic conditions.

A major hurdle is the need to evaluate GEMM algorithms for both single-core and multi-core executions, considering the intricate dynamics of parallelism, memory hierarchies, and processor architectures. Moreover, GEMM's role in CNN workloads introduces additional layers of complexity, as its performance cannot be assessed in isolation. Real-world CNN workloads impose diverse computational and memory access patterns, making it imperative to test GEMM algorithms within these realistic scenarios to identify bottlenecks effectively.

Another challenge lies in understanding the interactions between GEMM algorithms and the underlying hardware components. Variations in memory usage, cache behavior, data transfer latencies, and energy consumption across different architectures significantly influence algorithm performance. These interactions are often difficult to predict without detailed profiling and analysis.

Traditional approaches to GEMM evaluation, which involve independent algorithm testing on physical hardware [13], are time-consuming, costly, and limited in their ability to explore the vast design space of potential optimizations. Furthermore, they lack the flexibility to simulate hardware configurations that are unavailable or impractical to test physically.

This underscores the necessity for a systematic and comprehensive solution—one that enables the design, testing, and evaluation of GEMM algorithms within realistic CNN workloads across various hardware configurations. A robust simulation framework that can emulate diverse processor architectures and provide detailed insights into computation and memory bottlenecks is essential to bridge the gap between algorithm development and practical deployment. Such a framework would not only streamline the optimization process but also facilitate the development of efficient, hardware-aware GEMM implementations [14] for resource-constrained edge and embedded systems.

GEMM-ArchProfiler is a comprehensive simulation framework designed to study the performance analysis of GEMM algorithms across diverse CPU architectures, bridging the gap between theoretical evaluations and practical deployments. Leveraging the gem5 simulator [15–17] —a widely used tool in computer architecture research—the framework simulates and evaluates CNN workloads, integrating real-time implementations with customized Darknet open source CNN libraries [18] to drive GEMM computations. This enables detailed profiling of critical hardware-level metrics such as memory usage, cache behavior, execution latency, and energy consumption, particularly for resource-constrained environments.

The modular architecture of GEMM-ArchProfiler includes a flexible CPU configuration mechanism that allows easy customization of hardware parameters, a tailored CNN workload designed for realistic GEMM execution and performance profiling, and a custom log analyzer script to capture and evaluate key performance metrics. By delivering actionable insights into hardware performance and addressing computational bottlenecks, GEMM-ArchProfiler serves as a vital tool for designing and optimizing AI workloads for edge and embedded systems.

2. Software description

GEMM-ArchProfiler is a comprehensive software framework designed to address the challenges of evaluating and optimizing GEMM algorithms in resource-constrained environments. Traditionally, GEMM algorithms are developed and tested independently, often using stubs and drivers [19], which fail to provide a complete understanding of their performance in real-world applications. Testing these algorithms with realistic CNN workloads is essential to gain insights into how memory and cache systems influence data flow and performance parameters [20]. For instance, a basic GEMM algorithm that performs well on x86 or ARM architectures [21] may encounter bottlenecks when integrated into a real CNN workload due to memory usage patterns. This highlights the need for a robust evaluation framework like GEMM-ArchProfiler.

The framework integrates a Darknet-based CNN workload, where the GEMM source file and Makefile have been modified to include three GEMM algorithms with both single-threaded and multi-threaded capabilities. Researchers can also customize the framework to incorporate new GEMM algorithms for testing. GEMM-ArchProfiler leverages the gem5 simulation environment, enabling users to simulate and analyze GEMM performance on configurable CPU architectures. This allows for testing GEMM algorithms across different hardware configurations.

The simulation setup monitors every GEMM call within the workload, logging critical CPU events and GEMM-related metrics. Due to the detailed nature of system-level simulation, even simple CNN architectures can take several hours to simulate. To address this, the framework includes features like checkpoints and fast-forwarding, leveraging gem5's capabilities to reduce simulation time without compromising data accuracy.

Upon completing the simulation, a performance log analyzer processes the data to extract key metrics such as GEMM calls, matrix dimensions (M, N, K), cache usage, memory clock speeds, runtime, energy consumption, power usage, and operational intensity. These metrics are presented in a CSV format, providing actionable insights into the performance of GEMM algorithms under realistic workloads.

GEMM-ArchProfiler aids in identifying optimization opportunities and supports the collection of data for developing AI models that assess performance. This makes it an essential tool for researchers and developers aiming to optimize GEMM algorithms for efficient deployment in edge and embedded systems.

2.1. Software architecture

The GEMM-ArchProfiler framework is designed as a modular and extensible platform to analyze and optimize GEMM algorithms under realistic CNN workloads. The architecture integrates workload generation, configurable GEMM implementations, cycle-accurate hardware simulation, and detailed performance analysis. The system is divided into four core modules, each serving a distinct purpose in the overall profiling workflow. The detailed architecture of GEMM-ArchProfiler is given in Fig. 1.

2.1.1. CNN workload module

The first module is responsible for generating realistic GEMM calls within the flow of an actual deep learning workload. This is achieved using the Darknet library, a lightweight and efficient C-based deep learning framework. Unlike traditional approaches that rely on test stubs or synthetic drivers, this module executes complete CNN models where GEMM operations are invoked organically from convolutional and fully connected layers. This ensures that the generated memory access patterns, cache behaviors, and data dependencies accurately reflect those seen in practical AI applications. As a result, memory bottlenecks and cache-level interactions are captured in a realistic manner, forming a strong foundation for architectural evaluation.

2.1.2. GEMM algorithms module

The second module focuses on the actual GEMM algorithm implementations and their runtime configuration. It includes three different GEMM variants - a naive version, an optimized version, and a tiled version, each showcasing distinct computational strategies. These implementations are designed to be switchable between single-threaded and multi-threaded modes, helping researchers analyze the impact of thread-level parallelism on performance and memory usage. A simple preprocessor flag system enables flexible switching between implementations during compilation, allowing researchers to conduct controlled and reproducible experiments. Moreover, this module supports user-defined GEMM algorithms, which can be easily integrated into the framework and evaluated under full CNN execution. This design promotes extensibility and comparative analysis without altering the core CNN workload.

2.1.3. Architectural Simulation Module

The third module integrates the gem5 simulator to create a cycle-accurate hardware simulation environment. During execution, the CNN-driven GEMM workload is run inside gem5, enabling the capture of low-level architectural metrics such as memory latency, cache performance, instruction counts, and power consumption. The framework provides several baseline CPU configuration templates, allowing users to simulate different processor classes and customize the architectural features being evaluated. The simulation process generates two main types of logs: a GEMM-level log detailing matrix dimensions, thread usage, and GEMM identifiers; and a CPU-level log capturing execution cycles, cache access traces, energy consumption, and latency values. This module bridges algorithmic performance with architectural characteristics in a unified environment.

2.1.4. Log Analyzer module

The fourth module handles post-simulation analysis. It processes the raw logs generated by gem5 and extracts relevant metrics into a structured CSV format. The Log Analyzer module automatically extract and compile performance metrics for each GEMM operation executed during the simulation. This module reads the log files to identify the sequence of GEMM layers and their matrix dimensions. It also processes the log output from the gem5 simulator by separating it into blocks, each representing statistics for a specific GEMM call. For each layer, the module extracts useful simulation data such as floating-point operations, instruction count, memory bandwidth, and energy usage. Using this information, it calculates key performance values like runtime, FLOP rate, CPI, energy consumption, and operational intensity. The results for all GEMM layers are then compiled into a structured CSV file. This file can be used for further analysis, graphing, or machine learning-based predictions. Overall, this module helps link simulation results with actual GEMM operations in a CNN, making it easier for researchers to understand performance trends and system behavior across different CPU configurations in gem5.

The software architecture of GEMM-ArchProfiler brings together realistic CNN execution, configurable GEMM computation, detailed architectural simulation and post-simulation data processing in a cohesive and extensible framework. By leveraging Darknet for workload generation and gem5 for performance simulation, the system provides deep insights into the behavior of GEMM operations under actual deployment scenarios. Unlike traditional profilers that operate on fixed hardware, GEMM-ArchProfiler offers architecture customization, repeatable experimentation, and fine-grained analysis of GEMM performance. This makes it a powerful tool for algorithm designers and system architects working on AI acceleration for resource-constrained platforms.

2.2. Software functionalities

GEMM-ArchProfiler provides a comprehensive set of functionalities that allow researchers to evaluate the performance of GEMM algorithms within realistic CNN workloads. At its core, the software enables simulation of full CNN models where GEMM calls are naturally invoked through convolutional and fully connected layers. By integrating these calls into a working CNN flow, the software ensures that memory usage patterns, cache behaviors, and execution sequences closely resemble those encountered in real-world AI applications. This setup allows researchers to study GEMM performance under conditions that are representative of actual deployment scenarios, particularly in edge computing environments.

The software consists of four primary components that work together to support this goal. The first component, the CNN Workload Module, is built on the Darknet framework and is responsible for generating GEMM calls as part of actual model execution. The second component, the GEMM Algorithm Module, offers three different GEMM implementations that can be executed in either single-threaded or multi-threaded mode. This module is designed to be extensible—researchers can add their own GEMM implementations by modifying the source code and associated build scripts. Execution modes and algorithm selection are handled using simple preprocessor flags, allowing for controlled experimentation without altering the CNN structure.

The third component, the Architectural Simulation Module, integrates with the gem5 simulator to provide cycle-accurate performance insights at the CPU level. During simulation, gem5 captures detailed metrics such as memory latency, cache usage, energy consumption, and instruction-level execution behavior. This module supports configurable CPU parameters, enabling researchers to simulate a wide range of hardware architectures by adjusting cache sizes, memory bandwidth, and processor frequencies. Additionally, it leverages gem5's support for checkpointing and fast-forwarding to reduce simulation time and focus on regions of interest within the workload.

The final component is the Log Analysis Module, which processes the simulation output and converts raw logs into a clean and structured CSV format. This output can be used directly for performance analysis, visualization, or integration with machine learning models to predict trends and identify bottlenecks. The CSV data includes key information such as matrix dimensions, thread-level performance, cache access patterns, and energy usage, making it highly suitable for both quantitative studies and comparative analysis.

Users interact with the software primarily through command-line interfaces and configuration files. The workflow typically begins with selecting a GEMM implementation and execution mode in the Makefile, followed by compiling the Darknet-based CNN binary in static mode for compatibility with gem5. Once the simulation is complete, the logs are automatically parsed, and the resulting dataset can be analyzed using standard tools such as Python, pandas, or any CSV-compatible platform. The software is designed for Linux environments and supports gem5 full-system mode, making it suitable for profiling a wide range of CPU architectures including x86 and ARM. It requires standard open-source tools such as GNU Make, GCC, and Python for compilation and post-processing.

In typical use cases, GEMM-ArchProfiler is employed by researchers and developers who need to understand the low-level performance characteristics of GEMM algorithms in the context of deep learning models. It is particularly valuable in scenarios where memory efficiency, energy consumption, and execution latency are critical—such as in embedded or edge AI systems. The framework not only supports systematic performance evaluation but also enables reproducible experiments, architectural tuning, and early-stage design validation of AI workloads on CPU-based platforms.

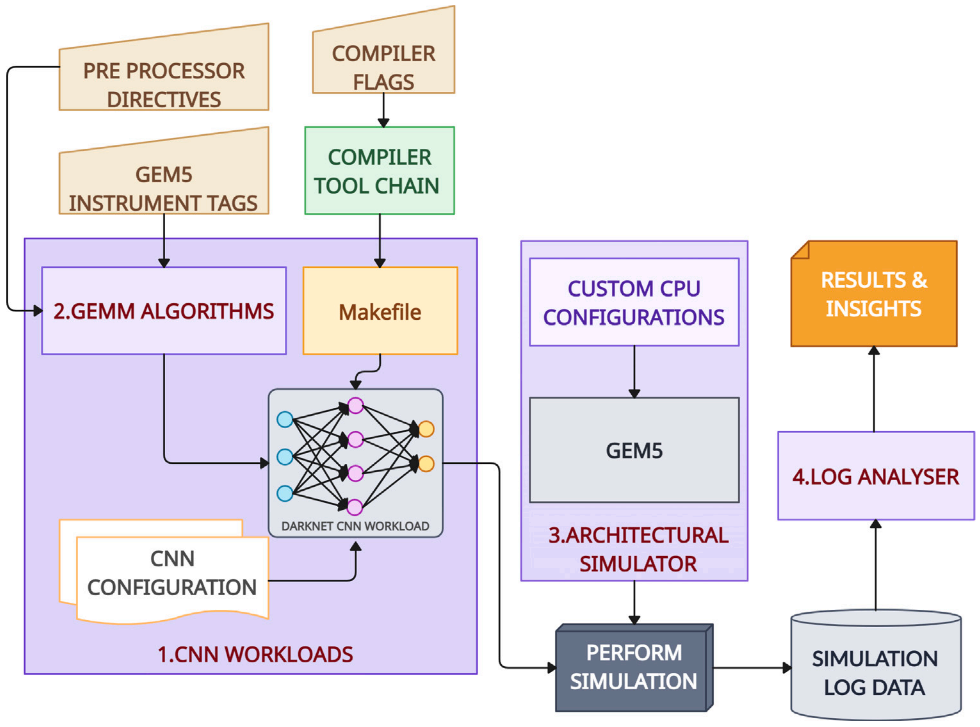

Fig. 1. Software Architecture of GEMM-ArchProfiler.

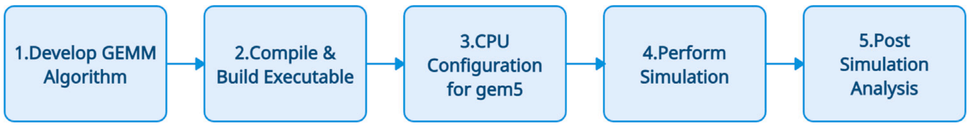
1.Develop GEMM Algorithm
2.Compile & Build Executable
3.CPU Configuration for gem5
4.Perform Simulation
5.Post Simulation Analysis

Fig. 2. Methodology for using GEMM-ArchProfiler.

3. Methodology

GEMM-ArchProfiler provides a structured approach for evaluating GEMM implementations across different architectures, integrating simulation-based profiling with real hardware validation. The methodology involves multiple stages, beginning with the independent development and validation of a GEMM implementation, followed by its integration into GEMM-ArchProfiler, compilation using a modified build system, execution within the gem5 simulation environment, and subsequent log processing and performance analysis. To ensure reliability, the profiling results are validated against real hardware execution using established profiling tools. The following subsections detail the step-by-step process of this methodology, as illustrated in Fig. 2.

3.1. Develop GEMM algorithm

Before incorporating a new GEMM implementation into GEMM-ArchProfiler, it is independently developed and tested to ensure correctness. This process includes unit testing for numerical accuracy and performance validation. Once verified, the implementation is integrated into GEMM-ArchProfiler by modifying the source file src/gemm.c. A preprocessor directive (#ifdef) is added to allow selective compilation of different GEMM implementations, ensuring flexibility in execution modes. The integration process ensures that the new implementation can be tested under different configurations while maintaining compatibility with the profiling framework.

3.2. Compile & build executable

To enable execution of the new GEMM implementation, the Makefile is modified to incorporate an additional menu selection. This modification allows users to choose the new implementation at compile time. The preprocessor directive defined in the source file is linked to the appropriate Makefile tags, ensuring proper recognition during the build process. The compilation is performed using the make utility, which generates a fully linked binary executable compatible with gem5. This executable serves as the foundation for running GEMM workloads within the profiling framework, enabling systematic performance evaluation under different hardware configurations.

3.3. CPU configuration for gem5

Once the executable is generated, the gem5 simulation environment is configured by defining processor-specific parameters such as core frequency, memory size, and cache hierarchy, in accordance with the gem5 CPU configuration specifications [22,23]. The configuration settings follow the specifications outlined in GEMMArchProfiler GitHub repository [24,25], which provides detailed guidelines for adjusting architectural parameters. GEMM workloads are embedded within three different neural network models—Darknet [26], DenseNet [27], and ResNet [28] —allowing for comprehensive evaluation across various computational structures. Depending on the study requirements, simulations can be executed on all models or a subset of them. The gem5 environment provides a controlled execution setting where different architectures can be tested under identical workload conditions, ensuring consistent comparisons.

3.4. Perform simulation

The execution process is initiated by running the simulate.sh script in the GEMM-ArchProfiler Github repository, which launches the GEMM profiling workflow in gem5. The simulation duration varies based on hardware capabilities, simulation depth, and workload complexity. To enhance efficiency, fast-forwarding and checkpointing techniques are employed. Fast-forwarding allows the simulation to skip initial non-critical phases, reducing overall execution time, while checkpointing saves intermediate system states, enabling efficient re-execution without restarting the entire simulation. Upon completion, gem5 generates two primary log files “stats.txt”, which contains low-level hardware performance metrics, and “gemm_calls.txt”, which records detailed GEMM execution statistics. These logs capture all architectural information related to execution latency, memory access behavior, and computational performance.

3.5. Post simulation analysis

Following simulation execution, the log files are processed using the “process.py”script in the GEMM-ArchProfiler Github repository, which extracts key performance metrics and compiles them into a structured CSV format. The extracted data includes GEMM layer dimensions, cache configurations, memory clock speed, execution runtime, CPI, floating point operations per second (MFLOP/s), memory bandwidth, total energy consumption, power usage, operational intensity, and theoretical peak FLOPs. The structured dataset allows researchers to analyze execution efficiency and identify computational bottlenecks across different architectures. The results are further analyzed to compare the impact of various CPU configurations and workload conditions, providing insights into optimization opportunities for GEMM execution.

4. Accuracy of gem5 based simulation on supported architectures

The simulation engine integrated into GEMM-ArchProfiler is the gem5 architectural simulator, a widely accepted tool for system-level architecture research. As a result, GEMM-ArchProfiler inherits compatibility with all processor architectures that are supported and validated by gem5. Among the prominently supported platforms are ARM, x86, x86_64, and RISC-V, along with several other modern CPU architectures [16,29–32].

Irene Wang et al. [33] present a comprehensive performance modeling and analysis framework using the gem5 simulator, specifically targeting ARM Cortex-R series CPU-based embedded System-on-Chips. The study emphasizes the relevance of ARM CPUs in embedded applications and highlights the challenges of integrating heterogeneous components under strict performance and power constraints. By tuning and validating gem5 against a real ARM SoC, the authors report a negligible average absolute error across the Embench suite [34].

Ayaz Akram et al. [35] present a detailed validation study of gem5’s x86 simulation accuracy, specifically targeting the Intel Haswell microarchitecture. Using hardware performance counters and the Linux perf tool, they compare gem5’s output against real hardware to identify inaccuracies in IPC and microarchitectural event modeling. The validation was driven by a comprehensive set of microbenchmarks that targeted key CPU subsystems.

Tuan Ta et al. [36]present a validated extension of gem5 for simulating multi-core RISC-V systems, incorporating support for thread-related system calls and synchronization primitives essential for multi-threaded workloads. The implementation was functionally validated using low-level assembly and C-based unit tests, while timing validation was performed through targeted micro-benchmarks comparing performance of key hardware units, such as multipliers, against Rocket chip baselines. Benchmarking was carried out using thirteen applications from the Ligra suite on real-world graph datasets, demonstrating the simulator’s scalability and efficiency.

5. Experimental setup and validation

The GEMM-ArchProfiler simulations were conducted on a system powered by an Intel Xeon Silver 4208 processor, running at 2.10 GHz. This system follows the x86_64 architecture and is equipped with 32 GB of DDR4 RAM operating at 3200 MHz. The processor features a three-level cache hierarchy, consisting of 512 KB of L1 cache, 8 MB of L2 cache, and 11 MB of L3 cache. The system runs on Ubuntu 22.04.5 LTS with the Linux kernel version 6.8.0-52-generic, providing a stable environment for executing and analyzing workloads.

For simulation, gem5 was configured to model an Intel Core i3-6100U CPU running at 2.30 GHz, which is based on the Intel Skylake architecture. The Intel Core i3-6100U is used in this study because it closely reflects the design and performance of many low-power CPUs commonly found in edge computing systems. The simulated system was configured with 8 GB of DRAM, reflecting a typical mid-range edge hardware environment. With its dual-core setup and 2.30 GHz base speed, it is well-suited for testing lightweight AI tasks. Its support for tools like gem5 and Likwid, along with clear technical documentation, helps ensure that the results are accurate and easy to reproduce. This makes the i3-6100U a practical and relevant choice for testing the profiling system in real-world edge AI use cases.

To validate the accuracy of the profiling results, the same workload was executed on physical hardware using Likwid [37]. The real hardware system used for this comparison was an Intel Core i3-6100U CPU @ 2.30 GHz with two cores. This system was configured with 8 GB of DRAM and ran on a lightweight Ubuntu Linux version without a graphical user interface, minimizing unnecessary background processes that could impact performance measurements. The operating system was based on the Linux kernel version 5.4.0-48, ensuring compatibility with profiling tools.

To capture accurate performance data, the system was configured to access processor performance monitoring counters (PMC) and read Model-Specific Register (MSR) values [38]. This setup allowed direct measurement of processor core activity and energy consumption. The results from the hardware-based profiling tools were systematically compared with those obtained from the gem5 simulation environment, ensuring the accuracy and reliability of GEMM-ArchProfiler in performance evaluation.

The GEMM-ArchProfiler was also tested and validated on the Intel Core i7-11370H [39], a modern processor based on the Tiger Lake architecture. As it adheres to the x86 instruction set and is fully supported by the Likwid profiling tool, this validation further demonstrates the framework’s applicability across a broad spectrum of platforms, ranging from legacy systems to current-generation hardware.

The GEMM-ArchProfiler was further evaluated in gem5 using architecture-specific CPU configurations for ARM-based platforms such as the Samsung Exynos 5422 [40] and AUP PYNQ-Z2 [41], as well as the RISC-V-based C-DAC VEGA AS4161 [42] processor. The corresponding configuration files are publicly available in the project’s GitHub repository to support reproducibility and community use. While results from these platforms were not independently validated, the underlying gem5 CPU models have been extensively verified in prior peer-reviewed studies [16,29–33,35,36], thereby supporting the reliability of simulation.

Table 1
Comparison of Profiling Tools.

Feature	GEMM-ArchProfiler	Likwid	Intel VTune	ARM DStream
Execution Model	Simulation-based	Hardware-based	Hardware-based	Hardware-based
Real Hardware Required	No	Yes	Yes	Yes
Code Instrumentation	No	Yes	No	Yes
Profiling Approach	Simulated Execution	Performance Counters	Function Call Monitoring	External Debugging
Supported Architectures	ARM, x86, x86_64, RISC-V [16]	Intel Only (x86, x86_64)	Intel Only (x86, x86_64)	ARM Only
Profiling Metrics	Execution time, Latency, Memory, Energy, Cache Usage	Execution time, Latency, Energy, Cache Usage	Execution Time, Hotspots	Execution Time, Debugging Insights
Additional Hardware	No	No	No	Yes (External Device)
Computational Overhead	High	Low	Moderate	Moderate
Multi-threading Support	Yes	Yes	Yes	Yes
Best Use Case	Architecture testing before deployment	Fine-grained performance tuning on Intel CPUs	System-wide function profiling on Intel CPUs	Low-level debugging on ARM CPUs

6. Benchmarking with existing profiling frameworks

GEMM performance profiling can be performed using various tools, each with its own methodology and limitations. Traditional tools like Likwid [37], Intel VTune [43], and ARM DStream [44] and ARM Debugging [45] rely on real execution data, requiring actual hardware and often code instrumentation or architecture-specific dependencies. While these tools provide valuable insights into performance metrics, their reliance on specific hardware platforms limits their flexibility. In contrast, GEMM-ArchProfiler adopts a simulation-based approach using the widely accepted gem5 simulator framework, enabling performance evaluation across multiple architectures without hardware-specific modifications. This reliance on gem5, a trusted tool in the research community, enhances the framework's credibility and makes it more suitable for both academic studies and industry-driven architectural exploration. A comprehensive comparison is presented in Table 1.

Traditional profiling tools rely on real execution, making them dependent on the target processor and often requiring modifications to source code or integration with vendor-specific libraries. Researchers and developers using these tools must execute their applications on multiple physical hardware platforms to gather insights, making cross-architecture performance evaluation complex.

The authors observed that profiling YOLOv3 implementation in darknet with naïve GEMM [18] on an Intel Skylake processor using a single core took less than two minutes with Likwid profiling tool. However, executing the same CPU configuration on GEMM-ArchProfiler required over eight hours due to the high computational demands of simulation. Despite the significant difference in execution time, the output of performance metrics remains comparable in both cases, accounting for the runtime overhead inherent in each approach. The Table 2 explains the benchmarking of profiling between real-time profiling using Likwid and the detailed architectural insights provided by simulation-based approach using GEMM-ArchProfiler. Since the authors used benchmarked CPU configurations in gem5 with SPEC CPU 2017 workloads [46], separate execution trace benchmarking is not necessary.

In terms of execution time, profiling granularity, and resource usage, hardware-based profiling tools introduce minimal overhead, with execution speeds closely matching real application performance. Overheads from code instrumentation, PMC monitoring, and external tracing remain negligible. In contrast, simulation-based profiling, such as GEMM-ArchProfiler using gem5, is significantly slower and resource-intensive. While it offers deeper architectural insights, it demands substantial computational resources and extended execution times, making it less practical for rapid performance evaluation.

7. Illustrative examples

A researcher seeking to evaluate the performance of a GEMM algorithm in a CNN workload on a custom CPU architecture with specific CPU, memory, and cache configurations can systematically achieve this using GEMM-ArchProfiler.

The process begins with configuring the CNN workload using the Darknet-based integration provided by GEMM-ArchProfiler. The researcher can select an appropriate CNN model, such as Darknet, ResNet, or DenseNet, along with relevant parameters as per the Darknet framework. A critical initial step involves identifying the correct GEMM implementation in the src/gemm.c file. This includes ensuring the presence of appropriate GEM5 instrumentation tags and preprocessor directives for compatibility. Researchers can also implement and integrate new GEMM algorithms by adding these tags and directives in a straightforward manner.

The next step is the build and compilation process. If the framework's pre-included GEMM algorithms are to be used, the researcher can select the desired algorithm during the make operation. Additionally, options for single-threaded or multi-threaded execution can be specified at this stage, making the compilation process both flexible and user-driven.

Following the compilation, the researcher customizes the CPU architecture using GEMM-ArchProfiler's configuration file. Parameters such as CPU frequency, L1, L2, and L3 cache sizes, memory allocation, and memory clock speed are defined to simulate the desired hardware environment. This configuration file also includes the simulation executable command, which must be appropriately updated to align with the specified architecture.

Once the configurations are in place, the researcher runs the simulation. The configured workload is executed in the gem5 simulator using the prescribed command. GEMM-ArchProfiler simulates the defined CPU architecture, capturing detailed logs of each GEMM call. The framework supports checkpoints and fast-forwarding to focus on computationally intensive layers, thereby reducing overall simulation time. However, depending on the host system's specifications, a full simulation may still require several hours, typically up to four hours or more.

After the simulation completes, GEMM-ArchProfiler generates two log files containing detailed hardware-level metrics. These logs are processed using the integrated log analyzer, which outputs a comprehensive CSV file. The metrics captured include matrix dimensions (M, N, K), cache usage patterns, memory bandwidth, execution latency, energy consumption, and performance indicators such as FLOPS.

This detailed data enables researchers to refine GEMM implementations, enhance cache utilization, and optimize CPU configurations for improved energy efficiency. By bridging the gap between theoretical evaluations and practical deployments, GEMM-ArchProfiler provides a systematic and robust approach to assessing GEMM operations in real-world CNN workloads.

Table 2
Comparison & Benchmarking : Likwid vs. GEMM-ArchProfiler.

Feature	Likwid	GEMM-ArchProfiler
Workload Used	YOLOv3 with naı̀ve GEMM	YOLOv3 with naı́ve GEMM
Processor	Intel Core i3-6100U CPU at 2.30 GHz, 32 KB L1 cache per core, 256 KB L2 cache per core, shared 3 MB L3 cache,8 GB of DRAM memory at 1600 MHz [47] (Real)	Intel Core i3-6100U CPU at 2.30 GHz, 32 KB L1 cache per core, 256 KB L2 cache per core, shared 3 MB L3 cache,8 GB of DRAM memory at 1600 MHz [48] (Simulated)
Execution Time	<2 mins	~8 hours
Execution Model	Real Execution	Simulation-Based
Profiling Granularity	Function-Level Metrics	Detailed Architectural Insights
Profiling Overhead	Low	High (Simulation Overhead)
CPU Utilization	Moderate (Real Execution)	High (Simulation)
Memory Usage	Minimal	High (Due to Simulation)
Profiling Depth	Limited to hardware counters	Deep insights into memory, cache, and execution
Instruction-Level Analysis	Limited	Detailed
Cache Performance Metrics	Supported	Highly Detailed
Branch Prediction Analysis	Limited	Detailed
Energy Consumption Profiling	Limited	Available
Thermal Profiling	Not Available	Available

8. Impact

GEMM-ArchProfiler is a powerful tool designed to support researchers in addressing critical questions related to AI workload optimization and hardware design. It provides a platform for detailed exploration of the performance of GEMM within CNNs on configurable CPU architectures. Researchers can utilize the framework to study the impact of different CPU configurations, such as cache sizes, memory hierarchies, and processor settings, on workload efficiency and energy consumption. The ability to test GEMM algorithms within realistic CNN workloads makes GEMM-ArchProfiler an invaluable resource for getting insights to optimize AI applications, especially for resource-constrained edge computing environments [49,50].

This framework enables researchers to advance existing studies by integrating realistic CNN workloads with the gem5 simulator, offering a comprehensive evaluation of hardware-level metrics such as memory usage, cache behavior, execution latency, and energy consumption. Traditional methods often rely on isolated or synthetic benchmarks, which fail to capture the complexities of real-world scenarios. GEMM-ArchProfiler addresses this gap by allowing researchers to test and refine their algorithms under conditions that closely mimic practical deployments. This systematic approach not only improves the precision of evaluations but also opens new avenues for research in optimizing GEMM algorithms and hardware configurations for edge systems.

By providing a modular architecture, GEMM-ArchProfiler simplifies the process of evaluating GEMM algorithms [51]. Researchers can customize CNN workloads and CPU configurations to simulate a variety of scenarios. The inclusion of configurable parameters, such as CPU frequency and cache sizes, along with the ability to incorporate new GEMM algorithms through a flexible Makefile system, ensures that the framework can adapt to diverse research needs. This design facilitates experimentation and supports the systematic evaluation of optimization strategies, making it an essential tool for researchers aiming to design efficient and reliable AI systems.

The framework's capability to generate detailed performance logs and actionable metrics further enhances its utility for researchers. By profiling GEMM operations within CNN workloads, researchers can gain insights into memory access patterns, cache efficiency, and energy consumption. These metrics are invaluable for identifying computational bottlenecks and making informed decisions about hardware optimization. The systematic evaluation enabled by GEMM-ArchProfiler reduces the reliance on trial-and-error approaches, thereby accelerating the research process and ensuring more robust outcomes.

GEMM-ArchProfiler holds the potential to significantly advance research in hardware-software co-design, edge computing, and AI performance profiling. While it is not yet widely adopted, it provides a critical foundation for researchers looking to systematically evaluate and optimize GEMM operations within realistic CNN workloads. By addressing the challenges of computational and memory constraints in edge systems, GEMM-ArchProfiler supports the development of innovative solutions that can drive advancements in AI applications across various domains.

Beyond inference optimization, GEMM-ArchProfiler will guide hardware-aware neural architecture search (NAS) on CPU-based systems [52] . By integrating GEMM execution profiling into NAS pipelines, researchers will predict optimal layer configurations, activation functions, and memory layouts that improve computational efficiency on CPU-driven deep learning models. This will have significant implications for applications such as AI-driven robotics, real-time video analytics, and intelligent sensor networks, where low-latency neural processing on CPUs will be a necessity.

Additionally, frameworks such as DeepSeek-R1 [53] and DeepSeek-V3 [54] have introduced new approaches to reducing GPU dependency, emphasizing the potential for optimizing AI workloads on CPU-driven systems. This shift presents an opportunity for further exploration into cost-effective AI deployment strategies, where GEMM-ArchProfiler can play a key role in enhancing CPU-based GEMM optimizations. By refining GEMM execution on CPUs, researchers will be able to minimize GPU usage while maintaining high computational efficiency, significantly improving the affordability and accessibility of AI applications, particularly for resource-constrained environments.

Moreover, in federated learning and distributed neural training [55], where model updates must be efficiently aggregated across CPU clusters, GEMM-ArchProfiler will help evaluate the impact of distributed GEMM computations on memory coherence and synchronization overheads. This will ensure scalable and power-efficient training methodologies, making it possible to deploy large-scale AI models across decentralized CPU-based infrastructures without incurring excessive computational costs.

The summary of the benchmarking experimental results is available in the GEMM-ArchProfiler GitHub repository for further analysis, providing insights into performance metrics, execution overheads, and scalability on Intel Skylake-based processors.

9. Generalization and extensibility of GEMM-ArchProfiler

GEMM-ArchProfiler is a simulation-based profiling tool that uses the gem5 simulator to evaluate GEMM performance across different CPU architectures. Unlike traditional profiling tools that depend on real hardware, it provides a flexible and architecture-independent approach, making it useful for testing various processor configurations, memory hierarchies, and cache designs before deployment.

A key strength of GEMM-ArchProfiler is its support for multiple architectures, including ARM, x86, and RISC-V. It allows users to test different CPU models, from simple in-order execution to advanced out-of-order pipelines, without requiring physical processors. It also enables custom microarchitecture configurations, allowing modifications in core count, cache sizes, memory bandwidth, and instruction scheduling. This makes it a valuable tool for benchmarking existing and future processor designs.

Despite its broad applicability, GEMM-ArchProfiler has some limitations. It does not directly support specialized AI accelerators such as GPUs [56], NPUs[57], or TPUs[58]. While it can simulate vectorized GEMM operations using SIMD instructions, it lacks direct profiling for tensor cores or deep-learning accelerators. Additionally, simulation introduces higher runtime overhead compared to real execution, though this can be improved with techniques like fast-forwarding and simplified simulation models.

While GEMM-ArchProfiler has been successfully used to simulate ARM and RISC-V based processor configurations using gem5, validation on real hardware for these architectures was not performed due to the limited accessibility of cycle-accurate profiling frameworks. However, it is important to note that these architectures have already been extensively validated within gem5 using standard benchmarking tools, and therefore, a separate validation for architectural correctness is not strictly mandatory. Nonetheless, to ensure the comparability and accuracy of profiling results based on specific CNN workloads, a separate validation using real hardware may be conducted in future studies.

Additionally, the framework can be extended to support GPU-based performance profiling by leveraging gem5's support for AMD VEGA-class GPUs. Such enhancements would broaden the scope of GEMM-ArchProfiler, enabling comprehensive analysis across heterogeneous computing platforms and improving its utility for modeling AI workloads on diverse processor and accelerator architectures. To extend its capabilities, future improvements could include integration with automated performance tuning using machine learning, and support for emerging architectures like RISC-V vector extensions. Despite these challenges, GEMM-ArchProfiler remains a powerful and flexible tool for evaluating CPU architectures before deployment, making it highly valuable for system designers and researchers.

10. Conclusions

GEMM-ArchProfiler provides a valuable framework for researchers to evaluate and optimize GEMM operations within realistic CNN workloads on configurable CPU architectures. By integrating the gem5 simulator and customized Darknet libraries, it enables detailed analysis of key hardware-level metrics, helping researchers address challenges in computational efficiency, memory usage, and energy consumption. Its modular design and flexibility allow users to test new GEMM algorithms and CPU configurations systematically, bridging the gap between theoretical studies and practical deployment scenarios. GEMM-ArchProfiler serves as a vital tool for advancing research in AI workload optimization and edge computing systems.

CRediT authorship contribution statement

Binu Ayyappan: Writing – review & editing, Writing – original draft, Visualization, Validation, Software, Resources, Project administration, Methodology, Investigation, Formal analysis, Data curation, Conceptualization. **G. Santhosh Kumar:** Writing – review & editing, Supervision, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

We gratefully acknowledge the open-source developers, maintainers, and communities behind Darknet and gem5 for making these foundational frameworks available through open-source distribution. The Darknet framework was utilized for executing real CNN workloads as test stubs, while the gem5 simulator enabled hardware-level performance profiling. The open-source availability of these systems significantly contributed to the development and validation of GEMM-ArchProfiler.

Funding: This research did not receive any grant from funding agencies in the public, commercial, or not-for-profit sectors.

References

[1] Yu K, Qi X, Zhang P, Fang J, Dong D, Wang R, et al. Optimizing general matrix multiplications on modern multi-core DSPs. In: 2024 IEEE international parallel and distributed processing symposium. 2024, p. 964–75. http://dx.doi.org/10.1109/IPDPS57955.2024.00090.

[2] Liu H, Shi S, Wang X, Jiang ZL, Chen Q. Performance analysis and optimizations of matrix multiplications on ARMv8 processors. In: 2024 design, automation test in Europe conference exhibition. 2024, p. 1–6. http://dx.doi.org/10.23919/DATE58400.2024.10546786.

[3] Sui B, Shen J, Sun C, Wang J, Zheng Z, Guo W. MACO: Exploring GEMM acceleration on a loosely-coupled multi-core processor. In: 2024 design, automation & test in europe conference & exhibition. 2024, p. 1–6. http://dx.doi.org/10.23919/DATE58400.2024.10546765.

[4] Ramírez C, Castelló A, Martínez H, Quintana-Ortí ES. Parallel GEMM-based convolution for deep learning on multicore RISC-V processors. J Supercomput 2024;80(9):12623–43. http://dx.doi.org/10.1007/s11227-024-05927-y, [Online]. Available: https://doi.org/10.1007/s11227-024-05927-y.

[5] Wu S, Zhai Y, Liu J, Huang J, Jian Z, Wong B, Chen Z. Anatomy of high-performance GEMM with online fault tolerance on GPUs. In: Proceedings of the 37th ACM international conference on supercomputing. New York, NY, USA: Association for Computing Machinery; 2023, p. 360–72. http://dx.doi.org/10.1145/3577193.3593715, [Online]. Available: https://doi.org/10.1145/3577193.3593715.

[6] Faingnaert T, Besard T, De Sutter B. Flexible performant GEMM kernels on GPUs. IEEE Trans Parallel Distrib Syst 2022;33(9):2230–48. http://dx.doi.org/10.1109/TPDS.2021.3136457.

[7] Xiaoteng, Liu, Halim P. Understanding GEMM performance and energy on NVIDIA ada Lovelace: A machine learning-based analytical approach. 2024, [Online]. Available: https://arxiv.org/abs/2411.16954.

[8] Ayyappan B, Kumar GS. Neuroprobe: a performance profiling feed forward neural network simulator for hardware interaction analysis. SN Computer Science 2025;6(5):414. http://dx.doi.org/10.1007/s42979-025-03943-0, https://doi.org/10.1007/s42979-025-03943-0.

[9] Li R, Xu Y, Sukumaran-Rajam A, Rountev A, Sadayappan P. Analytical characterization and design space exploration for optimization of CNNs. In: Proceedings of the 26th ACM international conference on architectural support for programming languages and operating systems. New York, NY, USA: Association for Computing Machinery; 2021, p. 928–42. http://dx.doi.org/10.1145/3445814.3446759, [Online]. Available: https://doi.org/10.1145/3445814.3446759.

[10] Liu Y, Wang Y, Yu R, Li M, Sharma V, Wang Y. Optimizing CNN model inference on CPUs. In: Proceedings of the 2019 USeNIX conference on usenix annual technical conference. USA: USENIX Association; 2019, p. 1025–40.

[11] Bochkovskiy A, Wang C, Liao HM. YOLOv4: Optimal speed and accuracy of object detection. 2020, CoRR, abs/2004.10934, [Online]. Available: https://arxiv.org/abs/2004.10934.

[12] Guerrouj FZ, Rodríguez Flórez S, Abouzahir M, El Ouardi A, Ramzi M. Efficient GEMM implementation for vision-based object detection in autonomous driving applications. J Low Power Electron Appl 2023;13(2). http://dx.doi.org/10.3390/jlpea13020040, [Online]. Available: https://www.mdpi.com/2079-9268/13/2/40.

[13] Dong KS, Nikiforov D, Soedarmadji W, Nguyen M, Fletcher C, Shao YS. Design space exploration of embedded SoC architectures for real-time optimal control. 2024, [Online]. Available: https://arxiv.org/abs/2410.12142.

[14] Alaejos G, Castelló A, Martínez Pérez H, Alonso P, Quintana-Orti ES. Generating hardware-aware code for matrix multiplication. 2023, http://dx.doi.org/10.13140/RG.2.2.19668.82565.

[15] Binkert N, Beckmann B, Black G, Reinhardt SK, Saidi A, Basu A, et al. The gem5 simulator. SIGARCH Comput Arch News 2011;39(2):1–7. http://dx.doi.org/10.1145/2024716.2024718, [Online]. Available: https://doi.org/10.1145/2024716.2024718.

[16] Lowe-Power J, Ahmad AM, Akram A, et al. The gem5 simulator: Version 20.0+. 2020, CoRR, abs/2007.03152, [Online]. Available: https://arxiv.org/abs/2007.03152.

[17] Qureshi YM, Simon WA, Zapater M, Olcoz K, Atienza D. Gem5-x: A many-core heterogeneous simulation platform for architectural exploration and optimization. ACM Trans Arch Code Optim 2021;18(4). http://dx.doi.org/10.1145/3461662, [Online]. Available: https://doi.org/10.1145/3461662.

[18] Redmon J. Darknet: Open source neural networks in C. 2013–2016, http://pjreddie.com/darknet/.

[19] Warangdaech N, Suwannasart T. A stub and driver generating approach for system testing using sequence and class diagram. In: 2022 7th international conference on computer and communication systems. 2022, p. 92–6. http://dx.doi.org/10.1109/ICCCS55155.2022.9846825.

[20] Ghasemi A, Ruaro M, Cataldo R, Diguet J-P, Martin KJM. The impact of cache and dynamic memory management in static dataflow applications. 2022, [Online]. Available: https://doi.org/10.1007/s11265-021-01730-7.

[21] Cronin P, Gao X, Wang H, Cotton C. An exploration of ARM system-level cache and GPU side channels. 2021, [Online]. Available: https://doi.org/10.1145/3485832.3485902.

[22] gem5. gem5: Creating a simple configuration script. 2022, https://www.gem5.org/documentation/learning_gem5/part1/simple_config/. [Accessed 11 March 2025].

[23] gem5. gem5: CPU models. 2023, https://www.gem5.org/documentation/general_docs/cpu_models/SimpleCPU. [Accessed 11 March 2025].

[24] Ayyappan B, Kumar GS. Detailed CPU and GEMM-ArchProfiler configuration with gem5. 2025, https://github.com/binooa/GEMM-ArchProfiler/blob/main/docs/gem5cpuconf.md. [Accessed 11 March 2025].

[25] Ayyappan B, Kumar GS. Detailed CPU customization instructions for GEMM-ArchProfiler. 2025, https://github.com/binooa/GEMM-ArchProfiler/blob/main/docs/gem5cpugeneric.md. [Accessed 11 March 2025].

[26] Zeid RB, Moubarak J, Bassil C. Investigating the darknet. In: 2020 international wireless communications and mobile computing. 2020, p. 727–32. http://dx.doi.org/10.1109/IWCMC48107.2020.9148422.

[27] Huang G, Liu Z, Van Der Maaten L, Weinberger KQ. Densely connected convolutional networks. In: 2017 IEEE conference on computer vision and pattern recognition. 2017, p. 2261–9. http://dx.doi.org/10.1109/CVPR.2017.243.

[28] He K, Zhang X, Ren S, Sun J. Deep residual learning for image recognition. In: 2016 IEEE conference on computer vision and pattern recognition. 2016, p. 770–8. http://dx.doi.org/10.1109/CVPR.2016.90.

[29] Lee S, Kim Y, Nam D, Kim J. Gem5-AVX: Extension of the Gem5 simulator to support AVX instruction sets. IEEE Access 2024;12:20767–78. http://dx.doi.org/10.1109/ACCESS.2024.3359296.

[30] Klein J, Qureshi Y, Zapater M, Atienza D. gem5 Extensions for RISC-V: Full System Manual. Lausanne, Switzerland: Embedded Systems Laboratory, EPFL and REDS Institute, HEIG-VD, HES-SO; 2020, Available from the Embedded Systems Laboratory, EPFL.

[31] Abudaqa AA, Al-Kharoubi TM, Mudawar MF, Kobilica A. Simulation of ARM and x86 microprocessors using in-order and out-of-order CPU models with Gem5 simulator. In: 2018 5th international conference on electrical and electronic engineering. 2018, p. 317–22. http://dx.doi.org/10.1109/ICEEE2.2018.8391354.

[32] Akram A, Sawalha L. A comparison of x86 computer architecture simulators. Technical report 1, Computer Architecture and Systems Research Laboratory (CASRL), Western Michigan University; 2016, [Online]. Available: https://scholarworks.wmich.edu/casrl_reports/1.

[33] Wang I, Chakraborty P, Xue ZY, Lin YF. Evaluation of gem5 for performance modeling of ARM Cortex-R based embedded SoCs. Microprocess Microsyst 2022;93:104599. http://dx.doi.org/10.1016/j.micpro.2022.104599, [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0141933122001442.

[34] Patterson D, Bennett J, Bennett M, Chelin H, Harris D, Hellar J, et al. Embench IOT 2.0 and DSP 1.0: Modern embedded computing benchmarks. Computer 2025;58(5):37–47. http://dx.doi.org/10.1109/MC.2024.3511352, [Online]. Available: https://doi.org/10.1109/MC.2024.3511352.

[35] Akram A, Sawalha L. Validation of the gem5 simulator for x86 architectures. In: 2019 IEEE/ACM performance modeling, benchmarking and simulation of high performance computer systems. 2019, p. 53–8. http://dx.doi.org/10.1109/PMBS49563.2019.00012.

[36] Ta T, Cheng L, Batten C. Simulating multi-core RISC-V systems in gem5. 2018, [Online]. Available: https://api.semanticscholar.org/CorpusID:44119937.

[37] Treibig J, Hager G, Wellein G. LIKWID: A lightweight performance-oriented tool suite for x86 multicore environments. In: 2010 39th international conference on parallel processing workshops. 2010, p. 207–16. http://dx.doi.org/10.1109/ICPPW.2010.38.

[38] Ayyappan B, Santhoshkumar G. EdgeInsight: An agent-driven framework for system health monitoring in edge computing platforms. In: Mathew L, Subramanian KG, Nagar AK, editors. Proceedings of international conference on theoretical and applied computing. Singapore: Springer Nature Singapore; 2025, p. 225–42.

[39] Intel Corporation. Intel® Core™ i7-11370H Processor (12M Cache, up to 4.80 GHz) - Specifications. 2024, https://www.intel.com/content/www/us/en/products/sku/196655/intel-core-i711370h-processor-12m-cache-up-to-4-80-ghz-with-ipu/specifications.html. [Accessed 23 May 2025].

[40] Samsung Newsroom. Samsung brings enhanced mobile graphics performance capabilities to new exynos 5 octa processor. 2013, https://news.samsung.com/global/samsung-brings-enhanced-mobile-graphics-performance-capabilities-to-new-exynos-5-octa-processor. [Accessed 23 May 2025].

[41] AMD University Program. PYNQ-Z2 board - AMD university program. 2024, https://www.amd.com/en/corporate/university-program/aup-boards/pynq-z2.html. [Accessed 23 May 2025].

[42] Centre for Development of Advanced Computing (C-DAC). VEGA AS4161 Processor - C-DAC India. 2024, https://cdac.in/index.aspx?id=product_details&productId=VEGAAS4161. [Accessed 23 May 2025].

[43] Intel,Corporation. Fix performance bottlenecks with Intel VTune Profiler. 2024, Available at: https://github.com/oneapi-src/oneAPI-samples/tree/master/Tools/VTuneProfiler. [Accessed 09 March 2025].

[44] Arm,Corporation. Introduction to ARM DSTREAM. 2024, Available at: https://developer.arm.com/documentation/dui0481/k/Introduction-to-ARM-DSTREAM/About-DSTREAM. [Accessed 09 March 2025].

[45] Ning Z, Wang C, Chen Y, Zhang F, Cao J. Revisiting ARM debugging features: Nailgun and its defense. IEEE Trans Dependable Secur Comput 2023;20(1):574–89. http://dx.doi.org/10.1109/TDSC.2021.3139840.

[46] Documentation SC. SPEC CPU 2017 benchmark. 2021, https://www.spec.org/cpu2017/. [Accessed 11 March 2025].

[47] Intel C. Intel core i3-6100 processor (3M cache, 3.70 GHz) - product specifications | Intel — intel.com. 2018, https://www.intel.com/content/www/us/en/products/sku/90729/intel-core-i36100-processor-3m-cache-3-70-ghz/specifications.html. [Accessed 11 March 2025].

[48] Ayyappan B, Kumar GS. Intel core i3-6100 processor gem5 configuration. 2025, https://github.com/binooa/GEMM-ArchProfiler/blob/main/cpuconf/IntelCorei3_6100U.py. [Accessed 11 March 2025].

[49] Ramírez C, Castelló A, Martínez H, Quintana-Ortí ES. Performance analysis of matrix multiplication for deep learning on the edge. In: Anzt H, Bienz A, Luszczek P, Baboulin M, editors. High performance computing. ISC high performance 2022 international workshops. Cham: Springer International Publishing; 2022, p. 65–76.

[50] Reggiani E, Pappalardo A, Doblas M, Moreto M, Olivieri M, Unsal OS, et al. Mix-GEMM: An efficient HW-SW architecture for mixed-precision quantized deep neural networks inference on edge devices. In: 2023 IEEE international symposium on high-performance computer architecture. 2023, p. 1085–98. http://dx.doi.org/10.1109/HPCA56546.2023.10071076.

[51] Taka E, Gourounas D, Gerstlauer A, Marculescu D, Arora A. Efficient approaches for GEMM acceleration on leading AI-optimized FPGAs. 2024, [Online]. Available: https://arxiv.org/abs/2404.11066.

[52] Chheda S, Curtis A, Siegmann E, Chapman B. Performance study on CPU-based machine learning with PyTorch. In: Proceedings of the HPC Asia 2023 workshops. New York, NY, USA: Association for Computing Machinery; 2023, p. 24–34. http://dx.doi.org/10.1145/3581576.3581615, [Online]. Available: https://doi.org/10.1145/3581576.3581615.

[53] DeepSeek-AI, Guo D, et al. DeepSeek-R1: Incentivizing reasoning capability in LLMs via reinforcement learning. 2025, [Online]. Available: https://arxiv.org/abs/2501.12948.

[54] DeepSeek-AI, Liu A, et al. DeepSeek-V3 technical report. 2025, [Online]. Available: https://arxiv.org/abs/2412.19437.

[55] Qi P, Chiaro D, Guzzo A, Ianni M, Fortino G, Piccialli F. Model aggregation techniques in federated learning: A comprehensive survey. Future Gener Comput Syst 2024;150:272–93. http://dx.doi.org/10.1016/j.future.2023.09.008, [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167739X23003333.

[56] Poremba M. Moving to full system simulation of GPU applications. 2023, https://www.gem5.org/2023/02/13/moving-to-full-system-gpu.html. [Accessed 11 March 2025].

[57] Shrivastava N, Sarangi SR. Securator: a fast and secure neural processing unit. In: Proceedings of the 2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA). Los Alamitos, CA, USA, March 2023: IEEE Computer Society; 2023, p. 1127–39. http://dx.doi.org/10.1109/HPCA56546.2023.10071091, https://doi.ieeecomputersociety.org/10.1109/HPCA56546.2023.10071091.

[58] Junior RLR, Rech P. Reliability of google's tensor processing units for convolutional neural networks. In: In Proceedings of the 2022 IEEE/IFIP International Conference on Dependable Systems and Networks — Supplemental Volume (DSN-S). 2022, p. 25–7. http://dx.doi.org/10.1109/DSN-S54099.2022.00018, IEEE, June 2022.