Grouped Adaptive Weight Sharing (GAWS): An Inference-Efficient Adaptation Method for Large Language Models

Eman Alsuradi[1], Junhyun Lee[2]*, Kyenghun Lee[2], Hyeonmok Ko[2], Fahed Jubair[3]*
[1] Samsung Research Jordan (SRJO)
[2] Samsung Research
[3] Department of Computer Engineering, University of Jordan
{eman.zaki, junhyun8.lee, kyenghun.lee, felix.ko}@samsung.com, f.jubair@ju.edu.jo

Abstract

Although Low-Rank Adaptation (LoRA) revolutionized parameter-efficient fine-tuning, it often incurs an inference overhead due to the extra computation required by adapter layers. While most literature focuses on maximizing accuracy or minimizing parameter counts, this paper prioritizes single-request inference performance in the unmerged adapter setting, where adapters must remain decoupled from the base model at runtime. By analyzing LoRA adapters on GPUs, we identify segmented function calls as the primary source of this latency. To address this, we propose **G**rouped **A**daptive **W**eight **S**haring (GAWS), a novel adapter design based on *structured Kronecker product decomposition*. Experiments on T5-3B, GPT-2 Large, LLaMA3.2-3B, and RoBERTa-Large show that GAWS reduces latency to about 40% of the gap between the unmerged LoRA and the base model, while maintaining parameter efficiency and comparable accuracy. This positions GAWS as a Pareto-efficient solution for deploying adapted LLMs in latency-sensitive settings, balancing the high latency of compressed adapters with the accuracy of LoRA. The source code is available at: https://github.com/SamsungLabs/GAWS.

1 Introduction

Pretrained Transformer-based large language models (LLMs) have achieved remarkable success across a wide range of complex AI tasks. In natural language processing, the standard approach for adapting these models to specific downstream tasks involves full fine-tuning, which updates all model parameters and incurs high computational and storage costs. To mitigate these costs, adapter-based methods have emerged as a compelling alternative (Houlsby et al., 2019; Hu et al., 2022).

These methods introduce lightweight, task-specific modules, called adapters, into the pre-trained model. By training and storing only these small modules while keeping the base model frozen, adapter-based approaches significantly reduce computational and storage overhead. They also help preserve the general knowledge encoded in the pretrained model and mitigate catastrophic forgetting (Vu et al., 2022).

Adapter-based methods also enable a single

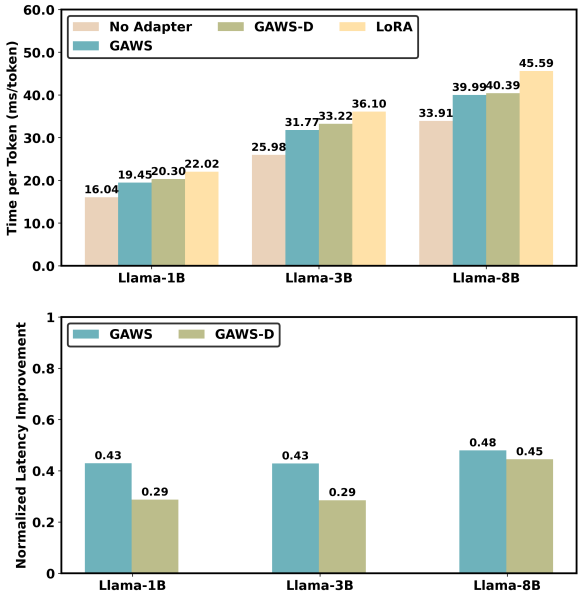


Figure 1: Inference latency (ms/token) measured on LLaMA models of three sizes (1B, 3B, and 8B) under four configurations: no-adapter, unmerged versions of LoRA, GAWS, and GAWS-D. All adapter configurations have similar parameter counts. **(Top)** absolute latency, **(Bottom)** normalized latency improvements, computed as $(\text{latency}_{\text{LoRA}} - \text{latency}_{\text{adapter}})/(\text{latency}_{\text{LoRA}} - \text{latency}_{\text{no-adapter}})$. Measurements were taken using an input sequence length of 1024 tokens and generating 128 output tokens with batch size = 1, under standard Pytorch eager mode CUDA execution. LoRA introduces an average latency overhead of 34% compared to the no-adapter baseline, whereas GAWS reduces this overhead by approximately 43%. Additional implementation details can be found in Appendix B.7.

*Corresponding authors.

Findings of the Association for Computational Linguistics: ACL 2026, pages 31790–31806

backbone model to support multiple tasks simultaneously through lightweight, task-specific modules, eliminating the need for model duplication (Yang et al., 2025; Feng et al., 2024).This modularity facilitates scalable deployment across diverse and personalized tasks settings. In such scenarios, adapters are kept unmerged and dynamically swapped at inference time, typically under single-request workloads (e.g., mobile AI, voice assistants, and AI code completion).

A straightforward strategy to minimize adapter-related runtime overhead is to merge adapter parameters into the base model weights prior to inference. However, this approach is often impractical in production environments. In resource-constrained settings where models are heavily quantized, for instance, weight merging can introduce numerical instabilities that lead to performance degradation (Fomenko et al., 2024). Furthermore, many production systems utilize statically compiled computational graphs that prohibit weight modifications at runtime, rendering dynamic merging incompatible with standard inference pipelines. Consequently, serving unmerged adapters remains the more practical alternative, despite the resulting inference time overhead.

As illustrated in Figure 1, using LoRA with LLaMA 3 models (Grattafiori et al., 2024) (1B, 3B, and 8B) results in an average latency increase of 34% compared to the base model, measured on an NVIDIA A100 80GB GPU. Notably, this overhead is most pronounced in the single-request (batch size = 1) setting, where kernel launch and function call fragmentation cannot be amortized across batches.

While adapter-based fine-tuning has improved significantly, most studies still focus mainly on accuracy and parameter efficiency, often overlooking the impact on inference speed. Recent system-level innovations address multi-tenant throughput via batched inference, but a critical gap remains in optimizing architectural latency for single-request, unmerged adapter serving. In this work, we target this gap by introducing **G**rouped **A**daptive **W**eight **S**haring (GAWS), a fine-tuning method that is both parameter- and latency-efficient, built on structured Kronecker product decomposition. GAWS is motivated by two key observations:

- **Latency Overhead in LoRA**: In LoRA, segmented CUDA kernel calls, caused by sequential matrix multiplications, introduce latency overhead. GAWS tackles this by using a structured Kronecker product decomposition, where one matrix is learnable and the other is fixed and structured. This design transforms the operation into a single, efficient matrix multiplication with grouped weight sharing, significantly lowering inference latency, as detailed in the Method section.
- **Maintaining Representational Power with Structured Efficiency**: Unlike LoRA, which relies on low-rank updates, GAWS employs a full-rank update matrix constructed through a Kronecker product decomposition. One of the decomposition factors is constrained to be a fixed structured matrix, which reduces the overall expressiveness of the full-rank update to a level comparable with LoRA. This structured constraint is a deliberate design choice that enables GAWS to preserve model quality while significantly accelerating inference. As a result, GAWS delivers LoRA-level performance with the added benefit of improved latency, offering a well-balanced trade-off between expressiveness and efficiency.

We fine-tuned a range of LLMs using our proposed adapter across multiple language generation and understanding tasks. On average, GAWS narrows the latency gap between LoRA and the base model by approximately 40%, while maintaining comparable performance (Experiments Section). We also benchmarked GAWS against other adapter methods, demonstrating its superior latency efficiency. Furthermore, our analysis revealed a high degree of similarity between the representations learned by models fine-tuned with LoRA and those fine-tuned with GAWS, which explains GAWS's similar performance to LoRA (see Ablation Experiments section).

2 Related Work

Adapter-Based Parameter-Efficient Fine-Tuning The growing size of Transformer models has motivated research in parameter-efficient fine-tuning (PEFT) methods that reduce the cost of adapting models to new tasks. Adapter-based approaches achieve this by inserting small, trainable modules into each Transformer layer while keeping the pretrained weights frozen. Among them, LoRA (Hu et al., 2022) introduces low-rank updates that significantly reduce trainable parameters.

Several extensions aim to improve LoRA's accuracy and training efficiency. AdaLoRA (Zhang et al., 2023), DoRA (Liu et al., 2024), and AutoLoRA (Zhang et al., 2024b) enhance expressiveness by adapting the rank or decomposing weight updates. Other approaches like QLoRA (Dettmers et al., 2023), LoRA+ (Hayou et al., 2024), VeRA (Kopiczko et al., 2024), and ResLoRA (Shi et al., 2024) improve training efficiency through quantization, learning rate tuning, training lightweight scaling vectors, or residual structures, respectively. Beyond LoRA, other adaptation methods target parameter efficiency: IA3 (Liu et al., 2022a) scales activations with learned vectors, and RoAd (Liao and Monz, 2024) applies simple 2D rotations to adapt LLM weights with minimal overhead.

Kronecker-based methods offer an alternative to low-rank approximations by enabling full-rank updates with fewer parameters. KronA (Edalati et al., 2022) applies Kronecker decompositions to improve expressiveness but lacks GAWS's structured constraints for fast inference. LoKr (Yeh et al., 2024) and AdaKron (Braga et al., 2024) build on this by combining Kronecker and low-rank decompositions and applying Kronecker products at the activation level. However, these methods primarily focus on training efficiency or accuracy, often overlooking inference overhead.

Adapter Serving for Multi-Task Inference Adapter-based architectures have emerged as efficient solutions for enabling large language models (LLMs) to handle multiple tasks concurrently. Recent systems such as S-LoRA (Sheng et al., 2023) and Punica (Chen et al., 2023a) focus on throughput, dynamically loading LoRA adapters or batching requests across users. In contrast, LoRA-Switch (Kong et al., 2024) targets latency reduction by selecting adapter paths at runtime through a mixture-of-experts approach and merging the chosen adapter with the base model to minimize overhead.

Despite their effectiveness, these approaches involve notable trade-offs. Throughput-oriented methods depend largely on low-level optimizations and do not directly address inference latency, while latency-focused approaches require merging adapters with the base model, an operation that is often impractical in production environments. GAWS addresses this gap by improving latency without relying on weight merging or specialized low-level optimizations, making it particularly suitable for real-world deployment.

Inference Efficiency in Adapter Models Improving inference efficiency in adapter-based models has gained attention through conditional computation and adapter compression. Methods such as DyT (Zhao et al., 2024) and CODA (Lei et al., 2023) reduce computation by selectively routing tokens through adapter paths. LoRAShear (Chen et al., 2023b) and LoRAPrune (Zhang et al., 2024a) apply structured pruning to reduce overhead with minimal loss in accuracy, while zFLoRA (Gowda et al., 2025) addresses the issue through structural layer fusion. GAWS takes an orthogonal approach, proposing a lightweight adapter that integrates smoothly with conditional computation and compression techniques for further gains.

3 Background & Motivation

3.1 LLM Adaptation

Large language models (LLMs) are commonly adapted to downstream tasks using *supervised fine-tuning* (SFT). Given an input x and a target output $y = \{y_1, y_2, \ldots, y_N\}$, the model is trained to maximize the autoregressive log-likelihood:

$$\mathcal{L}_{\text{SFT}}(\mathcal{W}) = \sum_{(x,y)\in\mathcal{Z}} \sum_{t=1}^{N} \log P_{\mathcal{W}}(y_t \mid x, y_{<t}). \quad (1)$$

Here, $\mathcal{W}$ denotes the full set of trainable parameters in the model, which includes the set of model's individual weight matrices $W^{(i)} \in \mathcal{W}$. In *full fine-tuning*, all weights are updated from their pre-trained values $\mathcal{W}_0$ via:

$$\mathcal{W} = \mathcal{W}_0 + \Delta\mathcal{W}. \quad (2)$$

For a weight matrix $W_0 \in \mathbb{R}^{d\times k}$, the forward pass becomes:

$$h = (W_0 + \Delta W)x. \quad (3)$$

While this approach offers full flexibility, it is costly in terms of memory and computation, as $\Delta\mathcal{W}$ is the same size as $\mathcal{W}_0$, requiring a full model copy per task.

Low-Rank Adaptation (LoRA) reduces this cost by constraining updates to a low-rank form. The update is parameterized as:

$$\Delta W = BA$$
$$B \in \mathbb{R}^{d\times r}, \quad A \in \mathbb{R}^{r\times k}, \quad r \ll \min(d, k). \quad (4)$$

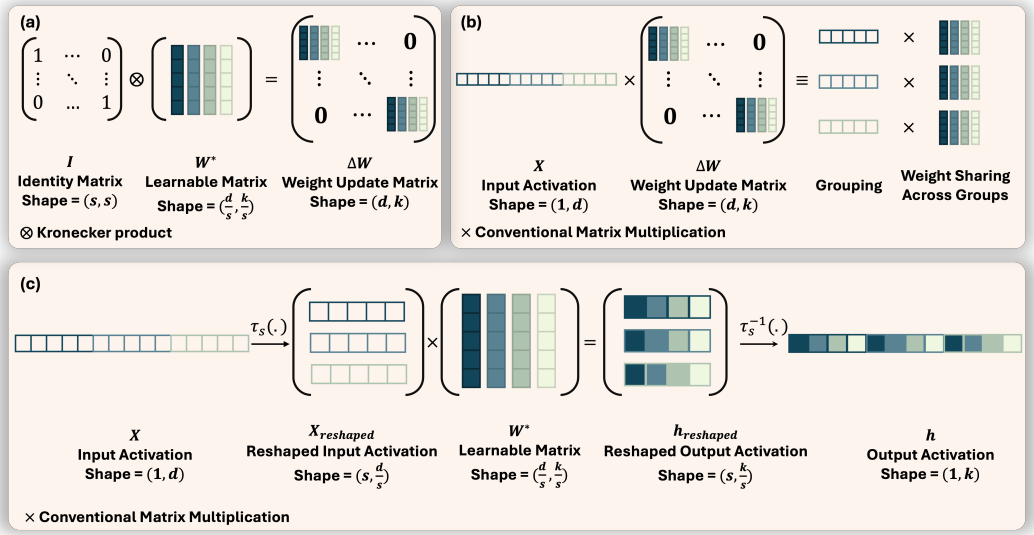


Figure 2: GAWS adaptation scheme. (a) The weight update matrix ΔW is formulated as a Kronecker product between a fixed structured identity matrix I and a trainable matrix W^*. (b) This structure enables efficient group-wise weight sharing by transforming the matrix product $X \times \Delta W$ into a group-wise operation, where each input group in X shares the same W^*. (c) The resulting design supports highly efficient computation, as illustrated.

The resulting forward pass is:

$$h = W_0 x + BAx. \tag{5}$$

LoRA keeps W_0 frozen and only trains A and B, and the training objective becomes:

$$\max_{A,B} \sum_{(x,y)\in\mathcal{Z}} \sum_{t=1}^{N} \log P_{\mathcal{W}_0+\Delta\mathcal{W}(A,B)}(y_t \mid x, y_{<t}). \tag{6}$$

This significantly reduces the number of trainable parameters. However, it introduces inference-time overhead, as each forward pass requires two additional matrix multiplications: Ax and $B(Ax)$, increasing memory access and kernel launch costs. This overhead becomes more pronounced in transformer-based LLM, where LoRA is applied to multiple projection matrices (e.g., query, key, value) across layers. Although merging ΔW into W_0 post-training can remove this overhead, such merging is infeasible in settings requiring model sharing across tasks, where adapters must remain modular and decoupled from the base model, to name a few.

To better analyze the source of this overhead, we profile GPU execution traces and observe a significant increase in kernel launch overhead. As shown in Table 1, LoRA introduces substantially more kernel launches than the base model (11,823 vs. 8,603), leading to higher total CUDA and CPU execution time.

Metric	Base	LoRA	GAWS
Kernel launches	8,603	11,823	10,199
Total CUDA time (ms/token)	357.88	426.44	385.15
Total CPU time (ms/token)	335.01	477.84	384.93
Total time (ms/token)	692.89	904.28	770.08

Table 1: PyTorch Profiler results on LLaMA-3.2-3B during token generation (input length 1024, output length 128, batch size 1), with comparable adapter parameter counts across methods.

These challenges motivate the need for new methods that retain LoRA's efficiency while improving inference-time performance. Our approach, introduced in the next section, addresses this trade-off.

3.2 Kronecker Product

The *Kronecker product* is a matrix operation that constructs a block-structured matrix from two input matrices. Given $X \in \mathbb{R}^{m\times n}$ and $Y \in \mathbb{R}^{p\times q}$, their Kronecker product, denoted $X \otimes Y$, produces a matrix of size $(mp) \times (nq)$, defined as:

$$X \otimes Y = \begin{bmatrix} x_{11}Y & x_{12}Y & \cdots & x_{1n}Y \\ x_{21}Y & x_{22}Y & \cdots & x_{2n}Y \\ \vdots & \vdots & \ddots & \vdots \\ x_{m1}Y & x_{m2}Y & \cdots & x_{mn}Y \end{bmatrix}. \tag{7}$$

Each scalar entry x_{ij} in X is multiplied by the

entire matrix Y, resulting in a larger matrix with repeated structure.

The Kronecker product is commonly used to combine low-dimensional matrices into higher-dimensional representations that retain internal structure. Conversely, it can also serve as a decomposition tool: a large matrix can be approximated by the Kronecker product of two smaller matrices.This decomposition forms the basis of our proposed adapter, which uses a structured Kronecker product as an alternative to LoRA's low-rank decomposition, preserving comparable representational capacity while enabling more efficient computation through its constrained full-rank design.

4 Grouped Adaptive Weight Sharing

We propose **Grouped Adaptive Weight Sharing (GAWS)**, a new adaptation method designed for parameter and latency efficiency. GAWS combines two powerful ideas: (1) parameter-efficient fine-tuning via a rank-preserving Kronecker product decomposition, ensuring high accuracy, and (2) low-latency inference by enforcing structure on one Kronecker component, allowing efficient group-wise matrix multiplication. As the name suggests, GAWS shares a trainable weight update matrix across grouped input segments, striking a balance between speed and accuracy. As in LoRA, GAWS keeps the pre-trained weight matrix $W_0 \in \mathbb{R}^{d\times k}$ frozen and learns an additive update $\Delta W \in \mathbb{R}^{d\times k}$ during fine-tuning. However, instead of approximating this update using two low-rank matrices, GAWS models it using a Kronecker product:

$$\Delta W = I \otimes W^* \tag{8}$$

where $I \in \mathbb{R}^{s\times s}$ is the identity matrix, $W^* \in \mathbb{R}^{\frac{d}{s}\times\frac{k}{s}}$ is a trainable matrix, and s is a grouping hyperparameter such that s divides both d and k. This yields a block-diagonal matrix where each block is a copy of W^*, as illustrated below:

$$\Delta W = I \otimes W^* = \begin{bmatrix} W^* & 0 & \cdots & 0 \\ 0 & W^* & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & W^* \end{bmatrix}. \tag{9}$$

Unlike LoRA, which uses two matrices A and B, GAWS employs only a single trainable matrix W^*, simplifying the adapter structure.

To efficiently implement the effect of the structured-Kronecker update without explicitly materializing ΔW, GAWS performs an equivalent computation directly on the input activation. Specifically, the input activation $x \in \mathbb{R}^{1\times d}$ is first reshaped using a function $\tau_s(\cdot)$, which partitions it into s contiguous groups, resulting in a matrix of shape $s \times \frac{d}{s}$. Each group is then multiplied by the shared trainable matrix $W^* \in \mathbb{R}^{\frac{d}{s}\times\frac{k}{s}}$, producing an intermediate output of shape $s \times \frac{k}{s}$. Finally, the function $\tau_s^{-1}(\cdot)$ concatenates the resulting groups to produce the final output vector $h \in \mathbb{R}^{1\times k}$.

The complete GAWS forward pass is given by:

$$h = W_0 x + \tau_s^{-1}(\tau_s(x) \times W^*). \tag{10}$$

Figure 2 illustrates the GAWS forward pass in full detail. It depicts the Kronecker-based structure of ΔW, the partitioning of the input vector x into groups via $\tau_s(x)$, and the application of the shared matrix W^* across groups. The figure also shows how the resulting outputs are concatenated using $\tau_s^{-1}(\cdot)$ to produce the final output vector h.

The operations $\tau_s(\cdot)$ and $\tau_s^{-1}(\cdot)$ can be implemented using tensor *view* operations in PyTorch, which execute in constant time by modifying tensor metadata, assuming memory contiguity, as in our setup.

In contrast to LoRA, which introduces two additional matrix multiplications, and consequently two separate computation kernels, per forward pass, GAWS requires only one. This reduces kernel launch overhead and enhances inference-time efficiency, particularly when applied across multiple layers of a transformer model.

We also introduce a second variant of our method, termed GAWS-D, illustrated in Figure 5 in Appendix A. This variant leverages the non-commutative property of the Kronecker product: by reversing the order of the decomposition, it yields a fundamentally different architecture defined as:

$$\Delta W = W^* \otimes I. \tag{11}$$

GAWS-D builds on the original GAWS framework but modifies the interaction between the reshaped input and the adapter weights. Specifically, the forward pass is computed as:

$$h = W_0 x + \tau_s^{-1}(W^* \times \tau_s(x)), \tag{12}$$

reversing the direction of matrix multiplication compared to standard GAWS.

In GAWS, each group of the reshaped input $\tau_s(x)$ is independently transformed by W^*, enabling localized adaptation. In GAWS-D, the input

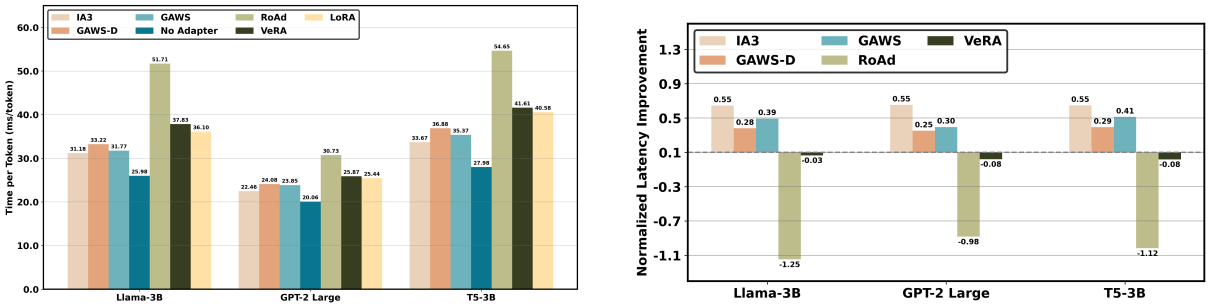


Figure 3: Inference latency (ms/token) for T5-3B, LLaMA3.2-3B, and GPT2-Large across various adapter methods. **(Left)**absolute latency, **(Right)** normalized latency, computed as $(\text{latency}_{\text{LoRA}} - \text{latency}_{\text{adapter}})/(\text{latency}_{\text{LoRA}} - \text{latency}_{\text{no-adapter}})$. Measurements were taken using an input sequence length of 1024 tokens and generating 128 output tokens with batch size = 1, under standard Pytorch eager mode CUDA execution. GAWS and GAWS-D consistently achieve the lowest latency among all evaluated adapters, demonstrating their efficiency for real-time deployment. Additional implementation details can be found in Appendix B.7.

is multiplied from the right, allowing each output group to incorporate information from all input groups. This naturally leads to a *dilation-like* interaction pattern, hence the name GAWS-D, enabling broader context aggregation while keeping the parameter count unchanged. We evaluate both GAWS and GAWS-D in our experiments and analyze the trade-offs between local and global interactions, as well as their respective impacts on accuracy and inference-time efficiency.

We also propose a third variant that introduces a learnable diagonal matrix D into the forward pass of both GAWS and GAWS-D. This matrix scales the columns of W^*, allowing the model to modulate the contribution of each output dimension independently. For GAWS, the forward pass becomes:

$$h = W_0 x + \tau_s^{-1} \left(\tau_s(x) \times (W^* D)\right) \tag{13}$$

For GAWS-D, the forward pass becomes:

$$h = W_0 x + \tau_s^{-1} \left((W^* D) \times \tau_s(x)\right) \tag{14}$$

This extension is motivated by prior works (Liu et al., 2022b; Kopiczko et al., 2024), which shows that incorporating lightweight, per-dimension scaling can improve convergence and stability during fine-tuning. By introducing D, we add a minimal degree of learnable flexibility while keeping the overall adapter structure simple and efficient. Importantly, this modification incurs no additional cost at inference time. Since D is a fixed diagonal matrix, it can be folded into W^* via standard matrix multiplication after training. This preserves the structure and computational footprint of the original forward pass, ensuring that runtime efficiency and memory usage remain unchanged.

5 Experiments

Overview Our experiment has two main phases: (1) inference latency benchmarking, where we measure the latency of various baseline adapters alongside our proposed adapters, (2) fine-tuning accuracy evaluation, comparing our adapters against these top-performing latency baselines. The following sections detail the models, adapter configurations, and experimental setups for each phase.

Models and Baseline Adapters To ensure a broad evaluation across model architectures and sizes, we include: (1) encoder-only: RoBERTa Large (355M) (Liu et al., 2019); (2) decoder-only: GPT-2 Large (774M), and LLaMA 3.2-3B (Radford et al., 2019; Grattafiori et al., 2024) (3) encoder-decoder: T5-3B (Raffel et al., 2023); and (4) Stable-Diffusion-v1-4 (1B)(Rombach et al., 2022). All models are sourced from Hugging Face (Wolf et al., 2020). Our baselines include the base model without adapters (full fine-tuning) and four PEFT methods: LoRA (Hu et al., 2022), IA3 (Liu et al., 2022a), VeRA (Kopiczko et al., 2024), and RoAd (Liao and Monz, 2024). We select IA3, VeRA, and RoAd as they are among the most parameter-efficient adapters proposed in the literature, and thus represent meaningful competitors to our latency-efficient design. LoRA is included as it represents the canonical and most widely used PEFT adapter. All baseline adapter implementations are obtained from the PEFT library (Mangrulkar et al., 2022), while our proposed adapters are implemented by extending this framework. Further details on models' and adapters' architectures, as well as adapters' target modules, are provided in Appendix B.

Model	Task	Items	FFT	LoRA	VeRA	IA3	RoAd	GAWS	GAWS$^{(d)}$	GAWS-D	GAWS-D$^{(d)}$
GPT2-Large	WebNLG	Param.	774M	8.85M	139K	138K	553K	7.08M	7.09M	7.08M	7.09M
		BLEU	55.30$_{(0.28)}$	**57.14**$_{(0.10)}$	56.79$_{(0.09)}$	48.20$_{(0.09)}$	53.36$_{(0.06)}$	56.38$_{(0.25)}$	56.87$_{(0.08)}$	56.24$_{(0.10)}$	56.95$_{(0.10)}$
		TER	0.42$_{(0.00)}$	**0.38**$_{(0.00)}$	0.38$_{(0.00)}$	0.43$_{(0.00)}$	0.40$_{(0.01)}$	0.39$_{(0.00)}$	0.39$_{(0.00)}$	0.39$_{(0.00)}$	0.38$_{(0.00)}$
	Dart	Param.	774M	4.42M	138K	138K	553K	1.77M	1.77M	1.77M	1.77M
		BLEU	47.23$_{(0.23)}$	**48.50**$_{(0.11)}$	47.87$_{(0.29)}$	43.88$_{(0.08)}$	46.04$_{(0.03)}$	48.19$_{(0.16)}$	48.35$_{(0.08)}$	48.14$_{(0.21)}$	48.20$_{(0.10)}$
		TER	0.46$_{(0.00)}$	**0.46**$_{(0.00)}$	0.48$_{(0.01)}$	0.50$_{(0.00)}$	0.49$_{(0.01)}$	0.47$_{(0.00)}$	0.47$_{(0.00)}$	0.47$_{(0.00)}$	0.47$_{(0.01)}$
RoBERTa-Large	MRPC	Param.	355M	1.83M	1.10M	1.10M	1.25M	1.83M	1.84M	1.83M	1.84M
		Acc.	90.20*	88.24$_{(0.10)}$	89.22$_{(0.10)}$	87.75$_{(0.08)}$	87.99$_{(0.05)}$	90.20$_{(0.10)}$	90.20$_{(0.15)}$	**90.93**$_{(0.10)}$	88.97$_{(0.10)}$
	CoLA	MCC	63.60*	**66.81**$_{(0.20)}$	65.30$_{(0.10)}$	59.06$_{(0.10)}$	64.06$_{(0.09)}$	64.63$_{(0.15)}$	65.32$_{(0.20)}$	65.82$_{(0.10)}$	65.30$_{(0.30)}$
	RTE	Acc.	78.7*	84.48$_{(0.10)}$	83.73$_{(0.12)}$	81.22$_{(0.10)}$	84.11$_{(0.15)}$	83.75$_{(0.20)}$	84.12$_{(0.10)}$	**85.92**$_{(0.15)}$	82.31$_{(0.10)}$
LLaMA3.2-3B	SquAD-v2	Param.	3B	12.85M	146K	143K	573K	10.09M	10.13M	10.09M	10.13M
		EM	71.69$_{(0.01)}$	75.79$_{(0.25)}$	69.88$_{(0.04)}$	70.75$_{(0.05)}$	72.07$_{(0.06)}$	76.05$_{(0.05)}$	76.94$_{(0.14)}$	**77.71**$_{(0.10)}$	74.45$_{(0.10)}$
		F1	79.28$_{(0.04)}$	83.06$_{(0.19)}$	77.12$_{(0.12)}$	78.05$_{(0.10)}$	79.50$_{(0.10)}$	83.21$_{(0.07)}$	84.16$_{(0.08)}$	**84.97**$_{(0.10)}$	81.49$_{(0.10)}$
	XSum	R-1	41.22$_{(0.40)}$	42.35$_{(0.50)}$	37.87$_{(0.20)}$	38.31$_{(0.05)}$	39.27$_{(0.05)}$	42.59$_{(0.07)}$	43.15$_{(0.21)}$	42.79$_{(0.17)}$	**43.25**$_{(0.11)}$
		R-2	18.03$_{(0.17)}$	18.96$_{(0.26)}$	14.96$_{(0.30)}$	15.46$_{(0.05)}$	16.23$_{(0.03)}$	19.15$_{(0.08)}$	19.77$_{(0.20)}$	19.37$_{(0.15)}$	**19.87**$_{(0.09)}$
		R-L	32.72$_{(0.21)}$	34.09$_{(0.26)}$	29.90$_{(0.05)}$	30.35$_{(0.03)}$	31.20$_{(0.01)}$	34.31$_{(0.11)}$	34.96$_{(0.20)}$	34.35$_{(0.17)}$	**35.06**$_{(0.09)}$
T5-3B	SquAD-v2	Param.	3B	17.69M	888K	885K	3.54M	14.16M	14.18M	14.16M	14.18M
		EM	**79.65**$_{(0.23)}$	79.65$_{(0.22)}$	76.53$_{(0.10)}$	77.60$_{(0.15)}$	78.41$_{(0.10)}$	79.05$_{(0.10)}$	79.13$_{(0.06)}$	79.44$_{(0.08)}$	79.42$_{(0.05)}$
		F1	**86.49**$_{(0.17)}$	86.48$_{(0.16)}$	83.30$_{(0.10)}$	84.48$_{(0.12)}$	85.17$_{(0.10)}$	85.98$_{(0.10)}$	86.06$_{(0.06)}$	86.27$_{(0.11)}$	86.24$_{(0.06)}$
	XSum	R-1	**46.12**$_{(0.05)}$	44.61$_{(0.04)}$	41.30$_{(0.08)}$	81.74$_{(0.10)}$	42.63$_{(0.10)}$	44.16$_{(0.05)}$	44.10$_{(0.10)}$	44.38$_{(0.04)}$	44.23$_{(0.10)}$
		R-2	**22.56**$_{(0.06)}$	20.79$_{(0.06)}$	17.58$_{(0.10)}$	17.97$_{(0.07)}$	18.76$_{(0.06)}$	20.32$_{(0.06)}$	20.31$_{(0.11)}$	20.61$_{(0.11)}$	20.57$_{(0.11)}$
		R-L	**37.86**$_{(0.05)}$	36.42$_{(0.06)}$	32.94$_{(0.10)}$	33.46$_{(0.12)}$	34.36$_{(0.10)}$	35.89$_{(0.05)}$	35.81$_{(0.11)}$	36.16$_{(0.09)}$	36.12$_{(0.11)}$

Table 2: Quantitative results of experiments. Superscript (d) indicates a diagonal matrix is incorporated into the adapter. * denotes numbers reported in prior work. Results are averaged over three seeds for GPT2-Large and RoBERTa-Large, two seeds for LLaMA3.2-3B and T5-3B. For all metrics, higher values indicate better performance, except for TER. MCC indicates Matthews correlation coefficient (See Appendix B.5).

5.1 Inference Latency Benchmarking

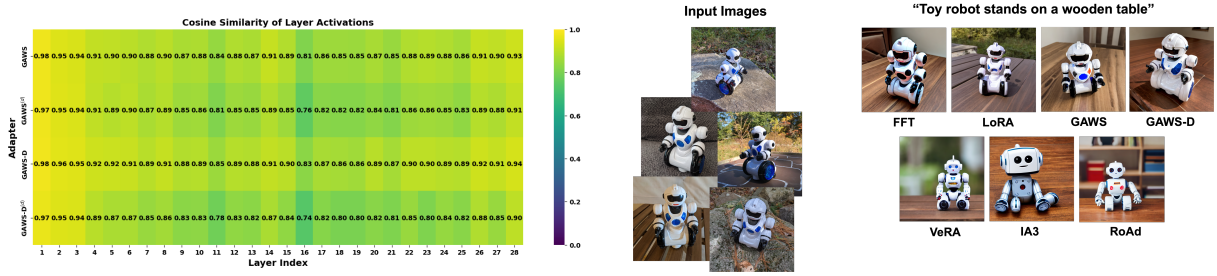
Experimental Setup We ensure fair and consistent inference latency measurements by standardizing our experimental setup: (1) *Parameter budget alignment*: Adapter configurations are chosen to keep parameter counts comparable across models (see Appendix for details). (2) *Fixed input and output lengths*: Models are evaluated with 1024-token inputs and, for generative models, 128-token outputs. (3) *Consistent hardware and measurements*: All experiments run on an NVIDIA A100 GPU (80 GB), with each latency measurement repeated 100 times for statistical reliability.

Results Figure 3 shows inference latency (ms/token) for T5-3B, LLaMA-3.2-3B, and GPT-2 Large, across baseline and proposed adapters. Latency is measured as token generation time, obtained by subtracting prompt processing time from total inference time. Prompt processing latencies, and latency plots for RoBERTa-Large and Stable Diffusion v1-4 are provided in the Appendix.

As shown in Figure 3, IA3 achieves the lowest latency, as it introduces only a lightweight scaling vector applied to layer activations. GAWS and GAWS-D consistently attain the second-lowest latency across all evaluated models. This performance advantage arises from their efficient design, which avoids the segmented CUDA kernel invocations required by competing methods such as VeRA and RoAd. Notably, GAWS closes approximately 40% of the latency gap between LoRA and the base model. A Wilcoxon signed-rank test confirms that GAWS is significantly faster than LoRA ($p < 0.001$). Although GAWS-D matches GAWS in terms of parameter count and FLOPs, it exhibits slightly higher latency due to a less cache-friendly, dilated memory access pattern. In contrast, GAWS relies on sequential memory accesses with improved data locality, leading to more efficient execution.

5.2 Fine-tuning Accuracy Evaluation

Experimental Setup To assess whether GAWS matches the accuracy of the selected baselines, we fine-tune all models on downstream tasks. GPT-2 Large is fine-tuned for text-to-table generation on WebNLG (Gardent et al., 2017) and DART (Nan et al., 2021). LLaMA 3.2-3B and T5-3B are fine-tuned on question answering (SQuAD v2) (Rajpurkar et al., 2018) and summarization (XSum) (Narayan et al., 2018). RoBERTa Large is fine-tuned on GLUE benchmarks MRPC, RTE, and CoLA (Wang et al., 2019). Stable Diffusion v1-4 is fine-tuned on a subject-driven DreamBooth dataset (Ruiz et al., 2023). Experiments with GPT-2 Large and RoBERTa Large are conducted on an NVIDIA P40 (24 GB), while LLaMA 3.2-3B, T5-3B, and Stable Diffusion v1-4 are run on an NVIDIA A100 (80 GB).



(a) (b)

Figure 4: (a) Cosine similarity of layer activations heatmap in LLaMA3.2-3B fine-tuned on SQuAD-v2 with LoRA and our adapters. Despite structural differences, GAWS and LoRA yield highly similar activation patterns. (b) Qualitative results for subject-driven text-to-image generation with Stable Diffusion v1-4. Our adapters consistently produce images that faithfully follow the prompt.

Hyperparameters, including learning rate, batch size, gradient accumulation, and adapter dimensions are selected via grid search. Full dataset and evaluation metrics details, adapter initialization, and hyperparameter configurations are provided in Appendix B.

Results Table 2 summarizes task performance for the baseline methods and our proposed adapters. GAWS consistently outperforms IA3 and RoAd in accuracy across all experiments. While IA3 achieves lower latency than GAWS, it suffers from substantially reduced accuracy. In contrast, GAWS lies on a more favorable Pareto frontier, providing a balanced solution for scenarios where both latency and task accuracy are critical. GAWS achieves accuracy comparable to LoRA and VeRA across all tasks, and in some cases matches or surpasses their performance. A one-sample t-test on relative performance further confirms that GAWS is non-inferior to LoRA within a 1% margin ($p < 0.05$), demonstrating that GAWS preserves accuracy while providing significantly improved latency.

For the subject-driven text-to-image generation task, Figure 4b presents qualitative results for a toy robot subject using a Stable Diffusion v1-4 model fine-tuned with both baseline and proposed adapters. Our adapters consistently generate high-quality images that accurately reflect the input prompts.

6 Ablation Experiments

We conduct a series of ablation studies to analyze GAWS from both architectural and system-level perspectives. Our evaluation focuses on four complementary aspects: (i) architectural variations, (ii) representational similarity, (iii) adapter behavior under compiler optimizations, and (iv) memory efficiency.

6.1 Architectural Variants

We evaluate two GAWS modifications: (i) reversing the matrix multiplication order (GAWS-D) and (ii) adding a diagonal scaling matrix. As shown in Table 2, both changes yield consistent, incremental performance gains. GAWS-D generally outperforms GAWS, indicating that reversing the multiplication order is beneficial. Adding diagonal scaling (GAWS$^{(d)}$) provides further improvements, and combining both modifications (GAWS-D$^{(d)}$) often matches or exceeds any other variant. These results suggest that matrix direction and diagonal scaling are complementary design choices that enhance GAWS's representational capacity with minimal parameter overhead.

6.2 Representation Similarity

Although GAWS and LoRA differ structurally, LoRA uses low-rank decomposition, while GAWS enforces a structured Kronecker product, it remains unclear whether they learn similar representations. To investigate this, we extract layer-wise activations from models fine-tuned with LoRA, GAWS, and GAWS-D by running inference on the fine-tuning test sets. We compute cosine similarity between LoRA activations and those of the other adapters. Figure 4a shows activations similarity heatmap for LLaMA 3.2-3B on SQuAD v2. Despite architectural differences, GAWS and GAWS-D exhibit activation patterns closely aligned with LoRA across layers, indicating convergence to comparable representations.

6.3 GAWS Behavior under Compiler Optimizations

Modern deep learning systems rely on compiler optimizations (e.g., `torch.compile`, `TensorRT`) to reduce execution overhead via graph fusion and kernel specialization. However, these gains largely assume static computation graphs and degrade in dynamic serving settings where adapters are swapped at runtime. In such cases, adapter parameters are treated as inputs, restricting fusion opportunities between the base model and adapter computations.

GAWS addresses this limitation at the architectural level by reformulating LoRA updates into a single-pass computation. This reduces dependence on compiler-level fusion and preserves latency benefits even under constrained compilation.

We evaluate base model, LoRA, and GAWS under `torch.compile` with `max-autotune`, using runtime-provided adapters to simulate dynamic swapping. Experiments are run on LLaMA3.2-3B with 1024-token prompts and batch size 1, averaged over 50 runs. The results are summarized in Table 3.

Method	Avg. Latency (ms/token)
Base Model	23.07
LoRA	27.79
GAWS	26.40

Table 3: Token generation latency under `torch.compile` with dynamic adapter inputs on LLaMA3.2-3B, evaluated with a 1024-token prompt and batch size 1.

As shown in Table 3, GAWS consistently reduces latency relative to LoRA, recovering approximately 30% of the gap between LoRA and the base model. This demonstrates that the architectural simplification of GAWS remains effective even when compiler optimizations are limited.

6.4 GAWS Memory Efficiency

Although GAWS uses a single matrix parameterization while LoRA relies on a two-matrix decomposition, both methods are configured with a comparable number of trainable parameters in our experiments. Consequently, their parameter and gradient storage requirements are effectively equivalent.

To empirically assess memory usage, we measure GPU allocated memory, GPU peak memory, and CPU memory during training on LLaMA3.2-3B. The results are reported in Table 4.

Method	GPU Allocated (GB)	GPU Peak (GB)	CPU (GB)
LoRA	12.38	45.17	2.26
GAWS	12.38	41.13	2.27

Table 4: Training memory usage comparison on LLaMA3.2-3B, reporting GPU allocated memory, GPU peak memory, and CPU memory.

As shown in Table 4, the allocated GPU memory is identical across methods, confirming comparable parameter and gradient storage. However, GAWS reduces peak GPU memory usage by approximately 4 GB, indicating improved runtime memory efficiency.

This reduction stems from the underlying architectural difference. LoRA's two-stage low-rank decomposition introduces intermediate activations (e.g., projections into the rank space) that must be materialized and retained for backpropagation. In contrast, GAWS reformulates the update as a single matrix multiplication, eliminating these intermediate tensors and their associated backward buffers. As a result, fewer transient activations are stored during forward and backward passes, leading to lower peak memory usage despite similar parameter counts.

7 Conclusion

We propose GAWS, a parameter- and latency-efficient fine-tuning method based on a structured Kronecker product decomposition for grouped weight sharing. This design reduces segmented CUDA kernel calls, improving inference speed while preserving rank and expressiveness. GAWS closes 40% of the latency gap between non-merged LoRA and a no-adapter baseline, while maintaining comparable parameter efficiency and accuracy. These properties make our adapter suitable for latency-sensitive, per-request deployment where adapters remain unmerged, such as multi-tenant serving with quantized base models or on-device dynamic adapter swapping.

Limitations

The GAWS adapter design introduces structural constraints tied to each layer's input and output dimensions: the grouping hyperparameter s must divide both, restricting valid adapter configurations. However, this constraint can be relaxed using standard padding, without affecting correctness or the underlying formulation.

References

Marco Braga, Alessandro Raganato, and Gabriella Pasi. 2024. AdaKron: An Adapter-based Parameter Efficient Model Tuning with Kronecker Product. In *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING 2024)*.

Lequn Chen, Zihao Ye, Yongji Wu, Danyang Zhuo, Luis Ceze, and Arvind Krishnamurthy. 2023a. Punica: Multi-Tenant LoRA Serving. *arXiv preprint arXiv:2310.18547*.

Tianyi Chen, Tianyu Ding, Badal Yadav, Ilya Zharkov, and Luming Liang. 2023b. LoRAShear: Efficient Large Language Model Structured Pruning and Knowledge Recovery. *arXiv preprint arXiv:2310.18356*.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2023. QLoRA: Efficient Finetuning of Quantized LLMs. *arXiv preprint arXiv:2305.14314*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805.

Ali Edalati, Marzieh Tahaei, Ivan Kobyzev, Vahid Partovi Nia, James J. Clark, and Mehdi Rezagholizadeh. 2022. KronA: Parameter Efficient Tuning with Kronecker Adapter. *arXiv preprint arXiv:2212.10650*.

Wenfeng Feng, Chuzhan Hao, Yuewei Zhang, Yu Han, and Hao Wang. 2024. Mixture-of-loras: An efficient multitask tuning for large language models. *Preprint*, arXiv:2403.03432.

Vlad Fomenko, Han Yu, Jongho Lee, Stanley Hsieh, and Weizhu Chen. 2024. A note on lora. *Preprint*, arXiv:2404.05086.

Claire Gardent, Anastasia Shimorina, Shashi Narayan, and Laura Perez-Beltrachini. 2017. The WebNLG challenge: Generating text from RDF data. In *Proceedings of the 10th International Conference on Natural Language Generation*, pages 124–133, Santiago de Compostela, Spain. Association for Computational Linguistics.

Dhananjaya Gowda, Seoha Song, Harshith Goka, and Junhyun Lee. 2025. zflora: Zero-latency fused low-rank adapters. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 21412–21429.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, and 1 others. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Soufiane Hayou, Nikhil Ghosh, and Bin Yu. 2024. LoRA+: Efficient Low Rank Adaptation of Large Models. *arXiv preprint arXiv:2402.12354*.

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin de Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. 2019. Parameter-efficient transfer learning for nlp. *Preprint*, arXiv:1902.00751.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. LoRA: Low-Rank Adaptation of Large Language Models. In *International Conference on Learning Representations (ICLR)*.

Rui Kong, Qiyang Li, Xinyu Fang, Qingtian Feng, Qingfeng He, Yazhu Dong, Weijun Wang, Yuanchun Li, Linghe Kong, and Yunxin Liu. 2024. LoRA-Switch: Boosting the Efficiency of Dynamic LLM Adapters via System-Algorithm Co-design. *arXiv preprint arXiv:2405.17741*.

Dawid J. Kopiczko, Tijmen Blankevoort, and Yuki M. Asano. 2024. VeRA: Vector-based Random Matrix Adaptation. In *International Conference on Learning Representations (ICLR)*.

Tao Lei, Junwen Bai, Siddhartha Brahma, Joshua Ainslie, Kenton Lee, Yanqi Zhou, Nan Du, Vincent Y. Zhao, Yuexin Wu, Bo Li, Yu Zhang, and Ming-Wei Chang. 2023. Conditional Adapters: Parameter-efficient Transfer Learning with Fast Inference. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*.

Baohao Liao and Christof Monz. 2024. 3-in-1: 2d rotary adaptation for efficient finetuning, efficient batching and composability. *Preprint*, arXiv:2409.00119.

Chin-Yew Lin. 2004. ROUGE: A package for automatic evaluation of summaries. In *Text Summarization Branches Out*, pages 74–81, Barcelona, Spain. Association for Computational Linguistics.

Haokun Liu, Derek Tam, Mohammed Muqeeth, Jay Mohta, Tenghao Huang, Mohit Bansal, and Colin Raffel. 2022a. Few-shot parameter-efficient fine-tuning is better and cheaper than in-context learning. *Preprint*, arXiv:2205.05638.

Haokun Liu, Derek Tam, Mohammed Muqeeth, Jay Mohta, Tenghao Huang, Mohit Bansal, and Colin Raffel. 2022b. Few-Shot Parameter-Efficient Fine-Tuning is Better and Cheaper than In-Context Learning. *arXiv preprint arXiv:2205.05638*.

Shih-Yang Liu, Chien-Yi Wang, Hongxu Yin, Pavlo Molchanov, Yu-Chiang Frank Wang, Kwang-Ting Cheng, and Min-Hung Chen. 2024. DoRA: Weight-Decomposed Low-Rank Adaptation. *arXiv preprint arXiv:2402.09353*.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *Preprint*, arXiv:1907.11692.

Sourab Mangrulkar, Sylvain Gugger, Lysandre Debut, Younes Belkada, Sayak Paul, and Benjamin Bossan. 2022. PEFT: State-of-the-art parameter-efficient fine-tuning methods. https://github.com/huggingface/peft.

Brian W Matthews. 1975. Comparison of the predicted and observed secondary structure of t4 phage lysozyme. *Biochimica et Biophysica Acta (BBA)-Protein Structure*, 405(2):442–451.

Linyong Nan, Dragomir Radev, Rui Zhang, Amrit Rau, Abhinand Sivaprasad, Chiachun Hsieh, Xiangru Tang, Aadit Vyas, Neha Verma, Pranav Krishna, Yangxiaokang Liu, Nadia Irwanto, Jessica Pan, Faiaz Rahman, Ahmad Zaidi, Mutethia Mutuma, Yasin Tarabar, Ankit Gupta, Tao Yu, and 5 others. 2021. Dart: Open-domain structured data record to text generation. *Preprint*, arXiv:2007.02871.

Shashi Narayan, Shay B. Cohen, and Mirella Lapata. 2018. Don't give me the details, just the summary! topic-aware convolutional neural networks for extreme summarization. *Preprint*, arXiv:1808.08745.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 311–318, Philadelphia, Pennsylvania, USA. Association for Computational Linguistics.

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2023. Exploring the limits of transfer learning with a unified text-to-text transformer. *Preprint*, arXiv:1910.10683.

Pranav Rajpurkar, Robin Jia, and Percy Liang. 2018. Know what you don't know: Unanswerable questions for squad. *Preprint*, arXiv:1806.03822.

Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. 2022. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10684–10695.

Nataniel Ruiz, Yuanzhen Li, Varun Jampani, Yael Pritch, Michael Rubinstein, and Kfir Aberman. 2023. Dreambooth: Fine tuning text-to-image diffusion models for subject-driven generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*.

Ying Sheng, Shiyi Cao, Dacheng Li, Coleman Hooper, Nicholas Lee, Shuo Yang, Christopher Chou, Banghua Zhu, Lianmin Zheng, Kurt Keutzer, Joseph E. Gonzalez, and Ion Stoica. 2023. S-LoRA: Serving Thousands of Concurrent LoRA Adapters. *arXiv preprint arXiv:2311.03285*.

Shuhua Shi, Shaohan Huang, Minghui Song, Zhoujun Li, Zihan Zhang, Haizhen Huang, Furu Wei, Weiwei Deng, Feng Sun, and Qi Zhang. 2024. ResLoRA: Identity Residual Mapping in Low-Rank Adaption. In *Findings of the Association for Computational Linguistics: ACL 2024*.

Matthew Snover, Bonnie Dorr, Rich Schwartz, Linnea Micciulla, and John Makhoul. 2006. A study of translation edit rate with targeted human annotation. In *Proceedings of the 7th Conference of the Association for Machine Translation in the Americas: Technical Papers*, pages 223–231, Cambridge, Massachusetts, USA. Association for Machine Translation in the Americas.

Tu Vu, Aditya Barua, Brian Lester, Daniel Cer, Mohit Iyyer, and Noah Constant. 2022. Overcoming catastrophic forgetting in zero-shot cross-lingual generation. *Preprint*, arXiv:2205.12647.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. 2019. Glue: A multi-task benchmark and analysis platform for natural language understanding. *Preprint*, arXiv:1804.07461.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, and 3 others. 2020. Huggingface's transformers: State-of-the-art natural language processing. *Preprint*, arXiv:1910.03771.

Yaming Yang, Dilxat Muhtar, Yelong Shen, Yuefeng Zhan, Jianfeng Liu, Yujing Wang, Hao Sun, Denvy Deng, Feng Sun, Qi Zhang, Weizhu Chen, and Yunhai Tong. 2025. Mtl-lora: Low-rank adaptation for multi-task learning. *Preprint*, arXiv:2410.09437.

Shih-Ying Yeh, Yu-Guan Hsieh, Zhidong Gao, Bernard B W Yang, Giyeong Oh, and Yanmin Gong. 2024. Navigating Text-To-Image Customization: From LyCORIS Fine-Tuning to Model Evaluation. *arXiv preprint arXiv:2309.14859*.

Mingyang Zhang, Hao Chen, Chunhua Shen, Zhen Yang, Linlin Ou, Xinyi Yu, and Bohan Zhuang. 2024a. LoRAPrune: Structured Pruning Meets Low-Rank Parameter-Efficient Fine-Tuning. In *Findings of the Association for Computational Linguistics: ACL 2024*.

Qingru Zhang, Minshuo Chen, Alexander Bukharin, Nikos Karampatziakis, Pengcheng He, Yu Cheng, Weizhu Chen, and Tuo Zhao. 2023. AdaLoRA: Adaptive Budget Allocation for Parameter-Efficient Fine-Tuning. *arXiv preprint arXiv:2303.10512*.

Ruiyi Zhang, Rushi Qiang, Sai Ashish Somayajula, and Pengtao Xie. 2024b. AutoLoRA: Automatically Tuning Matrix Ranks in Low-Rank Adaptation Based on

Meta Learning. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*.

Wangbo Zhao, Jiasheng Tang, Yizeng Han, Yibing Song, Kai Wang, Gao Huang, Fan Wang, and Yang You. 2024. Dynamic tuning towards parameter and inference efficiency for vit adaptation. *arXiv preprint arXiv:2403.11808*.

A GAWS-D Adaptation Scheme

Figure 5 illustrates the GAWS-D adaptation scheme. GAWS-D differs from GAWS in how ΔW is constructed: due to the non-commutative nature of the Kronecker product, altering the order of decomposition components leads to a distinct design.

B Experiment Details

B.1 Adapters

Multiple adapters were utilized in the inference latency benchmarking. A description of each adapter is provided below:

- **Low-Rank Adaptation (LoRA)** (Hu et al., 2022) injects small, trainable low-rank matrices into transformer layers. It decomposes the weight update matrix ΔW into two smaller matrices (down projection) and (up projection) of rank r , reducing the number of trainable parameters significantly. The weight update matrix is formulated as: $\Delta W = AB$.

- **Vector-based Random Matrix Adaptation (VeRA)** (Kopiczko et al., 2024) adapts models by freezing a shared pair of randomly initialized low-rank matrices across all layers and learning only small scaling vectors.

- **IA3** (Liu et al., 2022a) is a parameter-efficient adaptation method that applies learned, element-wise scaling vectors to intermediate activations.

- **2D rotary adaptation (RoAd)** (Liao and Monz, 2024) applies learned two-dimensional rotations to layer activations for efficient model adaptation.

B.2 Additional Adapting Methods Considered

We also explored Prefix Tuning. Although it provides substantial latency improvements (closing 92% of the gap), it leads to a notable reduction in accuracy (e.g., SQuAD-v2 EM of 58.77 on LLaMA3.2-3B), exhibiting a trade-off profile similar to IA3 that prioritizes speed at the cost of task performance.

B.3 Models

Below is a description of each model employed in the experiments section:

- **Text-To-Text Transfer Transformer (T5-3B)** (Raffel et al., 2023) is an encoder-decoder transformer model with 3 billion parameters, consisting of 24 encoder and 24 decoder layers and an embedding dimension of 1024. It is distinguished by its use of relative position encodings, which improve the model's handling of input sequences.

- **Large Language Model Meta AI (LLaMA 3.2-3B)** (Grattafiori et al., 2024) is a decoder-only transformer with 3 billion parameters and 28 decoder layers, featuring a larger embedding dimension of 3072. This model incorporates grouped-query attention (GQA) for faster inference, SwiGLU activation functions, and rotary positional embeddings (RoPE) for better positional awareness.

- **Generative Pre-trained Transformer (GPT-2 Large)** (Radford et al., 2019) is a decoder-only transformer with approximately 774 million parameters, 36 layers, and an embedding dimension of 1280. GPT-2 Large relies on a standard attention mechanism without rotary or grouped-query attention and uses learned positional embeddings.

- **Robustly Optimized BERT Approach (RoBERTa-Large)** (Liu et al., 2019) is an encoder-only model with approximately 355 million parameters, 24 layers, and a hidden size of 1024. It uses the standard self attention mechanism and employs learned positional embeddings. RoBERTa-large is trained with a masked language modeling (MLM) objective and removes the next sentence prediction task found in BERT (Devlin et al., 2018), enabling improved performance through dynamic masking.

B.4 Datasets

We carefully selected datasets that align with the strengths and architectural design of each model.

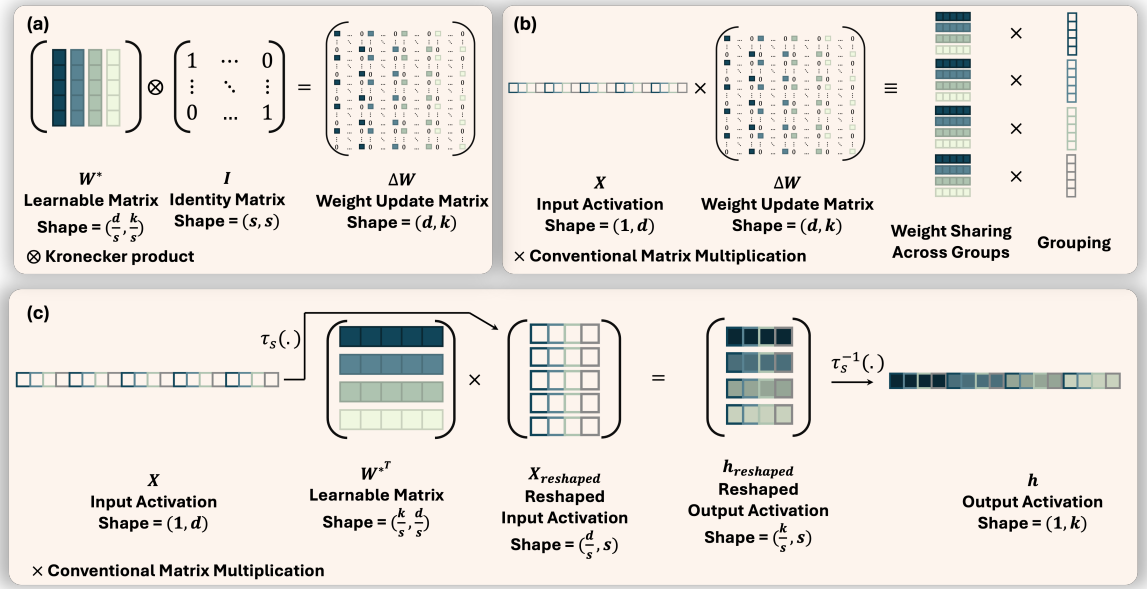


Figure 5: GAWS-D adaptation scheme. (a) The weight update matrix ΔW is formulated as a Kronecker product between a trainable matrix W^* and a fixed structured identity matrix I. (b) This structure enables efficient group-wise weight sharing by transforming the matrix product $X \times \Delta W$ into a group-wise operation, where each input group in X shares the same W^*. (c) The resulting design supports highly efficient computation, as illustrated.

For LLaMA3.2-3B and T5-3B, we utilize XSum and SQuAD v2, which are challenging text generation tasks that require strong contextual understanding and generation capabilities. GPT2-Large, on the other hand, is finetuned on DART and WebNLG, tasks that are less demanding and better suited for capabilites of smaller decoder only models. For RoBERTa-Large, we select three representative tasks from the GLUE benchmark, which are more aligned with encoder only architectures. A brief description of each selected dataset is provided below.

- **Extreme Summarization (XSum)** (Narayan et al., 2018) is designed for abstractive single-document summarization. It consists of BBC news articles from 2010 to 2017 across diverse domains such as News, Politics, Sports, Technology, and more. Each article is paired with a one-sentence summary. Dataset size: Total articles: 226,711, Training set: 204,045, Validation set: 11,332, Test set: 11,334.

- **Stanford Question Answering Dataset (SQuAD v2.0)** (Rajpurkar et al., 2018) is a reading comprehension dataset with questions created by crowdworkers on Wikipedia passages. It contains both answerable and unanswerable questions, where unanswerable questions are crafted to resemble answerable ones.Dataset size: Total questions: 142,192 (including 50,000 unanswerable), Training set: 129,819, Validation set: 500, Test set: 11,873.

- **Web Natural Language Generation (WebNLG)** (Gardent et al., 2017) focuses on generating text from structured data. It consists of sets of 1 to 7 resource description framework (RDF) triples (subject-predicate-object) from DBpedia and corresponding natural language verbalizations. The test set spans 15 domains, with 5 domains unseen during training. Dataset size: Training set: 18,025, Validation set: 2,258, Test set: 4,928.

- **DAta Record to Text generation (DART)** (Nan et al., 2021) contains RDF entity-relation triples annotated with sentence-level descriptions that comprehensively cover the facts. It aggregates data from multiple existing datasets including WikiTableQuestions, WikiSQL, WebNLG, and Cleaned E2E. Dataset size: Training set: 62,659, Validation set: 6,980, Test set: 12,552.

- **The General Language Understanding Evaluation Benchmark(GLUE)** (Wang

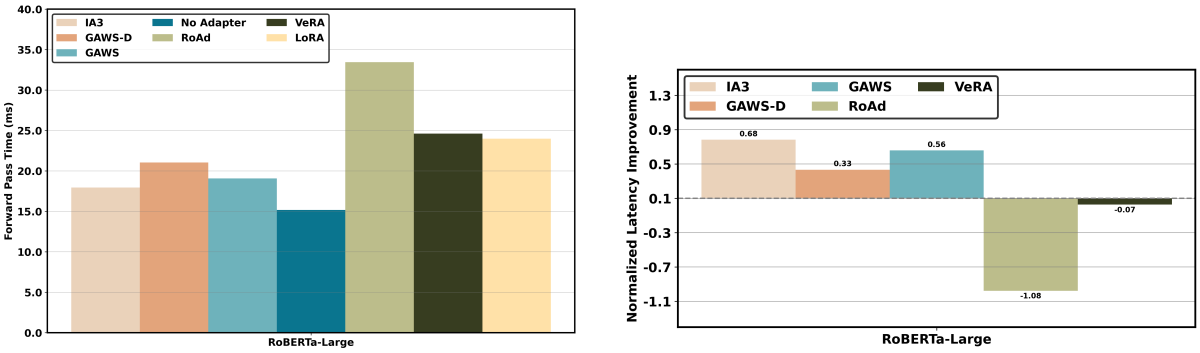


Figure 6: Inference latency (ms) for RoBERTa-Large across various adapter methods. **(Left)** absolute latency, **(Right)** normalized latency, computed as $(\text{latency}_{\text{LoRA}} - \text{latency}_{\text{adapter}})/(\text{latency}_{\text{LoRA}} - \text{latency}_{\text{no-adapter}})$. Measurements were taken using an input sequence length of 1024 tokens, with batch size = 1, under standard Pytorch eager mode CUDA execution. GAWS and GAWS-D consistently achieve the lowest latency among all evaluated adapters, demonstrating their efficiency for real-time deployment.

et al., 2019) is a collection of natural language understanding tasks designed to evaluate a model's performance across a broad range of linguistic capabilities. It focuses on three major task categories: (1) natural language inference, (2) similarity and paraphrase detection, (3) single-sentence classification tasks such as sentiment analysis and linguistic acceptability. GLUE consists of nine tasks in total, from which we selected three representative tasks, each corresponding to one of the three major categories. Table 5 presents the selected datasets along with their descriptions and dataset sizes.

Dataset	Description	Train	Val	Test
MRPC	Paraphrase detection dataset collected from news articles.	3.7k	408	1.7k
CoLA	Acceptability judgment dataset from linguistics publications.	8.5k	1k	1k
RTE	Inference dataset assessing whether a hypothesis follows from a premise.	2.5k	278	3k

Table 5: Selected GLUE benchmark tasks with descriptions and dataset sizes.

B.5 Metrics

In our experiments, we employed a range of task-specific evaluation metrics, meticulously chosen to align with the unique characteristics and objectives of each task. For example, Exact Match and F1 score are used for SQuAD-v2 to evaluate both precise answer correctness and partial overlap. Rouge is applied to XSum to assess summarization quality through n-gram overlap. BLEU and TER measure generation quality for DART and WebNLG, capturing both accuracy and edit distance. Matthews Correlation Coefficient is used for CoLA to handle imbalanced binary classification, while Accuracy serves as a straightforward performance measure for RTE and MRPC. Below, we provide a brief description of each metric.

- **Exact Match (EM)** is a strict metric that assigns a score of 1 if the predicted answer exactly matches the ground truth answer; otherwise, it assigns 0. Definition:

$$\text{EM} = \begin{cases} 1, & \text{if predicted = ground truth} \\ 0, & \text{otherwise} \end{cases} \tag{15}$$

- **F1 Score** evaluates the average overlap between the predicted and ground truth answers at the token level, balancing precision and recall. Definition:

$$\text{F1} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{16}$$

where

$$\text{Precision} = \frac{\text{Correctly predicted tokens}}{\text{Total predicted tokens}} \tag{17}$$

$$\text{Recall} = \frac{\text{Correctly predicted tokens}}{\text{Total tokens in ground truth}} \tag{18}$$

Model	Finetuning	SQuAD-v2				XSum			
		BS	LR	GAS	Epochs	BS	LR	GAS	Epochs
LLaMA3.2-3B	FFT	1	2.00E-05	64	3	1	4.00E-05	32	3
	$\text{LoRA}_{r=32}$	4	4.00E-04	16	2	4	8.00E-04	16	3
	IA3	4	1.00E-03	16	2	4	1.00E-03	16	3
	$\text{VeRA}_{r=32}$	4	1.00E-03	16	2	4	1.00E-03	16	3
	$\text{RoAd}_{v=4}$	4	1.00E-03	16	2	4	1.00E-03	16	3
	$\text{GAWS}_{s=6}$	1	1.00E-04	16	2	2	1.00E-04	16	3
	$\text{GAWS}^{(d)}_{s=6}$	2	4.00E-04	32	2	1	4.00E-04	16	3
	$\text{GAWS-D}_{s=6}$	1	1.00E-04	16	2	2	1.00E-04	16	3
	$\text{GAWS-D}^{(d)}_{s=6}$	1	4.00E-04	16	2	2	8.00E-04	16	3
T5-3B	FFT	1	4.00E-05	16	6	1	4.00E-05	16	6
	$\text{LoRA}_{r=16}$	4	4.00E-04	4	3	4	4.00E-04	4	3
	IA3	4	1.00E-03	4	3	4	1.00E-03	4	3
	$\text{VeRA}_{r=16}$	4	1.00E-03	4	3	4	1.00E-03	4	3
	$\text{RoAd}_{v=4}$	4	1.00E-03	4	3	4	1.00E-03	4	3
	$\text{GAWS}_{s=8}$	4	1.00E-03	4	3	4	1.00E-03	4	3
	$\text{GAWS}^{(d)}_{s=8}$	4	1.00E-03	4	3	4	1.00E-03	4	3
	$\text{GAWS-D}_{s=8}$	4	1.00E-03	4	3	4	1.00E-03	4	3
	$\text{GAWS-D}^{(d)}_{s=8}$	4	1.00E-03	4	3	4	1.00E-03	4	3

Table 6: LLaMA3.2-3B and T5-3B hyperparameter values utilized in SQuAD-v2 and XSum experiments. BS indicates batch size, LR indicates learning rate, and GAS indicates gradient accumulation steps.

Model	WebNLG							DART						
	Finetuning	BS	LR	GAS	Weight decay	Label smooth	Epochs	Finetuning	BS	LR	GAS	Weight decay	Label smooth	Epochs
GPT2-Large	FFT	6	5.00E-05	1	0.01	0.1	5	FFT	10	1.00E-05	1	0	0	10
	$\text{LoRA}_{r=32}$	8	2.00E-04	1	0.01	0.1	5	$\text{LoRA}_{r=16}$	8	2.00E-04	1	0	0	5
	$\text{VeRA}_{r=32}$	4	1.00E-03	2	0.01	0.1	5	$\text{VeRA}_{r=16}$	4	1.00E-03	2	0	0	5
	IA3	4	1.00E-03	2	0.01	0.1	5	IA3	4	1.00E-03	2	0	0	5
	$\text{RoAd}_{v=4}$	4	1.00E-03	2	0.01	0.1	5	$\text{RoAd}_{v=4}$	4	1.00E-03	2	0	0	5
	$\text{GAWS}_{s=5}$	4	4.00E-04	2	0.01	0.1	5	$\text{GAWS}_{s=10}$	4	4.00E-04	2	0	0	5
	$\text{GAWS}^{(d)}_{s=5}$	4	4.00E-04	2	0.01	0.1	5	$\text{GAWS}^{(d)}_{s=10}$	4	4.00E-04	2	0	0	5
	$\text{GAWS-D}_{s=5}$	4	4.00E-04	2	0.01	0.1	5	$\text{GAWS-D}_{s=10}$	4	4.00E-04	2	0	0	5
	$\text{GAWS-D}^{(d)}_{s=5}$	4	4.00E-04	2	0.01	0.1	5	$\text{GAWS-D}^{(d)}_{s=10}$	4	8.00E-04	2	0	0	5

Table 7: GPT2-Large hyperparameter values utilized in WebNLG and DART experiments. BS indicates batch size, LR indicates learning rate, and GAS indicates gradient accumulation steps.

- **BLEU (Bilingual Evaluation Understudy)** (Papineni et al., 2002) measures the quality of machine-generated text by comparing overlapping n-grams with one or more reference texts. It ranges from 0 to 100, with higher scores indicating better quality. BLEU incorporates a brevity penalty to penalize short outputs. Definition:

$$\text{BLEU} = \text{BP} \times \exp\left(\sum_{n=1}^{N} w_n \log p_n\right) \quad (19)$$

  Where:

  - p_n: Precision of n-grams (overlap between generated and reference n-grams)
  - w_n: Weight for each n-gram order (typically uniform)
  - BP: Brevity Penalty

- **Translation Error Rate (TER)** (Snover et al., 2006) measures the minimum number of edits (insertions, deletions, substitutions, shifts) needed to change a machine-generated text into a reference text. Lower TER indicates better quality. Definition:

$$\text{TER} = \frac{\text{Number of edits}}{\text{Total reference words}} \quad (20)$$

- **ROUGE (Recall-Oriented Understudy for Gisting Evaluation)** (Lin, 2004) evaluates summarization quality by measuring the overlap of n-grams or subsequences between the generated summary and reference summaries, focusing on recall.It ranges from 0 to 100, with higher scores indicating better quality. Definition:

$$\text{ROUGE} = \frac{\text{Number of overlapping n-grams}}{\text{Total n-grams in reference}} \quad (21)$$

  Rouge score variants:

  - ROUGE-N: Overlap of n-grams of length n.

Model	CoLA				MRPC				RTE			
	Finetuning	BS	LR	Epochs	Finetuning	BS	LR	Epochs	Finetuning	BS	LR	Epochs
RoBERTa-Large	$\text{LoRA}_{r=8}$	4	2.00E-04	20	$\text{LoRA}_{r=8}$	4	1.00E-04	20	$\text{LoRA}_{r=8}$	4	2.00E-04	20
	IA3	4	1.00E-03	20	IA3	8	1.00E-03	20	IA3	4	1.00E-03	20
	$\text{VeRA}_{r=8}$	4	1.00E-03	20	$\text{VeRA}_{r=8}$	8	1.00E-03	20	$\text{VeRA}_{r=8}$	4	1.00E-03	20
	$\text{RoAd}_{v=4}$	4	1.00E-03	20	$\text{RoAd}_{v=4}$	8	1.00E-03	20	$\text{RoAd}_{v=4}$	4	1.00E-03	20
	$\text{GAWS}_{s=8}$	10	4.00E-04	20	$\text{GAWS}_{s=8}$	4	3.00E-04	20	$\text{GAWS}_{s=8}$	4	4.00E-04	20
	$\text{GAWS}^{(d)}_{s=8}$	8	4.00E-04	20	$\text{GAWS}^{(d)}_{s=8}$	4	6.00E-04	20	$\text{GAWS}^{(d)}_{s=8}$	4	5.00E-04	20
	$\text{GAWS-D}_{s=8}$	8	4.00E-04	20	$\text{GAWS-D}_{s=8}$	4	4.00E-04	20	$\text{GAWS-D}_{s=8}$	8	6.00E-04	20
	$\text{GAWS-D}^{(d)}_{s=8}$	4	7.00E-04	20	$\text{GAWS-D}^{(d)}_{s=8}$	4	6.00E-04	20	$\text{GAWS-D}^{(d)}_{s=8}$	4	9.00E-04	20

Table 8: RoBERTa-Large hyperparameter values utilized in CoLA, MRPC and RTE experiments. BS indicates batch size, and LR indicates learning rate.

  - ROUGE-L: Measures the longest common subsequence (LCS) between candidate and reference texts.

- **Matthews Correlation Coefficient (MCC)** (Matthews, 1975) is a balanced measure for evaluating binary classification tasks, especially useful when classes are imbalanced. It takes into account true and false positives and negatives and returns a value between -1 and 1, where 1 indicates perfect prediction, 0 random prediction, and -1 total disagreement. Definition:

$$\text{MCC} = \frac{TP \cdot TN - FP \cdot FN}{\sqrt{(TP+FP)(TP+FN)}} \times \frac{1}{\sqrt{(TN+FP)(TN+FN)}} \tag{22}$$

  where TP = True Positives, TN = True Negatives, FP = False Positives, FN = False Negatives.

- **Accuracy** measures the proportion of correctly predicted instances over the total number of instances. It is simple and effective when the class distribution is balanced. The score ranges from 0 to 1, with higher values indicating better performance. Definition:

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}} \tag{23}$$

B.6 Hyperparameters

Tables 6, 7, and 8 present the hyperparameters used in the experiments. A grid search was performed to identify the optimal settings. The search ranges were as follows: batch size from 1 to 10 (incremented by 1), learning rate from 1e-4 to 1e-3 (incremented by 1e-4), and gradient accumulation steps from 1 to 32 (incremented by 2). All baseline adapters are initialized following their original implementation, while GAWS and GAWS-D are zero-initialized to avoid disrupting pretrained weights. Adapters are integrated into the key, query, and value projections of the attention mechanism for all models, except for RoBERTa, where they are applied only to the query and value weights.

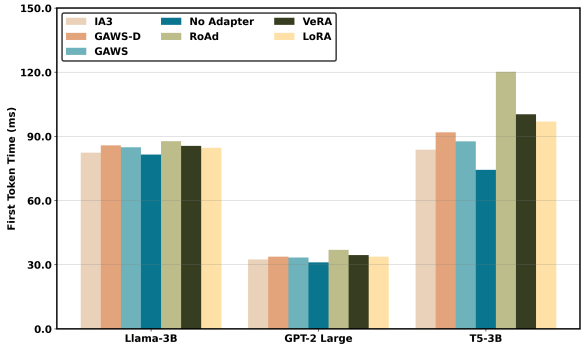


Figure 7: Prompt processing latency(ms) for LLaMA3.2-3B, GPT2-Large, T5-3B across various adapter methods. Measurements were taken using an input sequence length of 1024 tokens, with batch size = 1, under standard Pytorch eager mode CUDA execution. GAWS and GAWS-D consistently achieve the lowest latency among all evaluated adapters, demonstrating their efficiency for real-time deployment.

B.7 Inference Latency Benchmarking

This section provides additional details related to the latency evaluations presented in the main paper.

As shown in Figure 1, unmerged LoRA introduces latency overhead due to segmented CUDA kernel calls arising from sequential matrix multiplications. Table 9 offers further implementation details for the latency experiments in Figure 1. For each adapter, it lists the adapted modules, hyperparameters, and the total number of adapter parameters.

For the generative models used in our

Model	Adapter	Adapted Modules	Adapter Parameters	Time per Token (ms/token)
LLaMA3.2-1B	No Adapter	-	-	16.04
	LoRA$_{r=16}$	Q K V	2.36M	22.02
	GAWS$_{s=8}$	Q K V	3.15M	19.45
	GAWS-D$_{s=8}$	Q K V	3.15M	22.02
LLaMA3.2-3B	No Adapter	-	-	25.98
	LoRA$_{r=25}$	Q K V	10.04M	36.10
	GAWS$_{s=6}$	Q K V	10.09M	31.77
	GAWS-D$_{s=6}$	Q K V	10.09M	33.22
LLaMA3.1-8B	No Adapter	-	-	33.91
	LoRA$_{r=21}$	Q K V	12.39M	45.59
	GAWS$_{s=8}$	Q K V	12.58M	39.99
	GAWS-D$_{s=8}$	Q K V	12.58M	40.39

Table 9: Inference latency (ms/token) measured on LLaMA models of three sizes (1B, 3B, and 8B) under four config- urations: no-adapter, unmerged versions of LoRA, GAWS, and GAWS-D. The table summarizes the implementation details corresponding to Figure 1.

Model	Adapter	Adapted Modules	Adapter Parameters	First Token Time(ms)	Time per Token (ms/token)
LLaMA3.2-3B	No Adapter	-	-	81.50	25.98
	LoRA$_{r=25}$	Q K V	10.04M	84.62	36.10
	IA3	Q K V	143K	82.41	31.18
	VeRA$_{r=25}$	Q K V	145K	85.57	37.83
	RoAd$_{v=4}$	Q K V	9.63M	87.71	51.71
	GAWS$_{s=6}$	Q K V	10.09M	84.87	31.77
	GAWS-D$_{s=6}$	Q K V	10.09M	85.81	33.22
GPT2-Large	No Adapter	-	-	31.06	20.06
	LoRA$_{r=32}$	c_{attn}	5.90M	33.76	25.44
	IA3	c_{attn}	138K	32.44	22.46
	VeRA$_{r=32}$	c_{attn}	139K	34.48	25.87
	RoAd$_{v=4}$	c_{attn}	553K	36.90	30.73
	GAWS$_{s=5}$	c_{attn}	7.08M	33.33	23.85
	GAWS-D$_{s=5}$	c_{attn}	7.08M	33.72	24.08
T5-3B	No Adapter	-	-	74.35	27.98
	LoRA$_{r=12}$	Q K V	13.27M	96.90	40.58
	IA3	Q K V	884K	83.74	33.67
	VeRA$_{r=12}$	Q K V	887K	100.31	41.61
	RoAd$_{v=4}$	Q K V	3.54M	120.17	54.65
	GAWS$_{s=8}$	Q K V	14.16M	87.64	35.37
	GAWS-D$_{s=8}$	Q K V	14.16M	91.89	36.88

Table 10: Inference latency (ms/token) for T5-3B, LLaMA3.2-3B, and GPT-Large across various adapter methods. The table summarizes the implementation details corresponding to Figures 3 and 7. The subscript v in the RoAd adapter denotes the variant.

Model	Adapter	Adapted Modules	Adapter Parameters	Processing Time(ms)
RoBERTa-Large	No Adapter	-	-	15.18
	LoRA$_{r=32}$	Q K V	5.77M	23.97
	IA3	Q K V	1.13M K	17.95
	VeRA$_{r=32}$	Q K V	1.13M	24.60
	RoAd$_{v=4}$	Q K V	1.35M	33.44
	GAWS$_{s=4}$	Q K V	5.77M	19.06
	GAWS-D$_{s=4}$	Q K V	5.77M	21.03

Table 11: Inference latency (ms) for RoBERTa-Large across various adapter methods. The table summarizes the implementation details corresponding to Figure 6. The subscript v in the RoAd adapter denotes the variant.

experiments(GPT2-Large, LLaMA3.2-3B, and T5-3B)latency is divided into two stages: prompt processing and token generation. Figure 7 shows prompt processing latency across baseline adapters and our proposed adapter methods. This complements Figure 3, which reports latency per generated token. Table 10 provides implementation specifics for the experiments shown in Figures 3 and 7, including the adapted modules, adapter configurations, and parameter counts.

For RoBERTa-Large, which is an encoder-only model, Figure 6 presents processing latency across all adapters. Table 11 provides corresponding implementation details for this figure.

C Impact of Grouping Factor

We conducted ablation studies to observe the effect of varying the grouping factor s on both latency and task accuracy. While inference latency remains uniformly stable across different s values due to GPU parallelization hiding the FLOP variations, downstream task performance exhibits no universal behavior. As shown in Table 12, whether performance saturates or scales with the parameter count is highly dependent on the specific model architecture and task. Consequently, s should be treated as a tunable hyperparameter based on the target deployment.

Model	Task	s	Params	Metric 1	Metric 2
GPT2-Large	WebNLG	5	7.08M	56.38	0.39
		10	1.77M	56.60	0.39
	DART	5	7.08M	47.45	0.48
		10	1.77M	48.19	0.47
Llama3.2-3B	SQuAD-v2	6	10.09M	76.05	83.21
		8	4.82M	74.42	81.51
	XSum	6	10.09M	42.59	34.31
		8	4.82M	42.88	34.67
Llama3.2-3B (Query-only)	SQuAD-v2	6	7.34M	75.36	82.72
		12	1.84M	73.50	80.89
		24	459K	70.16	77.53
T5-3B	SQuAD-v2	8	14.16M	79.05	85.98
		16	3.54M	79.00	85.88
	XSum	8	14.16M	44.16	35.89
		16	3.54M	43.30	35.01

Table 12: Performance across models and tasks under varying grouping factors s. Metric columns Metric1/Metric2 correspond to BLEU/TER for DART and WebNLG, EM/F1 for SQuAD-v2, and ROUGE-1/ROUGE-L for XSum. Adapter placement follows the configuration described in Appendix B.6; the "Query-only" experiment applies adapters exclusively to the query matrix.