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

Adriano Marques Garcia[1,*], Giulio Malenza[1], Robert Birke[1] and Marco Aldinucci[1]

[1]Computer Science Department, University of Torino, Corso Svizzera 185, 10149 Torino, Italy

Abstract

The rising usage of compute-intensive AI applications with fast response time requirements, such as text generation using large language models, underscores the need for more efficient and versatile hardware solutions. This drives the exploration of emerging architectures like RISC-V, which has 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 large language models and other AI applications. This work aims to evaluate the BERT and GPT-2 language models 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 BLAS backends for PyTorch to enable vectorization. Enabling RVV in OpenBLAS improved the inference performance by up to 40% in some cases.

Keywords

RISC-V, RVV, PyTorch, LLM, XuanTie C920, SOPHON SG2042, OpenBLAS, BLIS

1. Introduction

In recent years, natural language processing (NLP) advancements have skyrocketed. This was largely propelled by developing sophisticated large language models (LLMs) such as 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 [1, 2]. This exceptional performance is fuelled by increasingly larger foundational models with billions of 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 [3]. 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 [4, 5, 6].

The main goal of this work is to assess the inference performance of the BERT and GPT-2 language models on the SOPHON SG2042 64-core RISC-V architecture with silicon-enabled vectors (RVV) v0.7.1 [7]. For this goal, we first built the PyTorch library using OpenBLAS and BLIS to leverage support for RVV v.0.7.1 instructions. 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.

BigHPC2024: Special Track on Big Data and High-Performance Computing, co-located with the 3rd Italian Conference on Big Data and Data Science, ITADATA2024, September 17 – 19, 2024, Pisa, Italy.

*Corresponding author.
adriano.marquesgarcia@unito.it (A. M. Garcia); giulio.malenza@unito.it (G. Malenza); robert.birke@unito.it (R. Birke); marco.aldinucci@unito.it (M. Aldinucci)
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)

The contributions of this work are as follows: (1) We provide a tutorial on how to enable RISC-V Vectors v0.7.1 support for PyTorch in the SOPHON SG204. (2) We evaluate how enabling RVV impacts the inference performance of large language models (BERT and GPT-2). (3) We evaluate an experimental version of the BLIS library that leverages RVV 0.7.1 on a real RISC-V architecture. (4) We analyze the scalability of model inference performance when increasing the parallelism of 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 the natural language. It is a model that predicts the likelihood of a sequence of words given the preceding context [8, 9]. Such models learn a language structure, grammar, and vocabulary from vast amounts of text data [10]. They then use that knowledge to perform various natural-language processing tasks, such as text generation, translation, summarization, sentiment analysis, and more [11, 12, 13]. 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 introduced the BERT model [1](LLM), which was notable for its dramatic improvement over previous state-of-the-art models. BERT introduced the use of the transformer architecture and attention mechanism, laying the way for the upcoming large language models (LLM). LLMs experienced a boost in popularity among the general public with the release of ChatGPT, a chat tool based on the series of Generative Pre-trained Transformer (GPT) [2] models developed by OpenAI [14].

GPT-2 [15] 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 [16]. It is considered a causal language model that predicts the next token (word) 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.

BERT [1] stands for Bidirectional Encoder Representations from Transformers. It is a language model based on the transformer architecture. BERT is not a causal language model but a masked one. It is an encoder-only architecture, lacking a decoder, which means that BERT can not be prompted or generate text. Although it was not originally designed to predict and generate the next sentences or words of a text, it still can be fine-tuned for this purpose. However, it increases the computational cost, and the quality of the generated text may not be comparable to that of other models, such as GPT-2.

2.2. 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 [17]. The first stable release, 1.0, has now been rectified. While most versions never made it into silicon [6], 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.

SOPHON SG2042 system-on-chip (SoC) contains 64 RISC-V cores divided into 16 clusters connected through a grid network [7]. 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 [18] 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 [19], 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 [20] when running on Intel CPUs or cuDNN [21] and MIOpen [22] when running on NVIDIA or AMD GPUs, or more generic ones such as various libraries from the BLAS family.

Colonnelli et al. [23] describe the first porting of PyTorch [24] 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 [25] 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 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 [26] and set `TARGET=C910V` to enable RVV. For BLIS, we used a modified version [25] based on BLIS v0.9.0, set "rv64iv0p7" as the target, and compiled it using the LLVM-EPI-0.7[1] compiler. 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 [23], 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 and GPT-2 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.

4.1. Benchmarks

We wrote a simple text generation benchmark application based on the GPT-2 and BERT language models to test and evaluate the performance of LLM models. This application receives a prompt as input and generates/predicts the next n tokens (words) as output. For GPT-2, we used the GPT-2 (Revision 909a290) model provided by OpenAI, with 124M parameters [27, 28]. For BERT, we used the bert-large-cased pre-trained version[2] (commit 06fa25d), which is similar in size to GPT-2 but has 24 layers, 4096 intermediate dimensions, 1024 hidden dimensions, 16 attention heads, and 336M parameters [1]. To make the generated text more human-like, we added extra routines to control some aspects, such as sentence length.

For all the experiments, we used the following input prompt both for BERT and GPT-2:

[1]https://repo.hca.bsc.es/gitlab/rferrer/llvm-epi
[2]https://huggingface.co/google-bert/bert-large-cased

– “*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.*”

And here is an example of the text generated by BERT:

– “*The quick brown fox ran. Run fast fox running hard. Go ahead. Running on faster the fox went forward quickly.*”

Notice that the generation depends on sampling the next token (word) 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. Performance Results

Figures 1 and 2 present the experimental results. The bars in the charts are the average of 10 executions, and the whiskers represent the standard deviation. It is important to note that we consider only the execution time of the model inference, not the total runtime of the application. We used our BERT and GPT-2 applications to generate 25 tokens (words) in all tests.

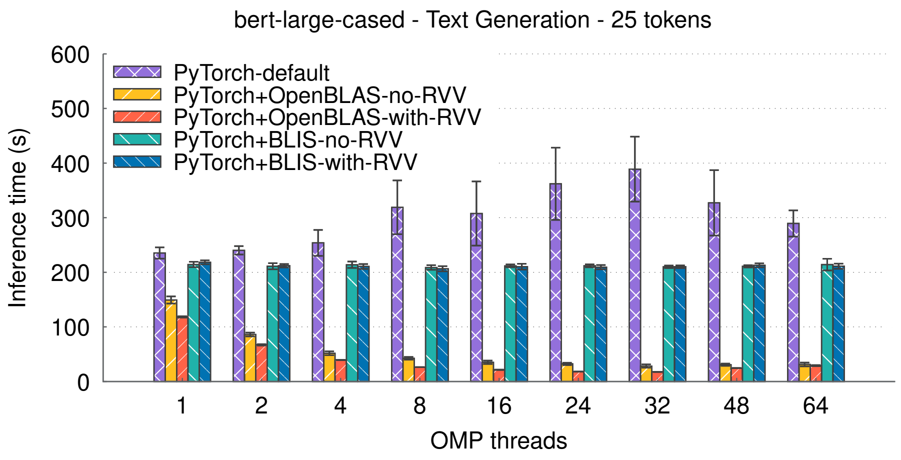

Figure 1: BERT inference performance using different BLAS runtimes on the SG2042 RISC-V CPU.

The chart with the performance results of BERT on the SOPHON SG2042 RISC-V CPU is presented in Figure 1. This chart compares the performance of the BERT benchmark with PyTorch using different BLAS runtimes. For this, we compiled five different versions of PyTorch: (1) PyTorch without OpenBLAS or BLIS (PyTorch-default); (2) PyTorch + BLIS compiled with generic target architecture; (3) PyTorch + an experimental BLIS library with support for RVV v0.7.1; (4) PyTorch + OpenBLAS targeting generic RISC-V architecture (no RVV); (5) PyTorch + OpenBLAS targeting the specific C910 architecture on the SOPHON SG2042 CPU.

For generating 25 tokens on the RISC-V CPU with a single thread, BERT with PyTorch-default is about 2 times slower than PyTorch + OpenBLAS with RVV. With 32 threads, it is 22 times slower. With 24 threads, PyTorch + OpenBLAS with RVV shows a 40% performance gain compared to OpenBLAS without RVV. This performance gain may be brought solely by SIMD operations. However, other optimizations may be applied to OpenBLAS when compiling it to target a specific architecture rather than a generic one.

On the other hand, BLIS shows no significant performance gains over the sequential execution with PyTorch-default. We wrote test programs in C++ to test OpenBLAS and BLIS’s capabilities in generating RVV instructions for GEMM operations, and both libraries passed the test. They generate RVV 0.7.1 instructions in the assembly code, which are properly executed. In these tests, BLIS with RVV behaves as expected and presents better performance. Our test programs implement the same main operation the models execute (`aten::addmm`). We also ensured that PyTorch was running the BLIS library as the backend for the main operations and that the parallelism was being set correctly. Therefore, a deeper investigation will be needed to understand the BLIS results.

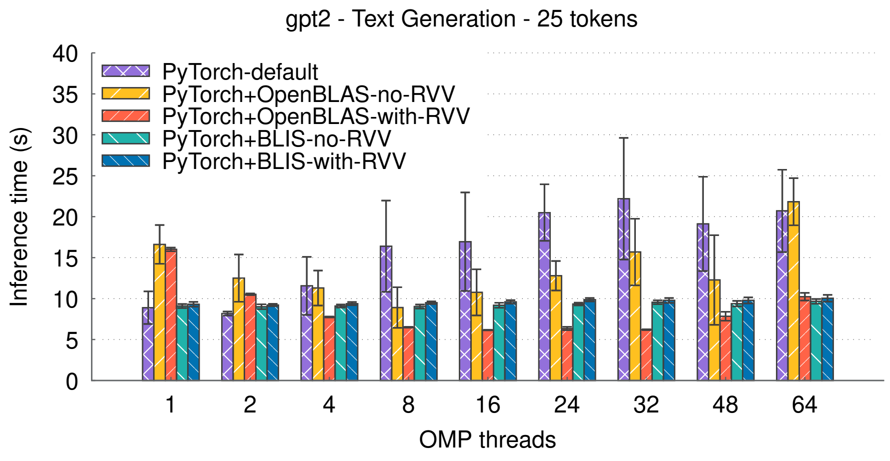

Figure 2: GPT-2 inference performance with different BLAS runtimes on the SG2042 RISC-V CPU.

The inference performance results using the GPT-2 model are presented in Figure 2. The behavior of the PyTorch-default is somewhat similar to the one observed with BERT. Increasing parallelism only leads to performance degradation. This is mainly due to the cost added by sharing data in the upper levels of the memory hierarchy, which is worsened in a mesh-like memory architecture of the SG2042 CPU [7]. PyTorch + BLIS also behaves the exact same way as in the BERT results. With OpenBLAS, performance scales only up to 24 threads. We hypothesize it is primarily due to overheads added by data communication plus a lighter load imposed by GPT-2 for generating 25 tokens if compared to BERT. In GPT-2, PyTorch + OpenBLAS with 1 and 2 threads perform unexpectedly poorly compared to PyTorch-default. Further investigation is needed to understand this behavior. However, with a higher number of threads, PyTorch + OpenBLAS with RVV enabled performs the best, improving over 30% the performance running with 16 threads compared to the sequential PyTorch-default. With 64 threads, the OpenBLAS with RVV is twice as fast as the OpenBLAS with no RVV. For both BERT and GPT-2, we ran experiments using different smaller model sizes and observed similar results.

```
       Name  Self CPU %    Self CPU  CPU total %  CPU total  CPU time avg  # of Calls
-----------  ----------  ----------  -----------  ---------  ------------  ----------
aten::addmm      94.50%      4.647s       95.36%     4.689s      32.116ms         146
 aten::gelu       2.31%   113.798ms        2.31%  113.798ms       4.552ms          25
```

Figure 3: PyTorch Profiler output when running BERT on a single core of the SOPHON SG2042 CPU.

```
       Name  Self CPU %    Self CPU  CPU total %  CPU total  CPU time avg  # of Calls
-----------  ----------  ----------  -----------  ---------  ------------  ----------
aten::addmm      60.21%   533.292ms       61.14%  541.477ms      11.281ms          48
   aten::mm      23.29%   206.269ms       23.29%  206.271ms     206.271ms           1
```

Figure 4: PyTorch Profiler output when running GPT-2 on a single core of the SOPHON SG2042 CPU.

Although the inference performance results show gains using PyTorch + OpenBLAS with RVV, it is unclear whether this gain comes from using RVV or other optimizations. OpenBLAS and BLIS do not provide tracing mechanisms to detail the type of instructions executed. OneDNN (oneAPI Deep Neural Network Library), for example, is a library that can be used with PyTorch to provide RVV support and can show which layers of the models execute vector instructions. However, the latest version of oneDNN only supports RVV v1.0 and still cannot vectorize any of the main layers of the models we tested, so we did not use it in this work. We tried using an experimental version of oneDNN that leverages RVV v0.7, but it is incompatible with other PyTorch dependencies.

Both language models should be highly vectorizable. Most of the operations involved in these two models are `aten::addmm` (matrix multiplication and addition) and also `aten::mm` for GPT-2. In addition, for BERT, we also notice that a minor percentage of processing time is spent on the `aten::gelu` (Gaussian Error Linear Units) primitive. For both models, the rest of the primitives comprise several calls of more negligible operations, as shown in Figures 3 and 4. Since OpenBLAS and BLIS are expected to properly vectorize matrix $\times$ matrix operations and both manage to do it in our tests without involving PyTorch, we are unsure why BLIS presented this expected behavior.

5. Related Work

To our knowledge, no other work in literature investigated the performance of language models on a RISC-V architecture similar to the one we use in this work. Brown et al. [29] 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.

Lee et al. [6] 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 in about 80%. However, many other factors may influence this result when comparing two different platforms. Both [29] and [6] point out the challenges of developing and running vectorized code on RISC-V due to the immaturity of tooling and hardware.

Igual et al. [30] 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.

The three related work we found [29, 6, 30] evaluate the RVV capabilities in similar CPUs as this work. However, only [29] investigated the SOPHON SG2042, with its complex NUMA design, but they did not focus on evaluating RVV performance. Although these works evaluated kernels with common operations found in inference model layers, evaluating the whole application adds extra complexity and can be more representative of real-world scenarios.

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. The experimental results showed that GPT-2 and BERT models perform better when using PyTorch + OpenBLAS with RVV in most test cases. However, it is unclear whether vectorization, different optimizations, or a wider combination of factors leverage this performance gain. Even though related work shows that OpenBLAS targeting RVV can provide up to 80% performance increase [30], they also do not guarantee that this performance gain comes only from using RVV. Unfortunately, to this point, the lack of performance analysis tools is a major drawback of the commercially available RISC-V architectures [6, 29].

In future work, a deeper and more comprehensive investigation should be carried out to understand better the RVV's impact on the performance of inference models on RISC-V architectures. For instance, verifying what model layers can be vectorized by OpenBLAS and BLIS in PyTorch may provide useful information on the maximum performance improvement that could be achieved solely through vectorization. Also, it would be paramount to investigate the effects of the SG2042 NUMA design on the performance, perhaps testing different task scheduling policies and comprehending what overheads are involved.

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 and the EuPilot project funded by EuroHPC JU under G.A. 101034126.

References

[1] J. Devlin, M. Chang, K. Lee, K. Toutanova, BERT: pre-training of deep bidirectional transformers for language understanding, in: J. Burstein, C. Doran, T. Solorio (Eds.), Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT, Association for Computational Linguistics, 2019, pp. 4171–4186. URL: https://doi.org/10.18653/v1/n19-1423. doi:10.18653/V1/N19-1423.

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

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

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

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

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

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

[8] I. Sutskever, O. Vinyals, Q. V. Le, Sequence to sequence learning with neural networks, in: Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, K. Weinberger (Eds.), Advances in Neural Information Processing Systems, volume 27, Curran Associates, Inc., 2014. URL: https://proceedings.neurips.cc/paper_files/paper/2014/file/a14ac55a4f27472c5d894ec1c3c743d2-Paper.pdf.

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

[10] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, D. Amodei, Language models are few-shot learners, in: H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, H. Lin (Eds.), Advances in Neural Information Processing Systems, volume 33, Curran Associates, Inc., 2020, pp. 1877–1901. URL: https://proceedings.neurips.cc/paper_files/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf.

[11] J. Li, T. Tang, W. X. Zhao, J.-Y. Nie, J.-R. Wen, Pre-trained language models for text generation:

A survey, ACM Comput. Surv. 56 (2024). URL: https://doi.org/10.1145/3649449. doi:10.1145/3649449.

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

[13] A. Karimi, L. Rossi, A. Prati, 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, 2020, pp. 8797–8803. URL: https://doi.org/10.1109/ICPR48806.2021.9412167. doi:10.1109/ICPR48806.2021.9412167.

[14] G. Brockman, I. Sutskever, Introducing openai, 2015. URL: https://openai.com/blog/introducing-openai/.

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

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

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

[18] C. Chen, X. Xiang, C. Liu, Y. Shang, R. Guo, D. Liu, Y. Lu, Z. Hao, J. Luo, Z. Chen, C. Li, Y. Pu, J. Meng, X. Yan, Y. Xie, X. Qi, 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, ISCA '20, IEEE Press, 2020, p. 52–64. URL: https://doi.org/10.1109/ISCA45697.2020.00016. doi:10.1109/ISCA45697.2020.00016.

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

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

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

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

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

[24] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, S. Chintala, Pytorch: An imperative style, high-performance deep learning library, in: Advances in Neural Information Processing Systems 32, Curran Associates, Inc., 2019, pp. 8024–8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

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

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

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

[28] HF Canonical Model Maintainers, gpt2 (revision 909a290), 2022. URL: https://huggingface.co/gpt2. doi:10.57967/hf/0039.

[29] N. Brown, M. Jamieson, J. Lee, P. Wang, 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, 2023, pp. 1566–1574.

[30] F. Igual, L. Piñuel, S. Catalán, H. Martínez, A. Castelló, E. Quintana-Ortí, 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, SC-W '23, Association for Computing Machinery, New York, NY, USA, 2023, p. 1523–1532. URL: https://doi.org/10.1145/3624062.3624229. doi:10.1145/3624062.3624229.