IRIS AperTO

UNIVERSITÀ DEGLI STUDI DI TORINO

AperTO - Archivio Istituzionale Open Access dell'Università di Torino

Inference Performance of Large Language Models on a 64-core RISC-V CPU with Silicon-Enabled Vectors

This is a pre print version of the following article:

Original Citation:

Availability:

This version is available http://hdl.handle.net/2318/2105617 since 2026-03-31T14:35:34Z

Published version:

DOI:10.1016/j.future.2025.108242

Terms of use:

Open Access

Anyone can freely access the full text of works made available as "Open Access". Works made available under a Creative Commons license can be used according to the terms and conditions of said license. Use of all other works requires consent of the right holder (author or publisher) if not exempted from copyright protection by the applicable law.

(Article begins on next page)

12 November 2026

Highlights

Inference Performance of Large Language Models on a 64-core RISC-V CPU with Silicon-Enabled Vectors

Adriano Marques Garcia,Giulio Malenza,Robert Birke,Marco Aldinucci

- Enabling RISC-V Vector 0.7.1 support for PyTorch on SOPHON SG2042
- Exploring the impact of RISC-V Vector instructions on LLM model inference
- Analyse scalability of LLM model inference across linear algebra libraries

PrePrint

Inference Performance of Large Language Models on a 64-core RISC-V CPU with Silicon-Enabled Vectors

Adriano Marques Garcia[a,*], Giulio Malenza[a], Robert Birke[a] and Marco Aldinucci[a]

[a]University of Turin, Corso Svizzera 185, Turin, 10149, Italy

ARTICLE INFO

Keywords:
RISC-V
RVV
PyTorch
LLM
XuanTie C920
SOPHON SG2042
OpenBLAS
BLIS
GEMM

ABSTRACT

The rising usage of compute-intensive AI applications with fast response time requirements, such as text generation using large language models (LLMs), underscores the need for more efficient and versatile hardware solutions. This drives the exploration of emerging architectures like RISC-V, which have the potential to deliver strong performance within tight power constraints. The recent commercial release of processors with RISC-V Vector (RVV) silicon-enabled extensions further amplifies the significance of RISC-V architectures, offering enhanced capabilities for parallel processing and accelerating tasks critical to LLMs.

This work evaluates the BERT, GPT-2, Gemma-2, LLaMA-3.2, and DeepSeek-LLM language model inference performance on the SOPHON SG2042 64-core RISC-V architecture with silicon-enabled RVV v0.7.1. We benchmarked the models with and without RVV, using OpenBLAS and BLIS as backends for PyTorch to enable vectorization. Our results show that the performance impact of RVV is closely tied to the matrix shape and arithmetic intensity. Indeed, vectorization can actually slow down GEMM operations due to memory-bound behavior, whereas higher batch sizes shift execution to the compute-bound region, where RVV shows clear benefits. We validate this behavior experimentally using roofline modeling and traced GEMM timing, revealing performance bottlenecks that are invisible to synthetic micro-benchmarks. While enabling RVV in OpenBLAS can speed up inference performance by up to 1.3×, its benefits are highly configuration-dependent. These insights suggest that workload characteristics, threading behavior, and datatype must be carefully aligned to unlock RVV's full potential. Our findings highlight both the promise and the current software limitations of running LLMs on RVV-enabled RISC-V platforms.

1. Introduction

In recent years, natural language processing (NLP) advancements have skyrocketed. This was propelled mainly by the development of sophisticated large language models (LLMs), starting with BERT (Bidirectional Encoder Representations from Transformers) and GPT (Generative Pre-trained Transformer). These models have showcased remarkable capabilities in understanding, generating, and even summarizing human-like language accurately and fluently [11, 19]. This exceptional performance is fueled by increasingly larger foundational models with billions of parameters; for example, LLama v3's largest model encompasses 400 billion parameters.

To cope with, on the one hand, the growing demand for more efficient and versatile NLP systems and, on the other hand, increasingly larger and resource-hungry models, it becomes crucial to assess the inference performance of such models across various hardware architectures. Among these architectures, the RISC-V (Reduced Instruction Set Computer-Five) architecture has emerged as a compelling option that stands out as an open-source instruction set architecture (ISA) that has gained considerable traction due to its flexibility, scalability, and potential for customization [17]. Besides, the RISC-V architecture is known for its potential to deliver strong performance within tight power constraints, which sets it apart from traditional Complex Instruction Set Computing (CISC) architectures [13, 1, 26].

This work extends the work from [15]. The main goal of this work is to assess the inference performance of large language models on the SOPHON SG2042 64-core RISC-V architecture with silicon-enabled vectors (RVV) v0.7.1 [44]. For this goal, we first built the PyTorch library using OpenBLAS and BLIS to leverage support for RVV v.0.7.1 instructions. BLIS and OpenBLAS are both optimized libraries that aim to provide efficient implementations of linear algebra routines, specifically implementing the BLAS (Basic Linear Algebra Subprograms) interface, differing in their approach and features. Then, we wrote a text-generation Python script for each model and ran experiments measuring the inference times. We compare the model inference performance with PyTorch without using OpenBLAS/BLIS, using OpenBLAS/BLIS built for generic RISC-V architectures, and using OpenBLAS/BLIS targeting the specific RISC-V architecture on the SOPHON SG2042. We further analyze the effects of model size via four different flavors of the GPT-2 model with parameters ranging from 136 million to 1.6 billion parameters and parameter format, and data types via LLama-v3.2 saved in `bf16` and `fp32`.

In our experiments, we observe that enabling RVV vectorization often introduces overheads in GEMM kernels used during LLM inference, especially in workloads with a batch size of $N = 1$, where the arithmetic intensity is

*Corresponding author
adriano.marques@unito.it (A.M. Garcia); giulio.malenza@unito.it (G. Malenza); robert.birke@unito.it (R. Birke); marco.aldinucci@unito.it (M. Aldinucci)
https://alpha.di.unito.it/adriano-garcia (A.M. Garcia)
ORCID(s): 0000-0003-4796-773X (A.M. Garcia); 0009-0006-4862-7429 (G. Malenza); 0000-0003-1144-3707 (R. Birke); 0000-0001-8788-0829 (M. Aldinucci)

low. This leads to cases where RVV performs worse than its scalar counterpart. We also show that many GEMMs executed by real LLMs fall into this low-intensity regime. Therefore, to better explain this behavior, we use roofline analysis with both theoretical and empirical hardware parameters, demonstrating that inference with small batches is memory-bound and ill-suited to vector acceleration. We validate this using traced GEMM execution data from real models.

The contributions of this work are as follows:

- We evaluate how enabling RVV impacts the inference performance of large language models (BERT, GPT-2, Gemma-2, Llama-3.2, and DeepSeek-LLM).
- We evaluate an experimental version of the BLIS library that leverages RVV 0.7.1 on a real in-silicon RISC-V architecture.
- We analyze the scalability of model inference performance when increasing the parallelism, LLM model size, and torch precision on the SOPHON SG2042 64-core RISC-V processor.

2. Background

2.1. Large Language Models (LLMs)

A language model can be defined as a probabilistic model of natural language. It is a model that predicts the likelihood of a sequence of words given the preceding context [41, 4]. Such models learn a language structure, grammar, and vocabulary from vast amounts of text data [5]. They then use that knowledge to perform various natural-language processing tasks, such as text generation, translation, summarization, sentiment analysis, and more [28, 25, 22]. The first language model was proposed in 1980, and researchers and the industry have been increasingly improving these models over the past decades. In 2017, researchers from Google utilized the transformer architecture and attention mechanism [11], laying the groundwork for the development of large language models.

BERT [11] (Bidirectional Encoder Representations from Transformers) is a foundational language model based on the transformer architecture. Unlike causal models that generate text from left to right, BERT uses a masked language modeling objective and a bidirectional encoder to learn contextual representations from both directions. BERT's encoder-only architecture makes it especially effective when fine-tuned for downstream natural language understanding applications. While it is not natively designed for text generation, BERT can still be adapted for generation tasks through fine-tuning or architectural modifications, though this typically comes with increased computational cost and less fluent output compared to models such as GPT-2.

GPT-2 [36] is a well-known state-of-the-art language model developed by OpenAI, the second in the series. It is also the last model fully disclosed by OpenAI before Microsoft acquired the exclusive rights to GPT-3 and follow-ups [20]. It is considered a causal language model that predicts the next token in the sequence (sentence) by only attending to the tokens to the left; that is, the model can not see into the future. Although the latest GPT version is GPT-4 Turbo, GPT-2 still presents incredible text generation capabilities. It is designed to generate human-like text based on the input it receives.

Gemma-2 [39] is a "lightweight" state-of-the-art open model from Google. Gemma improves the decoder-only transformer architecture with interleaving local-global attention and group-query attention. Furthermore, the model is trained using knowledge distillation in addition to next token prediction.

LLama-3.2 [16] is part of the open model series from Meta. While version 3.2 does not significantly deviate from the original LLaMa and LLaMA-2 architectures [42] it enhances performance using higher quality and more diverse training data and reinforcement learning with human feedback for increased training scale.

DeepSeek-LLM [10] largely follows the design of the LLaMA architecture [42] using a pre-norm structure with RMSNorm function, SwiGLU as activation function for the feed-forward network, and rotary embedding for positional encoding, but with different number of layers and a different use of grouped attention. DeepSeek-LLM outperforms LLaMa-2 on across different benchmarks.

2.2. LLM Inference Characteristics

The inference phase of LLMs can be divided into two distinct stages with different computational and memory access patterns. The first stage, *prefill*, processes the entire input sequence at once. This stage benefits from high data reuse across attention layers and typically exhibits high arithmetic intensity, making it compute-bound. However, prefill occurs only once per prompt. The results of this stage (keys and values for each token across attention heads) are stored in a dedicated key-value (KV) cache.

The second stage, *decode*, generates output tokens in an autoregressive manner, one token at a time. Since each step relies on cached context while introducing only a single new token, the operations involve smaller matrix multiplications with limited reuse. This reduces arithmetic intensity, making the decode stage more memory-bound, especially for small batch sizes. Batch size plays a crucial role in both stages. In latency-sensitive scenarios, such as interactive chat, batch size is usually equal to 1, which can lead to memory-bound GEMM operations with limited benefit from vectorization [24].

2.3. RISC-V Vectors (RVV) Enabled Silicon

The vector instructions for RISC-V are defined in the open "V" vector ISA extension standardized by RISC-V International [38]. The first stable release, 1.0, has now been rectified. While most versions never made it into silicon [26], one can find off-the-shelf hardware supporting either v0.7.1, e.g., SOPHON SG2042 SoC used here or v1.0, e.g., Canaan Kendryte K230 SoC. However, to our knowledge, v1.0 SoCs are still rare and target the low-medium compute range.

Table 1
Main characteristics of the Large Language Models used in this work.

Model	Parameters	Hidden Dim	# Layers	# Attention Heads	Context Length	FFN Dim	Data Type
GPT-2	137M	768	12	12	1024	3072	float32
GPT-2 Medium	380M	1024	24	16	1024	4096	float32
GPT-2 Large	812M	1280	36	20	1024	5120	float32
GPT-2 XL	1.61B	1600	48	25	1024	6400	float32
BERT-large-cased	335M	1024	24	16	512	4096	float32
Gemma-2-2B	2.61B	2304	26	8	8192	9216	float32
LLaMA-3.2-1B	1.24B	2048	16	32	131072	8192	bfloat16
DeepSeek-LLM-7B-base	7B	4096	30	32	4096	11008	bfloat16

SOPHON SG2042 system-on-chip (SoC) contains 64 RISC-V cores divided into 16 clusters connected through a grid network [44]. Each cluster comprises a XuanTie C920 4-core RISC-V CPU. Each core is equipped with 64KB of L1 instruction and data cache, and each cluster of 4 cores shares a 1MB of L2 cache. The unified L2 cache can handle two access requests in parallel within one cycle. The grid interconnect finally offers access to 64MB of level 3 cache shared among all 64 cores. Four DDR4-3200 memory controllers are used to manage access to the main memory system. For peripherals, the SG2042 is equipped with 32 PCIe Gen4 lanes.

XuanTie C920 [6] is a homogeneous high-performance 64-bit multi-core RISC-V CPU architecture designed by T-Head that supports 1 to 4 cores at a maximum operation frequency of 2 GHz. It targets high-performance applications and implements a 12-stage, out-of-order, multiple-issue superscalar pipeline. Based on the RISC-V instruction set architecture (ISA), this CPU provides the RV64GCV instruction set [43], which supports standard vector instructions extension version 0.7.1 (RVV 0.7.1). The vector processing unit's vector registers are 128 bits long and support the FP16, FP32, FP64, INT8, INT16, INT32, and INT64 data types.

3. PyTorch with Vector Support

PyTorch can delegate the computation to various low-level libraries to best leverage the capabilities of the underlying hardware, e.g., Intel MKL [14] when running on Intel CPUs or cuDNN [7] and MIOpen [23] when running on NVIDIA or AMD GPUs, or more generic ones such as various libraries from the BLAS family.

Colonnelli et al. [9] describe the first porting of PyTorch [34] v2.0 to the RISC-V ISA, but the underlying platform in use did offer limited acceleration capabilities, i.e., only fused multiply-add (FMA) support. Newer RISC-V silicons also support the RVV standard, providing vector instructions that can process multiple computations in parallel, following the same instruction multiple data (SIMD) parallel computing paradigm. To leverage such capability, we rely on the recently ported OpenBLAS or BLIS [32] libraries. These BLAS-like libraries can be compiled with and without RVV support based on the defined target. Notice that due to the different versions of the RVV standard, a careful match between the target architecture and a compatible compiler is required to compile vector instructions enabled code for the SG2042 SoC correctly.

More in detail, we first compiled OpenBLAS v0.3.26 using the XuanTie GNU Compiler Toolchain v2.8.0 [30] and set `TARGET=C910V` to enable RVV. For BLIS, we used a modified version [32] based on BLIS v0.9.0. This BLIS version requires an LLVM-based compiler. Therefore, we compiled it using the LLVM-EPI-0.7[1], a version of the EPI project's LLVM-based compiler designed to foster RVV. We configured it setting "rv64iv0p7" as the target architecture. Then, we compiled PyTorch v2.3 for Python v3.10.10, using Xuantie's GCC v13.2, OpenMP v4.5, OpenBLAS/BLIS, and enabling only the following build options: `USE_OPENMP=1` to leverage the multiple cores on the SG2042 SoC; `USE_BLAS=1` and `USE_LAPACK=1` to use the BLAS libraries as main computation backend; `USE_KINETO=ON` for profiling; and `USE_NUMPY=ON` for numpy support. Note that since the work in [9], the SLEEF vector math library now supports only RVV v1.0, which is incompatible with the SG2042 SoC.

4. Experimental Methodology and Results

This section presents the model inference performance of BERT, GPT-2 (different sizes), Gemma, Llama, and DeepSeek for text generation. We run the experiments on the Milk-V Pioneer Box, a commercial ready-to-use development platform in the form of a desktop computer equipped with a Pioneer motherboard powered by the SOPHON SG2042. The one we used in this paper is equipped with 128GB of RAM DDR4 (3200MHz) and a 1TB PCIe 3.0 SSD. The operating system is Linux fedora-riscv 6.1.31. All performance experiments report the average of 10 runs, along with the standard deviation represented by whiskers.

[1] https://repo.hca.bsc.es/gitlab/rferrer/llvm-epi

Model	Name	Self CPU %	Self CPU	CPU total %	CPU total	CPU time avg.	# of Calls
GPT-2	aten::addmm	69.55%	13.785s	70.51%	13.975s	5.823ms	2400
BERT	aten::addmm	96.55%	7.190s	97.03%	7.225s	49.489ms	3650
LLaMA	aten::mm	90.93%	55.545s	90.94%	55.549s	19.663ms	2825
Gemma	aten::mm	95.71%	308.439s	95.71%	308.448s	67.420ms	4575
DeepSeek	aten::mm	99.57%	2271.444s	99.57%	2271.452s	430.607ms	5275

Figure 1: PyTorch profiler output (main operation only) when running the models on a single core of the SOPHON SG2042 CPU.

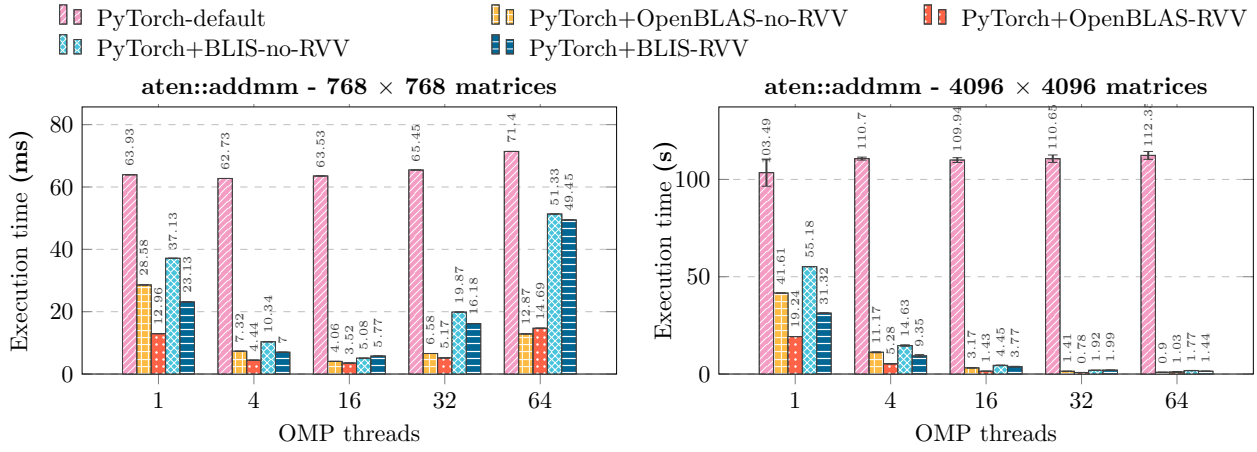


Figure 2: Aten::addmm kernel performance using 768 × 768 and 4096 × 4096 matrices when using PyTorch with different BLAS runtimes over multiple thread counts.

4.1. Benchmarks

We wrote simple text generation benchmark applications in Python based on the examples provided at the Hugging Face platform[2] for each language model to test and evaluate their performance. The text generation benchmark applications are pretty standard. They receive a prompt as input and generate/predict the next *n* token as output. For GPT-2 [37], we used the different model sizes (`gpt2`, `gpt2-medium`, `gpt2-large`, and `gpt2-xl`) provided by OpenAI on their profile at Hugging Face[3]. The other models we used are `bert-large-cased`[4] and `Gemma-2-2b`[5] from Google, `Llama-3.2-1B`[6] from Meta, and `deepseek-llm-7b-base`[7] from the DeepSeek-AI company. The main characteristics of all these models are available in Table 1.

For all the experiments with real models, we used the following input prompt:

– "*The quick brown fox ran*"

Here is an example of the generated output text from GPT-2:

– "*The quick brown fox ran off. This fox needs to think for a while. This fox needs a rest.*"

The other models generate similar outputs. Notice that the generation depends on sampling the next token probabilities via a random number generator. Hence, unless the seed of the random number generator is fixed, each execution leads to different sentences being generated.

4.2. Potential Performance Scalability

Large language models rely heavily on linear algebra operations, with matrix multiplication (MM) being one of the most computationally intensive components during inference [47]. In practice, operations such as `aten::addmm` and `aten::mm` account for a substantial portion of the CPU operations. To illustrate this, Figure 1 presents excerpts from the PyTorch traces for the models evaluated in this work.

As shown, in the case of `GPT-2`, the `aten::addmm` operation alone accounts for approximately 70% of total CPU computation (`aten::mm` is the second one for `GPT-2`, about 20%), while for `BERT`, this figure exceeds 96%. In newer and larger models such as `LLaMA`, `Gemma`, and `DeepSeek`, the dominant operation shifts to `aten::mm`, which represents over 99% of CPU operations in the case of `DeepSeek`. These observations confirm that the efficiency of matrix multiplication computation can directly and critically impact the inference performance of LLMs.

Given the dominance of MM operations, it is reasonable to consider targeted optimizations such as the use of tuned BLAS runtimes and architectural support for vector extensions. In particular, RISC-V architectures with RVV (RISC-V Vector Extension) provide opportunities for data-level parallelism, which can significantly accelerate dense

[2] https://huggingface.co/
[3] https://huggingface.co/openai-community
[4] https://huggingface.co/google-bert/bert-large-cased
[5] https://huggingface.co/google/gemma-2-2b
[6] https://huggingface.co/meta-llama/Llama-3.2-1B
[7] https://huggingface.co/deepseek-ai/deepseek-llm-7b-base

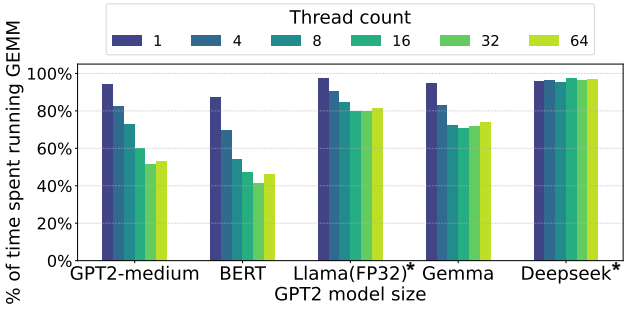


Figure 3: Time spent exclusively running GEMM computations by GPT-2 models compared to the total execution time across different thread counts. The models were run with **PyTorch + OpenBLAS with RVV enabled**. * Both Llama and Deepseek originally run with Bfloat16 precision, which is not supported by the BLAS libraries and the architecture used on these experiments. While we are running Llama with float32 precision, we kept the original Bfloat16 for Deepseek.

matrix operations. To explore this potential, we first implemented a microbenchmark that executes a single `aten::addmm` operation. This operation is a core linear algebra primitive provided by PyTorch's ATen backend. It performs a matrix multiplication followed by a matrix addition in a single call. Mathematically, the operation can be expressed as:

$$C = A \times B + C \tag{1}$$

where A, B and C are matrices of size $A \in \mathbb{R}^{M \times K}$, $B \in \mathbb{R}^{K \times N}$, and $C \in \mathbb{R}^{M \times N}$.

The operation described in Equation 1 fuses a matrix multiplication $A \times B$ with the addition of a pre-existing matrix C. Modern CPUs and vector instruction sets (such as RVV on RISC-V) can take advantage of *fused multiply-add* (FMA) instructions, which compute it in a single instruction, reducing both instruction count and rounding errors. Using FMA-capable hardware, this accumulation can be fused into efficient vector instructions, particularly when matrix shapes and memory alignment are favorable.

We tested the `aten::addmm` microbenchmark using two different square matrix sizes. In the first case, matrices A, B, and C are all 768×768, and size 4096×4096 in the second case. The smaller configuration reflects the hidden dimension size of lighter models such as `GPT-2`, while the larger configuration corresponds to heavier models like `DeepSeek`, as shown in Table 1.

The `aten::addmm` microbenchmark results, on Figure 2, demonstrate that switching to optimized BLAS runtimes yields consistent performance improvements over the default configuration. Moreover, enabling RVV leads to additional speedups, resulting in a doubling of performance gains in some scenarios. This trend is seen across different matrix sizes (also 1024×1024 and 8192×8192, not reported here). In the case of matrix size 768×768, we notice a performance loss when 32 and 64 threads are spanned. This is because generating and handling the threads takes more time than performing matrix multiplication for that size.

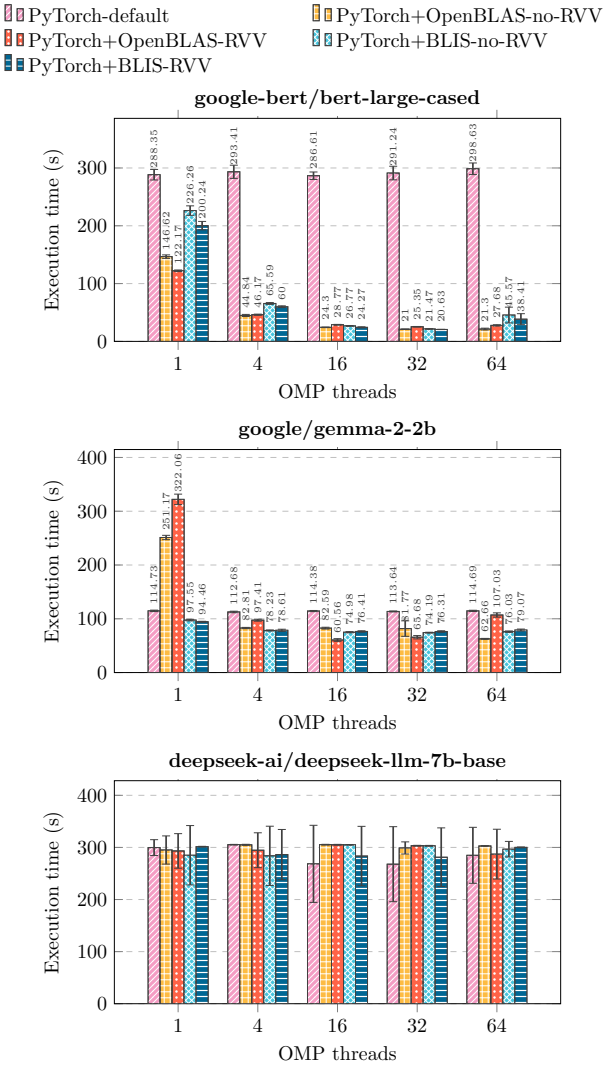


Figure 4: LLMs inference performance when generating 25 tokens and using PyTorch with different BLAS runtimes over multiple thread counts.

Overall, the results show good scalability with increasing thread count, with the best performance observed around 16 and 32 threads. This suggests that the RISC-V platform under evaluation can efficiently scale parallel workloads up to a moderate number of threads. Additionally, RVV proved to be an effective mechanism for boosting the performance of dense matrix operations.

It is important to note, however, that this microbenchmark cannot fully represent the behavior of a complete model. Even though most models' computation is over 90% GEMM operations, this does not imply that over 90% of the execution time was spent performing GEMM operations. To better visualize this, we profiled the PyTorch execution of the models and identified the direct call to the GEMM functions of the BLAS libraries within the ATen library. Then, we added routines to measure the time before and after the function call, and recompiled PyTorch using this modified ATen code. Figure 3 shows how much of the

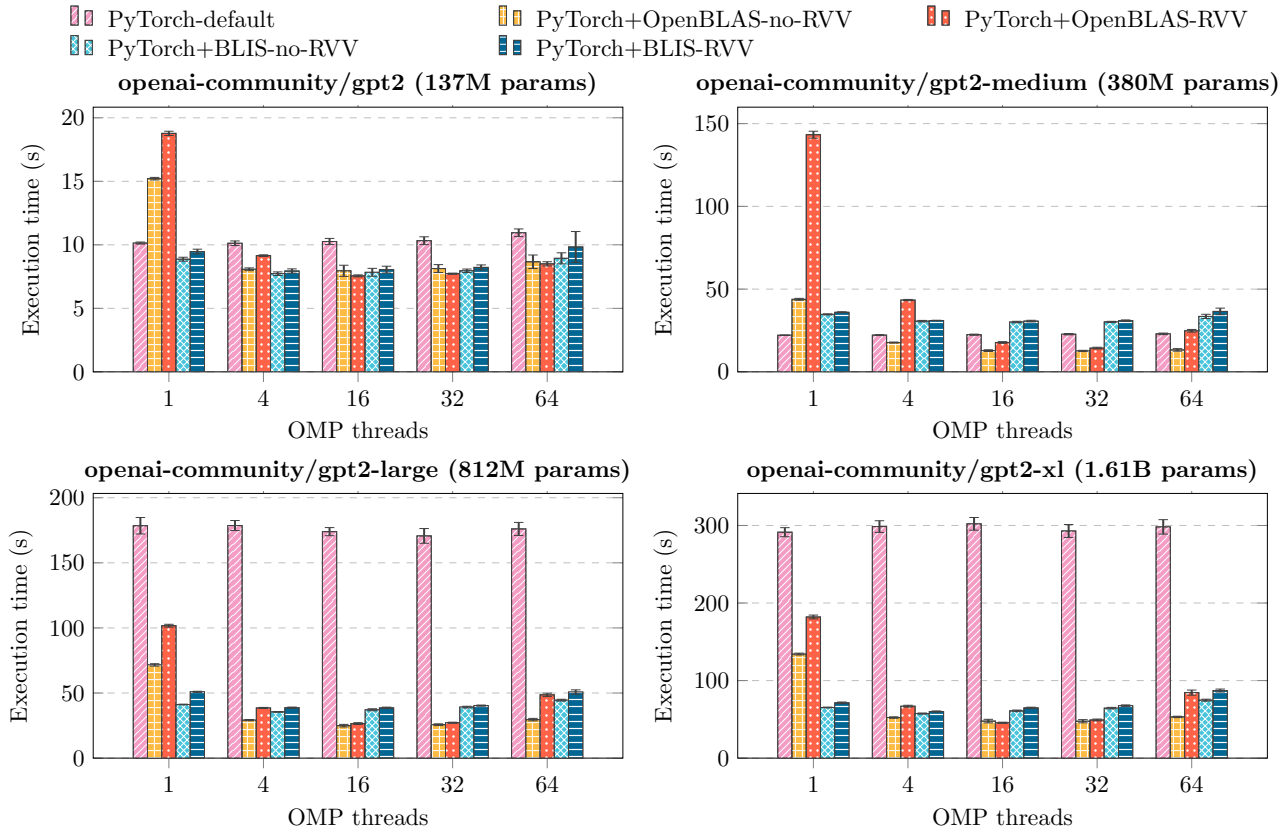


Figure 5: Performance of multiple GPT-2 model sizes, using PyTorch with different BLAS runtimes over multiple thread counts.

execution time is spent by GEMM operations on each model, with different parallelism degrees. For this experiment, we chose the PyTorch + OpenBLAS with RVV configuration, which yielded the best performance in experiments using the `aten::addmm` kernels, as shown in Figure 2.

The results in Figure 3 show that the GEMM share of the execution time can significantly drop when increasing the number of threads, going to almost 50% of the time with the GPT2-medium model and below it with BERT. The reason for this is that, while matrix multiplications dominate CPU operations, actual LLMs execute a complex sequence of operations, including varying tensor shapes, memory access patterns, nonlinear activations, normalization layers, and control-flow logic. Additionally, increasing parallelism can lead to other bottlenecks, such as insufficient computational intensity and increased memory traffic.

As such, while testing isolated kernels provides valuable insights into performance potential, they offer only a partial view. Indeed, much of the existing literature focuses on benchmarking specific operations (e.g., matrix multiplications) and extrapolating conclusions for real workloads [35, 2, 29, 21]. In contrast, the primary objective of this work is to evaluate how full, real-world models behave in practice and how their results diverge from those of isolated kernel analysis.

4.3. Whole Model Inference Performance

Figure 4 presents the performance results of three LLMs: BERT, Gemma-2, and DeepSeek, across our five PyTorch configurations. The results with BERT closely resemble the behavior observed with the `aten::addmm` kernel. As expected, performance improves with increasing thread count and further benefits from RVV acceleration. Among all configurations, PyTorch with BLIS consistently achieves the best performance. In contrast, the PyTorch default backend fails to scale in performance with more threads, which is a counterintuitive trend that we further analyze in subsequent sections.

With Gemma-2, however, an unexpected pattern emerges. When running with a single thread, the PyTorch + OpenBLAS configuration performs significantly worse than both PyTorch with BLIS and the default backend, with execution time up to 3× longer. Interestingly, when using 16 and 32 threads, PyTorch + OpenBLAS with RVV achieves the best execution times for this model. Specifically, at 16 threads, enabling RVV reduces the execution time from 82.5 to 60.5 seconds: a performance gain of approximately 26.7% over the same configuration without RVV support. The single-thread anomaly is not unique to Gemma and will be discussed later, as similar behavior is also observed on GPT-2 and LLama models.

The third chart in Figure 4 presents results for DeepSeek, which behaves quite differently. Across all configurations,

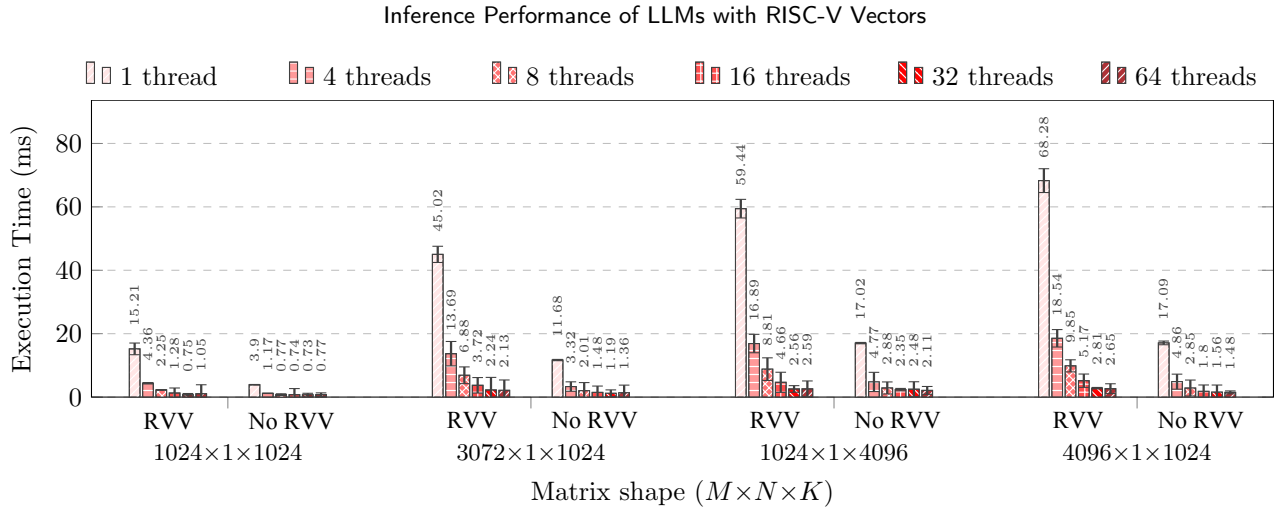


Figure 6: Performance of the main OpenBLAS GEMM operations (fused multiply–add) performed by the real GPT2-medium model, considering RVV enabled and disabled.

execution times are nearly identical, with only marginal differences well within the margin of error. Further investigation revealed that this lack of scalability is caused by the model's use of the BF16 data type, as noted in Table 1. As detailed in Section 2.3, the hardware under evaluation supports only FP16, FP32, FP64, INT8, INT16, INT32, and INT64. Since BF16 is not natively supported, the model likely triggers runtime conversions to compatible types such as FP32. This data type emulation introduces significant overhead, negating any performance benefit from multithreading and resulting in a flat performance profile regardless of the number of threads.

In summary, Figure 4 reveals three recurring performance patterns seen throughout this study: (a) BERT-like behavior: predictable scaling with increasing threads and clear benefits from RVV, except for PyTorch-default, which degrades with parallelism; (b) Gemma-like behavior: limited single-thread performance with OpenBLAS, yet strong multithreaded performance gains with RVV; and (c) DeepSeek-like behavior: complete lack of scalability due to unsupported data types and the associated emulation overhead.

In the following sections, we examine these behaviors more closely using additional models, GPT-2 and LLaMA, which exhibit similar trends.

4.3.1. Impact of Model Scaling on GPT-2 Performance

In this Section, we investigate how increasing the model size impacts the performance of GPT-2. Results are shown in Figure 5 for four different gpt2 model sizes. The PyTorch default backend exhibits two distinct behaviors. With `gpt2` and `gpt2-medium`, it consistently achieves low execution times, outperforming other backends in several cases, especially for `gpt2-medium` in the single-threaded configuration. However, the performance changes drastically when moving to `gpt2-large` and `gpt2-xl`.

We checked the hardware counters using `perf`, and they revealed important microarchitectural insights. For `gpt2-medium`, the CPU achieved an instruction per cycle (IPC) ratio of 0.95, with only 0.67 stalled cycles per instruction, suggesting good pipeline utilization. Both frontend and backend stalls were relatively low (approximately 59.5% and 63.8%, respectively). In contrast, `gpt2-large` shows a significant degradation in efficiency: IPC drops to 0.37, and the number of stalled cycles per instruction increases dramatically to 2.35. Backend stalls rise to 86.5%, and frontend stalls to 85.5%, indicating severe bottlenecks in both data supply and instruction fetch/issue stages.

This degradation in efficiency is reflected in execution time: PyTorch-default takes approximately 178.5 seconds with a single thread on `gpt2-large`, compared to just 22.1 seconds for `gpt2-medium`. This large gap remains even when the number of threads is increased to 64, with the default backend maintaining execution time above 170 seconds for `gpt2-large`. Cache behavior also plays a role in performance changes. L1 and LLC miss rates increase for `gpt2-large`, with LLC misses accounting for 46.8% of all LLC accesses (compared to 31.1% for `gpt2-medium`). This suggests that the larger model introduces more memory pressure, leading to higher cache miss penalties and stalled memory accesses.

Interestingly, for the relatively lightweight `gpt2-medium`, we observed a significant performance discrepancy when using the RVV-enabled OpenBLAS backend in the single-threaded configuration: its execution time is more than three times higher than that of the non-RVV OpenBLAS version. Since this behavior contrasts with trends observed in other models and core counts, we investigated it further.

To determine whether the poor RVV performance could be attributed to a specific subset of GEMM operations, we conducted a targeted analysis using a custom-instrumented version of the ATen library. This instrumented version records matrix shapes and execution times of each individual GEMM operation performed during inference, allowing us to isolate and measure the average execution time of every distinct GEMM shape triggered by running the real `gpt2-medium` model.

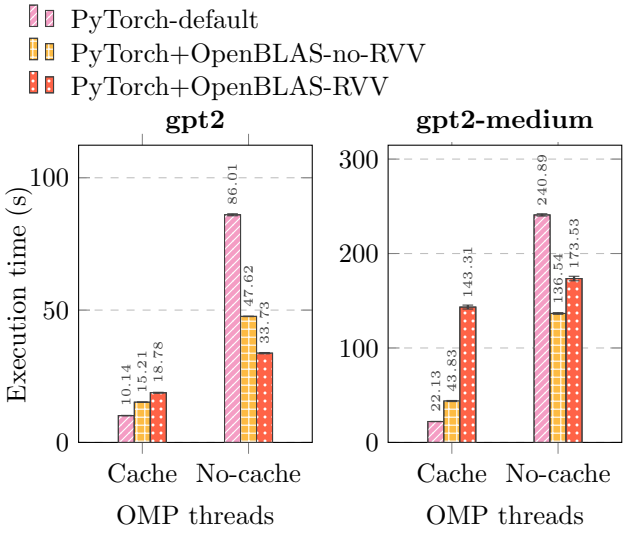


Figure 7: Single-thread performance of GPT-2 with the transformers caching mechanism (KV cache) enabled and disabled.

Figure 6 presents the execution times of the recorded GEMM shapes for GPT2-medium, with and without RVV, across multiple thread counts. The data reveal that the degradation in RVV performance is not due to a single pathological kernel, but rather is consistent across all common GEMM configurations used in the model, particularly those with matrix shapes such as $1024 \times 1 \times 1024$ and $4096 \times 1 \times 1024$. These operations dominate the inference workload and are executed in sequence with $N = 1$, as expected for autoregressive decoding.

This microbenchmark-driven investigation highlights the limitations of relying on peak performance or synthetic benchmarks when evaluating backend efficiency for real-world LLMs. It also reinforces the importance of considering actual model execution traces to identify performance bottlenecks.

4.3.2. Impact of RVV with and without Attention Caching (KV caching)

Continuing the analysis, we investigate the performance degradation observed when running `gpt2-medium` on PyTorch with OpenBLAS, particularly when RVV vectorization is enabled. A suspected contributing factor was the `use_cache` parameter from the HuggingFace Transformers architecture. This parameter, enabled by default, allows the model to reuse the key and value projections from previous tokens in the attention mechanism, which is expected to reduce redundant computation and improve inference time.

To evaluate the impact of `use_cache`, we conducted single-threaded runs of both `gpt2` and `gpt2-medium` under three configurations: PyTorch default, PyTorch with OpenBLAS, and PyTorch with OpenBLAS+RVV. For each case, we compared performance by setting `use_cache=True` and `use_cache=False`. Figure 7 shows the inference performance

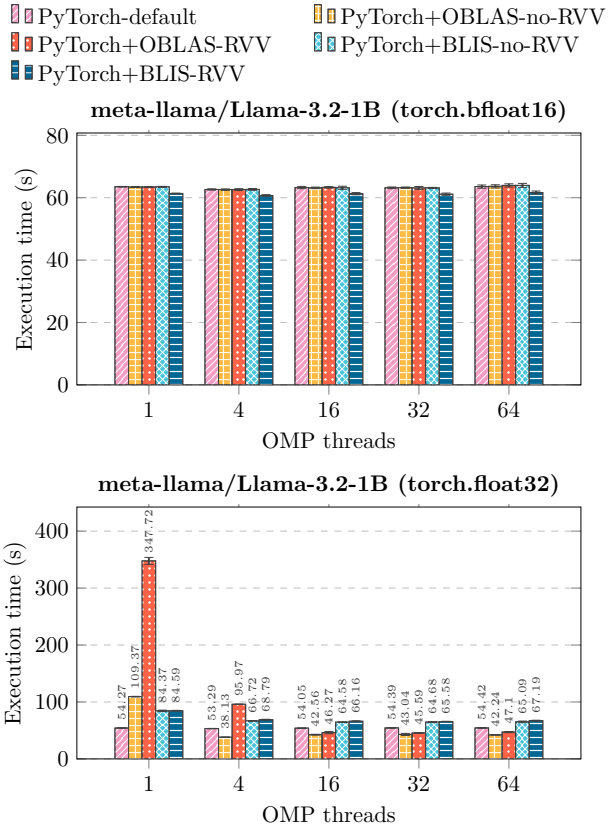
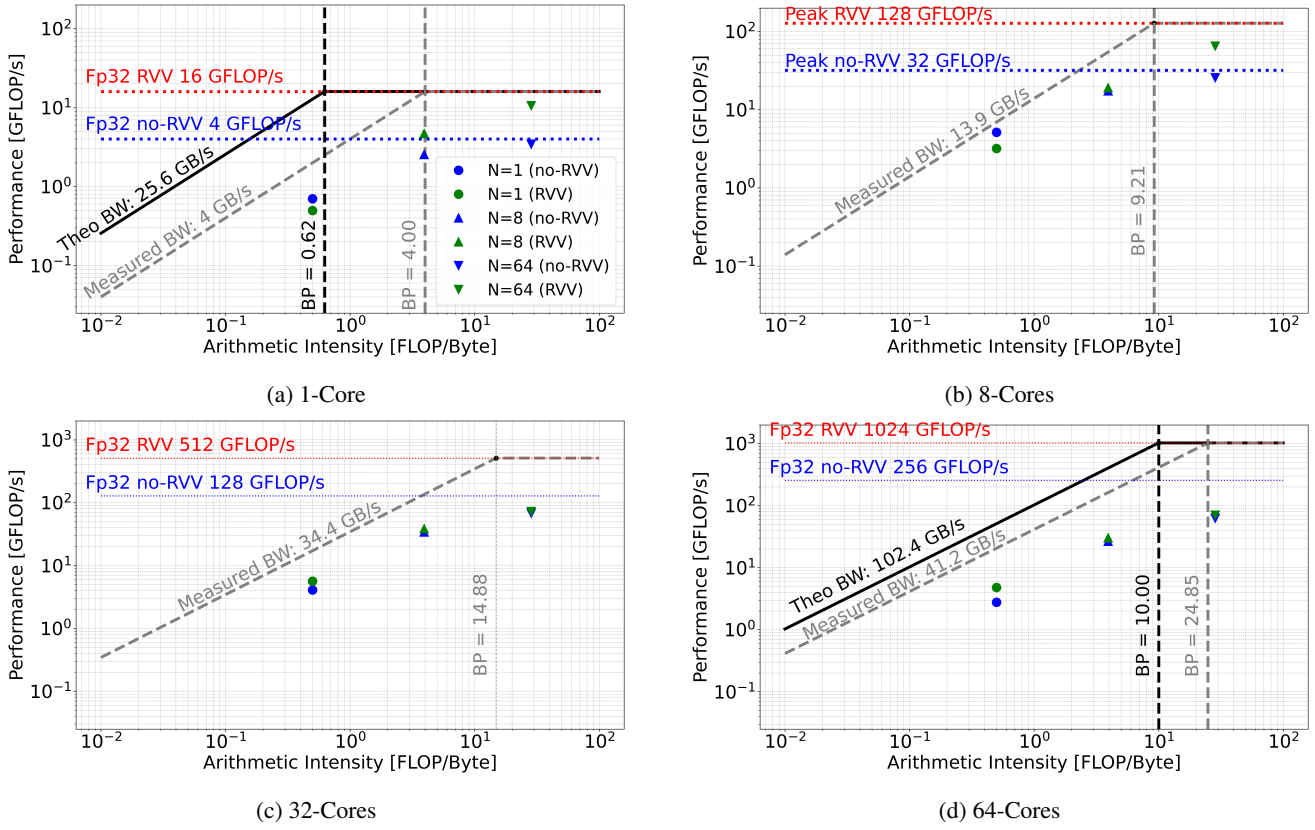


Figure 8: Inference performance of the LLaMA LLM on the SG2042 RISC-V CPU using PyTorch with different BLAS runtimes and thread counts. Results are shown for the model using its original `torch.bfloat16` weight representation and after conversion to `torch.float32`.

for `gpt2` and `gpt2-medium` using PyTorch built with OpenBLAS with RVV and without RVV. Contrary to expectations, enabling the attention cache (`use_cache=True`) brings limited benefit under RVV execution. While `use_cache=True` reduces the instruction count significantly for both models (66.6B → 32.3B for `gpt2` with OpenBLAS+RVV, for instance), the improvement in runtime is modest when compared to the default PyTorch backend. More notably, the performance gap between `use_cache=True` and `use_cache=False` is much smaller under RVV. For instance, the speedup from caching in `gpt2-medium` is only 1.2× with OpenBLAS+RVV, compared to 3.2× for OpenBLAS-no-RVV and 11× for PyTorch-default. This suggests that RVV vectorization is more effective when all attention layers are recomputed, i.e., when caching is disabled.

In summary, the results show that RVV benefits more from the regular computation path (`use_cache=False`), where the overhead of attention mechanisms allows better vector utilization. The attention cache appears to reduce computational intensity in a way that limits RVV's potential advantage. While generally beneficial, this behavior highlights that the caching feature can fail to leverage the underlying libraries or hardware features (e.g., vectorization with RVV) that may not be fully optimized for this usage pattern.



(a) 1-Core (b) 8-Cores

(c) 32-Cores (d) 64-Cores

Figure 9: Comparison of roofline models for RVV and no-RVV on 1, 8, 32, and 64 cores, varying N while keeping $M = K = 1024$.

4.3.3. Handling Unsupported Data Types: BF16 vs FP32 on RISC-V

Some models, such as Llama-3.2-1B and DeepSeek, use `bfloat16` (`torch.bfloat16`) by default for inference, offering a reduced memory footprint, particularly in GPU-based systems where VRAM is a bottleneck. However, our RISC-V platform lacks native hardware support for BF16 arithmetic, and neither of the CPU-based BLAS libraries used in this study (OpenBLAS and BLIS) supports BF16 operations, not even via software emulation.

As a result, PyTorch internally dispatches BF16 matrix multiplications to alternative non-BLAS kernels, possibly falling back to NumPy-based or native CPU implementations. These fallbacks are not only inefficient but also largely single-threaded, which we confirmed experimentally. Even when setting `OPENBLAS_NUM_THREADS` or `OMP_NUM_THREADS` to values greater than one, we observed that PyTorch launches threads but only utilizes one or two of them in practice. Consequently, all BF16 configurations result in flat performance regardless of the backend or thread count, as shown in Figure 8 (left), with runtimes stagnating around 63 s.

To address this limitation, we converted the Llama model to `torch.float32` using the `save_pretrained()` method and reran the experiments. With `float32`, PyTorch properly dispatches GEMM operations to the linked BLAS libraries, allowing for multi-threaded execution and hardware vectorization. The performance improved significantly under several backends, especially for OpenBLAS and RVV-enabled builds (see Figure 8, right). For instance, with four threads, PyTorch+OpenBLAS-no-RVV improved from 63 s (`BF16`) to 38 s (`FP32`), despite the increase in arithmetic precision.

That said, some backends such as PyTorch+BLIS perform worse under FP32 at low thread counts due to internal overheads introduced by the higher precision. Nonetheless, the key insight remains: converting BF16 models to FP32 is essential on RISC-V platforms where BF16 is unsupported. This conversion avoids costly runtime fallbacks, unlocks thread-level parallelism, and enables the use of vectorized libraries like RVV-enabled OpenBLAS.

4.4. LLMs' GEMM Performance

To gain a deep understanding of the performance behavior, we analyzed the recursive pattern of matrix-by-matrix multiplications executed during the inference phase in detail. We compute the roofline model [45] for representative matrix multiplications under different thread counts. We computed the theoretical single precision peak performance Fp^{32}_{Peak} with and without RVV using the following equation:

$$Fp^{32}_{Peak}[GFLOP/s] = \#C \times CF \times \#FPC \quad (2)$$

where, $\#C$ is the number of total core, CF is the clock frequency (2 GHz/s) and $\#FPC$ is the number of flops per cycle. On a single-core, assuming execution of a single fused multiply-add (FMA) instruction per cycle, it results in 2 flops per cycle without RVV. With RVV, the 128 bits long vector

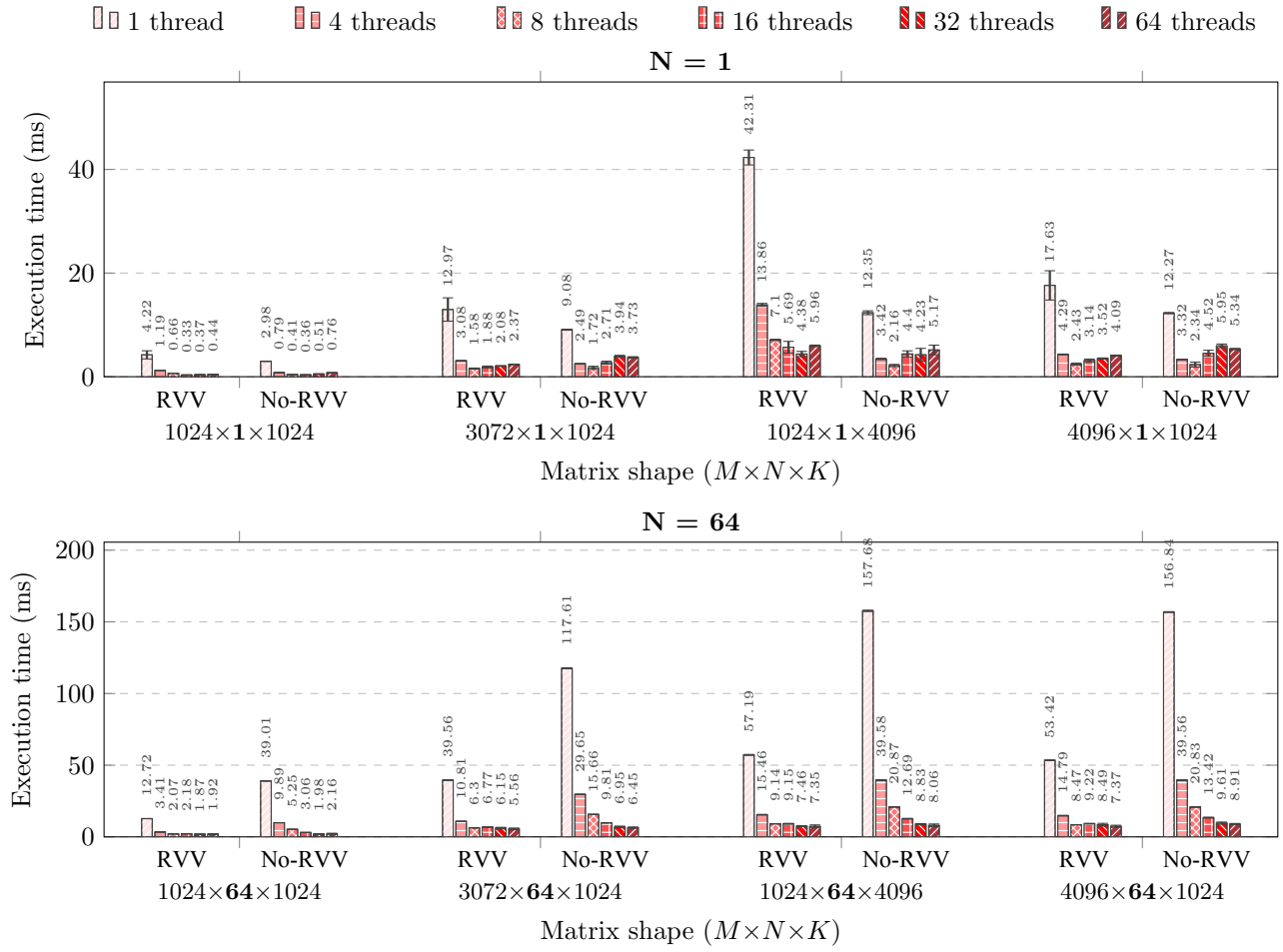


Figure 10: Performance of OpenBLAS GEMM operations (fused multiply–add) simulating the GPT2-medium model GEMM computation, considering two different sizes for N, and RVV enabled and disabled.

Table 2
Theoretical Peak Performance [GFLOP/s] vs Number of Cores

Cores	1	4	8	16	32	64
Fp^{32}_{Peak}	4	16	32	64	128	256
$Fp^{32}_{Peak-Rvv}$	16	64	128	256	512	1024

registers of the underlying architecture can accommodate four 32-bit FMA operations per cycle, resulting in a peak performance of 8 flops per cycle. The resulting peak performance for different numbers of cores is reported in Table 2.

On SG2042, we have four memory controllers DDR4-3200, which theoretically can provide a maximum bandwidth of $25.6 \times 4 = 102.4$ GB/s. In addition, we measured the effective bandwidth using [27] and obtained 41.2 GB/s with 64 cores, which aligns with [33].

Figure 9 represents the roofline models in the case of matrices with sizes $M = 1024$, $K = 1024$, and varying $N = 1, 8$, and 64. Each subfigure represents the performance achieved by the kernels, varying the number of cores to 1, 8, 32, and 64. The kernel with $N = 1$ consistently falls in the memory-bound region under both theoretical and measured memory bandwidth ceilings. This is because, with $N = 1$ the estimated arithmetic intensity AI is:

$$\begin{aligned} AI &\approx \frac{O(2 \times M \times N \times K)}{O(4 \times (M \times N + N \times K + K \times M))} \\ &\approx \frac{O(M \times K)}{O(M \times K)} \\ &\approx O(1) \end{aligned} \tag{3}$$

which means BLAS Level 2 [12], where performance is constrained by memory bandwidth. Indeed, across all thread counts, the arithmetic intensity of the $N = 1$ case falls below the machine balance point BP:

$$\text{BP} = \frac{Fp_{peak}}{B_{peak}} \quad \left[\frac{\text{FLOP}}{\text{Byte}}\right] \tag{4}$$

If the arithmetic intensity of a kernel is below this value, it is memory-bound; otherwise, it is compute-bound.

On a single core (see Figure 9a), the measured memory bandwidth is ($B_{real} = 4$ GB/s). This places the $N = 1$ kernel, $AI = 0.49$ deep into the memory-bound region, especially for RVV having $BP_{RVV} = \frac{16}{4} = 4$ FLOP/Byte.

In this scenario the non-RVV implementation outperforms the RVV one. We impute this to vector instructions placing greater demands on memory bandwidth due to loading/storing multiple values simultaneously. Notice that we are close to the limit imposed by the memory bandwidth. Increasing the problem size via a larger N increases the arithmetic intensity, improving the utilization of RVV instructions. Already with $N = 8$, RVV surpasses the non-RVV performance, but we are at the BP border. At $N = 64$, the computation fully enters the compute-bound region, allowing for the full exploitation of RVV instructions. Notice that in this case, both RVV and non-RVV are able to reach close to their respective optimal performance limits given by Fp_{peak}.

Increasing the number of threads increases both F_{peak} and the measured peak memory bandwidth B_{peak}. This leads to two main effects. First, BP is pushed to the right because F_{peak} increases faster than the measured memory bandwidth. Second, the higher memory bandwidth generally leads to more efficient utilization of vector registers. However, managing multiple threads introduces overhead due to factors such as thread synchronization, contention for shared resources, and increased pressure on memory bandwidth and caches, which can offset the performance gains, mainly in the RVV cases.

With 8 cores, see Figure 9b, the $N = 1$ kernel remains memory-bound close to the memory-imposed limit, with the non-RVV version outperforming the RVV one. In this configuration, even the $N = 4$ kernel is memory-bound due to the larger BP. Furthermore, distributing the workload across eight cores reduces the number of floating-point operations per core, limiting performance and preventing effective utilization of RVV instructions. The $N = 64$ kernel remains compute-bound and has a size that allows it to take advantage of the vector instructions. However, as the number of cores increases to 32 and 64, see Figure 9c and 9d, the work per thread decreases substantially, and even if the relative positions of the kernel's arithmetic intensities with respect to BP stay approximately the same, see, e.g., 32 threads case, the performance gap to the optimal increases, and the benefits of RVV diminish. We attribute this to the overhead of data movement and thread synchronization, which outweighs the benefits of extra compute capacity. This effect is particularly detrimental for small batch sizes, such as $N = 1$, which dominate LLM inference workloads.

4.4.1. Validation on GPT2-medium

To validate the theoretical insights drawn from the roofline model, we measured the execution time of all `GEMM` operations performed during a full inference pass of the `GPT2-medium` model, both with and without RVV-enabled OpenBLAS. Since the idea here is to test higher N values, we implemented a program that simulates only the GEMM operations performed by the model, using the exact same matrix shapes and running them in the same order as the real model. We compared two representative configurations: $N = 1$, which corresponds to the most common case in autoregressive decoding, and $N = 64$, which simulates a higher-batch workload. The results are summarized in Figure 10, which compares the average execution times of the GEMM kernels with varying core counts, grouped by matrix shape and batch size.

As predicted by the roofline analysis, for $N = 1$, the arithmetic intensity remains low, placing the computation in the memory-bound region. In this regime, RVV vector instructions provide no performance benefit and, in most cases, degrade performance due to increased memory bandwidth demands and register pressure. This is especially visible at low thread counts, where the non-RVV version consistently outperforms the RVV implementation across all tested matrix shapes.

In contrast, when N increases to $N = 64$, the arithmetic intensity rises significantly, pushing the computation into the compute-bound region. Under these conditions, RVV becomes more effective, and we observe substantial performance improvements with RVV over non-RVV execution, particularly at low-to-moderate thread counts (1–16 cores). This confirms that RVV can provide tangible acceleration benefits when the working set per thread is large enough to amortize vectorization overhead and when the compute-to-memory ratio crosses the machine balance point. However, at higher core counts (32 and 64), the performance gap narrows or reverses in some cases, likely due to diminishing returns from threading, increased contention on memory channels, and reduced per-thread workload, which limits the utilization of vector registers.

These results reinforce the roofline interpretation: RVV instructions are beneficial primarily when the arithmetic intensity is sufficiently high and the workload per core is large enough to justify vectorization overhead. This highlights the challenges of leveraging RVV for typical LLM inference workloads, which often consist of many small matrix multiplications with $N = 1$ and limited opportunities for compute-bound scaling.

5. Related Work

To our knowledge, no other work in the literature has investigated the performance of language models on a RISC-V architecture similar to the one we use in this work. Brown et al. [3] evaluated the parallel workloads from the RAJAPerf benchmarking suite on the SOPHON SG2042 SoC and compared the performance with state-of-the-art ARM and x86 architectures. The authors explored the RVV v0.7.1 capabilities using the XuanTie GCC compiler provided by the vendor to enable single-precision auto-vectorization on the RAJAPerf kernels. They observed that the SG2042 CPU delivers up to ten times more performance per core than the nearest widely available RISC-V hardware. Still, it was 4 to 8 times outperformed by the x86 CPUs in multi-threaded scenarios. They added that using custom thread mapping strategies is essential for leveraging performance on this

architecture. For this work, we tested various mapping strategies to enhance data locality with different degrees of parallelism, including the strategies used by Brown et al. [3]. However, compared to no-mapping, the strategies adopted did not show any significant performance gains.

Lee et al. [26] also tested the RAJPerf kernels on a RISC-V architecture with silicon-enabled RVV v0.7.1. But it is the T-Head C906 single-core CPU. They compared the RVV performance against the ARM NEON and SVE instruction sets and against the SiFive U74 RISC-V, which does not implement RVV. In some cases, the vectorized code on the C906 outperformed the U74 CPU with no vectors by about 80%. However, many other factors may influence this result when comparing two different platforms. Both [3] and [26] highlight the challenges of developing and running vectorized code on RISC-V, due to the immaturity of tooling and hardware.

Igual et al. [21] evaluated general matrix multiplication (GEMM) kernels on the C906 and C910 T-Head RISC-V architectures, both implementing RVV v0.7.1. Evaluating GEMM kernels is important because it is frequently the type of operation found inside language model layers. They evaluated the performance of the SGEMM kernels using OpenBLAS targeting RISC-V and RVV, as well as a version of OpenBLAS targeting a generic architecture with no particular support for RVV. They report performance gains up to 80% when using OpenBLAS with RVV. However, they also do not distinguish specific vector operations from any other optimizations that could be added by OpenBLAS.

Liu et al. [29] explore optimization strategies for running LLM inference on edge devices, focusing on efficiency under constrained compute and memory. While their approach emphasizes model-level techniques such as quantization and pruning, our work complements theirs by analyzing low-level GEMM performance under architectural constraints, particularly vectorization with RVV and thread scalability.

Banchelli et al. [2] highlight the limitations of the available batched BLAS APIs and propose and implement a new batched GEMM library specifically designed for small matrices. Their synthetic benchmarks achieved a maximum speedup of 32.6. Even though the authors evaluate RVV performance with GEMM and vary M and K parameters on $M \times N \times K$, the biggest matrix size they investigate is defined by $20 \times 9 \times 10$.

Olas et al. [33] evaluate the same SG2042 RISC-V platform in their work. Their experiments with the NAS Parallel Benchmark Suite show that only the embarrassingly parallel problem was able to scale performance up to 64 parallel threads. Peak performance in the other problems was with 32 or 48 threads. To improve the performance scalability of a memory-bound application, they applied several optimization techniques to increase computational intensity and decrease memory traffic, including the use of vectorization. By applying all the optimization techniques to fine-tune the application, they manage to scale performance up to 64 threads. Therefore, the authors focused on heavy manual optimizations on a single parallel application with a very specific parallel implementation to improve performance. Our work addresses the challenges of scaling the performance of state-of-the-art LLMs on current HPC RISC-V architectures.

Pirova et al. [35] investigate the acceleration of banded matrices on RISC-V processors with RVV support, focusing on the exploitation of structured sparsity to reduce memory accesses and improve vector utilization. Their approach targets matrices with narrow bandwidths, allowing them to use custom storage formats and alignment strategies to maximize vector lane occupancy. In contrast, our work focuses on dense matrix multiplications (`addmm`) that arise during LLM inference, particularly with small batch sizes (N=1), where the irregular shapes (e.g., $M \times 1 \times K$) challenge effective vectorization. While both works analyze RVV performance, our study highlights the difficulties of applying RVV to dense GEMMs in end-to-end transformer models, especially under thread contention and limited instruction-level parallelism. Furthermore, we explore how the surrounding model context and threading behavior affect real-world performance, aspects that are not addressed in [35], which evaluates isolated kernel performance in synthetic scenarios.

Several prior works have examined SIMD/vectorization for deep-learning workloads on other ISAs. On ARM, studies exploiting NEON/SVE instructions (often combined with quantization and kernel fusion) show substantial latency and throughput gains for transformer and quantized workloads [31, 46], and recent work applied SVE on the training of full LLMs (LLaMA-2), reporting up to 4.25× speedups with compiler-assisted vectorization [40]. Related RISC-V efforts evaluated vectorized posit arithmetic and reported large speedups for small matrix–vector and GEMM kernels when comparing RISC-VV to ARM SVE [8]. A study about accelerating CNN inference on long vector architectures [18] investigates manual/vectorized kernel optimizations, vector length, and cache sizing on RVV and ARM-SVE and shows that longer vectors and larger caches, together with BLIS-style GEMM tuning, can dramatically improve CNN kernel performance.

While these works collectively demonstrate that SIMD provides large benefits for many deep learning kernels, they differ from our study in workload and scope: they focus on CNNs, quantized models, or synthetic kernels and often use simulators (gem5) or ARM silicon, whereas we analyze full LLM inference on real silicon RVV v0.7.1 and explicitly isolate the impact of RVV on GEMM within whole model runs. Importantly, whereas prior ARM/x86/SVE studies generally report consistent SIMD gains, our experiments reveal that RVV's effectiveness for LLMs is highly dependent on arithmetic intensity and batch size. This underlines the need for ISA-specific, workload-aware evaluations of vector extensions such as RVV.

6. Conclusion

This paper assessed the inference performance of pre-trained language models on a multi-core RISC-V CPU with silicon-enabled vectors version 0.7.1. The first challenge

involved building the PyTorch acceleration library with support for vectorization on RISC-V architectures using OpenBLAS and BLIS.

Our experimental evaluation demonstrates that enabling hardware vectorization with RVV on RISC-V can yield performance gains for large language model inference, but these gains are highly dependent on configuration choices. In particular, we found that disabling the `use_cache` feature of the Transformers library impacts the performance less when using RVV, which indicates that RVV has its potential shaded because of the reduced amount of computation caused by the caching mechanism.

However, our deeper investigation reveals that RVV-enabled matrix multiplication is frequently slower than scalar implementations when inference workloads operate on small batch sizes (e.g., $N = 1$). This behavior is explained by the low arithmetic intensity of such GEMMs, which leads to memory-bound performance where vectorization cannot be fully utilized.

We confirm this through roofline modeling and experimental validation, using traced execution data from the PyTorch ATen library. These findings highlight the importance of profiling actual workloads rather than relying solely on microbenchmarks.

Our results also highlight that running large language models in data types not natively supported by the hardware, such as `bfloat16` on RISC-V, leads to flat and non-scalable performance due to the lack of support for this data type in the OpenBLAS and BLIS libraries. By converting models to `float32`, which is well-supported, we enable performance improvements and scalability across thread counts and BLAS backends. In this setting, vectorized backends like OpenBLAS with RVV deliver measurable benefits, particularly when multiple threads are used. These findings highlight the importance of data type compatibility in unlocking the potential of hardware-specific optimizations, such as RVV. Nonetheless, even when models use compatible data types, vectorization does not guarantee speedup. Workload characteristics such as matrix shape, number of threads, and arithmetic intensity must align for RVV to be effective. Overall, the results suggest that RVV is most beneficial in compute-bound configurations that avoid caching and rely on native data types.

This work opens several avenues for future exploration. The tested RISC-V architecture supports F16 data types. However, preliminary experiments showed that running BF16 models converted to F16 results in significant performance overhead compared to F32 converted models. Therefore, this unexpected behavior should be investigated in future work. Additionally, our experiments demonstrate that enabling RVV does not always result in consistent scalability. For instance, with certain models and thread counts, RVV performance plateaus or even regresses. Understanding how threading overheads, memory bandwidth saturation, and core-level contention affect RVV efficiency will be critical to future optimizations. We intend to expand our analysis to include quantized models and sparse representations, which may further amplify RVV benefits. We also plan to investigate further the impact of running unsupported data types on available RISC-V architectures against other state-of-the-art architectures (e.g., x86 and ARM). It would also be paramount to investigate the effects of the SG2042 NUMA design on performance, possibly by testing more complex task scheduling policies and understanding the associated overheads.

Acknowledgments

This work has been partially supported by the Spoke "FutureHPC & BigData" of the ICSC - Centro Nazionale di Ricerca in "High Performance Computing, Big Data and Quantum Computing", funded by the EU - NextGenerationEU, the EUPilot project funded by EuroHPC JU under G.A. 101034126, and the DARE SGA1 project funded by EuroHPC JU under G.A. 101202459.

CRediT authorship contribution statement

Adriano Marques Garcia: Methodology, Software deployment and execution, Data analysis, and Writing. **Giulio Malenza:** Methodology, Data analysis, and Writing. **Robert Birke:** Data curation, Writing - Original draft preparation. **Marco Aldinucci:** Conceptualization of this study, Resources provision.

References

[1] Asanović, K., Patterson, D.A., 2014. Instruction sets should be free: The case for risc-v. EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2014-146 .

[2] Banchelli, F., Jurado, D., Garcia-Gasulla, M., Mantovani, F., 2025. Exploring risc-v long vector capabilities: A case study in earth sciences. Future Generation Computer Systems 174, 107932. doi:https://doi.org/10.1016/j.future.2025.107932.

[3] Brown, N., Jamieson, M., Lee, J., Wang, P., 2023. Is risc-v ready for hpc prime-time: Evaluating the 64-core sophon sg2042 risc-v cpu, in: Proceedings of the SC'23 Workshops of The International Conference on High Performance Computing, Network, Storage, and Analysis, pp. 1566–1574.

[4] Brown, P., Dellapietra, V., Souza, P., Lai, J., Mercer, R., 1992. Class-based n-gram models of natural language. Computational Linguistics 18, 467–479.

[5] Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J.D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., Amodei, D., 2020. Language models are few-shot learners, in: Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., Lin, H. (Eds.), Advances in Neural Information Processing Systems, Curran Associates, Inc.. pp. 1877–1901. URL: https://proceedings.neurips.cc/paper_files/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf.

[6] Chen, C., Xiang, X., Liu, C., Shang, Y., Guo, R., Liu, D., Lu, Y., Hao, Z., Luo, J., Chen, Z., Li, C., Pu, Y., Meng, J., Yan, X., Xie, Y., Qi, X., 2020. Xuantie-910: a commercial multi-core 12-stage pipeline out-of-order 64-bit high performance risc-v processor with vector extension, in: Proceedings of the ACM/IEEE 47th Annual International Symposium on Computer Architecture, IEEE Press. p.

52–64. URL: https://doi.org/10.1109/ISCA45697.2020.00016, doi:10.1109/ISCA45697.2020.00016.

[7] Chetlur, S., Woolley, C., Vandermersch, P., Cohen, J., Tran, J., Catanzaro, B., Shelhamer, E., 2014. cudnn: Efficient primitives for deep learning. CoRR abs/1410.0759. URL: http://arxiv.org/abs/1410.0759, arXiv:1410.0759.

[8] Cococcioni, M., Rossi, F., Ruffaldi, E., Saponara, S., 2021. Vectorizing posit operations on risc-v for faster deep neural networks: experiments and comparison with arm sve. Neural Computing and Applications 33, 10575–10585. doi:10.1007/s00521-021-05814-0.

[9] Colonnelli, I., Birke, R., Aldinucci, M., 2023. Experimenting with pytorch on risc-v, in: RISC-V Summit Europe 2023, Barcelona, Spain. URL: https://iris.unito.it/retrieve/429bf344-9090-42c3-809c-1b8ac320a930/2023-06-08-Iacopo-COLONNELLI-abstract.pdf. poster.

[10] DeepSeek-AI, 2024. Deepseek llm: Scaling open-source language models with longtermism. arXiv preprint arXiv:2401.02954 URL: https://github.com/deepseek-ai/DeepSeek-LLM.

[11] Devlin, J., Chang, M., Lee, K., Toutanova, K., 2019. BERT: pre-training of deep bidirectional transformers for language understanding, in: Burstein, J., Doran, C., Solorio, T. (Eds.), Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT, Association for Computational Linguistics. pp. 4171–4186. doi:10.18653/V1/N19-1423.

[12] Dongarra, J.J., Du Croz, J., Hammarling, S., Hanson, R.J., 1988. An extended set of fortran basic linear algebra subprograms. ACM Trans. Math. Softw. 14, 1–17. URL: https://doi.org/10.1145/42288.42291, doi:10.1145/42288.42291.

[13] Elsadek, I., Tawfik, E.Y., 2021. Risc-v resource-constrained cores: A survey and energy comparison, in: 2021 19th IEEE International New Circuits and Systems Conference (NEWCAS), pp. 1–5. doi:10.1109/NEWCAS50681.2021.9462781.

[14] Foundation, U., 2024. oneapi math kernel library (onemkl) interfaces. https://github.com/oneapi-src/oneMKL.

[15] Garcia, A.M., Malenza, G., Birke, R., Aldinucci, M., et al., 2024. Assessing large language models inference performance on a 64-core risc-v cpu with silicon-enabled vectors, in: BigHPC2024: Special Track on Big Data and High-Performance Computing. CEUR Workshop Proceedings. volume 3785, pp. 1–9. URL: https://ceur-ws.org/Vol-3785/paper110.pdf.

[16] Grattafiori A. et al., 2024. The llama 3 herd of models. URL: https://arxiv.org/abs/2407.21783, arXiv:2407.21783.

[17] Greengard, S., 2020. Will risc-v revolutionize computing? Communications of the ACM 63, 30–32.

[18] Gupta, S.R., Papadopoulou, N., Pericàs, M., 2023. Accelerating cnn inference on long vector architectures via co-design, in: 2023 IEEE International Parallel and Distributed Processing Symposium (IPDPS), pp. 145–155. doi:10.1109/IPDPS54959.2023.00024.

[19] Han, X., Zhang, Z., Ding, N., Gu, Y., Liu, X., Huo, Y., Qiu, J., Yao, Y., Zhang, A., Zhang, L., Han, W., Huang, M., Jin, Q., Lan, Y., Liu, Y., Liu, Z., Lu, Z., Qiu, X., Song, R., Tang, J., Wen, J., Yuan, J., Zhao, W.X., Zhu, J., 2021. Pre-trained models: Past, present and future. AI Open 2, 225–250. doi:10.1016/J.AIOPEN.2021.08.002.

[20] Hao, K., 2020. Openai is giving microsoft exclusive access to its gpt-3 language model. MIT Technology Review .

[21] Igual, F., Piñuel, L., Catalán, S., Martínez, H., Castelló, A., Quintana-Ortí, E., 2023. Automatic generation of micro-kernels for performance portability of matrix multiplication on risc-v vector processors, in: Proceedings of the SC '23 Workshops of The International Conference on High Performance Computing, Network, Storage, and Analysis, Association for Computing Machinery, New York, NY, USA. p. 1523–1532. doi:10.1145/3624062.3624229.

[22] Karimi, A., Rossi, L., Prati, A., 2020. Adversarial training for aspect-based sentiment analysis with BERT, in: 25th International Conference on Pattern Recognition, ICPR 2020, Virtual Event / Milan, Italy, January 10-15, 2021, IEEE. pp. 8797–8803. URL: https://doi.org/10.1109/ICPR48806.2021.9412167, doi:10.1109/ICPR48806.2021.9412167.

[23] Khan, J., Fultz, P., Tamazov, A., Lowell, D., Liu, C., Melesse, M., Nandhimandalam, M., Nasyrov, K., Perminov, I., Shah, T., Filippov, V., Zhang, J., Zhou, J., Natarajan, B., Daga, M., 2019. Miopen: An open source library for deep learning primitives. arXiv:1910.00078.

[24] Kim, S., Hooper, C., Wattanawong, T., Kang, M., Yan, R., Genc, H., Dinh, G., Huang, Q., Keutzer, K., Mahoney, M.W., Shao, Y.S., Gholami, A., 2023. Full stack optimization of transformer inference: a survey. doi:10.48550/arXiv.2302.14017, arXiv:2302.14017.

[25] Koroteev, M.V., 2021. BERT: A review of applications in natural language processing and understanding. CoRR abs/2103.11943. URL: https://arxiv.org/abs/2103.11943, arXiv:2103.11943.

[26] Lee, J.K.L., Jamieson, M., Brown, N., Jesus, R., 2023. Test-driving risc-v vector hardware for hpc, in: Bienz, A., Weiland, M., Baboulin, M., Kruse, C. (Eds.), High Performance Computing, Springer Nature Switzerland, Cham. pp. 419–432.

[27] Lemire, D., 2024. Code used on daniel lemire's blog (2024-01-13). https://github.com/lemire/Code-used-on-Daniel-Lemire-s-blog/tree/master/2024/01/13. Accessed: 2025-07-26.

[28] Li, J., Tang, T., Zhao, W.X., Nie, J.Y., Wen, J.R., 2024. Pre-trained language models for text generation: A survey. ACM Comput. Surv. 56. URL: https://doi.org/10.1145/3649449, doi:10.1145/3649449.

[29] Liu, Z., Peng, L., Wang, W., Li, K., Zeng, B., Yu, J., Liu, X., 2025. Accelerating llm inference on risc-v edge devices via vector extension optimization, in: Huang, D.S., Zhang, C., Zhang, Q., Pan, Y. (Eds.), Advanced Intelligent Computing Technology and Applications, Springer Nature Singapore, Singapore. pp. 515–526.

[30] Ltd., T.H.S.C., 2024. T-head gnu compiler toolchain. https://github.com/T-head-Semi/xuantie-gnu-toolchain.

[31] Martínez, H., Catalán, S., Castelló, A., Mestre, J.I., Quintana-Ortí, E.S., 2025. Latency-critical quantized inference with transformer decoders on arm and risc-v cpus. IEEE Internet of Things Journal 12, 25676–25690. doi:10.1109/JIOT.2025.3560382.

[32] Nassyr, S., Mood, K.H., Herten, A., 2023. Programmatically Reaching the Roof: Automated BLIS Kernel Generator for SVE and RVV. Technical Report. Jülich Supercomputing Center. doi:10.34734/FZJ-2023-03437.

[33] Olas, T., Szustak, L., Wyrzykowski, R., Olas, M., Lapegna, M., 2025. Advances in adapting memory-bound cfd computations to risc-v multicore architecture, in: Lees, M.H., Cai, W., Cheong, S.A., Su, Y., Abramson, D., Dongarra, J.J., Sloot, P.M.A. (Eds.), Computational Science – ICCS 2025, Springer Nature Switzerland, Cham. pp. 151–166.

[34] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., Chintala, S., 2019. Pytorch: An imperative style, high-performance deep learning library, in: Advances in Neural Information Processing Systems 32. Curran Associates, Inc., pp. 8024–8035.

[35] Pirova, A., Vodeneeva, A., Kovalev, K., Ustinov, A., Kozinov, E., Liniov, A., Volokitin, V., Meyerov, I., 2025. Performance optimization of blas algorithms with band matrices for risc-v processors. Future Generation Computer Systems 174, 107936. doi:https://doi.org/10.1016/j.future.2025.107936.

[36] Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I., et al., 2019a. Language models are unsupervised multitask learners. OpenAI blog 1, 9.

[37] Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I., et al., 2019b. Language models are unsupervised multitask learners. OpenAI blog 1, 9.

[38] RISC-V, 2024. Risc-v "v" vector extension. URL: https://github.com/riscv/riscv-v-spec/blob/master/v-spec.adoc.

[39] Rivière, M. et al., 2024. Gemma 2: Improving open language models at a practical size. CoRR abs/2408.00118. URL: https://doi.org/10.48550/arXiv.2408.00118.

[40] Rossi, F., Cococcioni, M., Saponara, S., 2025. Llama-2 acceleration using the arm scalable vector extension, in: Ruo Roch, M., Bellotti,

F., Berta, R., Martina, M., Motto Ros, P. (Eds.), Applications in Electronics Pervading Industry, Environment and Society, Springer Nature Switzerland, Cham. pp. 261–268.

[41] Sutskever, I., Vinyals, O., Le, Q.V., 2014. Sequence to sequence learning with neural networks, in: Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N., Weinberger, K. (Eds.), Advances in Neural Information Processing Systems, Curran Associates, Inc. URL: https://proceedings.neurips.cc/paper_files/paper/2014/file/a14ac55a4f27472c5d894ec1c3c743d2-Paper.pdf.

[42] Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M.A., Lacroix, T., Rozière, B., Goyal, N., Hambro, E., Azhar, F., Rodriguez, A., Joulin, A., Grave, E., Lample, G., 2023. Llama: Open and efficient foundation language models. URL: https://arxiv.org/abs/2302.13971, arXiv:2302.13971.

[43] Waterman, A., Lee, Y., Patterson, D., Asanovic, K., level Isa, V.I.U., Waterman, A., Lee, Y., Patterson, D., 2014. The risc-v instruction set manual. Volume I: User-Level ISA', version 2, 1–79.

[44] Wei, C., 2023. Sg2042 technical reference manual. Available at https://github.com/milkv-pioneer/pioneer-files/blob/main/hardware/SG2042-TRM.pdf.

[45] Williams, S., Waterman, A., Patterson, D., 2009. Roofline: an insightful visual performance model for multicore architectures. Commun. ACM 52, 65–76. doi:10.1145/1498765.1498785.

[46] Zhang, C., Zhu, X., Chen, L., Yang, T., Pan, E., Yu, G., Zhao, Y., Wu, X., Li, B., Mao, W., Han, G., 2025. Enhancing llm inference performance on arm cpus through software and hardware co-optimization strategies. Integrated Circuits and Systems 2, 49–57. doi:10.23919/ICS.2025.3568404.

[47] Zhao, W.X., Zhou, K., Li, J., Tang, T., Wang, X., Hou, Y., Min, Y., Zhang, B., Zhang, J., Dong, Z., Du, Y., Yang, C., Chen, Y., Chen, Z., Jiang, J., Ren, R., Li, Y., Tang, X., Liu, Z., Liu, P., Nie, J.Y., Wen, J.R., 2025. A survey of large language models. doi:10.48550/arXiv.2303.18223, arXiv:2303.18223.

Adriano Marques Garcia is a postdoctoral researcher at the University of Turin. He received his PhD in Computer Science at the Pontifical Catholic University of Rio Grande do Sul with a thesis on easing the benchmarking of parallel stream processing. He also received a Master's Degree in Electrical Engineering from the Federal University of Pampa with a master's thesis on a new benchmark suite for evaluating parallel frameworks' performance and energy consumption. His research interests include parallel stream processing, benchmarking, and HPC for AI.

Giulio Malenza (Student Member, IEEE) is a PhD student at the University of Turin. He received his BE degree in physics and MS in mathematical engineering at the University of Padua. He also received a master's in high-performance computing in 2021 at Trieste. His research focuses on the performance portability of different parallel programming frameworks used to port and optimize scientific codes on accelerators.

Robert Birke (Senior Member, IEEE) is a tenured Assistant Professor with the University of Turin, Italy. He has been a Visiting Researcher with IBM Research Zurich, Switzerland, and a Principal Scientist with ABB Corporate Research, Switzerland. He has published more than 120 papers. He is currently an Associate Editor of the IEEE TRANSACTIONS ON NETWORK AND SERVICE MANAGEMENT and COMPUTER COMMUNICATIONS. His research interests are in the broad area of virtual resource management, including network design, workload characterization, and AI and big-data application optimization.

Marco Aldinucci is a Full Professor and the Head of the Parallel Computing research group at the University of Turin. He has authored over 150 scientific articles and received several prestigious awards, including the HPC Advisory Council University Award in 2011, the NVidia Research Award in 2013, the IBM Faculty Award in 2015, and the Autodesk Award in 2021. His research interests lie in the broad area of parallel and distributed systems: from languages and programming models to tools for HPC, Cloud and large AI systems.