Evaluating AMD Instinct™ MI300A APU: Performance Insights on LLM Training via Knowledge Distillation

Dennis Dickmann
dennis.dickmann@seedbox-ai.com
Seedbox
Stuttgart, Germany

Philipp Offenhäuser
philipp.offenhaeuser@hpe.com
HPE
Böblingen, Germany

Rishabh Saxena
rishabh.saxena@hlrs.de
HLRS, University of Stuttgart
Stuttgart, Germany

George S. Markomanolis
georgios.markomanolis@amd.com
AMD
Lille, France

Alessandro Rigazzi
alessandro.rigazzi@hpe.com
HPE HPC/AI EMEA Research Lab
Basel, Switzerland

Patrick Keller
patrick.keller@hpe.com
HPE
Schwalbach, Germany

Kerem Kayabay
kerem.kayabay@hlrs.de
HLRS, University of Stuttgart
Stuttgart, Germany

Dennis Hoppe
dennis.hoppe@hlrs.de
HLRS, University of Stuttgart
Stuttgart, Germany

CCS Concepts

• **Hardware**; • **Mathematics of computing** → *Probability and statistics*; • **Computing methodologies** → **Artificial intelligence**;

Keywords

AI, ML, LLM, HPC, Computer Science, Deep Learning, MI300A

ACM Reference Format:
Dennis Dickmann, **Philipp Offenhäuser**, **Rishabh Saxena**, **George S. Markomanolis**, **Alessandro Rigazzi**, **Patrick Keller**, **Kerem Kayabay**, and **Dennis Hoppe**. 2025. Evaluating AMD Instinct™ MI300A APU: Performance Insights on LLM Training via Knowledge Distillation. In *Proceedings of Cray User Group Conference (CUG).* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

Abstract

Advanced Micro Devices, Inc. (AMD) has recently launched the MI300A accelerated processing unit (APU), which integrates central processing unit (CPU) and graphical processing unit (GPU) compute in a single chip with unified high bandwidth memory (HBM). This study assesses the AMD Instinct™ MI300A capabilities and examines how this new architecture handles real-world generative artificial intelligence (AI) workloads. While performance data for large language model (LLM) use cases exists for the MI250X and MI300X, to the best of our knowledge, such assessments are absent for the MI300A. We apply a knowledge distillation (KD) use case to distil the knowledge of the Mistral-Small-24B-Base-2501 teacher model to a student model that is 50% sparse using a 2:4 sparsity pattern. The results show quasi-linear scaling of raw performances on up to 256 APUs. Lastly, we discuss the challenges, research, and practical implications.

CUG, New York City, NY

ACM ISBN 978-x-xxxx-xxxx-x/YYYY/MM
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 Introduction

With AMD systems making up about 60 percent of the total compute in the November TOP500 list [31], the company has reached its exascale heterogeneous processor milestone with the MI300A APU. According to TOP500 in November 2024, the MI300A powers the world's top exascale high performance computing (HPC) system, El Capitan [31].

Machine learning (ML) and AI influence HPC, as there is a demand for greater computational power to train larger models. Simultaneously, HPC use cases increasingly incorporate ML techniques. This poses several questions: Can HPC centres effectively use the MI300A for ML and Generative AI workflows? How does this new architecture perform compared to traditional architectures? AMD also released MI300X, which replaces the CPU chiplets in MI300A with GPU ones to serve solely as an accelerator [27]. Performance data for LLM workflows is available for the MI250X and MI300X, but there's limited information on the MI300A [7, 21]. This study evaluates the MI300A using KD, a state-of-the-art LLM compression method employed in MiniLLM [11], Gemma-2 [29], and Apple Intelligence Foundation Language Models [12], and tries to show the standalone performance of the MI300A on an HPC system called Hunter [14] (Section 3.1).

Our primary objective is to demonstrate the effectiveness of a hybrid APU-based architecture, exemplified by the AMD MI300A, in scaling the distributed training of the latest versions of LLMs. Leveraging state-of-the-art deep learning frameworks, we focused on achieving efficient scaling on a unified memory architecture-based system, exploiting unique optimizations tailored for the MI300A. We first compiled a 20 billion token multilingual corpus spanning 24 European languages, enriched with high-quality synthetic text, to continuously pre-train a teacher model using tailored learning rates for embeddings and linear layers. We then tested two pipelines

for compressing and fine-tuning over-parametrized LLMs through structured 2:4 sparsity and advanced knowledge distillation. A layer-wise optimal brain-surgeon algorithm, guided by Hessian inverses, prunes redundant weights while preserving linguistic performance. The student model is subsequently distilled with a combined cross-entropy, Kullback-Leibler (KL) divergence, and Mean Squared Error (MSE) objective over hidden states, requiring fewer data-points than KL-only methods. Experiments demonstrate that this integrated pruning and distillation approach achieves substantial compression using structured sparsity, without compromising model fidelity.

Along with these two pipelines as our real-world AI workload, we also present strong scaling results for training Meta AI's LLM (Llama) 3.1 70B with DeepSpeed [23] done as a preliminary test-case using 4 APUs per node, one rank per APU, and the AWS-OFI RCCL plugin [2] as back-end for collectives (Section 4.1). The learnings from this test case were then used to design the two main experiments presented in this paper (Sections 4.2 and 4.3). The results demonstrate quasi-linear scaling in terms of system performance (measured as the tokens processed during training per second).

By meticulously balancing workloads across four APUs per node and strategically utilizing the system's shared 512 GB memory pool, we effectively mitigate memory-induced bottlenecks and demonstrate near-linear scaling up to 64 nodes. These results substantiate the viability and advantages of hybrid architectures, such as the AMD MI300A, as versatile solutions for large-scale HPC and AI deployments. Lastly, we discuss the challenges, research, and practical implications. A comparative analysis of the results of the MI300A with other such accelerators is left as future work.

1.1 Contributions

This paper presents the following key contributions based on the results of the conducted experiments:

- **Hardware-Aware Model Optimization:** Among the unique optimizations for MI300A, we introduce SimplePrune (Section 3.5), a multi-stage pruning pipeline designed for hardware constraints. These optimizations demonstrate that parameter efficient models can remain effective while overcoming the architectural limitations of unified memory systems.
- **Implementation Framework for Novel Architecture Adoption:** We document challenges and solutions across three implementation stages in Section 6.1, providing a practical roadmap for researchers and engineers to overcome barriers when working with emerging hybrid architectures.
- **First Performance Analysis of MI300A for LLM Distillation:** To the best of our knowledge, this paper presents the first comprehensive performance characterization of AMD MI300A hybrid architecture for knowledge distillation of LLMs, establishing a baseline metric previously unavailable in the literature.
- **Decision Support for Novel Architecture Adoption:** Our results offer valuable insights for HPC centres evaluating trade-offs between unified hybrid architectures versus deploying dedicated hardware solutions for AI, with implications for resource allocation, operational complexity, and sustainable AI development at scale.

1.2 Paper Outline

The paper is organized as follows. Section 2 introduces the core concepts of the distributed model training pipeline. Section 3 describes our setup, including infrastructure, software environment, datasets, models, and algorithms used during the experiments. Section 4 presents the experiments done (i.e., the two pipelines for fine-tuning and KD), and Section 5 reports the corresponding results for the training and inference of the models, as well as the scaling performance of the system. Finally, Section 6 discusses challenges, implications, and future directions.

2 Background

This section looks at the various concepts, tools, techniques, frameworks presented in this paper and used for the four use cases, namely the (1) fine-tuning of SmolLM2-1.7B model, (2) distillation of SmolLM2-1.7B model using Mistral-Small-24B-Base-2501 model as a teacher, (3) fine-tuning of Mistral-Small-24B-Base-2501 model, and (4) distillation of Mistral-Small-24B-Base-2501 model using the same model as the teacher for 2:4 sparsity. It starts with the background of the core concept of knowledge distillation, as well as the methodology used for fully-sharded training using DeepSpeed, PyTorch, RCCL, and MPI.

Since the launch of language models such as ChatGPT [5], interest in LLMs has been steadily growing. At the time of writing, the largest models being published require more than 10^{25} Floating Point Operations per Second (FLOPS) and millions of dollars to be pre-trained from scratch [9]. Such models can then be further trained using various techniques, such as fine-tuning [3] which can improve the performance of the model by either tuning the outputs of a model to a particular task, as well as knowledge distillation, where a larger "teacher" model is used to train a smaller "student" model, either from scratch or from a pre-trained model, in order to improve the smaller model. These operations, while also computationally expensive, do not require systems as large as the ones used for pre-training from scratch. As such, this paper looks at evaluating one such system, called Hunter [14], for performing knowledge distillation on two models, as well as introducing a new model called KafkaLM. We further expand the concept of knowledge distillation in the next section.

2.1 Knowledge Distillation

KD emerged to avoid deploying model ensembles for numerous users, which can be complex and resource-intensive. Buciluǎ et al. [6] demonstrated that the knowledge of ensemble models can be compressed into a single model, and Hinton et al. [15] achieved strong results on MNIST using this approach. With the rapid advancement of LLMs, KD has gained popularity due to their substantial computational demands; even modest reductions in model size can significantly improve inference speed and resource efficiency while preserving most performance capabilities. KD is particularly valuable for compressing the expansive knowledge of large teacher models, even when their parameters are inaccessible. As a result, KD not only reduces computational overhead but also helps narrow the performance gap between proprietary and open-source models [33].

In the context of LLMs, two primary approaches to KD have emerged [11, 33]. Black-box KD fine-tunes the student model using only the prompt-response pairs generated by a teacher model via API access, making it suitable for distilling knowledge from closed-source LLMs. In contrast, white-box KD leverages the teacher model's output distributions, intermediate features, or activations. For example, MiniLLM is trained with access to the teacher's output distributions, enabling richer knowledge transfer [11]. This richer transfer allows the student model to better replicate the teacher's internal reasoning and feature representations, resulting in superior performance across tasks, especially in complex or knowledge-intensive applications where output alone fails to capture the full decision-making process.

Accordingly, this study uses white-box KD with full access to model weights. Instead of relying solely on the teacher's output distributions, we employ a dual approach that focuses on both the hidden embedding layers and output logits. Our fused loss function combines MSE over pooled hidden layer embeddings and KL divergence for output logit distributions. We observed that this combined approach speeds up convergence compared to logit-only distillation tests (Section 4.3).

2.2 Zero Redundancy Optimizer

A set of algorithms used during the training were the Zero Redundancy Optimizer (ZeRO) [22] algorithms that improve the orchestration and memory management of large-scale model training. They are an alternative to the 3D parallelism (Data [19], Model [25], and Pipeline [13]) that is usually used when training models across multiple-nodes with multiple GPUs. The goal of data parallelism is to distribute the training dataset across multiple GPUs in a cluster. Each GPU processes different batches of data using identical model copies. Both forward and backward passes are performed independently on each GPU. Subsequently, gradients from all GPUs are centrally aggregated to synchronously update the model parameters. As a prerequisite, the model (all the parameters, gradients, and optimizer states) must fit into the memory of a single GPU, which may not be the case as the model size goes beyond a certain number of parameters. To solve this problem, Model Parallelism tries to split the model itself onto different devices (GPUs) such that each layer, or a set of layers, is present on a device. The drawback to this approach is that the large amount of data being communicated becomes a challenge for the communication bandwidth between the nodes, and hence this approach is used for devices residing in the same node and connected via high-bandwidth interconnect. Finally, pipeline parallelism splits the model horizontally such that each instance of the model (or a set of layers of the model) is a stage, and each stage performs the forward and backward pass for a subset of the dataset and communicates the results, leading to higher utilization of the resources. For all these methods, the obvious drawback is the high communication cost, as well as duplication of various aspects of the model in each device.

To simplify this process and remove redundancy, Rajbhandari et al. [22] proposed ZeRO, splitting it into two component algorithms, one being ZeRO-DP, and the other being ZeRO-R. ZeRO-DP partitions the states of the model across the different devices instead of replicating them, where the states are the optimizer states, gradient states, and the parameter states, while keeping the overhead of communication the same as in DP. ZeRO-R further reduces the memory per device used by the model by partitioning the activation states on the devices and reconstructing them at runtime, as well as making temporary buffers static and defragmenting the memory at runtime. The implementation of these algorithms was done using the DeepSpeed Python framework, as explained in detail in Section 3.2.2.

Another component of ZeRO that was not implemented due to limitation of the underlying hardware was the ZeRO-Offload [24] algorithm which aims to reduce overall accelerator device memory consumption by offloading the model states to the CPU, i.e., the node DRAM, thereby increasing the size of the models that can be trained on the system by converting memory-bound model training to compute-bound model training. Since the MI300A has a unified memory architecture, the shared memory of the accelerator devices means that the total available memory for the CPU and GPU is the same, and hence, no benefit can be achieved due to offloading.

In the next section, we look at some implementation details regarding the libraries and frameworks used for the experiments.

3 Setup

This section looks at the components that were needed for the use cases that are presented later in this paper. These represent the constraints and design choices that were made during the presented experiments.

3.1 Infrastructure

In this section, we outline the hardware specifications of the APU partition of Hunter, the HPE Cray Supercomputing EX4000 located at High-Performance Computing Centre Stuttgart (HLRS), on which the presented experiments were run.

3.1.1 MI300A Node. Each node has four AMD Instinct™ MI300A APUs, connected through Infinity Fabric™ (XGMI) as shown in Figure 1. Each APU is attached to a 200 Gbps endpoint of the underlying HPE Slingshot [8] 11 Dragonfly network [16]. The MI300A integrates CPU and GPU compute in a single module with unified HBM memory and other system-on-a-chip components in one processor package [27]. The CPU and GPU chiplets share 128 GB of unified HBM-based memory, with a peak theoretical memory bandwidth of about 5.3 TB/s across 128 memory channels. Each channel is paired with a 2 MB slice of memory-side cache, boosting the bandwidth to up to 17 TB/s [27].

3.1.2 Slingshot Interconnect. As mentioned in Section 3.1.1, the Hunter system is connected through a Dragonfly network topology using the Slingshot Interconnect technology from HPE. It uses Rosetta switches which contain 32 tiles and a crossbar implementing 64 ports, with 200Gb/s bandwidth in each direction. Each port has 4 lanes, each supporting 56Gb/s per direction bandwidth. Using these switches, the system is connected in a Dragonfly topology, which is a hierarchical direct topology where each switch is connected to the compute nodes and the other switches. The system supports RoCEV2 as well as InfiniBand verbs, the latter being used for our experiments.

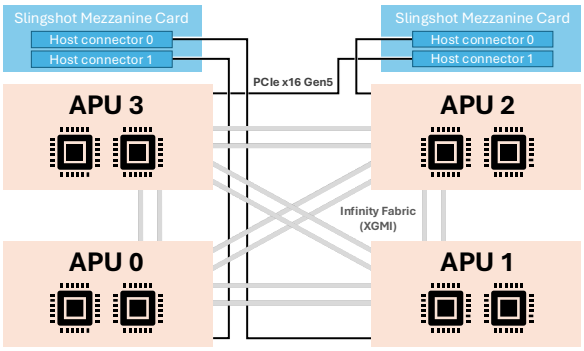

Figure 1: Simplified view of the APU node architecture of Hunter.

3.2 Frameworks and Libraries

To conduct the experiments and evaluation on the MI300A for fine-tuning and KD, the programming language of choice was Python, which is supported by many machine learning, deep learning, and distributed large-scale model training libraries that make it easier for developers to perform their experiments. This section looks at the major tools and libraries used in all the experiments and evaluations conducted.

3.2.1 PyTorch. To implement the core model training functionalities, we used PyTorch [20], which is a library initially developed by Meta AI, implementing GPU acceleration through underlying libraries like CUDA and ROCm. For our use case, we used the ROCm (Version 6.2.2) backend for PyTorch, which is optimised for AMD GPUs. The library also contains data parallel and Fully-Sharded Data Parallel training paradigms, but those were not used for this experiment in favor of DeepSpeed, as explained in the following section. PyTorch was chosen as the core library for the implementation due to its syntactical simplicity, easy integration with GPU acceleration backends, its ability to integrate custom loss functions, and the large user base that allows superior support.

3.2.2 DeepSpeed. Base PyTorch provides implementation of parallel algorithms such as Fully-Sharded Data Parallel (FSDP), but these were not chosen due to their complexity and difficulty in scaling to larger ranks. We used another library called DeepSpeed [23], which contains orchestration strategies and optimization algorithms for large-scale model training. Section 2.2 describes one such optimization algorithm used during the training. As mentioned in Section 3.2.1, we chose DeepSpeed for its ease-of-use, along with the added benefits of memory optimization, parallelism algorithms, reduction in communication overhead, and finally broad support among researchers.

3.2.3 ROCm Collective Communication Library. For intra and inter node communication the ROCm Collective Communication Library (RCCL) [2] was used. RCCL provides multi-GPU and multi-node collective communication primitives optimized for AMD GPUs. RCCL provides routines such as all-gather, all-reduce, reduce-scatter, and many more collectives as well as point-to-point (GPU-to-GPU) communication. RCCL is optimized to achieve high bandwidth and low latency on high-speed interconnects. To use the full potential of the Slingshot interconnect of Hunter, the AWS-OFI-RCCL [2] plug-in was needed along with the RCCL libary. The plug-in enabled the usage of libfabric as a network provider while running RCCL based applications. Since libraries such as PyTorch or DeepSpeed [23] have integrated RCCL to accelerate deep learning training on multi-GPU multi-node systems, this combined setup provided an easy-to-use interface to develop the experiments.

3.3 Models

We used two pre-trained models for these experiments, namely the SmolLM2-1.7B model from HuggingFace and the Mistral-Small-24B-Base-2501 model from MistralAI.

3.3.1 SmolLM2-1.7B. smollm2 [1] is a family of models published by HuggingFace that were designed for inferencing on local devices due to their small size. The three models introduced were of 135 million, 360 million, and 1.7 billion parameters. For this experiment, we choose the 1.7 billion parameter model since it performed relatively well compared to its size. They were pre-trained on a 1 trillion token corpus of training data, smollmcorpus [4], which consists of high-quality educational and synthetic data designed for training small language models. This model was suitable as a student model due to its small size and limited corpus, along with its incorporation of the Llama architecture [32].

3.3.2 Mistral-Small-24B-Base-2501. Mistral-Small-24B-Base-2501 is a latency-optimized, 24 billion parameter model [30], developed by MistralAI and published on HuggingFace, as well as accessible on MistralAI's website. The model performs well when compared to similar sized models, and was chosen by the authors due to its robust evaluation metrics, as well the Llama architecture [32]. The Mistral-Small-24B-Base-2501 model was used as the teacher model in the KD process, as well fine-tuned using the custom dataset proposed in the next section.

3.4 Dataset

We constructed a diverse 20B-token dataset encompassing mathematics, programming, general internet text, and multilingual Wikipedia entries specifically tailored for post-pruning knowledge distillation. While contemporary approaches typically demand 50-100B tokens or more, our novel loss function, further explained in Section 3.6.3, inspired by Squarehead's sparsity-aware methodology [17], substantially reduces data requirements while effectively restoring model performance after pruning. The memory architecture of the AMD MI300A compute node, with its 512 GB unified memory capacity, necessitated the implementation of a custom data streaming collator. The collator dynamically feeds optimized pre-fetched mini-batches asynchronously to GPUs through a persistent pipeline, eliminating the prohibitive memory requirements of storing pre-tokenized multi-billion token datasets in memory. Our approach maintains near-optimal computational, between 88% and 90% GPU utilization, while minimizing memory overhead. This optimization can not only benefit the MI300A but also other systems where memory management is important.

3.5 SimplePrune

To address the architectural needs of the MI300A along with the requirement for hardware-compatible sparsity patterns [28], we introduce **SimplePrune**, a data-aware hierarchical pruning framework tailored for distributed LLM compression. SimplePrune is explicitly designed to align with the architectural characteristics of hybrid HPC-AI platforms. It employs a multi-stage pruning pipeline that integrates structured sparsity constraints compatible with GPU acceleration, ensuring that compression gains translate to actual runtime and memory benefits. The pruning process is organized hierarchically: it begins with coarse-grained block pruning and progressively refines the model through finer-grained operations, reducing model complexity while preserving functional capacity.

Performance degradation is quantified using a composite of perplexity - a metric that quantifies how well a language model predicts the next tokens in a text by measuring the average number of significant possibilities a model generates when predicting the next word and KL divergence metrics. Our results indicate that SimplePrune achieves substantial reductions in model size with negligible impact on task-relevant performance, even at higher compression ratios.

3.5.1 Motivation and Goal. Despite their success, LLMs are often significantly over-parametrized, leading to inefficiencies in inference and deployment, particularly when scaled across distributed GPU clusters. A large fraction of parameters contributes marginally to model output, enabling opportunities for targeted removal. Moreover, traditional pruning strategies tend to treat all layers uniformly, ignoring the layered sensitivity and varying structural importance across the model's depth.

SimplePrune addresses these limitations by implementing a top-down, sensitivity-aware pruning mechanism. The framework eliminates inactive neurons and superfluous parameters in a staged fashion, with each stage designed to minimize representational damage while progressively reducing the computational footprint.

Following pruning, we employ large-scale knowledge distillation on multi-billion token corpora to recover any residual loss in task fidelity. The distillation pipeline scales efficiently across MI300A-class hybrid CPU-GPU clusters, leveraging unified memory architectures and high-bandwidth interconnects. Empirical evaluations confirm that pruned models with up to 40% parameter reduction (down to ~15B parameters) maintain strong alignment with their uncompressed counterparts across perplexity and divergence-based metrics. These findings underscore the practicality of SimplePrune for scalable LLM deployment in memory-constrained distributed environments.

3.5.2 Data Collection and Analysis. The initial step in SimplePrune involves detailed profiling of activation magnitudes and Hessian-based sensitivity measures across model layers and neurons. Given the extensive data volume produced, we implement an asynchronous I/O strategy, capturing and storing model activations during forward passes to disk, effectively circumventing out-of-memory (OOM) issues on nodes equipped with 512 GB unified memory (e.g., AMD MI300A). This asynchronous approach ensures uninterrupted training and maximizes computational resource utilization for pruning tasks.

Due to highly skewed weight distributions in LLMs, conventional pruning algorithms are unable to differentiate critical neurons from redundant ones effectively. To address this, SimplePrune incorporates specialized, outlier-aware statistical analyses utilizing both empirical activation distributions and Hessian-based second-order metrics. These methods robustly classify neurons by importance, further refined by additional ranking procedures to align pruning decisions with global model importance.

3.5.3 Initial Pruning Solution. SimplePrune initiates pruning with a greedy algorithm targeting removable blocks and channels within Multi-Layer Perceptron (MLP) expansion layers. This process systematically assesses parameter removals based on minimal degradation of perplexity and KL divergence across a multi-task calibration dataset. By constraining the search to specific structural components, we reduce complexity while maintaining overall architectural integrity.

3.5.4 Combinatorial Optimization. Following the initial greedy pruning stage, SimplePrune employs a Tabu Search meta-heuristic [10] to explore and optimize a broader pruning configuration space. Leveraging the parallel capabilities of the MI300A APU cluster, this approach efficiently identifies cross-layer pruning synergies, significantly reducing evaluation time and mitigating performance regressions.

3.5.5 Post-Optimization Pruning with Structured Sparsity. We further enhance computational efficiency by enforcing 2:4 structured sparsity on selected attention and MLP layers. Interestingly, some layers demonstrated unexpected improvements, such as minor reductions in perplexity and KL divergence, highlighting structured sparsity's potential role in enhancing model regularization and generalization.

A more detailed explanation and optimization of the initial SimplePrune algorithm is planned as future work by the authors.

3.6 Algorithms

To attain optimal performance and scalability, we implemented a targeted training strategy explicitly addressing critical bottlenecks inherent in distributed, memory-constrained environments. Key components of this strategy include precise GPU resource management, optimized internode communication protocols, and memory-aware workload partitioning. Although detailed mathematical analyses and algorithmic specifics will be presented in a forthcoming study, the methodologies employed reflect contemporary best practices in data-parallelism and pipeline-parallelism for distributed LLM training. This section looks at the various optimizations that were implemented before and during the fine-tuning and KD experiments.

3.6.1 Teacher Model Calibration. To ensure the teacher model provides stable, high-quality supervision for distillation, we first calibrated the teacher model on our curated dataset. This calibration phase is essential since we lack precise information about whether portions of our distillation corpus overlap with the teacher model's pretraining data. Without calibration, the teacher might produce overconfident predictions for familiar examples or inconsistent logit distributions for out-of-distribution data, both of which would

compromise distillation quality. By fine-tuning the unpruned model on the curated corpus, we obtained more reliable and consistent logit distributions that serve as optimal distillation targets.

3.6.2 Knowledge Distillation. We implemented an optimized knowledge distillation pipeline leveraging dual DeepSpeed plugins simultaneously orchestrated by Hugging Face's Accelerate library. This configuration allowed us to efficiently transfer knowledge from the calibrated teacher model to the pruned student model while minimizing memory overhead. Unlike traditional inference engines that employ tensor-parallel per-node-sharding, our approach properly sharded the teacher model across all available APUs, reducing memory footprint by approximately 43% while maintaining computational efficiency. Training proceeded using DeepSpeed's hybrid engine with ZeRO-3, featuring both activation partitioning and optimizer state sharding to maximize the utility of the MI300A's unified memory architecture. We deliberately disabled CPU offloading, as mentioned in Section 2.2, due to MI300A's unified memory design. Instead, we implemented Non-Uniform Memory Access (NUMA)-aware memory pinning and strategic memory prefetching, achieving stable throughput exceeding 35,000 tokens/second. The distillation process utilized a per-device batch size of 2, gradient accumulation steps of 1, and a global learning rate of $2 \cdot 10^{-5}$, regulated through cosine annealing with a 2,000-step warm-up period to ensure stable convergence.

3.6.3 Fused Loss Function. Our fused loss function synergistically integrates three complementary mechanisms that significantly enhance the efficiency and effectiveness of KD. Central to our method is the flex_squarehead_loss ($\gamma = 1.0$), which aligns intermediate hidden representations from the teacher and student models—even when they differ structurally—by adaptively mapping layers based on relative depth and dimensionally matching features using adaptive pooling:

$$\mathcal{L}_{\text{SQH}} = \frac{1}{L} \sum_{i=1}^{L} \frac{\mathbb{E}\left[\left(h_i^S - h_{\phi(i)}^T\right)^2\right]}{\mathbb{E}\left[\left(h_{\phi(i)}^T\right)^2\right] + \epsilon} \tag{1}$$

Here, h_i^S represents the student's hidden state at layer i, and $h_{\phi(i)}^T$ denotes the teacher's hidden state at the corresponding mapped layer $\phi(i)$. Unlike conventional distillation techniques that solely mimic final output distributions, the intermediate feature matching provided by this loss offers a richer and more informative supervision signal across multiple abstraction layers, substantially accelerating convergence—reducing training time compared to logit-only methods. This loss is complemented by the KL divergence loss ($\beta = 1.0$), ensuring a close alignment of the student's output distribution to the teacher's softened logits:

$$\mathcal{L}_{\text{KL}} = T^2 \cdot \text{KL}\left(\text{softmax}\left(\frac{z^S}{T}\right) \middle\| \text{softmax}\left(\frac{z^T}{T}\right)\right) \tag{2}$$

Here, z^S and z^T are student and teacher logits respectively, and T is the temperature parameter. Finally, the Cross-Entropy (CE) loss component ($\alpha = 0.25$) serves predominantly as a regularization term, anchoring training to the ground-truth targets to prevent the student from overfitting to the teacher's predictions alone:

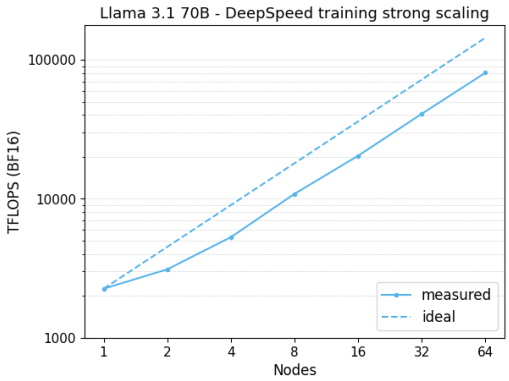

Figure 2: Strong scaling results from the pre-production run (Section 4.1). Configuration: 70.6B-parameters model Llama 3.1 70B with DeepSpeed, 4 APUs per node, one rank per APU, the AWS-OFI RCCL plugin as backend for collectives.

$$\mathcal{L}_{\text{CE}} = -\sum_{c=1}^{C} y_c \log(p_c^S) \tag{3}$$

Here, y_c represents the ground-truth label and p_c^S the student's predicted probability for class c. The collective integration of these three loss components creates a robust training signal, leveraging intermediate feature alignment, distributional consistency, and regularization, to significantly enhance parameter optimization and model convergence, thus ensuring high efficiency and superior performance in HPC environments.

The next section outlines the performed experiments in detail.

4 Experiments

This section outlines the results obtained from a pre-production test case on the Hunter system that guided this work by providing first performance metrics, and the 4 use-cases, or 2 experiment pipelines.

4.1 Pre-Production

Prior to these experiments, the authors conducted some preliminary tests on the Hunter system to provide a first-look into the performance and scaling of the system. This pre-production experiment looked at fine-tuning an LlaMa 3.1 70 billion parameter model. We scaled the workflow from a single node to 64 nodes, and saw strong scaling with respect to TFLOPS, indicating that the system was capable of large-scale AI workloads, as shown in Figure 2. Still, we recognized room for improvement, particularly when scaling from 1 to 4 nodes. Therefore, we investigated the communication overhead in Section 5.3. Based on the results, we decided to deploy more complicated workloads on the system to see how it performs under real-world AI workloads.

4.2 Experiment 1: Fine-Tuning

We conducted a comprehensive fine-tuning phase to optimize our teacher models for subsequent knowledge distillation. For the SmolLM2-1.7B model, we utilized an 8-node MI300A cluster with a per-device batch size of 4 and gradient accumulation steps of 2,

yielding an effective batch size of 64. Training proceeded with a cosine learning rate schedule starting at $5 \cdot 10^{-5}$ and 1% warm-up steps over a single epoch with sequence length of 1024 tokens. The larger Mistral-Small-24B-Base-2501 model required a more substantial 32-node MI300A configuration with per-device batch size reduced to 2 (maintaining the same gradient accumulation) and an elevated learning rate of 10^{-4} to compensate for the reduced effective batch size relative to model scale. Performance scaling was near-linear up to 16 nodes, after which communication overhead began to dominate, particularly for the smaller SmolLM2-1.7B model. The MI300A's unified memory architecture proved advantageous, allowing us to maintain full activation checkpoints while leveraging mixed-precision training with bfloat16.

4.3 Experiment 2: Knowledge Distillation

Our knowledge distillation protocol leveraged the calibrated teacher models to train compact, high-performance student models. For SmolLM2-1.7B, we employed an 8-node configuration with increased gradient accumulation steps of 4 to provide more stable gradient updates, effectively doubling the batch size compared to fine-tuning while reducing the learning rate to $2 \cdot 10^{-5}$. We extended training to 2 full epochs and implemented a temperature scaling factor of 3.0 in the KL divergence component to smooth teacher probability distributions. Larger configurations overheated the setup, causing out of memory (OOM) errors due to various aspects of distillation processing interacting simultaneously on numerous ranks. On the other hand, the Mistral-Small-24B-Base-2501 model required a substantial 64-node MI300A deployment with aggressive communication optimization to maintain throughput. We configured a per-device batch size of 2 without gradient accumulation, leveraging the increased node count to maintain update frequency while preserving batch statistics. The learning rate and temperature parameters matched the SmolLM2-1.7B configuration, ensuring methodological consistency across scales. Both distillation runs benefited from our three-component loss function. The squarehead component was particularly effective for the Mistral-Small-24B-Base-2501 model, where we observed that intermediate layer matching accelerated convergence compared to logit-only distillation in preliminary tests. Memory utilization remained efficient throughout the extended training runs due to our custom dual-DeepSpeed plugin configuration, which properly sharded both model states across all available compute units while minimizing communication overhead. A notable implementation detail was our handling of hidden state alignment between architecturally divergent teacher-student pairs. Rather than enforcing exact dimensional correspondence or employing complex projection layers, our adaptive pooling approach dynamically aligned feature dimensions based on relative layer depth, reducing both implementation complexity and computational overhead while maintaining distillation quality. Figure 3 shows the overall workflow implemented in Sections 4.2 and 4.3.

5 Results

This section looks at the results from the experiments mentioned in Section 4, specifically looking at two aspects of the training, namely the model training performance to see how well the distilled student model performs by looking at the training loss through the steps, and how the workflow scales to 64 nodes. All code has been published at https://github.com/Seedbox-Ventures/kafkalm and the model can be publicly accessed at https://huggingface.co/doubledsbv/KafkaLM-15B-Base.

5.1 Training Performance

Figure 4 shows the convergence behaviour during knowledge distillation training of the 15B pruned model across different compute setups. All runs employed identical optimization and loss settings. After meeting pre-defined convergence criteria, the distilled model achieved 96% accuracy recovery across various performance metrics. Further performance benchmarks will be conducted after accomplishing the post-training pipeline covering several steps from supervised fine-tuning, preference optimization and reinforcement learning with verifiable feedback, which at the time writing this article, is still under execution.

5.2 Inference Performance

Comparative benchmarking of the pruned Kafka 15B model against the Mistral-Small-24B-Base-2501 revealed significant performance differentials across critical inference metrics, as shown in Figure 5. This inference benchmarking pipeline was done on 1 MI300A node using the vLLM [18] library's batch inferencing mode and using 100 diverse prompts. Following parameter reduction from 24B to 15B parameters (a 37.5% decrease in model size), Kafka 15B demonstrated consistently superior computational efficiency despite its structural modifications when evaluated on a standardized test set of 100 identical prompts. Specifically, the pruned model exhibited a 50% reduction in Time to First Token (TTFT) (2.46s vs. 4.91s, $p < 0.01$), suggesting substantially reduced computational overhead during the initial decoding phase. Throughput measurements further corroborated this performance advantage, with Kafka 15B processing 6.55 prompts/second compared to Mistral's 4.7 prompts/second—representing a 38.3% improvement in overall inference capacity. This enhancement in processing velocity was paralleled by token generation metrics, where Kafka 15B achieved 812 tokens/second versus Mistral's 579 tokens/second (40% differential). Notably, these performance gains were achieved despite a substantial 37.5% reduction in parameter count, while maintaining comparable response quality as measured by perplexity and KL divergence metrics (detailed in Section 3.5.5), demonstrating that the hierarchical pruning methodology successfully preserved the model's functional capabilities despite the significant parameter reduction. These results suggest that SimplePrune effectively targets redundant neural pathways while maintaining critical information processing channels, resulting in models optimized for deployment in resource-constrained HPC environments.

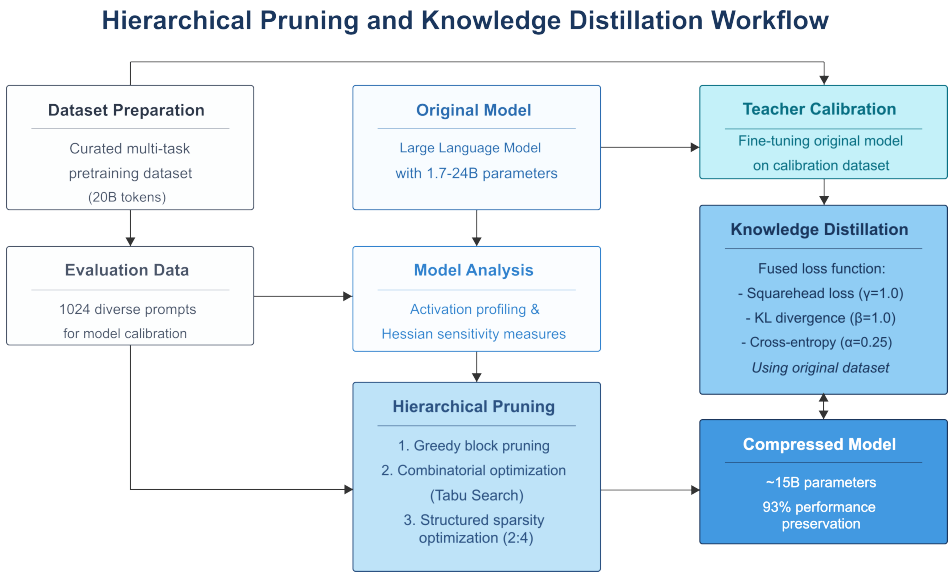

Figure 3: Diagram showcasing the steps involved in the knowledge distillation process. The complex frameworks required iterative development for the full realization of the pipeline.

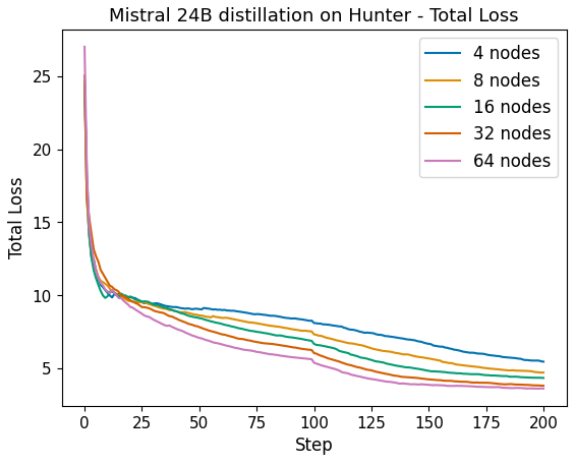

Figure 4: Total training loss for the 15B model (distilled from Mistral 24B) across different node counts, showing faster convergence with larger global batch sizes. The effective batch size is computed as: number of GPUs × gradient accumulation (2) × per-GPU batch size (1). The best convergence is achieved with a global batch size of 512 (64 nodes).

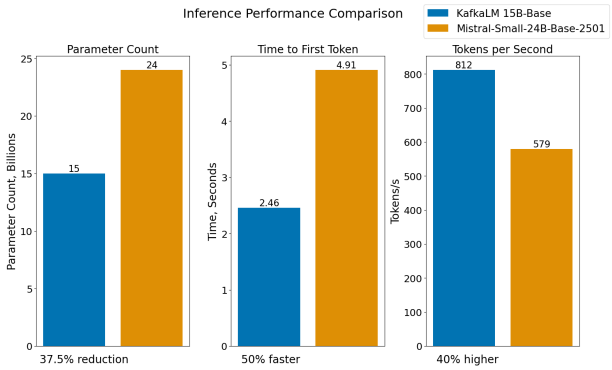

Figure 5: Comparison of Time to First Token and Tokens Per Second between KafkaLM-15B-base and Mistral-Small-24B-Base-2501, showing the performance increase between the two models, which results in minimal loss between outputs.

5.3 Scaling Performance

In the course of our experiments—particularly during the transition from a limited number of nodes to a larger-scale deployment—we sought to gain a deeper understanding of the workflow's scalability characteristics. Within AI workflows, performance evaluation relies on several key metrics, notably Tokens per Second (Tokens/sec) and FLOPs, often segmented by data type (e.g., FP16, BF16, INT8). These metrics are analysed in the context of the known theoretical peak performance of the underlying APU architecture, providing insight into both computational efficiency and scalability. As previously mentioned, due to the complicated nature of the models in the experiments, a certain FLOPs measurement was infeasible, and so we rely on tokens per second as our metric of choice.

We investigated the impact of communication overhead in this workflow by first profiling RCCL and analysing the collected data. RCCL profiling provided detailed metrics on collective communication operations, including operation type (e.g., all-reduce, all-gather), message size, communication time, and both algorithmic (algbw) and actual bus bandwidth (busbw). These metrics were essential for identifying performance bottlenecks and assessing scalability across distributed nodes, which also helped in providing insights into the optimization of the experiment pipelines.

In DeepSpeed, RCCL profiling can be enabled by adding the following configuration snippet to the training JSON file:

```
"comms_logger": {
    "enabled": true,
    "verbose": true,
    "prof_all": true,
    "debug": false
},
```

Then for every RCCL call, we printed all information to the standard output and logs of the training run, such as:

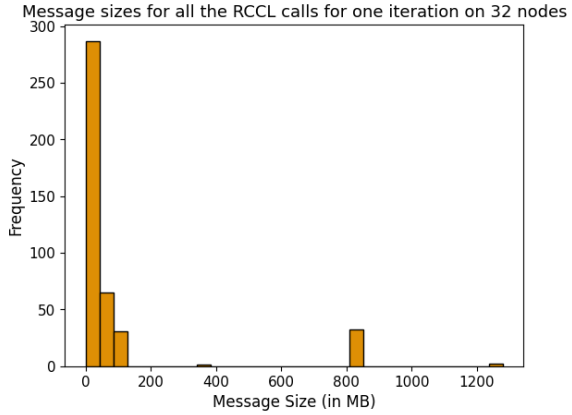

Figure 6: Histogram of the message sizes for 32 nodes. It can be seen that most messages were small in size.

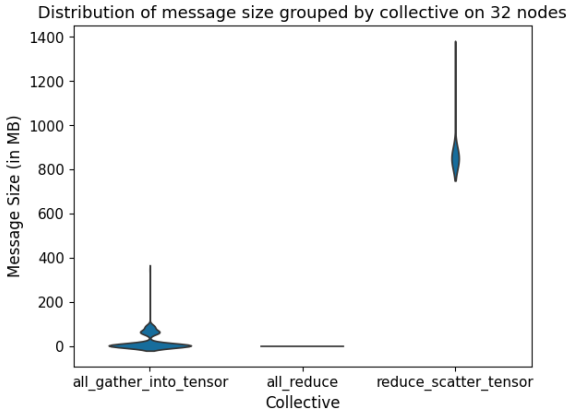

Figure 7: Distribution of message sizes grouped by collectives on 32 nodes. Most messages originated from the all_gather_into_tensor calls, but were really small in size.

```
[2025-04-03 17:01:45,873] [INFO]
[logging.py:128:log_dist]
[Rank 0] comm op: all_gather_into_tensor |
time (ms): 4.89 | msg size: 7.81 MB |
algbw (Gbps): 428.54 | busbw (Gbps): 415.15
```

Each log entry provided information on the collective operation type, execution time, message size, and both the algorithmic (algbw) and actual bus bandwidth (busbw), all of which were extracted for analysis. We analyzed the RCCL logs for one iteration as they remained the same across all the iterations.

As shown in Figure 6, a significant portion of RCCL communication consisted of small message sizes, with many falling below a few megabytes. In general, larger message sizes tended to perform better because of the fixed latency of the communication infrastructure, which had a more significant impact on smaller messages, reducing overall efficiency.

From Figure 7, we can see that the reduce_scatter_tensor uses enough large messages for higher performance where the other collectives use smaller messages. Observing the time spent per

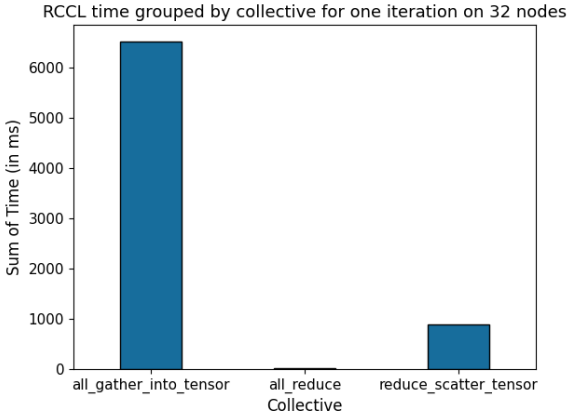

Figure 8: Collective communication time. Due to the high number of messages and their small size, the all_gather_into_tensor takes the majority of the communication time.

collective, we see that the all_gather_into_tensor required the most time to communicate, since all the data and the parameters were stored in a tensor format, as shown in Figure 8.

While the all_reduce operation appears to contribute the least to the overall time, as shown in Figure 8, this is attributed to the fact that only four calls were made, each involving a payload of just 8 bytes, leading to a minimal measured duration, presumably to transmit the initial run information to the other ranks.

The total data movement for the all_gather_into_tensor collective exceeded 7 GB, whereas for reduce_scatter it reached approximately 30 GB. Despite the smaller data volume, all_gather_into_tensor took longer to complete due to the large number of small messages, as shown in Figure 8.

We investigated various RCCL environment variables (aliased as NCCL environment variables for interoperability of DeepSpeed and PyTorch) that could potentially improve performance. While a detailed analysis is beyond the scope of this paper, we identified and adopted a set of parameters that provided measurable benefits. We used the following environment variables:

```
NCCL_DEBUG=INFO
NCCL_NET_GDR_LEVEL=3
NCCL_MIN_NCHANNELS=2
NCCL_MAX_NCHANNELS=16
```

The consensus in the HPC community recommends setting the environment variable NCCL_MIN_NCHANNELS to values like 32 or 72. Our experiments (conducted on fewer than hundreds of nodes) demonstrated improved efficiency with smaller values.

The workflow scaled efficiently from 4 nodes, where the model was already restricted, to 64 nodes, as it is demonstrated in Figure 9. Here, Estimated-Time-of-Arrival (ETA) is the time it takes for one step, multiplied by the number of steps in the epoch to process the data. With each doubling of the node count, the estimated time to completion (ETA) consistently decreased by a factor of approximately 2 to 2.2.

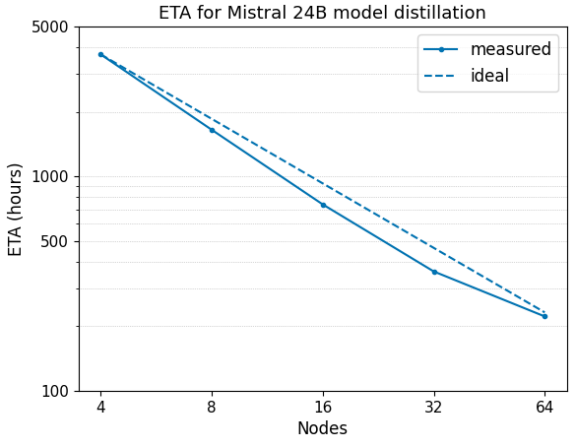

Figure 9: ETA scaling from 4 to 64 nodes of Mistral-Small-24B-Base-2501 on Hunter. The performance increases almost linearly due to the reduction in the time needed to perform the distillation.

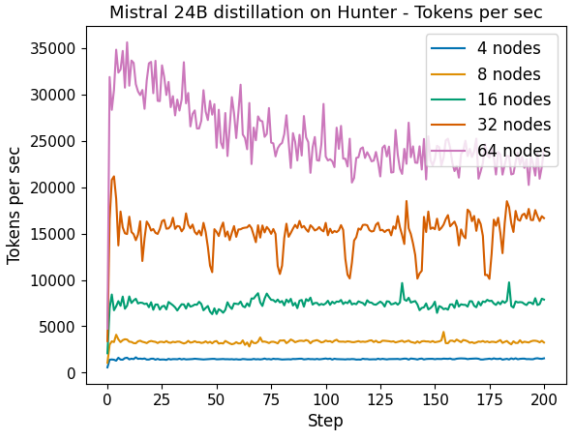

Figure 10: Tokens/sec scaling from 4 to 64 nodes of Mistral-Small-24B-Base-2501 on Hunter. One aspect to be noted is the unstable nature of the 64 node run, possibly due to a change in network congestion during the run on a shared system.

Figure 10 and Table 1 present the achieved tokens per second as the number of nodes scales from 4 to 64. The results exhibit the expected trend, demonstrating consistent performance improvements with increased node count. A point to be noted is the unstable nature of the 64 node training run, which could possibly be due to the network congestion on a shared system. As previously mentioned, we avoid using TFLOPS as a metric for these experiments due to the complicated nature of the interaction between the teacher and student models, which does not adequately represent the performance of the system. Instead, we choose tokens/sec as our benchmark metrics since the scaling of the workflow shows almost linear scaling for the node count of the experiments.

Finally, we demonstrate that a MI300A-based HPC-AI cluster can show promising results for complicated AI/ML workloads. In the next section, we summarize the findings of this paper.

6 Discussion and Conclusion

This section looks at summarizing our work, mentioning the challenges we faced during these experiments, and providing an outlook into possible future work.

# Nodes	Tokens/sec
4	1461.56 ± 76.85
8	3327.28 ± 219.97
16	7422.58 ± 587.03
32	15285.58 ± 1865.05
64	25455.49 ± 3740.09

Table 1: Mean value and standard deviation of tokens per second vs. number of nodes for Mistral-Small-24B-Base-2501 distillation.

6.1 Challenges

Due to the novel AMD MI300A APU architecture, we encountered challenges both in setting up the pipeline and during its execution. To address this, we introduced a pre-production run between the initial setup and the actual training. The main challenges and our approaches to overcoming them are discussed under the respective stages in the following subsections.

6.1.1 Pipeline Initialization. To get the pipeline running, we manually compiled DeepSpeed (0.16.1) with PyTorch (2.6.0), using the appropriate versions of ROCm (6.2.2), the RCCL OFI plugin for ROCm, among the AWS-OFI plugin (commit 17d41cb). The full software stack became available shortly before the pre-production run. Since the pipeline included numerous dependencies, we created unit tests with minimal software requirements to ensure that individual workflow components functioned correctly. This allowed us to focus our efforts on optimizing the pipeline.

6.1.2 Pre-Production Stage. Once the pipeline was running, we noticed scaling performance issues during the pre-production stage, particularly when scaling from one to four nodes, as shown in Figure 2. To address this, we investigated communication overhead by profiling RCCL and analysing the collected data. We identified several NCCL environment variables that delivered measurable performance gains. Such performance tuning should consider not only chip architecture but also factors like cluster size, as demonstrated in Section 5.3. These findings helped us perform the final experiments mentioned in Section 4.

6.1.3 Model Training Stage. The architecture of the AMD MI300A cluster imposed several non-trivial constraints that required us to rethink conventional model training and memory management strategies. Each node in our system comprises 4 GPUs with access to 512 GB of unified memory—significantly different from typical HPC configurations where nodes often provide a large, dedicated GPU memory, complemented by a similarly large amount of host

memory, making memory orchestration across model components particularly challenging.

To accommodate this, we deliberately avoided incorporating full 3D parallelism strategies such as Megatron-LM [26] at the initial stage of training. Instead, we adopted a hybrid strategy focused on aggressive memory optimization and control. This included sharding the teacher model across devices and implementing a custom dataset streaming and prefetching pipeline that dynamically loads batches during training, minimizing peak memory usage. Despite these efforts, operating close to the memory ceiling led to sporadic out-of-memory (OOM) events, often occurring after several hours of training. This fragility is a direct consequence of the unified memory layout, where all components—including model weights, optimizer states, gradients, and input sequences—compete within the same memory pool without fallback to system RAM.

We empirically identified configurations for the batch size that balanced throughput and memory utilization, but these configurations proved to be highly sensitive to seemingly minor changes in sequence length, optimizer state size, or model architecture. Additionally, the use of conventional profiling tools such as nsys and rocprof further exacerbated memory pressure, frequently triggering OOM errors and limiting our ability to perform detailed bottleneck analysis in-situ. These challenges provide us with valuable insights into using HPC-optimised hardware for AI/ML workloads, showing that these systems can be used for both purposes effectively, while only needing a better understanding of memory-management.

6.2 Implications

This section discusses the broader impact of our work on the MI300A architecture for knowledge distillation of LLMs. Our study bridges important gaps in the current literature while offering actionable insights for system architects, AI practitioners, and HPC centres.

6.2.1 Research Implications. Our study provides the first comprehensive performance data for the novel MI300A architecture in LLM knowledge distillation workflows. This addresses a significant gap in the literature, as previously available performance data existed only for the MI250X and MI300X architectures. By establishing these baseline metrics, researchers can now make more informed comparisons when considering platforms for LLM training and distillation tasks.

The performance characterization we provide creates a benchmark for future research on hybrid architectures designed for both simulation and AI workloads. Our findings could influence future architectural designs that aim to balance these different computational paradigms more effectively. This dual-purpose optimization represents an emerging trend in HPC where the traditional boundaries between HPC and AI workloads are increasingly blurred.

By presenting inference results from a state-of-the-art knowledge distillation pipeline, we demonstrate that models with substantially reduced parameter counts can maintain effectiveness, contributing to the LLM knowledge distillation body of knowledge. Our work provides empirical evidence for the viability of parameter-efficient models in specific contexts, potentially redirecting research efforts toward optimization techniques rather than simply scaling model size.

Furthermore, our results contribute to the broader discourse on computational efficiency in AI research. We introduce SimplePrune in Section 3.5, a multi-stage pruning pipeline designed with the architecture of MI300A in mind. This work aligns with the growing interest in sustainable AI development, where hardware-aware model design and training methods are becoming increasingly important.

6.2.2 Practical Implications. These results show the first results of SimplePrune as a pruning methodology, as well as KafkaLM-15, a 15 billion parameter model trained on European languages. It showcases the performance and training on sparse models, which perform on par with their large counterparts, reaffirming the viability of sparse models as substitutes for larger models. The successful parameter reduction demonstrated in our distilled models has profound implications for **organizations deploying AI models at scale**. Since real-world energy consumption is driven largely by inference rather than training, these efficiency gains could translate to substantial operational cost reductions and decreased environmental impact when deployed at scale.

Our preliminary results provide crucial decision-making support for **HPC centres** evaluating whether to:

- Deploy a single hybrid architecture like MI300A that can handle both simulation and AI workloads
- Invest in separate dedicated hardware architectures for different computational tasks

This decision has significant implications for resource allocation, software stack maintenance, and operational complexity.

The documented challenges and solutions across three implementation stages (Pipeline Initialization, Pre-Production, and Model Training) in Section 6.1 serve as a practical guide for **researchers and engineers** working with novel architectures. By sharing our approach to overcoming architectural limitations, we reduce the barrier to entry for utilizing cutting-edge hardware. This bridges the often-delayed implementation gap between hardware release and software optimization, accelerating the practical utility of emerging architectures for AI research.

Lastly, **chip designers** can benefit from our findings as they highlight specific architectural bottlenecks and optimization opportunities that affect AI workload performance. Our detailed analysis of the MI300A's behaviour during knowledge distillation workflows reveals particular hardware constraints regarding the unified memory layout, as identified in Section 6.1.3.

6.3 Outlook

While the current study demonstrates the effectiveness of a staged, hierarchical pruning pipeline—comprising block-level, channel-level, and fine-grained structured sparsity, each evaluated using perplexity and KL divergence—followed by a separate knowledge distillation phase. Such a design, while pragmatic, introduces inefficiencies in both the time-to-convergence and the optimality of the resulting model configurations.

Our future research direction would look towards a scenario in which pruning, performance evaluation, KD and Quantization Aware Training are fused into a single process allowing an efficient end-to-end optimizing of LLMs at larger scale and automation. This integrated paradigm would enable real-time trade-offs between

sparsity, accuracy, and computational cost, ideally leveraging reinforcement learning, differentiable sparsification objectives, or meta-learning strategies. Furthermore, such an approach could be made hardware-aware, optimizing the resulting models not only for parameter efficiency but also for FLOPs, memory bandwidth, and latency on heterogeneous compute architectures like MI300A-class clusters. As we see more and more HPC systems moving towards such architectures, it is crucial to analyse and benchmark AI/ML workloads on this architecture, for pure ML pipelines, as well as hybrid simulation-AI pipelines.

Alongside this, we would also like to bridge the gap between MI300A-style systems that are now becoming common in the HPC-community, and AI/ML users who want to deploy and use such systems. This would include a more comprehensive and comparative analysis of the performance metrics of such large-scale AI jobs.

Acknowledgments

The authors thank the High-Performance Computing Center Stuttgart (HLRS) for providing access to computing resources on their HPC system, Hunter, through the enabling pre-access program for project ELATO-2 (Application No: 62106). We also thank the AI Factory "HammerHAI"—co-funded by the European High Performance Computing Joint Undertaking (JU) and Germany under grant agreement number 101234027—for their close collaboration and invaluable support. We thank Ali Mohammed from the HPE HPC/AI EMEA Research Lab for his support.

References

[1] Allal, L. B., Lozhkov, A., Bakouch, E., Blázquez, G. M., Penedo, G., Tunstall, L., Marafioti, A., Kydlíček, H., Lajarín, A. P., Srivastav, V., et al. Smollm2: When smol goes big–data-centric training of a small language model. *arXiv preprint arXiv:2502.02737* (2025).

[2] AMD ROCm Contributors. AWS OFI RCCL. https://github.com/ROCm/aws-ofi-rccl, 2025. Accessed: 2025-05-02.

[3] Anisuzzaman, D., Malins, J. G., Friedman, P. A., and Attia, Z. I. Fine-tuning large language models for specialized use cases. *Mayo Clinic Proceedings: Digital Health 3*, 1 (2025), 100184.

[4] Ben Allal, L., Lozhkov, A., Penedo, G., Wolf, T., and von Werra, L. Smollm-corpus, Jul. 2024.

[5] Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. Language models are few-shot learners. *Advances in neural information processing systems 33* (2020), 1877–1901.

[6] Buciluǎ, C., Caruana, R., and Niculescu-Mizil, A. Model compression. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining* (2006), pp. 535–541.

[7] Dash, S., Lyngaas, I. R., Yin, J., Wang, X., Egele, R., Ellis, J. A., Maiterth, M., Cong, G., Wang, F., and Balaprakash, P. Optimizing distributed training on frontier for large language models. In *ISC High Performance 2024 Research Paper Proceedings (39th International Conference)* (2024), pp. 1–11.

[8] De Sensi, D., Di Girolamo, S., McMahon, K. H., Roweth, D., and Hoefler, T. An in-depth analysis of the slingshot interconnect. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis* (2020), IEEE, pp. 1–14.

[9] Epoch AI. Key trends and figures in machine learning, 2023. Accessed: 2025-03-28.

[10] Glover, F., and Laguna, M. *Tabu search*. Springer, 1998.

[11] Gu, Y., Dong, L., Wei, F., and Huang, M. MiniLLM: Knowledge distillation of large language models. In *The Twelfth International Conference on Learning Representations* (2024). Accessed: Jan. 22, 2025.

[12] Gunter, T., Wang, Z., Wang, C., Pang, R., Narayanan, A., Zhang, A., Zhang, B., Chen, C., Chiu, C.-C., Qiu, D., et al. Apple intelligence foundation language models. *arXiv preprint arXiv:2407.21075* (2024).

[13] Harlap, A., Narayanan, D., Phanishayee, A., Seshadri, V., Devanur, N., Ganger, G., and Gibbons, P. Pipedream: Fast and efficient pipeline parallel dnn training. *arXiv preprint arXiv:1806.03377* (2018).

[14] High-Performance Computing Center Stuttgart (HLRS). "hunter" supercomputer goes into service in stuttgart. *HLRS Press Release* (Jan. 2025). Accessed: 28.03.2025.

[15] Hinton, G., Vinyals, O., and Dean, J. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531* (2015).

[16] Kim, J., Dally, W. J., Scott, S., and Abts, D. Technology-driven, highly-scalable dragonfly topology. *ACM SIGARCH Computer Architecture News 36*, 3 (2008), 77–88.

[17] Kurtic, E., Kuznedelev, D., Frantar, E., Goin, M., and Alistarh, D. Sparse fine-tuning for inference acceleration of large language models, 2023.

[18] Kwon, W., Li, Z., Zhuang, S., Sheng, Y., Zheng, L., Yu, C. H., Gonzalez, J. E., Zhang, H., and Stoica, I. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles* (2023).

[19] Li, S., Zhao, Y., Varma, R., Salpekar, O., Noordhuis, P., Li, T., Paszke, A., Smith, J., Vaughan, B., Damania, P., et al. Pytorch distributed: Experiences on accelerating data parallel training. *arXiv preprint arXiv:2006.15704* (2020).

[20] Paszke, A. Pytorch: An imperative style, high-performance deep learning library. *arXiv preprint arXiv:1912.01703* (2019).

[21] Patel, D., Nishball, D., and Knuhtsen, R. MI300X vs H100 vs H200 Benchmark Part 1: Training – CUDA Moat Still Alive. SemiAnalysis, 2024. Accessed: Jan. 22, 2025.

[22] Rajbhandari, S., Rasley, J., Ruwase, O., and He, Y. Zero: Memory optimizations toward training trillion parameter models. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis* (2020), IEEE, pp. 1–16.

[23] Rasley, J., Rajbhandari, S., Ruwase, O., and He, Y. Deepspeed: System optimizations enable training deep learning models with over 100 billion parameters. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining* (New York, NY, USA, 2020), KDD '20, Association for Computing Machinery, p. 3505–3506.

[24] Ren, J., Rajbhandari, S., Aminabadi, R. Y., Ruwase, O., Yang, S., Zhang, M., Li, D., and He, Y. {Zero-offload}: Democratizing {billion-scale} model training. In *2021 USENIX Annual Technical Conference (USENIX ATC 21)* (2021), pp. 551–564.

[25] Shazeer, N., Cheng, Y., Parmar, N., Tran, D., Vaswani, A., Koanantakool, P., Hawkins, P., Lee, H., Hong, M., Young, C., et al. Mesh-tensorflow: Deep learning for supercomputers. *Advances in neural information processing systems 31* (2018).

[26] Shoeybi, M., Patwary, M., Puri, R., LeGresley, P., Casper, J., and Catanzaro, B. Megatron-lm: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053* (2019).

[27] Smith, A., Loh, G. H., Schulte, M. J., Ignatowski, M., Naffziger, S., Mantor, M., Kalyanasundharam, M. F. N., Alla, V., Malaya, N., Greathouse, J. L., Chapman, E., and Swaminathan, R. Realizing the amd exascale heterogeneous processor vision : Industry product. In *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)* (2024), pp. 876–889.

[28] Sreenivas, S. T., Muralidharan, S., Joshi, R., Chochowski, M., Mahabaleshwarkar, A. S., Shen, G., Zeng, J., Chen, Z., Suhara, Y., Diao, S., Yu, C., Chen, W.-C., Ross, H., Olabiyi, O., Aithal, A., Kuchaiev, O., Korzekwa, D., Molchanov, P., Patwary, M., Shoeybi, M., Kautz, J., and Catanzaro, B. Llm pruning and distillation in practice: The minitron approach, 2024.

[29] Team, G., Riviere, M., Pathak, S., Sessa, P. G., Hardin, C., Bhupatiraju, S., Hussenot, L., Mesnard, T., Shahriari, B., Ramé, A., et al. Gemma 2: Improving open language models at a practical size. *arXiv preprint arXiv:2408.00118* (2024).

[30] Team, M. A. Mistral Small 3 | Mistral AI.

[31] TOP500.org. TOP500 List - November 2024. TOP500, 2024. Accessed: Jan. 22, 2025.

[32] 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., and Lample, G. Llama: Open and efficient foundation language models, 2023.

[33] Xu, X., Li, M., Tao, C., Shen, T., Cheng, R., Li, J., Xu, C., Tao, D., and Zhou, T. A survey on knowledge distillation of large language models, 2024.