Efficient transformer adaptation for analog in-memory computing via low-rank adapters

NEUROMORPHIC
Computing and Engineering

OPEN ACCESS

RECEIVED
30 September 2025

REVISED
14 January 2026

ACCEPTED FOR PUBLICATION
2 February 2026

PUBLISHED
17 February 2026

PAPER

Efficient transformer adaptation for analog in-memory computing via low-rank adapters

Chen Li[1,4], **Elena Ferro**[2,4], **Corey Lammie**[2], **Manuel Le Gallo**[2], **Irem Boybat**[2] and **Bipin Rajendran**[1,3,*]

[1] Department of Engineering, King's College London, London, United Kingdom
[2] IBM Research Europe, 8803 Rüschlikon, Switzerland
[3] Current address: Institute for Intelligent Networked Systems, Northeastern University London, London E1 8BT, United Kingdom.
[4] These authors contributed equally.
[*] Author to whom any correspondence should be addressed.

E-mail: bipin.rajendran@kcl.ac.uk

Keywords: analog in-memory computing, AHWA-LoRA training, transformers

Supplementary material for this article is available online

Abstract

Analog in-memory computing (AIMC) offers a promising solution to the von Neumann bottleneck. However, deploying transformer models on AIMC remains challenging due to their inherent need for flexibility and adaptability across diverse tasks. For the benefits of AIMC to be fully realized, weights of static vector-matrix multiplications must be mapped and programmed to analog devices in a weight-stationary manner. This poses two challenges for adapting a base network to hardware and downstream tasks: (i) conventional analog hardware-aware (AHWA) training requires retraining the entire model, and (ii) reprogramming analog devices is both time- and energy-intensive. To address these issues, we propose AHWA low-rank adaptation (AHWA-LoRA) training, a novel approach for efficiently adapting transformers to AIMC hardware. AHWA-LoRA training keeps the analog weights fixed as meta-weights and introduces lightweight external LoRA modules for both hardware and task adaptation. We validate AHWA-LoRA training on SQuAD v1.1 and the GLUE benchmark, demonstrate its scalability to larger models, and show its effectiveness in instruction tuning and reinforcement learning. We further evaluate a practical deployment scenario that balances AIMC tile latency with digital LoRA processing using optimized pipeline strategies, with RISC-V-based programmable multi-core accelerators. This hybrid architecture achieves efficient transformer inference with only a 4% per-layer overhead compared to a fully AIMC implementation.

Deep learning has revolutionized various fields, from computer vision to Natural Language Processing (NLP), achieving unprecedented performance in the solution of complex tasks [1]. Much of this progress stems from scaling up neural network (NN) architectures and datasets, enabling models to capture more information and approximate complex functions that can be generalized to unseen samples. Among these advancements, the transformer architecture stands out as the backbone of large-language models (LLMs), allowing more effective network scaling and data compression [2]. However, as NNs grow in size and complexity, they demand more computational resources, leading to higher power consumption and carbon emissions [3]. This raises sustainability concerns and drives research into more energy-efficient architectures tailored to NN computation.

Analog in-memory computing (AIMC) has emerged as a promising computing paradigm to tackle these challenges, offering improved performance and energy-efficiency through computation directly within the memory array [4, 5]. However, distinct properties of AIMC, namely device noise and circuit non-idealities, introduce additional complexity to NN training and deployment [6]. Analog devices are inherently non-deterministic and subject to temporal variations, impacting NN accuracy when deployed on AIMC-based accelerators [7–10]. Analog hardware-aware (AHWA) training techniques have been demonstrated to enhance model robustness under these constraints for various NN architectures,

effectively mitigating accuracy losses by injecting Gaussian noise during forward-propagation and simulating circuit-non-idealities [11–13]. However, transformer-based architectures introduce several unique challenges that make the direct application of AHWA training techniques difficult. We summarize these limitations below, which motivate the development of a novel approach to efficiently adapt transformers for AIMC hardware.

First, training large-scale networks with a high parameter count quickly becomes computationally challenging. Transformers, in particular, require significantly more parameters than traditional architectures like convolutional NNs and long short-term memory that have been the primary focus of AIMC research. While the larger size of transformers contributes to their strong performance across many tasks, it also makes AHWA training challenging. Training such large models with simulated hardware constraints using existing AHWA approaches often exceeds graphics processor unit (GPU) memory limits and results in prohibitive computational costs.

Second, while pre-trained transformer models generalize remarkably well across a wide range of natural language processing tasks due to extensive training on large and diverse corpora [14], conventional AHWA training methodologies typically optimize performance for only one task at a time [12]. Such narrowly-focused, task-specific tuning under-utilizes the generalization potential of pre-trained models, resulting in hardware-optimized models that perform well on one task but struggle to generalize to others. While aggregating multiple tasks into a single unified corpus and conducting AHWA training on it might seem like a remedy, it often leads to degraded performance, as conflicting task objectives can interfere with the model's learned representations.

Another critical limitation is that existing AHWA methods are not designed for continual adaptation—a necessity for many real-world applications. In dynamic environments, transformers must adapt to new data and shifting user needs. However, current AHWA approaches achieve adaptation by reloading and retraining the full model weights on AIMC hardware—a process that is both time-consuming and energy-intensive, making frequent updates impractical [15].

Finally, real-world AIMC-based accelerators suffer from various device noise and circuit non-idealities. Industry-fabricated phase change memory (PCM) devices, for example, display state-dependent programming and read noise, as well as drift in conductance states over time [7–10]. While AHWA training has proven adept at addressing these hardware imperfections, a more advanced training methodology should preserve this strength while addressing the aforementioned limitations.

Hence, we introduce AHWA low-rank adaptation(AHWA-LoRA) training—a method that leverages the principles of LoRA [16] to address the challenges faced by AIMC systems [17]. As shown in figure 1, AHWA-LoRA training preserves the original, pre-trained weights of the transformer model, i.e. the meta-weights. These meta-weights are obtained through extensive pretraining and possess strong generalization capabilities. In conventional AHWA training, these meta-weights are overwritten during task-specific fine-tuning. We argue that these meta-weights should be preserved and instead introduce lightweight LoRA adapters to alter the effective weight values for different tasks. For task-specific retraining, this approach drastically reduces the memory overhead of AHWA training. During inference, the inclusion of LoRA modules equips AIMC with on-chip adaptation ability, allowing models to dynamically respond to hardware constraints and evolving task demands. This flexibility is difficult to achieve using conventional methods.

The complete AHWA-LoRA training pipeline consists of three main steps. First, pre-trained transformer weights are directly mapped onto AIMC hardware (meta weight deployment). Second, hardware constraints are applied to the meta-weights, but only the LoRA weights are updated (AHWA-LoRA training). Third, the trained LoRA weights are deployed onto Digital Processing Units (DPUs), allowing them to operate in parallel with analog computation on the meta-weights (LoRA weight deployment). A detailed description of these steps is provided in the Methods. We investigate the accuracy of our method using a realistic heterogeneous AIMC-based hardware configuration and statistical PCM device model calibrated on hardware measurements from a chip containing one million PCM devices.

Simulation results confirm that our method is effective on a 25.3 M-parameter transformer model, a practical size suitable for deployment on currently available AIMC chips [18, 19]. We demonstrate the scalability of our method to larger encoder-only transformer architectures of up to approximately 300 million parameters, including BERT-Base and BERT-Large, which may be supported by future AIMC chips. Results indicate our method constrains the size of the LoRA components at just 1% of the total model parameters. Furthermore, we extend our method to even larger decoder-only transformers, successfully applying it to the LLaMA 3.1 8B model for tasks in instruction tuning and mathematical reasoning, through supervised learning and reinforcement learning, respectively. Finally, we explore an optimized implementation that integrates AIMC computation with digital LoRA adaptation using RISC-V-based programmable multi-core accelerators (PMCAs) as DPUs. By strategically partitioning the

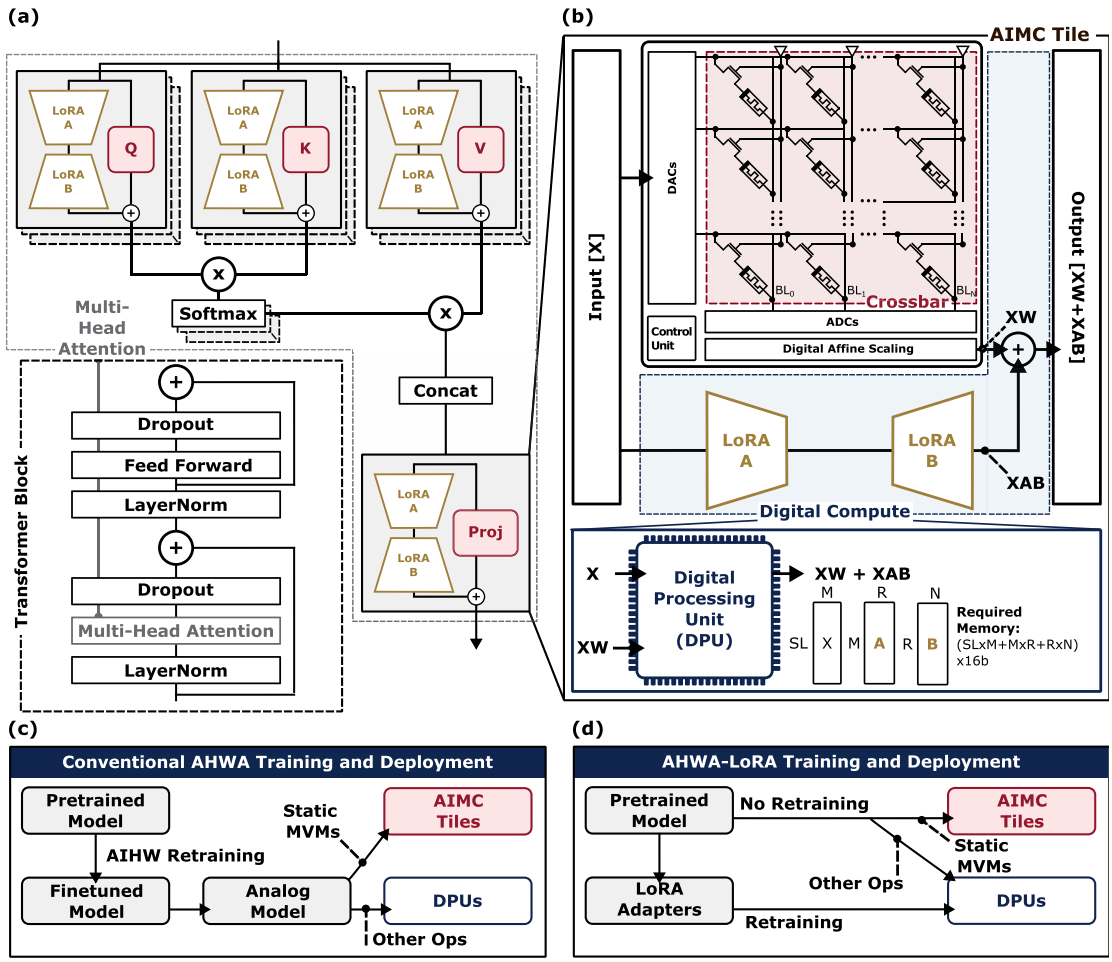

Figure 1. Implementation of AHWA-LoRA training to the multi-head attention block in the standard transformer architecture. (a) Within each transformer block, the weights of linear (dense) layers with fixed weights ($W \in \mathbb{R}^{m\times n}$) are mapped to AIMC tiles. The LoRA weight matrices $A \in \mathbb{R}^{m\times r}$ and $B \in \mathbb{R}^{r\times n}$ can be trained to adapt the effective weights for different downstream tasks without altering the meta-weights $W \in \mathbb{R}^{m\times n}$. (b) The high-level proposed architecture, where each AIMC tile is paired with a PMCA. The latency of AIMC tiles and PMCAs is balanced. Computation of *XAB* is performed within the PMCA, alongside the required addition operation. (c) Conventional AHWA training and deployment methods require retraining of the meta-weights of the pre-trained model. (d) AHWA-LoRA training preserves the meta-weights from the pre-trained model and deploys them directly to the AIMC hardware. Only the LoRA weight matrices are trained and executed by the DPUs.

workload, i.e. assigning the meta-weights to AIMC tiles and mapping LoRA layers onto digital PMCA units, our design offers a practical and efficient framework for serving transformer models onto these AIMC chips [20].

1. Methods

1.1. AHWA-LoRA training

The complete AHWA-LoRA training pipeline consisted of three main steps. During the first step, the meta weights of the model were directly deployed to AIMC hardware without any training. The hardware-specific parameters, such as weight clipping thresholds and the bit resolutions of AIMC peripherals (e.g. analog-to-digital converters (ADCs) and digital-to- analog converters (DACs)), were determined and used to simulate hardware constraints in the subsequent training process. In the second step, hardware constraints were simulated by incorporating them into the forward pass of the meta weights. Gradients propagated back through these simulated constraints, but learning happened in LoRA and only the LoRA weights were updated. This structure allowed the meta-weights to 'sense' the hardware limitations, while LoRA learned to compensate for them. Importantly, the model was trained using data from the downstream task, ensuring that it was optimized for the target application. This enabled joint optimization for both hardware compatibility and task performance via the introduced LoRA component. The loss function remained unchanged from standard AHWA training; no modification was needed to incentivize this behavior. Finally, the trained LoRA weights were deployed onto DPUs. These LoRA weights, though high in precision, were relatively small in number, making it feasible to parallelize

their computation with the analog computation performed on the fixed meta-weights residing in AIMC tiles.

In AHWA-LoRA training, the meta-weights, which are the pretrained base-model weights, are programmed onto AIMC hardware once and then kept fixed during adaptation. This choice improves long-term reusability: the same analog-programmed base model can be reused across multiple downstream tasks by swapping lightweight LoRA adapters, avoiding costly and frequent array reprogramming. Moreover, by not updating analog-mapped weights, AHWA-LoRA training avoids error accumulation commonly associated with repeated programming or on-chip learning. Temporal non-idealities after deployment (e.g. conductance drift) are mitigated via global drift compensation; when necessary, LoRA adapters can be refreshed off-chip and redeployed without reprogramming the AIMC arrays.

1.2. Model mapping to hardware and assumed hardware configuration

Our approach mapped all linear layers of MobileBERT (comprising 20.4 M parameters, ∼81% of the total parameters) onto AIMC tiles. This included the embedding transformation layer, the final output layer, and the linear layers within both feed-forward network (FFN) and the QKV projection of multi-head attention. The AIMC tiles had 512×512 unit cells, and 8-bit DACs and ADCs. A digital affine scaling operation was applied after the ADCs. Due to the dynamic nature of matrix-matrix operations, the computation of attention scores was handled by PMCAs, as non-volatile memory-based AIMC computation was not well-suited for such operations. Additionally, PMCAs managed the LoRA adapters and integrated their outputs with those from the AIMC tiles. To accurately model the behavior and constraints of AIMC hardware, we used AIHWKIT, an open-source simulator for AIMC devices [21]. AIHWKIT offered detailed models of PCM devices based on extensive experimental data and accurately modeled the peripheral circuitry. A differential channel-wise weight mapping scheme was applied to all analog weights with maximum conductance $G_{\max} = 25\,\mu$S, and each channel was clipped to 3-sigma based on the fitted weight distribution.

1.3. Training and inference details

During forward propagations, we injected hardware constraints on the model, including Gaussian noise [18] with an amplitude of 6.7% on analog weights and 4.0% on ADCs, among other hardware constraints. The 6.7% weight-noise level is not an immutable physical constant of PCM devices; rather, it is an *effective* noise amplitude used in our simulator to approximate the dominant stochastic behavior of the AIHWKIT PCM analog tile model [21], which is calibrated to measured device statistics and includes multiple non-idealities and parameters. For training efficiency, we summarize these effects using a zero-mean Gaussian perturbation model as a first-order approximation. Importantly, this amplitude is a tunable hyperparameter in our framework; we provide an ablation over noise amplitudes in the supplementary information (table II), and select 6.7% as a strong accuracy–cost trade-off under our training budget.

Noise is 'injected' on-the-fly during each forward pass by forming a temporary noisy instance of the fixed meta-weights. Concretely, we maintain clean master meta-weights W throughout training, and for each minibatch we sample an i.i.d. perturbation and compute $\tilde{W} = W + \Delta W$, where ΔW follows a zero-mean Gaussian distribution with relative amplitude set by the chosen noise level (e.g. 6.7%). The noisy weights $\tilde{W}$ are used only for the current forward/backward computation and are discarded afterwards, ensuring that noise is uncorrelated across minibatches and that the expected noisy weights remain unbiased around W. In AHWA-LoRA training, gradients flow through the noisy forward path, but only the LoRA parameters are updated, while the meta-weights remain fixed. The Adam optimizer was used with an initial learning rate of 2×10^{-4}, following a linear decay schedule over 15 epochs. The maximum sequence length (SL) was set to 320. For inference, we evaluated accuracy and robustness under realistic hardware constraints, including programming noise, conductance drift, and read noise for a period ranging from 0 s to 10 years. All results were averaged over 10 trials. We used global drift compensation [22] to mitigate temporal variations. We evaluated the impact of different rank values r on performance and computational overhead, determining that a rank of 8 provided an optimal balance (see figure 2(a)).

1.4. Instruction tuning and reinforcement learning

For instruction tuning, we utilized a carefully curated version of the Alpaca dataset [23], which enhanced data quality and consistency. Our experiments were conducted using the LLaMA-3.1 8B model, incorporating a LoRA rank of 16. This corresponded to approximately 0.52% of the model's total parameters. The model mapping strategy was consistent with the method used in MobileBERT. All linear layers were mapped to AIMC tiles and attention was managed by PMCAs. During forward propagation,

hardware constraints were emulated by injecting Gaussian noise with an amplitude of 6.7% by adding it to the meta weights, alongside additional hardware-induced limitations. Explicit modeling of ADC and DAC components was omitted, assuming these components operated at high resolution. Unlike the approach taken in MobileBERT experiments, we did not apply weight clipping, following recent insights from weight quantization research in LLMs [24, 25]. Training was executed with the AdamW optimizer, using an initial learning rate of 2×10^{-4}, a linear decay schedule over 6470 steps, and a brief warm-up period of 5 steps. The batch size and weight decay were set to 8 and 0.01, respectively. The maximum SL was limited to 2,048 tokens.

In the reinforcement learning setup, we leveraged the GSM8K dataset to foster the model's reasoning abilities, particularly encouraging the effective use of long chain-of-thought (CoT). We employed the group relative policy optimization (GRPO) algorithm [26] for training, with the instruction-tuned Llama-3.1 8B model again featuring a LoRA rank of 16. Consistent with our earlier instruction-tuning approach, we applied the same model mapping strategy and refrained from applying weight clipping. However, the amplitude of Gaussian noise injected during RL training was reduced to 3.0%. We emphasize that this reduction does not imply that the underlying PCM noise is physically reduced; rather, it is a practical training choice for stable and compute-efficient reinforcement learning. In GRPO, the update relies on *relative* quality differences among multiple sampled completions for the same prompt to form effective advantage estimates. With 6.7% noise, the initial policy frequently produces uniformly low-quality (near-random) generations, yielding near-zero or non-informative reward differences within a sampling group and substantially weakening the learning signal. Using 3.0% preserves a meaningful gradient signal early in training while still operating in a noisy analog regime. A straightforward way to enable higher-noise RL is a curriculum strategy (e.g. supervised adaptation at the target noise level before any reinforcement learning), which we leave for future work. Modeling of ADC and DAC components was reserved for future investigation. The maximum SL during RL training was 1024 tokens. Our RL approach incorporated four complementary reward functions to incentivize correct reasoning patterns and precise mathematical answers, allowing a maximum achievable reward of 9.5. For each GSM8K question, we generated 16 sample outputs, which were grouped and utilized for advantage calculations by GRPO [26]. RL training consisted of 500 optimization steps, including a 50-step warm-up phase, with a learning rate set at 5×10^{-6} and a comparatively higher weight decay of 0.1.

To efficiently handle memory overhead during training, we adopted the aihwkit-lightning framework [27]. Its low memory footprint, combined with LoRA, allowed us to perform all training and evaluations on a single GPU with 80 GB of memory. During inference, we evaluated model robustness against Gaussian noise applied to weights, following methodologies established in prior studies [22, 28]. The Gaussian noise levels during evaluation were set to match those used during training: 6.7% for instruction tuning and 3.0% for reinforcement learning. Additional evaluation results for a wider range of noise levels, as well as results using the PCM model with zero-second drift, were provided in the Supplementary Information.

1.5. PMCA performance estimation simulations

We adopted a small version of the Snitch cluster [29], a RISC-V-based PMCA optimized for energy-efficient floating-point (FP) computation. It consisted of nine in-order RV32IMAF Snitch cores, each featuring a 32-bit SIMD-capable, mixed-precision FP unit (FPU). Two key ISA extensions, FP Repetition [29] and Stream Semantic Registers [30] enhanced execution efficiency by automating FP loops and reducing memory overheads, enabling pseudo-dual-issue performance and achieving up to ∼90% FPU utilization on dense workloads. The cluster architecture included a shared L1 instruction cache and a 128KiB interleaved tightly coupled data memory (TCDM) connected via a single-cycle interconnect. One of the cores managed the direct memory access unit [31], while the remaining eight executed parallel computations. We integrated the RedMulE [32] accelerator to the cluster, an open-source, parametrizable accelerator optimized for reduced-precision matrix-vector multiplications (MVMs) using FP16 and FP8. We configured RedMulE to have 32 fused-multiply-accumulate blocks to minimize the final area of the cluster. We performed cycle-accurate register-transfer level simulations to obtain the performance metrics on the PMCA.

2. Results

2.1. Accuracy validation

We first validate the accuracy of the proposed method and compare it to conventional AHWA training [22]. Table 1 presents the performance validation of AHWA-LoRA training applied to MobileBERT

Table 1. Comparison of AHWA training and AHWA-LoRA training. The dataset used is SQuAD v1.1 and the model is MobileBERT.

			Score after conductance drift						
Training method	Metric	Baseline	0 s	1 h	1 d	1 w	1 m	1 y	10 y
AHWA training	*F*1	90.01	89.47	89.07	88.66	88.19	87.73	86.59	85.14
	EM	82.60	82.42	81.75	81.28	80.60	79.94	78.41	76.40
AHWA-LoRA training	*F*1	89.17	89.06	88.71	88.36	87.97	87.49	86.51	85.36
	EM	82.06	81.93	81.41	80.94	80.39	79.77	78.41	76.92

on the SQuAD v1.1 dataset under different drift durations. MobileBERT was chosen due to its relatively small size, making it practical for deployment on current AIMC chips [18, 19]. We will validate the effectiveness of our method on larger models in later sections. Evaluation metrics used in the experiments include the *F*1 score and exact match (EM). The baseline results in the table represent the performance of the digital model without hardware constraints during training or inference.

The results demonstrate that AHWA-LoRA training achieves performance comparable to conventional AHWA training, with *F*1 and EM scores within 1% of those obtained through full AHWA training. Notably, at a drift time of 10 years, our approach outperforms the previous AHWA training method, achieving an *F*1 score of 85.36 versus 85.14, and an EM score of 76.92 versus 76.40. We attribute this improvement to the difference in update mechanisms: standard AHWA training updates all model parameters, which may cause the model to deviate significantly from the local minima reached during pretraining. In contrast, AHWA-LoRA training updates only the low-rank components, helping the model stay closer to the flatter local minimum found during pretraining, which improves robustness under significant drift.

It is also worth noting that AHWA-LoRA training updates only the LoRA weights, which make up approximately 6.6% of the model parameters. Despite this small proportion of trainable parameters, the model demonstrates strong robustness to hardware constraints, as indicated by *F*1 and EM scores that are on par with conventional AHWA training. This suggests that adapting a model to hardware constraints may not require the adaption of all parameters. Instead, hardware adaptation could be inherently a low-rank problem, where LoRA-style updates are sufficient. Since hardware adaptation is a critical challenge in AIMC, these findings offer new insights and indicate that full model retraining may not always be necessary.

2.2. Training performance evaluation

Table 2 benchmarks the number of trainable parameters and GPU memory usage during training. The results show that AHWA-LoRA training reduces the number of trainable parameters to approximately one million, which is more than a 15× reduction compared to conventional AHWA training. Additionally, GPU memory usage is reduced by 13%, saving over 4 GB of VRAM, making training more accessible. Although MobileBERT only has 24.67 M trainable parameters, the memory usage during AHWA training is significantly higher than during standard digital training due to the high overhead of modeling hardware constraints in both the forward and backward passes. This suggests that memory overhead is a key bottleneck in scaling AIMC to larger transformer models. LoRA offers a promising solution to reduce both trainable parameters and memory requirements. Moreover, optimizing the training framework can further improve training efficiency. For instance, recent work using Triton to reduce memory usage and accelerate AHWA training presents another promising direction [33].

We further break down the effects of LoRA placement and rank. Applying LoRA only to the FFN achieves lower parameter counts and memory usage than full AHWA-LoRA training, while restricting adaptation to the QKV projections reduces parameters even further (0.22 M) with the lowest memory footprint (28.02 GB). For varying LoRA ranks ($r = 1, 2, 4, 8, 16$), the number of parameters scales nearly linearly with r, whereas GPU memory usage remains largely unchanged.

2.3. Multi-task inference evaluation

We also evaluate our method on multi-task inference. Conventional AHWA methods require at least N AIMC chips to handle N tasks. In contrast, our proposed AHWA-LoRA training method employs an elegant strategy wherein a single analog model is mapped to one AIMC chip with multiple sets of LoRA weights, each adapted to a specific task.

We report the results on the 8 tasks from the GLUE benchmark in table 3. These tasks were achieved using a single analog model mapped to AIMC tiles, with 8 sets of LoRA parameters computed on DPUs, each containing 1.6 M parameters, one for each task. To support all 8 tasks, a total of 8×1.6M LoRA

Table 2. Comparison on trainable parameters and GPU memory usage across different training methods. The model used is MobileBERT and the dataset is SQuAD v1.1. Experiments are conducted on an NVIDIA H100 80GB GPU with a batch size of 32.

Method	Trainable parameters (M)	GPU Memory usage (GB)
AHWA	24.67	37.72
AHWA-LoRA	1.63	32.92
AHWA-LoRA (FFN)	1.40	31.88
AHWA-LoRA (QKV)	0.22	28.02
AHWA-LoRA ($r = 1$)	0.20	32.90
AHWA-LoRA ($r = 2$)	0.41	32.90
AHWA-LoRA ($r = 4$)	0.82	32.91
AHWA-LoRA ($r = 8$)	1.63	32.92
AHWA-LoRA ($r = 16$)	3.27	32.94

Table 3. AHWA-LoRA training on the GLUE benchmark for MobileBERT.

		Score after conductance drift						
GLUE Task	Score	0 s	1 h	1 d	1w	1 m	1 y	10 y
SST-2	92.8	91.2	90.9	90.8	90.6	90.6	90.4	90.0
MNLI-m/mm	83.3/82.6	82.8/83.3	82.6/83.0	82.4/82.7	82.1/82.6	82.0/82.4	81.7/82.0	81.2/82.6
MRPC	88.8	87.6	87.5	87.5	87.2	87.3	87.1	86.4
QNLI	90.6	90.9	90.9	90.8	90.6	90.3	90.0	89.5
QQP	70.2	86.9	87.0	86.9	86.7	86.6	86.4	86.2
RTE	66.2	52.9	53.4	54.3	55.4	54.2	53.3	53.4
STS-B	84.4	87.5	87.3	87.0	86.9	86.7	86.2	85.6
CoLA	50.5	46.0	44.6	42.7	40.6	39.8	39.1	37.3
GLUE	78.8	78.8	78.6	78.3	78.1	77.8	77.4	76.9

parameters are required, in addition to the 20.4 M mappable parameters on AIMC tiles and 4.9 M unmappable parameters on DPUs, resulting in a total of 38.1 M parameters. In contrast, a conventional AHWA training approach [22] would require developing and programming 8 separate models into the hardware, requiring at least $(8 \times 20.4 + 4.9)$ M parameters. Our method, therefore, achieves a more than 4-fold reduction in the number of parameters. Furthermore, our approach demonstrates robustness to the inherent constraints of AIMC hardware, with most tasks showing minimal performance degradation over time due to drift.

AHWA-LoRA training also offers significant advantages, including on-chip task switching and on-chip adaptation to user data. For example, when AIMC hardware is initially configured for the SST-2 task, it can be easily switched to the MNLI task by simply updating the 1.6 M LoRA weights from SST-2 to MNLI, without needing to reload AIMC weights. Additionally, the LoRA weights allow for on-chip adaptation; when new user-generated data becomes available, the LoRA weights can be updated accordingly, while the pre-trained model on the AIMC tiles remains unchanged.

2.4. Optimal allocation of LORA parameters

Optimizing the allocation of LoRA parameters is crucial for balancing model performance and resource efficiency. This challenge represents a typical resource allocation problem [34]. While several methods exist for optimizing LoRA allocation [35, 36], our study evaluates two predefined strategies for the sake of simplicity and consistency, avoiding the use of adaptive LoRA allocation techniques.

As shown in figure 2(a), increasing the LoRA rank generally improves the $F1$ score across all drift durations, though the gains eventually plateau. For instance, raising the LoRA rank from 1 to 2 results in an $F1$ score increase of approximately 2.2 for a 10 year drift. However, increasing the rank from 8 to 16 yields only a marginal improvement of 1.0. Since a higher rank introduces greater computational overhead, this diminishing return suggests that selecting an appropriate rank can effectively balance accuracy gains with efficiency. Specifically, we find that a rank of 8 represents an optimal sweet spot for practical AIMC constraints: it provides sufficient capacity to compensate for analog noise and drift (accuracy constraint) while remaining lightweight enough to fit within the memory and latency budget of the digital accelerators (hardware constraint).

Figure 2(b) investigates the impact of different LoRA allocation strategies on model accuracy over varying drift times. The results indicate that applying LoRA to all linear layers reaches an initial accuracy

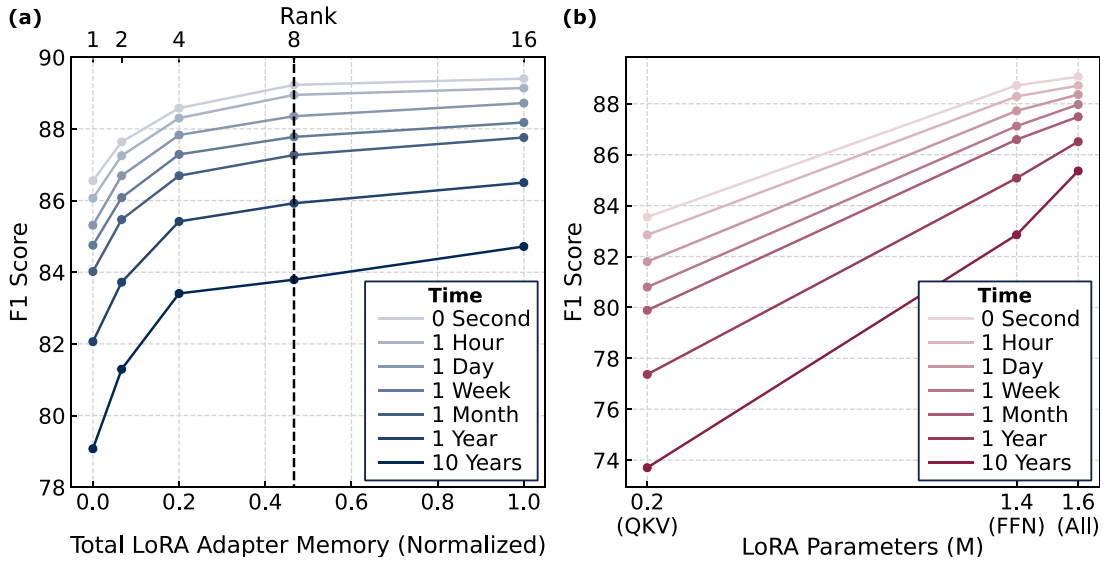

Figure 2. Resource optimization study for MobileBERT and SQuADv1.1 at different drift times. (a) Pareto front of the *F*1 score and total LoRA adapter memory for different rank values. A rank of 8 – highlighted using a vertical dashed line—provides a good tradeoff between these quantities. (b) The total number of LoRA parameters when adapters are applied to different layers of the multi-head attention block.

of 89.06 (at 0 s), then shows a gradual performance decrease over time, reaching 85.36 after 10 years. This strategy has the highest LoRA parameter count (1.6 M). In contrast, restricting LoRA to only the QKV linear layers or FFN layers reduces the parameter count to 0.2 M and 1.4 M, respectively, but results in consistently lower *F*1 scores across all drift times. These findings suggest that applying LoRA to all linear layers is essential for maintaining competitive *F*1 scores.

2.5. Dynamic adaptation

In conventional approaches, once a model is deployed, its weights remain fixed, limiting adaptability to evolving hardware conditions. Our proposed AHWA-LoRA training enables dynamic adaptation, which is crucial for real-world applications. For instance, as shown in figure 3(a), reducing the ADC/DAC precision from 8-bit to 6-bit results in a significant 25% *F*1 score drop after a 10 year drift period. Our approach addresses this issue by updating only the LoRA weights. This on-chip adaptation effectively mitigates performance degradation, improving the *F*1 score from 60.81 to 74.23 after ten years. Additionally, our method can adapt to other environmental variations, such as noise pattern shifts due to temperature fluctuations or changes in ADC reference current, by leveraging LoRA modules for targeted adjustments.

2.6. Scalability studies

As illustrated in figure 3(b), our method scales effectively to both BERT-Base (108 M parameters, with 1.3 M LoRA weights) and BERT-Large (334 M parameters, with 3.5 M LoRA weights), optimizing these models for AIMC hardware by training only the LoRA components. The results demonstrate that larger models not only achieve higher *F*1 scores but also exhibit increased robustness against hardware-induced degradation. For instance, a 10 year hardware drift leads to a performance drop of nearly 4 points in general; however, BERT-Base experiences only a 0.63-point reduction, while BERT-Large sees just a 0.48-point drop. This indicates that larger models are more resilient to AIMC hardware limitations, highlighting AIMC's potential to support transformer-based models at scale.

Furthermore, BERT-Large is approximately 12× larger than MobileBERT, yet the LoRA parameter count increases by only about 2×, and a LoRA rank of 8 remains sufficient. This highlights that our proposed method scales efficiently with minimal overhead. Encouraged by these results on encoder-based transformers, we next explore the application of our approach to decoder-only transformers, such as large language model (LLMs).

2.7. Instruction tuning and reinforcement learning on LLMs

While previous sections have focused on encoder-only transformers, primarily due to their practical size for current LoRA hardware, we additionally demonstrate the broader applicability of our AHWA-LoRA training approach to decoder-only LLMs. Specifically, we investigate the LLaMA-3.1 model, containing

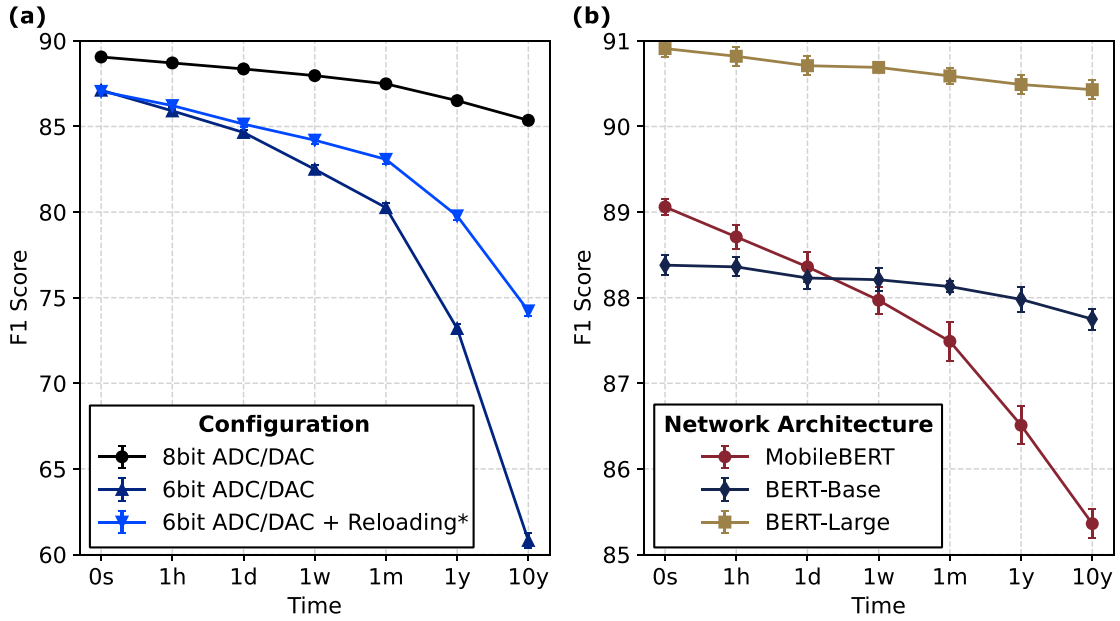

Figure 3. Dynamic adaptation and scalability studies at different drift times. (a) Different configurations for AHWA-LoRA training evaluated on MobileBERT and SQuAD v1.1. *LoRA weight reloading. (b) Scalability analysis using BERT-Base and BERT-Large.

Table 4. Zero-shot accuracy (%) of LlaMA 3.1 8B across nine benchmarks under three settings: (1) baseline digital model, (2) analog model before AHWA-LoRA training, and (3) analog model after AHWA-LoRA training. The analog model shows substantial accuracy degradation compared to the digital baseline. Applying AHWA-LoRA training substantially narrows this gap, yielding consistent improvements across all tasks. Additional results in the supplementary information show that evaluation at lower noise levels can further boost accuracy, bringing it closer to the digital baseline.

Model variant	HellaSwag	BoolQ	PIQA	WinoGrande	ARC-c	ARC-e	SciQ	COPA	OpenBookQA
Baseline: LlaMA 3.1 8B (Digital)	78.91	82.11	81.23	73.88	53.41	81.10	94.60	87.00	44.80
Analog LlaMA 3.1 8B (Pre-AHWA-LoRA)	29.48	48.35	50.44	51.78	25.34	29.42	42.10	55.00	26.20
Analog LlaMA 3.1 8B (Post-AHWA-LoRA)	67.71	69.36	69.64	55.80	35.67	54.71	80.10	71.00	33.60

8 billion parameters, with a LoRA rank of 16, comprising approximately 0.52% of the model's total parameters. Detailed experimental configurations are provided in the Methods.

We first evaluate our approach in the context of instruction tuning using the Alpaca dataset, as shown in table 4. The baseline digital Llama 3.1 8B achieves strong zero-shot accuracy across diverse benchmarks, whereas the analog counterpart prior to AHWA-LoRA training suffers severe degradation, with performance drops exceeding 40% on several tasks. This substantial loss is primarily attributed to analog hardware noise, which affects reasoning and knowledge-retrieval capabilities in decoder-only LLMs. Notably, applying our AHWA-LoRA training consistently restores a large portion of the lost performance, yielding absolute improvements of up to 38.23 percentage points (e.g. on HellaSwag) compared to the pre-tuning analog model. These results highlight that our method enables the model to recover general-purpose instruction-following abilities, demonstrating robust cross-task generalization without requiring full-precision retraining.

We extend our evaluation to reinforcement learning using the GSM8K dataset, aiming to enhance the model's reasoning capability through a long CoT, as shown in table 5. The digital baseline benefits from LoRA fine-tuning, improving accuracy from 67.63% to 85.06%. In the analog setting, the unadapted model starts at a much lower accuracy of 37.98%, reflecting its difficulty in conducting reasoning under hardware noise. After applying our AHWA-LoRA training, the analog model reaches 70.74% accuracy, representing a remarkable 32.76 percentage point gain and narrowing the gap to the digital post-LoRA model by over half. These results show the versatility of AHWA-LoRA training, extending its benefits beyond supervised instruction tuning to reinforcement learning, and demonstrating its ability to optimize models for complex, reasoning-intensive tasks.

Table 5. GSM8K evaluation accuracy (%) with chain-of-thought (CoT) reasoning. The LLM is trained using reinforcement learning to produce outputs in the following format: <start_working_out> reasoning steps <end_working_out> <SOLUTION> final answer </SOLUTION>. We compare the digital baseline (*LLaMA 3.1 8B*) and its analog counterpart, both before and after reinforcement learning via LoRA training. In the digital case, LoRA training improves performance from 67.63% to 85.06%. For the analog model, AHWA-LoRA training boosts accuracy from 37.98% to 70.74%, reducing the analog–digital performance gap from nearly 30% to about 15%. Additional results in the Supplementary Information show that, under reduced evaluation noise, this gap can shrink to 2.5%.

	Digital (LlaMA 3.1 8B)		Analog (LlaMA 3.1 8B)	
Benchmark	Pre-LoRA	Post-LoRA	Pre-AHWA-LoRA	Post-AHWA-LoRA
GSM8K	67.63	85.06	37.98	70.74

2.8. Latency analysis of hardware components and network layers

We analyze the performance optimization of the proposed hybrid implementation, which utilizes AIMC tiles and PMCAs, with a focus on balancing their latencies. If the latency of the PMCAs is well balanced with that of the AIMC tiles, an efficient AIMC-PMCA pipeline parallelism can be implemented, leading to minimal performance loss due to the additional computations introduced by the LoRA modules. Then, we can enjoy the high accuracy on long drift time, efficient multi-task processing ability, and dynamic adaptation ability with minimal latency cost. We examine this balance by analyzing our architecture across different layer sizes of MobileBERT, with the rank of LoRA matrices fixed at 8. We consider AIMC tile integration times of 128, 256, and 512 ns, based on values reported in the literature for AIMC-based accelerators [18]. Since processing one token at a time using a PMCA does not fully exploit the parallelization capabilities of the cluster, we evaluate scenarios where multiple tokens are processed in parallel. Specifically, once the required static XW (i.e. activation×weight) MVMs on AIMC tiles are executed for t tokens and transferred to the PMCA, the PMCA then performs the required XAB MVM LoRA computations and element-wise addition operations. We consider t values of 8, 16, 32, 64, and 128.

The first step in our evaluation consists of analyzing and comparing the performance of AIMC tiles and PMCAs when generating different numbers of parallel tokens, t. Figure 4(a) shows the latency of AIMC tiles and PMCAs for different layer sizes, AIMC tile integration times, and t values. The results show that for a larger weight matrix size (512×128) and longer integration times (256 ns and 512 ns), the latencies of PMCA and AIMC are well-balanced. For the same matrix size with a shorter integration time of 128 ns, PMCA latency becomes the bottleneck. In the case of smaller weight matrix size (128×128) and longer integration times (256 ns and 512 ns), AIMC latency dominates, whereas for shorter integration time (128 ns) PMCA and AIMC latencies are balanced. For a smaller weight matrix size (128×128), an optimal balance between PMCA and AIMC computation latencies is achieved when processing 128, 8, and 8 parallel numbers of tokens t for integration times of 128, 256, and 512 ns, respectively. This corresponds to PMCA-to-AIMC latency ratios of approximately 1.04, 0.63, and 0.32, respectively. For the larger matrix size (512×128), the optimal balance is achieved for 128, 128, and 8 parallel tokens, corresponding to PMCA-to-AIMC ratios of approximately 2.57, 1.29, and 0.7 for the above respective integration times.

Increasing the number of parallel tokens can enhance computational balance, but it also affects memory requirements. Figure 4(b) shows the memory requirements to store the inputs and low-rank weight updates in the PMCAs as a function of the number of parallel tokens t. We observed that the required PMCA memory varies significantly depending on the number of parallel tokens. For smaller weight matrices, the memory requirement ranges from 8.2KiB to 21KiB, while for larger matrices, it increases to between 70KiB to 172KiB. Considering the PMCAs in our architecture have a TCDM size of 128KiB, performing MVMs with large weight matrices and many parallel tokens requires either a larger TCDM or additional TCDM-SRAM communication.

To further assess the impact of the proposed methodology, we evaluate the total MVM latency for processing a SL of 320 tokens. Figure 4(c) shows the total latency for different MobileBERT layers across different AIMC tile integration times. It compares the performance of the proposed AHWA LoRA approach against a baseline AIMC system without LoRA integration. We implement the AIMC-PMCA pipeline while accounting for data transfer latency between AIMC and PMCA. As shown in figure 4(c), the layer size, AIMC integration time, and the parallel number of tokens, all impact the performance overhead introduced by the additional LoRA matrices. However, regardless of the integration time, when AIMC and PMCA latencies are well balanced, the additional latency from the LoRA matrices remains minimal. For large weight matrices (512×128), the latency overhead from adopting AHWA LoRA remains limited to 2.72×, 1.39×, and 1.05× for integration times of 128, 256, and 512 ns, respectively.

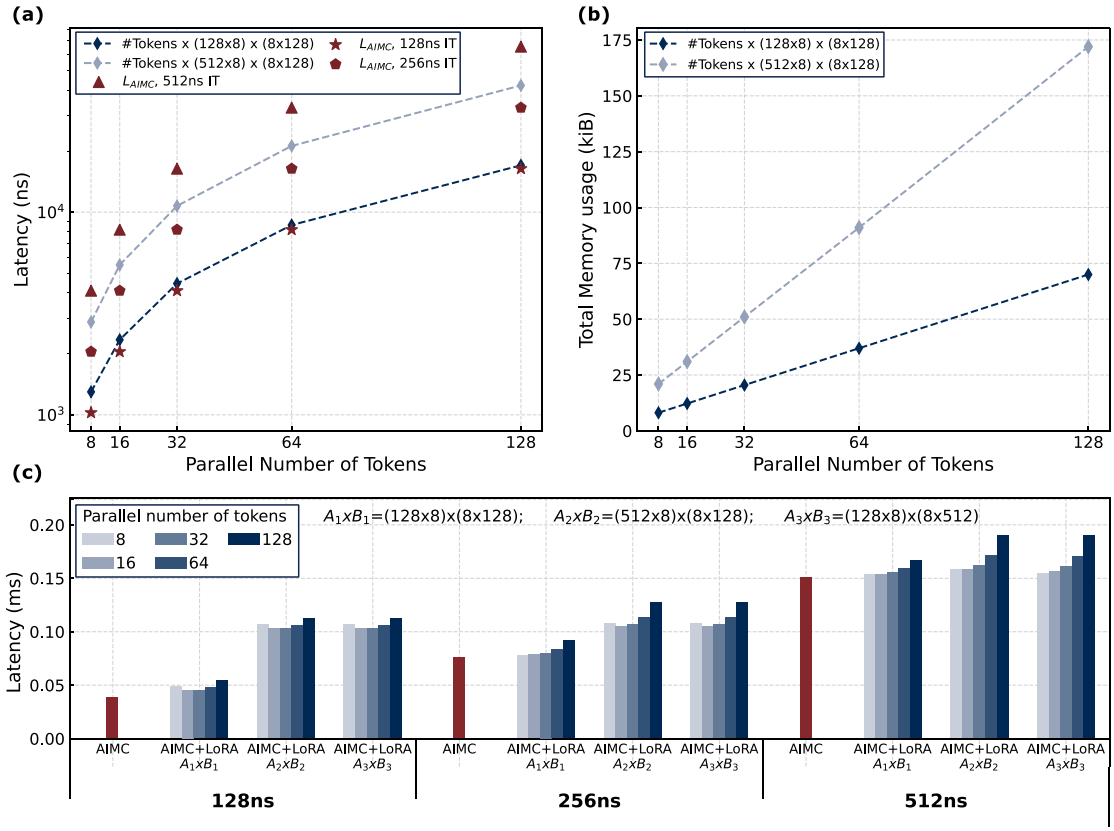

Figure 4. Performance analysis and latency balancing of the proposed architecture with AIMC tiles and coupled PMCAs. (a) The latency of two different MobileBERT layers with varying AIMC tile integration times and (b) PMCA's TCDM requirement as a function of the number of parallel tokens. (c) The total latency for all the different layers of MobileBERT with an optimized AIMC-PMCA pipeline balancing the latency of the AIMC tiles and PMCAs. The latency without any LoRA adapters (AIMC) is also reported.

Notably, in the best scenario, our method incurs only a 4% latency overhead compared to the AIMC baseline. This evaluation highlights that balancing AIMC and PMCA latencies minimizes the overall layer latency and keeps AHWA LoRA's performance cost low.

This evaluation highlights the importance of balancing the AIMC and PMCA latencies. When the AIMC latency to perform the static MVMs aligns closely with the PMCA latency, the overall latency for each layer is minimized and the performance cost associated with the additional LoRA matrices is kept minimal.

3. Discussion

When deploying NNs on AIMC hardware, accuracy is often degraded due to hardware constraints. To mitigate this, researchers have introduced AHWA training, which adapts models to specific hardware characteristics. However, the trained models become tightly bound to the specific hardware settings and the dataset used during training, leading to over-specification. This limits both their flexibility and generalization ability.

One of the core motivations for using transformer models is their strong generalization capacity—an advantage that is in direct conflict with the rigid nature of conventional AIMC deployment. Our proposed AHWA-LoRA training method addresses this by retaining the meta-weights and introducing minimal overhead through LoRA. This design significantly improves generalization performance while maintaining accuracy comparable to conventional AHWA training methods. Furthermore, it enables additional capabilities for AIMC, such as multi-task inference and continual adaptability using a shared set of analog weights.

Importantly, our results provide concrete evidence that AHWA-LoRA training scales efficiently across model sizes and architectures. For encoder-only transformers, we evaluate MobileBERT (25 M), BERT-Base (108 M), and BERT-Large (334 M), and observe that larger models exhibit smaller performance degradation under long-horizon drift (e.g. under 10-year simulated drift, MobileBERT drops by

∼4 points, while BERT-Base and BERT-Large degrade substantially less; see table 1). For decoder-only LLMs, we apply AHWA-LoRA training to LLaMA 3.1 8B for instruction tuning and reinforcement learning, where LoRA accounts for only 0.52% of parameters, demonstrating that the adaptation overhead remains low even at billion-parameter scale. A plausible intuitive explanation is that larger models may possess stronger and more redundant representations, making them inherently more robust to perturbations and reducing the burden on the LoRA adapters to compensate hardware noise. Together, these experiments highlight that the same AHWA-LoRA training mechanism transfers cleanly across architectures (encoder/decoder), diverse task families (SQuAD/GLUE), model sizes (million-scale model/billion-scale model), and training paradigms (finetuning/RL).

The primary cost of our method lies in serving the LoRA components, whose overhead is proportional to their rank. Interestingly, we find that a rank of 8 is sufficient for most scenarios. This rank supports both learning a specific task and adapting to hardware variations—indicating that the noise inherent in analog hardware with limited dynamic range can be effectively and efficiently compensated for. These positive results are expected to encourage broader adoption of analog hardware, allowing more users to benefit from its high speed and ultra-low power consumption.

Additionally, we found that the proposed architecture is well-suited for hardware implementations that parallelize AIMC tiles and PMCA for efficient execution. Our evaluation demonstrates that when the AIMC latency is well balanced with the PMCA latency, the latency overhead introduced by LoRA is minimal. In summary, our approach offers substantial improvements in the versatility and usability of AIMC, with only a modest latency cost.

For future improvements, insights from LoRA and NN training can be adapted to the AIMC context with suitable modifications [37, 38]. While this study builds on existing LoRA methods, it also contributes back to that domain by demonstrating that LoRA can be used to compensate for noisy weights-a scenario that was not considered in early LoRA studies, which focused on full-precision [17] or low-precision weights [24], but not statistical (noisy) weights.

ADC plays a crucial role in bridging the analog and digital domains, but its limited precision can impact overall system performance. Enhancing ADC optimization to develop models resilient to lower bit-widths can accelerate inference or improve accuracy at fixed bit settings. Advances in transformer activation quantization are particularly relevant here [39], as they help mitigate the adverse effects of quantization noise. Notably, recent work by Büchel *et al* [28] demonstrates that knowledge distillation can be used to overcome the precision limitations of ADC, highlighting a promising direction for future research. Our proposed method is inherently compatible with such quantization and distillation techniques, making it well-suited for integration with these advances. Future work will incorporate these methods to enhance both performance and robustness in our AIMC system design.

Our initial results on instruction tuning and reinforcement learning provide promising insights into the broader applicability of the proposed methods and their scalability to LLMs. We emphasize that the low-memory footprint of the proposed AHWA-LoRA training is even more critical for LLMs than for MobileBERT, as it enables training LLMs with AIMC constraints on a single GPU, making the process much more accessible to a wider range of researchers. Other benefits of AHWA-LoRA training also extend naturally to LLMs, including multi-task support through LoRA reloading or multi-LoRA serving, as well as dynamic adaptation to user-specific data. A key challenge in this direction is to further improve the robustness of LLM training under hardware constraints. Our qualitative results in the Supplementary Information show that models can still generate coherent and meaningful outputs despite these hardware constraints. On the other hand, our quantitative results indicate that evaluation accuracy begins to degrade at a noise level of 4.0% for instruction tuning and 2.0% for reinforcement learning. A similar inflection point of 2–4% was also reported by Buchel *et al* [28]. This indicates that there remains significant room for optimization in model training. Advancements in AIMC hardware would also contribute to this goal by enabling operation at lower noise levels.

Our results on GSM8K are closely related to the idea of noisy networks in reinforcement learning, which demonstrates that introducing controlled noise into NNs can significantly enhance exploration capabilities and improve performance across both on-policy and off-policy methods. Similarly, the injection of noise in our method may encourage exploration. From this perspective, the inherent noise within AIMC is not a limitation to be overcome, but rather a beneficial feature that drives more effective learning. How to maximize this potential benefit can be an interesting future work.

The proposed method in this paper may also be connected to the idea of mortal computations [40, 41], which involves using analog hardware to perform NN operations. In such systems, the algorithm is inherently tied to the analog substrate, meaning the computation is mortal—it dies with the analog hardware. While this tight integration enables high-speed and energy-efficient processing, it also presents challenges in training effectiveness, knowledge sharing, and scalability, as

identified in a previous study [41]. Our method introduces minimal digital parameters (i.e. LoRA weights) to augment the analog weights components. This approach addresses learning limitations without requiring precise knowledge of the analog weights' exact values; instead, it only relies on their statistical noise profiles, making the mortal system learnable. The LoRA modules effectively compensate for the noise in the mortal system, enabling robust functionality. As a result, the architecture achieves an elegant balance: analog weights remain fast and large in number, while the digital weights, though less-efficient and slower, are kept minimal. When adaptation is needed, only the minimal digital weights are updated. This update process is simpler and more efficient than modifying all analog weights.

Data availability statement

The data that support the findings of this study are openly available at the following URL/DOI: https://github.com/chenlicodebank/lora_on_analog_hardware [42]. Data will be available from 30 September 2025.

Supplementary Material available at https://doi.org/10.1088/2634-4386/ae405e/data1.

Acknowledgments

This work is supported in part by the NeuroSoC Project funded under Horizon Europe Grant Agreement 101070634. The work of Bipin Rajendran was supported in part by EPSRC Open Fellowship under Grant EP/X011356/1 and in part by EPSRC under Grant EP/X011852/1. We also gratefully acknowledge valuable discussions with Dr Abu Sebastian and help from Julian Büchel regarding the usage of AIHWKIT.

Author contributions

C L proposed the initial concept of AHWA LoRA training and performed the accuracy validation experiments. E F conducted the latency analysis. C L and E F co-authored the initial draft of the manuscript, with all authors contributing substantial feedback and revisions. B R, M G, and I B provided overall project supervision and guidance.

ORCID iDs

Chen Li 0000-0002-9806-7128
Elena Ferro 0000-0002-8618-8643
Manuel Le Gallo 0000-0003-1600-6151

References

[1] LeCun Y, Bengio Y and Hinton G 2015 Deep learning *Nature* **521** 436–44
[2] Vaswani A, Shazeer N, Parmar N, Uszkoreit J, Jones L, Gomez A N, Kaiser L u and Polosukhin I 2017 Attention is all you need *Advances in Neural Information Processing Systems* vol 30, ed I Guyon, U V Luxburg, S Bengio, H Wallach, R Fergus, S Vishwanathan and R Garnett (Curran Associates, Inc.)
[3] Patterson D, Gonzalez J, Le Q, Liang C, Munguia L-M, Rothchild D, So D, Texier M and Dean J 2021 Carbon emissions and large neural network training (arXiv:2104.10350)
[4] Gallo M L, Sebastian A, Mathis R, Manica M, Tuma T, Bekas C, Curioni A and Eleftheriou E 2018 Mixed-precision in-memory computing *Nat. Electron.* **1** 246–53
[5] Boybat I *et al* 2024 Heterogeneous embedded neural processing units utilizing PCM-based analog in-memory computing *2024 IEEE Int. Electron Devices Meeting (IEDM)*
[6] Lammie C, Benmeziane H, Simon W, Ferro E, Vasilopoulos A, Büchel J, Le Gallo M, Boybat I and Sebastian A 2025 Deep learning software stacks for analogue in-memory computing-based accelerators *Nat. Rev. Electr. Eng.* **2** 621–33
[7] Nandakumar S R, Boybat I, Joshi V, Piveteau C, Le Gallo M, Rajendran B, Sebastian A and Eleftheriou E 2019 Phase-change memory models for deep learning training and inference *2019 26th IEEE Int. Conf. on Electronics, Circuits and Systems (ICECS)*
[8] Sebastian A, Le Gallo M, Khaddam-Aljameh R and Eleftheriou E 2020 Memory devices and applications for in-memory computing *Nat. Nanotechnol.* **15** 529–44
[9] Nandakumar S R, Boybat I, Han J-P, Ambrogio S, Adusumilli P, Bruce R L, BrightSky M, Rasch M, Le Gallo M and Sebastian A 2020 Precision of synaptic weights programmed in phase-change memory devices for deep learning inference *2020 IEEE Int. Electron Devices Meeting (IEDM)* p 29.4.1–29.4.4
[10] Boybat I, Kersting B, Sarwat S G, Timoneda X, Bruce R L, BrightSky M, Gallo M L and Sebastian A 2021 Temperature sensitivity of analog in-memory computing using phase-change memory *2021 IEEE Int. Electron Devices Meeting (IEDM)*
[11] Le Gallo M *et al* 2023 Using the IBM analog in-memory hardware acceleration kit for neural network training and inference *APL Mach. Learn.* **1** 041102
[12] Rasch M J *et al* 2023 Hardware-aware training for large-scale and diverse deep learning inference workloads using in-memory computing-based accelerators *Nat. Commun.* **14** 5282

[13] Lammie C, Büchel J, Vasilopoulos A, Le Gallo M and Sebastian A 2025 The inherent adversarial robustness of analog in-memory computing *Nat. Commun.* **16** 1756
[14] Devlin J 2018 Bert: pre-training of deep bidirectional transformers for language understanding (arXiv:1810.04805)
[15] Benmeziane H, Lammie C, Vasilopoulos A, Boybat I, Gallo M L, Tsai H, Maghraoui K E and Sebastian A 2024 Multi-task neural network mapping onto analog-digital heterogeneous accelerators *NeurIPS 2024 Workshop Machine Learning With new Compute Paradigms*
[16] Hu E J, Shen Y, Wallis P, Allen-Zhu Z, Li Y, Wang S, Wang L and Chen W Lora: Low-rank adaptation of large language models 2021 (arXiv:2106.09685 [cs.CL])
[17] Hu E, Shen Y, Wallis P, Allen-Zhu Z, Li Y, Wang S, Wang L and Chen W Lora: low-rank adaptation of large language models arXiv:2106.09685v1)
[18] Le Gallo M *et al* 2023 A 64-core mixed-signal in-memory compute chip based on phase-change memory for deep neural network inference *Nat. Electron.* **6** 680–93
[19] Wen T-H *et al* 2024 Fusion of memristor and digital compute-in-memory processing for energy-efficient edge computing *Science* **384** 325–32
[20] Mohanty A, Du X, Chen P-Y, Seo J-S, Yu S and Cao Y 2017 Random sparse adaptation for accurate inference with inaccurate multi-level RRAM arrays *2017 IEEE Int. Electron Devices Meeting (IEDM)*
[21] Rasch M J, Moreda D, Gokmen T, Le Gallo M, Carta F, Goldberg C, El Maghraoui K, Sebastian A and Narayanan V 2021 A flexible and fast PyTorch toolkit for simulating training and inference on analog crossbar arrays *2021 IEEE 3rd Int. Conf. on Artificial Intelligence Circuits and Systems (AICAS)*
[22] Joshi V, Le Gallo M, Haefeli S, Boybat I, Nandakumar S R, Piveteau C, Dazzi M, Rajendran B, Sebastian A and Eleftheriou E 2020 Accurate deep neural network inference using computational phase-change memory *Nat. Commun.* **11** 2473
[23] Taori R, Gulrajani I, Zhang T, Dubois Y, Li X, Guestrin C, Liang P, and Hashimoto T B 2023 Stanford alpaca: an instruction-following llama model (available at: https://github.com/tatsu-lab/stanford_alpaca)
[24] Dettmers T, Pagnoni A, Holtzman A and Zettlemoyer L 2023 Qlora: efficient finetuning of quantized llms *Advances in Neural Information Processing Systems* vol 36 pp 10088–115
[25] Frantar E, Ashkboos S, Hoefler T and Alistarh D 2022 Gptq: accurate post-training quantization for generative pre-trained transformers (arXiv:2210.17323)
[26] Guo D *et al* 2025 Deepseek-r1: incentivizing reasoning capability in llms via reinforcement learning (arXiv:2501.12948)
[27] Büchel J, Simon W A, Lammie C, Acampa G, Maghraoui K E, Gallo M L and Sebastian A 2024 AIHWKIT-lightning: a scalable HW-aware training toolkit for analog in-memory computing *NeurIPS 2024 Workshop Machine Learning With new Compute Paradigms*
[28] Büchel J, Chalas I, Acampa G, Chen A, Fagbohungbe O, Tsai S, Maghraoui K E, Gallo M L, Rahimi A and Sebastian A 2025 Analog foundation models (arXiv:2505.09663)
[29] Zaruba F, Schuiki F, Hoefler T and Benini L 2021 Snitch: a tiny pseudo dual-issue processor for area and energy efficient execution of floating-point intensive workloads *IEEE Trans. Comput.* **70** 1845–60
[30] Schuiki F, Zaruba F, Hoefler T and Benini L 2021 Stream semantic registers: a lightweight risc-v isa extension achieving full compute utilization in single-issue cores *IEEE Trans. Comput.* **70** 212–27
[31] Benz T, Rogenmoser M, Scheffler P, Riedel S, Ottaviano A, Kurth A, Hoefler T and Benini L 2023 A high-performance, energy-efficient modular DMA engine architecture *IEEE Trans. Comput.* **73** 263–77
[32] Tortorella Y, Bertaccini L, Rossi D, Benini L and Conti F 2022 Redmule: a compact fp16 matrix-multiplication accelerator for adaptive deep learning on RISC-V-based ultra-low-power SoCs *Proc. 2022 Conf. & Exhibition on Design, Automation & Test in Europe*
[33] Büchel J, Simon W A, Lammie C, Acampa G, El Maghraoui K, Le Gallo M and Sebastian A 2024 Aihwkit-lightning: a scalable hw-aware training toolkit for analog in-memory computing *NeurIPS 2024 Workshop Machine Learning With new Compute Paradigms*
[34] Lammie C, Wang Y, Ponzina F, Klein J, Benmeziane H, Zapater M, Boybat I, Sebastian A, Ansaloni G and Atienza D 2025 Lionheart: a layer-based mapping framework for heterogeneous systems with analog in-memory computing tiles *IEEE Trans. Emerg. Top. Comput.* **13** 1383–95
[35] He J, Zhou C, Ma X, Berg-Kirkpatrick T and Neubig G 2021 Towards a unified view of parameter-efficient transfer learning (arXiv:2110.04366)
[36] Zhang Q, Chen M, Bukharin A, Karampatziakis N, He P, Cheng Y, Chen W and Zhao T 2023 Adalora: adaptive budget allocation for parameter-efficient fine-tuning (arXiv:2303.10512)
[37] Liu S-Y, Wang C-Y, Yin H, Molchanov P, Wang Y-C F, Cheng K-T and Chen M-H 2024 Dora: weight-decomposed low-rank adaptation *41st Int. Conf. on Machine Learning*
[38] Jordan K, Jin Y, Boza V, Jiacheng Y, Cesista F, Newhouse L, and Bernstein J 2024 Muon: an optimizer for hidden layers in neural networks (available at: https://kellerjordan.github.io/posts/muon/)
[39] Xiao G, Lin J, Seznec M, Wu H, Demouth J and Han S 2023 Smoothquant: accurate and efficient post-training quantization for large language models *Int. Conf. on Machine Learning* (PMLR) pp 38087–99
[40] Ororbia A and Friston K, 2023 Mortal computation: a foundation for biomimetic intelligence (arXiv:2311.09589)
[41] Hinton G 2022 The forward-forward algorithm: Some preliminary investigations (arXiv:2212.13345)
[42] Li C 2024 *LoRA on analog hardware* (available at: https://github.com/chenlicodebank/lora_on_analog_hardware)