mathematics

MDPI

Article

Eliminating Packing-Aware Masking via LoRA-Based Supervised Fine-Tuning of Large Language Models

Jeong Woo Seo and Ho-Young Jung *

Department of Artificial Intelligence, Kyungpook National University, Daegu 41566, Republic of Korea; sjw6328@gmail.com

* Correspondence: hoyjung@knu.ac.kr

Abstract

Packing approaches enhance training efficiency by filling the padding space in each batch with shorter sequences, thereby reducing the total number of batches per epoch. This approach has proven effective in both pre-training and supervised fine-tuning of large language models (LLMs). However, most packing methods necessitate a packing-aware masking (PAM) mechanism to prevent cross-contamination between different text segments in the multi-head attention (MHA) layers. This masking ensures that the scaled dot-product attention operates only within segment boundaries. Despite its functional utility, PAM introduces significant implementation complexity and computational overhead during training. In this paper, we propose a novel method that eliminates the need for PAM during supervised fine-tuning with packing. Instead of masking, we introduce a learnable tensor derived from Low-Rank Adaptation (LoRA) with the query and value parameters of the attention mechanism. This tensor is trained to attenuate the subspace corresponding to cross-contamination, effectively replacing the function of PAM. Through component-wise decomposition of attention head outputs, we isolate the contamination component and demonstrate that it can be attenuated using the LoRA-derived tensor. Empirical evaluations on 7B-scale LLMs show that our method reduces training time and runtime overhead by completely removing the implementation associated with PAM. This enables more scalable and efficient supervised fine-tuning with packing, without compromising model integrity.

Keywords: training efficiency; supervised fine-tuning; large language model; packing; training-time overhead; implementation complexity

MSC: 68T50

check for updates

Academic Editors: Hwei Jen Lin and Ching-Ting Tu

Received: 4 September 2025
Revised: 8 October 2025
Accepted: 16 October 2025
Published: 20 October 2025

Citation: Seo, J.W.; Jung, H.-Y. Eliminating Packing-Aware Masking via LoRA-Based Supervised Fine-Tuning of Large Language Models. *Mathematics* **2025**, *13*, 3344. https://doi.org/10.3390/math13203344

Copyright: © 2025 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

Transformer-based architectures [1] have become the backbone of numerous pre-trained models and large language models (LLMs) [2–6], demonstrating state-of-the-art performance across a wide range of natural language processing tasks. These models are typically pre-trained on large-scale datasets and subsequently fine-tuned using supervised fine-tuning (SFT) for specific downstream applications. However, in practical scenarios, SFT is often constrained by limited computational resources, thereby motivating the need for more efficient training strategies.

One widely adopted approach to improve training efficiency is packing [7], which effectively utilizes the padding space within batches by replacing unused tokens with shortened sequences from the dataset. Packing has been successfully applied in both the

pre-training [8–10] and SFT [11–16], significantly increasing GPU utilization and reducing the number of batches per epoch and training time. Existing packing strategies generally fall into two categories: offline [7,17] or online [18] packing. Offline approaches pre-pack sequences prior to training, while online approaches perform packing dynamically during training. Despite their efficiency gains, both approaches need packing-aware masking (PAM).

This PAM departs from traditional key-padding masks and causal (diagonal) masks in both structure and semantics. These traditional masks are often provided as two-dimensional tensors and are broadcast uniformly across all matrices in a three-dimensional attention tensor along a fixed axis. This operation minimizes both generation and transmission overhead. For example, in PyTorch-based training scenarios, the mask tensor's data type (i.e., dtype) and device can be set once at the model level and then reused across attention layers; such reuse incurs minimal casting overhead because the casted mask is forwarded rather than recomputed. In contrast, PAM, whether in offline or online packing, requires distinct block-diagonal masks for each matrix within a batch tensor. Generating the mask tensor and applying it in scaled dot-product attention needs time and has implementation costs [7]. Since the mask tensor is inherently three-dimensional ($\mathbb{R}^{B\times N\times N}$), constructing these masks on the CPU and stacking or transferring them to GPU devices at every training step introduces a significant runtime overhead. The cost further increases if the mask uses an integer dtype (e.g., long) instead of the supported boolean or floating-point formats, thereby triggering implicit casts. Additional overhead arises when unnecessary operations such as '`clone()`' or '`contiguous()`' are repeatedly applied.

Unlike two-dimensional masks that can often be precomputed and reused, PAM entails per-matrix mask construction, which raises casting and data-movement overheads. Moreover, PyTorch's native scaled dot-product attention API automatically dispatches among FlashAttention-2, memory-efficient, and math backends based on the inputs; supplying an explicit three-dimensional attention mask generally precludes the fused Flash/memory-efficient paths unless masking is expressed via supported flags (e.g., causal), leading to fallback to a slower backend with reduced computational efficiency. These constraints also increase the implementation complexity of masking in training code. In sum, three-dimensional PAM entails more considerations than two-dimensional masking and can lead to training speed degradation, especially when per-matrix mask construction and backend fallback are involved.

To mitigate the overheads associated with PAM, several prior works have proposed alternatives, such as packing with applying Flash attention [18] and boundary-ware positional encoding [19]. While effective in controlled settings, these approaches are often non-trivial to integrate into standard SFT pipelines and frequently require custom implementations or backend overrides. Even when mask generation is optimized, runtime overheads may persist due to device-specific behaviors or library/package level leakage, depending on hardware and software configurations.

Meanwhile, LoRA (Low-Rank Adaptation) [20] has emerged as a widely adopted parameter-efficient fine-tuning method for large-scale pre-trained language models. Rather than updating all model parameters, LoRA introduces low-rank matrices into attention and feed-forward layers, significantly reducing both memory consumption and computational cost, while maintaining or even surpassing the performance of full fine-tuning. Variants such as Delta-LoRA [21], NoRA [22], and DoRA [23] further improve upon this framework by enhancing stability, inference efficiency, and parameter reduction. In practice, most SFT scenarios focus on adapting only the query and value projections, as this achieves strong performance with minimal adaptation overhead.

Recently, LoRA has been combined with sequence packing during SFT to further improve training efficiency. However, PAM remains a critical bottleneck in this setup due to its masking requirements. To address this, we propose a novel approach that eliminates the need for PAM during training, while maintaining the benefits of both packing and LoRA. Specifically, we begin by analyzing and formalizing the subspaces of tensors in the scaled dot-product attention mechanism under two conditions: when PAM is omitted, allowing potential cross-contamination between packed sequences, and when LoRA is applied to query and value projections (denoted as LoRA-Q and LoRA-V). Based on this analysis, we demonstrate that the learnable subspace induced by LoRA-Q and LoRA-V can naturally suppress or attenuate the contamination subspace that arises from removing PAM. This insight enables us to construct an efficient SFT framework that forgoes PAM entirely, thereby eliminating its implementation and runtime costs. The proposed approach erases both the implementation complexity and factors degrading training speed fundamentally. Furthermore, this combination guarantees not only simple and fast implementation but also decreases the packing and training time for SFT with packing and LoRA. We validate our method on multiple pre-trained models and SFT datasets available via Hugging Face, demonstrating its practical utility, scalability, and training-time improvements across diverse settings.

This paper is organized as follows: Section 2 reviews the scaled dot-product stage attention in Transformer with packing and LoRA. Section 3 outlines the possibility to replace cross-contamination to the LoRA space, without packing-aware masking. The pre-trained model and datasets for supervised fine-tuning, hyperparameters, and devices for the experiments are described in Section 4. The results and discussion for each experiment are presented with figures and tables. Finally, Section 5 summarizes the contributions of our approach.

2. Background and Related Work

2.1. Preliminaries

The operations in the scaled dot-product stage attention (SDPA) in traditional multi-head attention layer (MHA) in Transformer [1] are represented as follows:

$$q_i = W_{Qi}^T x \,, \tag{1}$$

$$k_i = W_{Ki}^T x \,, \tag{2}$$

$$v_i = W_{Vi}^T x \,, \tag{3}$$

$$S_{i0} = q_i^T k_i / \sqrt{d_k} \,, \tag{4}$$

$$\mathcal{A}_{i0} = \texttt{softmax}(S_{i0}) \,, \tag{5}$$

$$\mathcal{H}_{i0} = \mathcal{A}_{i0} v_i^T \,, \tag{6}$$

where $x \in \mathbb{R}^{d \times N}$ and $W_{Qi}, W_{Ki}, W_{Vi} \in \mathbb{R}^{d \times d_k}$. They denote, respectively, the embedded input tensor and the learnable weights for i-th head in MHA. d_k is equal to d/h. N, d, and h are the sequence length, model dimension, and the number of heads, respectively. $\mathcal{H}_{i0}$ in each head are concatenated and projected by a learnable output matrix. The result is added to the input x via a residual connection, followed by layer normalization.

2.2. SDPA with Packing

If packing is applied, a packing-aware mask is generated based on how the sequences are packed in the batch and added into the dot-product tensor [7] in the SDPA stage. The masking matrix consists of zero and negative infinite values with a block-diagonal pattern.

$$\begin{aligned} &S_{i\mathcal{M}} = q_i^T k_i / \sqrt{d_k} + \mathcal{M}_i \ , \\ &\textit{where } \mathcal{M} \in \{0, -\infty\}^{n \times n} \text{ with a block-diagonal structure .} \end{aligned} \tag{7}$$

Negative elements lead the attention scores between queries and keys from different sequences to approach zero after softmax, while ignoring cross-contamination values.

$$\frac{e^{-\infty}}{\sum_{m,n=1}^{N} e^{S_{i\mathcal{M}\{m,n\}}}} \approx 0 \ . \tag{8}$$

Therefore, the cross-contamination area cannot affect to head output.

In practice, masking is employed by adding large negative constants instead of $-\infty$ to designated positions, thereby forcing their probabilities to near zero and preventing numerical overflow or NaN values. This results in masked positions producing small but nonz-ero values after the softmax operation. However, these residual values contribute to the denominator in the normalization step, which increases the sensitivity of the probability distribution to differences among the scores of valid tokens [24]. Furthermore, as the size of the masked area grows and the number of valid tokens decreases, the same score differences yield proportionally larger changes in the resulting probability distribution, amplifying instability in attention weights [25].

2.3. SDPA with LoRA

The weights for query and key adaptation with LoRA are, respectively, added to the original weights ($W'_{Qi} = W_{Qi} + \Delta W_{Qi}$, $W'_{Ki} = W_{Ki} + \Delta W_{Ki}$). ΔW_{Qi} and ΔW_{Ki} are decomposed to learnable low-rank weights $B_{Qi,Ki}$ and $A_{Qi,Ki}$. Hence, S_i is expressed as follows:

$$S_i = ((W_{Qi} + \Delta W_{Qi})^T x)^T (W_{Ki} + \Delta W_{Ki})^T x / \sqrt{d_k} \tag{9}$$

$$= x^T W_{Qi} W_{Ki}^T x / \sqrt{d_k} + (x^T W_{Qi} \Delta W_{Ki}^T x + x^T \Delta W_{Qi} W_{Ki}^T x + x^T \Delta W_{Qi} \Delta W_{Ki}^T x) / \sqrt{d_k} \tag{10}$$

$$\equiv S_{io} + \Delta S_i \ , \tag{11}$$

where S_{io} and ΔS_i are equal to the first term of expansion of S_i and the sum of the others, respectively. Therefore, $\mathcal{H}_i$ is described as follows:

$$\mathcal{H}_i = \mathcal{A}_i v_i^T = \texttt{softmax}(S_{i0} + \Delta S_i) v_i^T \not\equiv \mathcal{H}_{i0} + \Delta \mathcal{H}_i \ . \tag{12}$$

Since softmax is not the linear function, $\mathcal{H}_i$ cannot be directly defined to $\mathcal{H}_{i0} + \Delta \mathcal{H}_i$.

In the case of only adding learnable parameters to value weight ($W'_{Vi} = W_{Vi} + \Delta W_{Vi}$), $\mathcal{H}_i$ can be decomposed to sum of head values and each term is defined as follows:

$$\mathcal{H}_i = \mathcal{A}_{i0}(x W_{Vi}^T + x \Delta W_{Vi}^T) \equiv \mathcal{A}_{i0} v_i^T + \mathcal{A}_{i0} \Delta v_i^T \equiv \mathcal{H}_{i0} + \Delta \mathcal{H}_i \ . \tag{13}$$

$\mathcal{H}_{i0}$ is equal to the head output tensor from pre-trained parameters (Equation (6)) and constant term for trainable term $\Delta \mathcal{H}_i$. $\Delta \mathcal{H}_i$ directly influences attention-weighted output and shifts the output corresponding to the fine-tuning dataset.

In LoRA experiments, adapting the ΔW_{Qi} and ΔW_{Vi} (i.e., LoRA-Q and LoRA-V) achieves performance comparable to full fine-tuning [20,26,27]. Therefore, the terms with ΔW_{Ki} in ΔS_i in Equations (10) and (11) become zero. Consequently, $\mathcal{H}_i$ applied LoRA-Q and LoRA-V can be reformulated as follows:

$$\mathcal{H}_i = \texttt{softmax}(S_{i0} + \Delta S_i)(v_{i0} + \Delta v_i) \tag{14}$$

$$= \texttt{softmax}(S_{i0} + \Delta S_i)xW_{Vi}^T + \texttt{softmax}(S_{i0} + \Delta S_i)x\Delta W_{Vi}^T \,, \tag{15}$$

$$\textit{where } \Delta S_i = x^T \Delta W_{Qi} W_{Ki}^T x \equiv (\Delta q_i^T) k_i / \sqrt{d_k} \,. \tag{16}$$

3. Methodology

3.1. Decomposing SDPA Output Without PAM

Regardless of whether the input x is packed, any element of $\mathcal{H}_i$ can be written as sum of products of attention tensor $\mathcal{A}_i$ and value entry v_i:

$$\mathcal{H}_{i\{m,p\}} = \sum_{n=1}^{N} \mathcal{A}_{i\{m,n\}} v_{i\{n,p\}}^T \,, \tag{17}$$

where $m \in [1, N]$ and $p \in [1, d_k]$. If the input x is a packed sequence with two segments and boundary is t-th token of x, Equation (17) can be decomposed as follows:

$$\mathcal{H}_{i\{m,p\}} = \sum_{n=1}^{t-1} \mathcal{A}_{i\{m,n\}} v_{i\{n,p\}}^T + \sum_{n=t}^{N} \mathcal{A}_{i\{m,n\}} v_{i\{n,p\}}^T \,. \tag{18}$$

For the elements of the first segment $\mathcal{H}_{i\{m<t,p\}}$, the first term in Equation (18) is the valid attention values and the second is the cross-contamination term. Conversely, the opposite holds for the elements of the second segment $\mathcal{H}_{i\{m\geq t,p\}}$. Without PAM (i.e., allowing cross-contamination), $\mathcal{H}_i$ with packing can be decomposed and defined as follows:

$$\mathcal{H}_i \equiv \mathcal{H}_{i0} + \mathcal{H}_{iC} \,, \tag{19}$$

where each term, respectively, corresponds to the valid attention and cross-contamination tensor.

3.2. Eliminating PAM via LoRA

Building on Equations (13) and (19), the PAM to avoid cross-contamination are not necessary if $\Delta\mathcal{H}_i$ can cover up $\mathcal{H}_{iC}$. Rather, the space of $\mathcal{H}_{iC}$ can be reused and updated as a space for SFT. In the case of adapting only LoRA-V for fine-tuning while allowing cross-contamination ($\Delta S_i = 0, \Delta\mathcal{H}_i \neq 0$), $\mathcal{H}_i$ can be reformulated as follows:

$$\mathcal{H}_{i\{m,p\}} = \sum_{n=1}^{N} \mathcal{A}_{i\{m,n\}} v_{i\{n,p\}}^T + \sum_{n=1}^{N} \mathcal{A}_{i\{m,n\}} \Delta v_{i\{n,p\}}^T \tag{20}$$

$$= \sum_{n=1}^{t-1} \mathcal{A}_{i\{m,n\}} v_{i\{n,p\}}^T + \sum_{n=t}^{N} \mathcal{A}_{i\{m,n\}} v_{i\{n,p\}}^T + \sum_{n=1}^{N} \mathcal{A}_{i\{m,n\}} \Delta v_{i\{n,p\}}^T \tag{21}$$

$$\equiv \mathcal{H}_{i0\{m,p\}} + \mathcal{H}_{iC\{m,p\}} + \Delta\mathcal{H}_{i\{m,p\}} \,. \tag{22}$$

Since $\mathcal{H}_{iC}$ is obtained from pre-trained parameters, it is a constant term for trainable $\Delta\mathcal{H}_i$. Therefore, B_{Vi} and A_{Vi} train to attenuate the cross-contamination while adapting to the SFT dataset. Namely, we can overlap $\mathcal{H}_{iC}$ while training LoRA-V and removing the PAM stage (Equation (7)) during fine-tuning with packing.

Similar to Equation (19), the locations of each valid attention and cross-contamination regions are unchanged by softmax, while only the values are rescaled. Therefore, the scaled dot-product before softmax with packing (S_i) can be decomposed by subspaces of valid attention and cross-contamination as follows:

$$S_i = S_{i0} + S_{iC} \,, \tag{23}$$

where S_{iC} is the cross-contamination term. As explained in Equation (12), S_{iC} cannot be directly mapped to $\mathcal{H}_{iC}$. However, without PAM, LoRA-Q is also updated with perturbing cross-contamination before softmax.

$$S_i = S_{i0} + S_{iC} + \Delta S_i \,, \tag{24}$$

3.3. Perturbation Influence in LoRA Without PAM

With the first-order Taylor approximation for $\mathcal{H}_i$-applied LoRA-Q and LoRA-V, the perturbation of $\mathcal{H}_i$ under ΔS_i and Δv_i can be expressed as follows:

$$\mathcal{H}_i(\Delta S_i, \Delta v_i) \equiv \mathcal{H}_i(S_{i0} + \Delta S_i, v_{i0} + \Delta v_i) - \mathcal{H}_i(S_{i0}, v_{i0}) \tag{25}$$

$$\approx \left(\frac{\partial \mathcal{H}_i}{\partial S_i}\right)[\Delta S_i] + \left(\frac{\partial \mathcal{H}_i}{\partial v_i}\right)[\Delta v_i] = J_{\texttt{softmax}}(S_{i0})[\Delta S_i] v_{i0} + \texttt{softmax}(S_{i0}) \Delta v_i \,, \tag{26}$$

where $J_{\texttt{softmax}}$ is softmax Jacobian. The detail deviation between Equations (25) and (26) is provided in Appendix A, including why higher-order terms can be safely ignored. Δv_i is applied via direct multiplication, whereas ΔS_i first passes through the softmax Jacobian, which tends to suppress perturbations for ΔS_i. Additionally, ΔS_i is generally a quadratic form involving LoRA-Q, making its contribution less significant relative to LoRA-V. In experiments, the gradient percentage of LoRA-V in the whole gradient per step is much greater than that of LoRA-Q during fine-tuning. This characteristic implies that the loss contribution associated with mitigating cross-contamination is concentrated after the softmax operation rather than before it.

Although the gradient percentage of LoRA-Q is much less than that of LoRA-V, LoRA-Q can be analyzed to train to attenuate the cross-contamination before softmax. As mentioned in Section 2.2, the masking with small but non-zero values and the masking area can affect to normalization of softmax and stability in attention weights. Since these sensitivities are increased when PAM is not applied, the additional learnable parameter LoRA-Q needs to be trained more than with PAM. While LoRA-Q is updated with the interaction of LoRA-V during training as expressed in Equation (26), the gradient contribution for LoRA-Q becomes larger, whereas that for LoRA-V becomes smaller compared to the case without PAM, particularly in highly packed steps. Estimating and comparing the gradient percentage of LoRA-Q and LoRA-V per step are described in Section 4.2.

3.4. Runtime Overheads of PAM

One of implementation complexities for packing is masking generation. For efficient execution during training, each mask is generated per training iteration and fed with inputs to the model. Algorithm 1 describes the packing-aware mask of the block-diagonal shape, which is imported in mask generation per training iteration. Let N be the sequence length. On the Flash attention [18] path, constructing a key-padding mask is $\mathcal{O}(N)$ in time and memory, whereas the causal mask is handled implicitly by the kernel (thus $\mathcal{O}(1)$). In contrast, the packing-aware (block-diagonal) attention mask, when materialized as an $N \times N$ keep-mask, requires $\mathcal{O}(N^2)$ time and memory; with batch size B, the cost grows linearly to $\mathcal{O}(BN^2)$. If cross-contamination is allowed, generating packing-aware mask with $\mathcal{O}(BN^2)$ is passed. The time difference with and without packing-aware mask generation is reported in Table 1.

Algorithm 1 Packing-aware mask generation

function MAKEBLOCKDIAGKEEPMASKANDPOS($\mathbb{B}, N$, pad)
  $Bnds \leftarrow \text{sorted}(\{b \in \mathbb{Z} \mid 0 \le b < N\} \cap \mathbb{B})$
  if $Bnds = \varnothing$ or $Bnds[0] \neq 0$ **then**
    $Bnds \leftarrow [0] \,\|\, Bnds$
  end if
  $Bnds \leftarrow Bnds \,\|\, [N]$ ▷ append end sentinel
  $M \leftarrow \mathbf{0}^{N \times N}$; $p \leftarrow \mathbf{0}^{N}$ ▷ keep-mask (1 = keep, 0 = mask)
  for $k = 0$ to $|Bnds| - 2$ **do**
    $s \leftarrow Bnds[k]$, $e \leftarrow Bnds[k{+}1]$ ▷ segment $[s : e)$
    if $s < e$ **then**
      $M[s{:}e,\ s{:}e] \leftarrow 1$ ▷ place a 1-block on the diagonal
    end if
  end for
  $P \leftarrow \text{pad} \cdot \text{pad}^{\top}$ ▷ outer product; zero padded rows/cols
  $M \leftarrow M \odot P$ ▷ M stays a keep-mask for Algorithm 2
  return (M, p)
end function

Table 1. Detailed timing results for batch generation, per-step, and per-epoch computations for 'NousResearch/Llama-2-7b-hf' model with 'databricks/databricks-dolly-15k' dataset. "Allow CC" denotes that cross-contamination is allowed (setting without PAM). All numbers except packing time are averages; units are seconds (s) or milliseconds (ms). The 'Mask' column is computed as 'mask gen.' + 'mask h2d.' per step.

Exp.	Allow CC	Packing Time (s)	Step Time (s)	Mask gen. (ms)	Mask h2d. (ms)	Mask (ms)	Step Time -Mask (s)	Epoch Time (s)
Base		–	0.8307	0.1000	0.1641	0.1641	0.8307	390.1433
Online (Fast)	×	60.0617	0.7493	1.8111	0.2653	2.0764	0.7472	166.1000
	✓	58.9256	0.7265	0.1027	0.1300	0.2326	0.7263	161.0433
	Δ	−1.1361	−0.0228	−1.7085	−0.1353	−1.8438	−0.0210	−5.0567
	Δ (%)	−1.8916	−3.0428	−94.3294	−50.9989	−88.7979	−2.8045	−3.0444
Online (Delay)	×	60.4502	0.7509	2.3954	0.2660	2.6613	0.7482	166.4400
	✓	58.8972	0.7353	0.0600	0.1214	0.1814	0.7351	162.9933
	Δ	−1.5530	−0.0155	−2.3354	−0.1446	−2.4800	−0.0130	−3.4467
	Δ (%)	−2.5691	−2.0775	−97.4952	−54.3609	−93.1838	−1.7535	−2.0708
Offline (Fast)	×	92.7946	0.7491	2.2307	0.2656	2.4963	0.7466	154.0567
	✓	91.4681	0.7406	0.0806	0.0610	0.1417	0.7405	152.3067
	Δ	−1.3266	−0.0085	−2.1500	−0.2046	−2.3546	−0.0061	−1.7500
	Δ (%)	−1.4296	−1.1347	−96.3868	−77.0331	−94.3236	−0.8231	−1.1359
Offline (Delay)	×	95.0622	0.7413	2.3773	0.4832	2.8605	0.7384	152.4633
	✓	93.6683	0.7395	0.1182	0.0877	0.2059	0.7393	152.1033
	Δ	−1.3939	−0.0018	−2.2591	−0.3955	−2.6546	0.0009	−0.3600
	Δ (%)	−1.4663	−0.2428	−95.0280	−81.8502	−92.8020	0.1157	−0.2361

Algorithm 2 describes the collator that constructs attention masks and measures their generation time ('mask_gen_ms'). When cross-contamination is not allowed, the block-diagonal mask generation (Algorithm 1) is invoked at line 16, producing a full 3D attention mask. In contrast, when cross-contamination is allowed, a simple padding mask is generated. The variable 'mask_gen_ms' records the CPU-side wall-clock time required to generate the attention mask, enabling quantitative analysis of the masking overhead. The host-to-device (H2D) transfer time of the attention mask is measured separately by synchronizing CUDA streams before and after the data transfer. Specifically, we record the timestamps immediately before and after calling `attention_mask.to(device)` with `torch.cuda.synchronize()` on both sides, and compute the elapsed time in milliseconds as the H2D latency ('mask_h2d_ms') for every training iteration. Both 'mask_gen_ms' and 'mask_h2d_ms' are represented in Section 4.2 as 'mask gen.' and 'mask h2d', respectively.

Algorithm 2 Collate with masking and timing

```
1: function COLLATE(Examples)
2:     Examples = [ {ex_i}_{i=1}^{B}, pad_id, allow_contamination, build_position_ids, mask_dtype ]
3:     t_0 ← TIMERNOWMS()
4:     Stack input_ids and labels from {ex_i} to tensors of shape [B, N]
5:     boundaries ← { ex_i.boundaries or ∅ }
6:     if allow_contamination then
7:         attention_mask ← (input_ids ≠ pad_id) as long in [B, N]
8:         position_ids ← None
9:     else
10:        Initialize attention_mask ∈ R^{B×N×N} with zeros
11:        if build_position_ids then
12:            Initialize position_ids ∈ N^{B×N} with zeros
13:        end if
14:        for b ← 1 to B do
15:            pad ← (input_ids[b] ≠ pad_id) as long
16:            (m, p) ← MAKEBLOCKDIAGKEEPMASKANDPOS (boundaries[b], N, pad)
17:            attention_mask[b] ← m
18:            if build_position_ids then
19:                position_ids[b] ← p
20:            end if
21:        end for
22:    end if
23:    mask_gen_ms ← TIMERNOWMS() − t_0
24:    return {input_ids, labels, attention_mask, boundaries, mask_gen_ms, position_ids}
25: end function
```

Furthermore, in practice, applying PAM introduces several training-time overheads. Each factor is detailed below, with references to the corresponding pseudocode lines in Algorithm 3.

1. For a 3D block mask of size $\mathbb{R}^{B\times N\times N}$, converting the mask to `float32` (line 10) and transforming the values in masking area to negative values (line 12), perform full-tensor reads/writes and arithmetic at $\mathcal{O}(BN^2)$, making the step bandwidth-bound and increasing kernel launches relative to simply forwarding the mask.
2. Repeated preprocessing without semantic effect such as additional no-ops with '`.contiguous()`' and '`.clone()`' introduces extra kernels and full-tensor copies, as described in lines 13 to 15. When repeated `rep` times, they scale linearly with `rep`, further amplifying GPU memory traffic without changing computation semantics.
3. Forcing the SDPA backend to `math` instead of Flash attention [18] (i.e., disabling `flash/mem_efficient`) removes the most optimized attention kernels as shown in line 23. Even with identical masks, the attention computation itself becomes slower, compounding the overheads introduced by 1 and 2 factors.
4. Performing the above transformations inside each attention layer causes the same mask to be reprocessed L times per forward (for L layers). In contrast, a cached, additive mask prepared once at the model level would amortize this cost; thus, layer-local preprocessing multiplies both kernel count and memory traffic (lines 8 to 17).

If PAM is eliminated, training speed degradation and its contributing factors do not arise. Furthermore, the implementation for PAM stage including mask generation (Algorithm 1) and forwarding the mask to attention layer (Algorithm 3) are not required. The experiment setting with all of the above four factors is named '**Delay**', which mimics worst-case masking slowdowns per Algorithm 3, while Fast skips them for optimization. We compare the training results without '**Delay**', which is named '**Fast**', as tabulated in Tables 1 and 2.

Algorithm 3 Runtime overheads in attention layer with PAM

function ComplexCustomAttentionForward(H, M, *original_forward*, *config*)
▷ $H \in \mathbb{R}^{B\times N\times d}$, M may be `None`, $M \in \{B\times N,\ B\times N\times N\}$
  $M_{\text{ext}} \leftarrow$ `None`
  $ms \leftarrow$ *config*.`mask_stress`
    ▷ user options: `enable`, `preprocess_in_forward`, `repeat_preproc`, `disable_sdp`
  if $M \neq$ `None` **and** rank$(M) = 3$ **then** ▷ supports $B\times N\times N$
    $m \leftarrow M$
    if *ms*.`enable` **and** *ms*.`preprocess_in_forward` **then**
      if dtype$(m) \neq$ `float32` **then**
        $m \leftarrow$ AsType(m, `float32`)
      end if
      $m \leftarrow (1-m)\cdot(-10^9)$ ▷ *0/1 keep-mask → additive mask*: keep→ 0, mask→ $-\infty$
      for $i \leftarrow 1$ **to** max$(0,$ *ms*.`repeat_preproc`$)$ **do**
        $m \leftarrow m + 0.0$; $m \leftarrow$ Contiguous(m); $m \leftarrow$ Clone(m)
          ▷ intentional kernel/memory stress without changing values
      end for
    end if
    $M_{\text{ext}} \leftarrow$ Unsqueeze(m, axis $= 1$) ▷ $B\times 1\times N\times N$
  else if $M \neq$ `None` **then** ▷ $B\times N$ padding mask path
    $M_{\text{ext}} \leftarrow M$[:, `None`, `None`, :]
  end if
  if *ms*.`enable` **and** *ms*.`disable_sdp` **then**
    return *original_forward*(`self`, H, `attention_mask` $= M_{\text{ext}}$, `sdpkernels` $=$ MathOnly)
      ▷ force math-only attention (disable Flash/SDPA)
  else
    return *original_forward*(`self`, H, `attention_mask` $= M_{\text{ext}}$)
  end if
end function

Table 2. Average and standard deviation of loss and parameter-update percentage differences ($\Delta = \checkmark - \times$) per step for the 'NousResearch/Llama-2-7b-hf' model on the 'databricks/databricks-dolly-15k' dataset. All parameter-update values are reported as percentages (%).

Exp.	Δ Step Loss		Δ Epoch Last Loss		Δq_rel_mean		Δq_p95		Δq_max		Δv_rel_mean		Δv_p95		Δv_max	
	Avg.	Std.	Avg.	Std.	Avg.	Std.	Avg.	Std.	Avg.	Std.	Avg.	Std.	Avg.	Std.	Avg.	Std.
Online(Fast)	0.036	0.576	−0.010	0.001	−0.805	0.197	−1.366	0.639	−3.400	1.178	0.465	0.352	0.470	0.357	0.295	0.216
Online(Delay)	0.044	0.597	0.001	0.014	−0.795	0.199	−1.260	0.596	−3.707	1.452	0.260	0.230	0.621	0.401	0.594	0.499
Offline(Fast)	−0.046	0.106	0.003	0.004	−1.306	0.373	−2.049	0.597	−5.717	2.077	0.206	0.262	0.008	0.285	0.470	1.252
Offline(Delay)	−0.044	0.101	−0.004	0.005	−1.351	0.385	−2.668	0.791	−5.935	2.225	0.260	0.269	0.327	0.304	0.654	1.295

4. Results and Discussion

4.1. Experimental Setup

'databricks/databricks-dolly-15k' dataset [28] consists of 15,000 samples of instruction-response pairs. 'tatsu-lab/alpaca' and 'yahma/alpaca_cleaned' datasets [29] contain 52,000 samples. For supervised fine-tuning, we use the released version of the datasets from Hugging Face. Five parameter sets of pre-trained models are used to supervise fine-tuning, as shown in Table 3. All of the pre-trained models can be downloaded from Hugging Face Transformers [30]. Unless otherwise stated, we fine-tune all models with LoRA adapters on the query and value projections. For all 7B pre-trained models, we adopt $r = 8$, following prior evidence that small ranks can be competitive with larger-rank configurations [20], and use $\alpha = 32$, LoRA dropout of 0.1, and a learning rate of 2×10^{-4}. The maximum sequence length per batch is set to 256 tokens with the batch size of 4 per device, except for 'mistralai/Mistral-7B-v0.1' model, which uses the batch size of 2 tokens per device.

All pre-trained models are fine-tuned for 3 epochs using distributed data parallelism (DDP) on 8 NVIDIA RTX A6000 GPUs and AMD EPYC 7513 32-Core Processor CPUs. Experiments are conducted on an NVIDIA server with driver 525.105.17 (CUDA 12.0) using

PyTorch 2.1.2, Transformers 4.39.3, PEFT 0.10.0, Accelerate 0.27.2, and Datasets 2.19.1. We apply masking as an additive bias to $q_i k_i^T$. The final mask is $M = M_{pad} + M_{pack} + M_{causal}$, where M_{pad} masks padding regions, M_{pack} blocks cross-segment attention for packed inputs, and M_{causal} is a strict upper-triangular bias implementing causality. Packing is applied at batch construction, and packing-aware attention masks are built on the fly per training step.

Table 3. Comparison of four publicly available 7B-scale language models from Hugging Face.

Model Name	Base Architecture	# of Parameters
NousResearch/Llama-2-7b-hf [1] (**NR**)	Llama 2	7B
huggyllama/llama-7b [2] (**HL**)	LLaMA 1	6.74B
openlm-research/open_llama_7b [2] (**OR**)	OpenLLaMA	7B
mistralai/Mistral-7B-v0.1 [31] (**MR**)	Mistral 7B	7B
NousResearch/Llama-2-13b-hf [3] (**NR(13B)**)	Llama 2	13B

[1] https://huggingface.co/NousResearch/Llama-2-7b-hf (1 September 2025) [2] https://huggingface.co/huggyllama/llama-7b (1 September 2025) [3] https://huggingface.co/NousResearch/Llama-2-13b-hf (1 September 2025).

4.2. Reducing Training Time by Eliminating PAM

Tables 1 and 2 describe the comparisons for PAM existence including online and offline packing, respectively marked as **Online** and **Offline**. The experiment setting with mimicked worst-case masking slowdowns per Algorithm 3 is named **Delay**, while Fast skips them for optimization. The training results without **Delay** are named **Fast**. The pre-trained model is 'NousResearch/Llama-2-7b-hf' and trained by 'databricks/databricks-dolly-15k' dataset. The title of column 'Allow CC' denotes that cross-contamination is allowed, where '×' is applied PAM and ✓ is not.

As shown in Table 1, the reported packing time incudes batch generation under packing and the accumulated mask-generation time across training steps. Step time denotes the per-step runtime, including both the forward and backward passes. Values in the 'Epoch time' column are computed as the sum of step times over each epoch. Values in Δ rows are computed by subtracted values from allowing cross contamination to without allowing. Average and standard deviation values in the Δ rows for each experiment in Table 1 are derived from differences ($\checkmark - \times$) in the packing time, per step and per epoch. Δ (%) denotes the decreasing percentage ($(\checkmark - \times)/\times$). Therefore, the smaller the Δ value (or Δ (%)) is (i.e., the more negative), the faster the training.

The values in Table 2 are also derived from differences (Δ) for per-step loss, last loss per epoch, and update percentage for adaptive parameters (LoRA-Q and LoRA-V) per step. For each q or v projection l-th layer, the relative update magnitude is defined as

$$rel_l = \frac{\|\Delta W_l\|_F}{\|W_l\|_F + \epsilon} \times 100 \text{ , (\%)} \tag{27}$$

where $\Delta W_l = (B_l A_l) \cdot scaling$ is the effective LoRA-Q increment and W_l is the pre-trained base weight. $\|\cdot\|_F$ denotes Frobenius norm. We aggregate rel_l across all query layers to report three summary statistics: 'q_rel_mean' (arithmetic mean), 'q_rel_p95' (95th percentile) and 'q_rel_max' (maximum). All values are dimensionless and reported in percentage. For the LoRA-V, we aggregate rel_l across all value layers and respectively summarize them: 'v_rel_mean', 'v_rel_p95' (95th percentile), and 'v_rel_max' (maximum). Unless otherwise noted, the statistics are computed after each optimizer step (post-update) with the baseline W_l taken at the start of training; $\epsilon = 10^{-12}$ prevents division by zero. Frobenius norms are used throughout.

As shown in Δ rows for each experiment in Table 1, the packing time, step time, and epoch time are decreased when PAM is not applied by erasing the PAM stages. Since the packing-aware mask generation stage is passed, the values for packing time in Δ row are less than zero. Average values per step time for all Δ rows also decrease by eliminating the stage to forward packing-aware mask into the attention layer. For the packing time and average time values when both allowing or not allowing cross-contamination with **Online**, **Fast** is lower than **Delay**. Comparing **Online** and **Offline** experiments, **Online** packing is faster than **Offline** but the average time values is much faster in **Offline** since each batch is more compactly packed. **Base** is supervised fine-tuning with LoRA-Q and LoRA-V adaptive parameters to compare the value of packing itself. Both experiments shows shorter training time than **Base** for all time columns. The packing time for **Base** is not necessary to estimate.

Table 1 also reports the per-step masking overhead. Except the 'packing time' column, all values i the table are averages. 'mask gen.' and 'mask h2d' denote the average per-step times for mask generation and host-to-device transfer, respectively. 'Mask' is their per-step sum (Mask = mask gen. + mask h2d). All three columns are reported in milliseconds. 'step time - Mask' equals the step time minus 'Mask' in step level. The reductions reported in the Δ rows for 'mask gen.', 'mask h2d', and 'Mask' arise because the packing-aware mask generation (Algorithm 1) is entirely bypassed when cross-contamination is allowed. Generating PAM requires materializing an $N \times N$ keep-mask per sequence, leading to $\mathcal{O}(BN^2)$ time and memory, which is substantially higher than the costs for a key-padding mask $\mathcal{O}(BN)$ and the (flag-based) causal constraint $\mathcal{O}(B)$ on the Flash attention path. Consequently, 'mask gen.' under 'Allow CC' decreases by more than 20 times compared to not allowing cross-contamination. 'mask h2d' under 'Allow CC' is roughly halved, since a one-dimensional padding vector is transferred instead of an $N \times N$ mask. A small non-zero masking time remains due to computing the padding vector and enforcing the causal constraint inside the attention kernel. Therefore, the time for 'mask gen.' and 'mask h2d.' when allowing cross-contamination (not PAM) show similar values to those for **Base**.

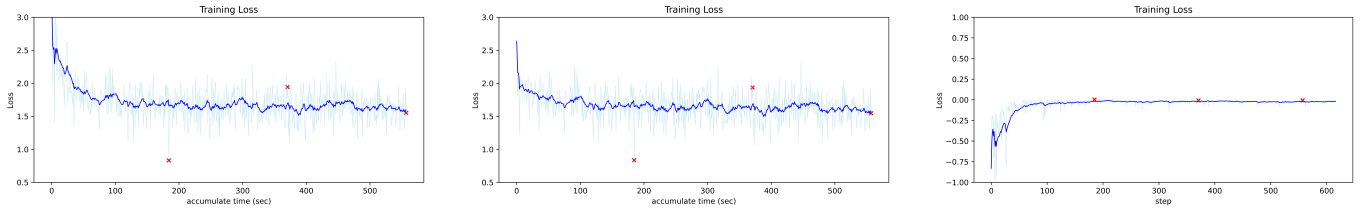
We evaluate on an 8-subject subset of MMLU [32] in the K-shot (default $K = 5$) multiple-choice setting. Table 4 reports per-subject accuracies only. For each subject, we concatenate K in-context examples and the test question with four options (A–D), followed by "Answer:". The model selects the option whose answer letter attains the highest conditional log-likelihood (no sampling). The rows labeled ✓ − **B** represent the absolute difference between allowing contamination for each experiment and **Base**. The rows labeled ✓ − × reports the absolute difference between the two experiments following the definitions Δ used in Table 1. The results show performance comparable to the baseline, indicating that our approach shows similar performances for downstream tasks. The scores on all subjects for **Online** experiments vary only within -0.08 to $+0.05$, confirming that the proposed method maintains task-level performance while reducing overhead. The scores across all subjects in the **Offline** experiments vary only within a range of -0.12 to $+0.02$, which is slightly lower than those of the **Online** experiments. This is mainly because offline packing involves a more complex segment composition and is inherently more difficult to mitigate cross-contamination. However, performance comparison between allowing cross-contamination and **Base** shows similar or higher scores, varying within a range of -0.05 to $+0.06$ as shown in ✓ − **B** rows.

Although the unnecessary attention of cross-contamination is expected to make fine-tuning the models difficult, the loss difference between the training with and without PAM is not remarkable, as described in 'Δ step loss' and 'Δ epoch last loss' columns in Table 2. To compare the detail trajectory of loss, the figures in Figure 1 show the difference between two losses in **Offline(Delay)** experiment. Although cross-contamination

is expected to require much more training cost, except at the beginning of training, the loss without PAM converges quickly to the loss with PAM. Rather, the training loss when allowing cross-contamination is lower than when not allowing. This lower loss distribution is also represented as negative average values for Δ step loss and Δ epoch loss columns in Table 2. In the remaining experiments, the average values for Δ loss are $<\pm 0.05$, indicating that the models trained without PAM rapidly approach the performance of their with-PAM counterparts.

Table 4. Evaluation results for eight subjects in MMLU benchmark with fine-tuned 'NousResearch/Llama-2-7b-hf' model with 'databricks/databricks-dolly-15k' dataset. "Allow CC" denotes that cross-contamination is allowed (setting without PAM). **B** denotes **Base** setting.

Exp.	Allow CC	Abstract Algebra	Computer Security	Econometrics	College Biology	Astronomy	High School	Clinical Knowledge	Professional Law
Base		0.300	0.510	0.289	0.347	0.303	0.252	0.370	0.330
Online (Fast)	×	0.280	0.470	0.325	0.465	0.355	0.258	0.375	0.365
	✓	0.330	0.480	0.316	0.382	0.316	0.232	0.385	0.360
	✓ − ×	0.050	0.010	−0.009	−0.083	−0.039	−0.026	0.010	−0.005
	✓ − **B**	0.030	−0.030	0.027	0.035	0.013	−0.020	0.015	0.030
Online (Delay)	×	0.320	0.520	0.307	0.479	0.309	0.219	0.430	0.350
	✓	0.310	0.480	0.342	0.403	0.336	0.219	0.400	0.350
	✓ − ×	−0.010	−0.040	0.035	−0.076	0.026	0.000	−0.030	0.000
	✓ − **B**	0.010	−0.030	0.053	0.056	0.033	−0.033	0.030	0.020
Offline (Fast)	×	0.340	0.530	0.325	0.472	0.342	0.298	0.475	0.355
	✓	0.340	0.500	0.298	0.410	0.362	0.252	0.405	0.325
	✓ − ×	0.000	−0.030	−0.026	−0.062	0.020	−0.046	−0.070	−0.030
	✓ − **B**	0.040	−0.010	0.009	0.063	0.059	0.000	0.035	−0.005
Offline (Delay)	×	0.340	0.580	0.333	0.479	0.355	0.272	0.490	0.340
	✓	0.350	0.460	0.289	0.410	0.342	0.252	0.425	0.280
	✓ − ×	0.010	−0.120	−0.044	−0.069	−0.013	−0.020	−0.065	−0.060
	✓ − **B**	0.050	−0.050	0.063	0.039	0.039	0.000	0.055	−0.050

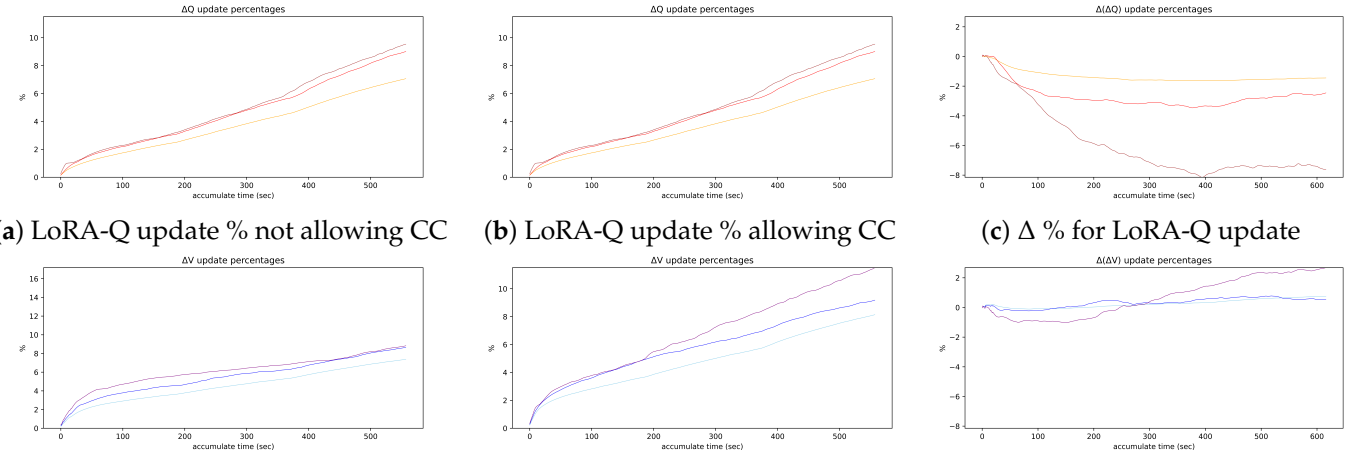


(**a**) Not allowing CC (**b**) Allowing CC (**c**) Δ loss per step

Figure 1. The loss comparison for **Offline(Delay)** as described in Table 2. The figures (**a**,**b**) with the x-axis in the time domain show how the values change based on the cumulative time at each step. Red points are the last loss values per epoch, thin blue lines represent training loss, and thick lines are applied EMA smoothing ($\alpha = 0.1$) for each loss.

As mentioned in Equation (27), we measure the update percentages for LoRA-Q and LoRA-V per step and summarize to three statistics as shown in Table 2 and Figure 2. When cross-contamination is not allowed, all update percentages for LoRA-Q (Figure 2a) steadily increase more rapidly than percentages for LoRA-V (Figure 2d) during training, except the early part of the training. However, for the training when allowing cross-contamination as shown in Figure 2b,e, the percentages for LoRA-V increase much higher than percentages for LoRA-V, whereas they comparable to percentages for LoRA-V with not allowing. This means a higher perturbation of the step-wise gradient budget is allocated to value parameters, as explained in Section 3.3. These percentage differences between the two experiments are shown in Figure 2c,f. These results are represented as the negative

average values for the columns (named with 'Δq_') in Table 2, while the average values with columns (named with 'Δv_') are positive.



(**a**) LoRA-Q update % not allowing CC (**b**) LoRA-Q update % allowing CC (**c**) Δ % for LoRA-Q update

(**d**) LoRA-V update % not allowing CC (**e**) LoRA-V update % allowing CC (**f**) Δ % for LoRA-V update

Figure 2. The parameter-update percentage comparison for **Offline(Delay)** as described in Table 2. The figures (**a**,**b**,**d**,**e**) with the x-axis in the time domain show how the values change based on the cumulative time at each step, while figures (**c**,**f**) are step level with x-axis. Orange, red, and brown lines are 'q_rel_mean', (arithmetic mean), 'q_rel_p95' (95th percentile) and 'q_rel_max' (maximum) for updating LoRA-Q parameter, respectively. Skyblue, blue, and purple lines are 'v_rel_mean' (arithmetic mean), 'v_rel_p95' (95th percentile), and 'v_rel_max' (maximum) for updating the LoRA-V parameter, respectively.

In the above experiment, we apply $r = 8$ as the default LoRA rank, following the original LoRA [20], which demonstrated it as a broadly effective setting. To further address other ranks, we additionally conduct experiments with $r = 4$ and $r = 16$ for **Offline(Delay)** setting in Tables A6 and A7. The results confirm that the performance trends—such as mask generation and transmission time as well as step loss reduction—are consistent with those observed at $r = 8$.

Tables 5 and 6 report the differences of SFT results for four pre-trained models under the **Offline(Delay)** setting on each dataset to assess the generality of our approach. The original training result values for each dataset in Table 5 are represented separately for each dataset in Appendix B. In the rows labeled 'databricks/databricks-dolly-15k' in Table 5, almost-average values are negative (✓ $<$ ×), consistent with the previous experiment. In particular, values for mask generation and transmission stage are highly decreased by more than 90% and 60%, respectively. These decreases affect the decrease in time for packing, per step, and epoch. Table 6 shows the decreases for the average losses and update percentages, as similar to Table 2. The detailed timing results are tabulated in Table A1. The averages for LoRA-V update percentage columns (named with 'Δv_') are also negative, but the averages for LoRA-Q (named with 'Δq_') are much less. Thus, the relative update percentage in LoRA-Q decreases more rapidly than in LoRA-V, with LoRA-V exhibiting larger perturbations to attenuate cross-contamination.

'tatsu-lab/alpaca' rows in Tables 5 and 6 report SFT results for four pre-trained models on the 'tatsu-lab/alpaca' dataset. In Table 5, all time entries in the Δ rows are negative, consistent with the previous experiment tables. The detailed timing results are tabulated in Table A2. Table 6 shows the decreases for the average losses and update percentages, as similar to Table 6. The averages for each LoRA-V update percentage columns are also negative, but the corresponding averages for LoRA-Q is much less. Therefore, LoRA-V exhibits larger perturbations to attenuate cross-contamination.

Table 5. Detailed timing results for batch generation, per-step, and per-epoch computations for pre-trained models with three datasets and **Offline(Delay)** setting. Δ, Δ (%) and units (s, ms) follow the same definition as in Table 1. All numbers except packing time are averages. The detailed values are represented in tables in Appendix B.

Datasets	Models	Δ	Packing Time (s)	Step Time (s)	Mask gen. (ms)	Mask h2d. (ms)	Mask (ms)	Step Time -Mask (s)	Epoch Time (s)
databricks databricks -dolly-15k	**NR**	✓ − ×	−1.3939	−0.0018	−2.2591	−0.3955	−2.6546	0.0009	−0.3600
		Δ (%)	−1.4663	−0.2428	−95.0280	−81.8502	−92.8020	0.1157	−0.2361
	HL	✓ − ×	−1.1749	0.0024	−1.9042	−0.2856	−2.1899	0.0046	0.4933
		Δ (%)	−1.0437	0.3325	−94.0534	−62.9352	−88.3559	0.6382	0.3323
	OR	✓ − ×	−1.2786	−0.0172	−2.2353	−0.3215	−2.5568	−0.0146	−3.2800
		Δ (%)	−1.7260	−2.3354	−95.5256	−80.4555	−93.3272	−1.9957	−2.3356
	MR	✓ − ×	−1.1817	−0.0086	−0.9880	−0.1656	−1.1536	−0.0074	−3.4267
		Δ (%)	−1.5654	−0.9808	−90.1469	−62.0315	−84.6430	−0.8506	−0.9803
tatsu-lab /alpaca	**NR**	✓ − ×	−1.3939	−0.0018	−2.2591	−0.3955	−2.6546	0.0009	−0.3600
		%	−1.4663	−0.2428	−95.0280	−81.8502	−92.8020	0.1157	−0.2361
	HL	✓ − ×	−1.1749	0.0024	−1.9042	−0.2856	−2.1899	0.0046	0.4933
		%	−1.0437	0.3325	−94.0534	−62.9352	−88.3559	0.6382	0.3323
	OR	✓ − ×	−1.2786	−0.0172	−2.2353	−0.3215	−2.5568	−0.0146	−3.2800
		%	−1.7260	−2.3354	−95.5256	−80.4555	−93.3272	−1.9957	−2.3356
	MR	✓ − ×	−1.1817	−0.0086	−0.9880	−0.1656	−1.1536	−0.0074	−3.4267
		%	−1.5654	−0.9808	−90.1469	−62.0315	−84.6430	−0.8506	−0.9803
yahma/ alpaca _cleaned	**NR**	✓ − ×	−6.9868	−0.0038	−2.0580	−0.2902	−2.3482	−0.0015	−4.2733
		Δ (%)	−0.6457	−0.5112	−94.2607	−78.4171	−91.9637	−0.1960	−0.5080
	HL	✓ − ×	−7.3857	−0.0246	−2.1755	−0.2896	−2.4651	−0.0221	−27.8633
		Δ (%)	−0.7290	−3.2178	−96.2100	−75.6333	−93.2264	−2.9054	−3.2207
	OR	✓ − ×	−6.2812	−0.0079	−1.9985	−0.2809	−2.2794	−0.0057	−8.3233
		Δ (%)	−0.6913	−1.0725	−95.7411	−68.6461	−91.3002	−0.7695	−1.0651
	MR	✓ − ×	−6.8929	−0.0057	−1.0580	−0.1658	−1.2238	−0.0045	−12.3400
		Δ (%)	−0.7307	−0.6639	−90.7220	−61.3620	−85.2001	−0.5222	−0.6618

Table 6. Average and standard deviation of loss and parameter-update percentage differences (Δ = ✓ − ×) per step for pre-trained models with three datasets and **Offline(Delay)** setting. All parameter-update values are reported as percentages (%).

Datasets	Models	Δ Step Loss		Δ Epoch Last Loss		Δq_rel_mean		Δq_p95		Δq_max		Δv_rel_mean		Δv_p95		Δv_max	
		Avg.	Std.	Avg.	Std.	Avg.	Std.	Avg.	Std.	Avg.	Std.	Avg.	Std.	Avg.	Std.	Avg.	Std.
databricks/ databricks -dolly-15k	**NR**	−0.045	0.101	−0.004	0.005	−1.351	0.385	−2.668	0.791	−5.935	2.225	0.260	0.269	0.327	0.304	0.654	1.295
	HL	−0.039	0.133	−0.017	0.009	−1.355	0.344	−1.880	0.814	−0.898	1.567	−0.017	0.257	−0.751	0.161	−2.196	0.786
	OR	−0.074	0.125	−0.042	0.027	−1.197	0.356	−1.994	0.993	−5.736	3.386	0.074	0.322	−0.897	0.393	−0.800	0.453
	MR	−0.032	0.071	−0.016	0.035	−2.907	0.879	−3.332	1.321	−10.206	4.398	−0.726	0.475	−0.808	0.426	−1.942	1.329
tatsu-lab/ alpaca	**NR**	−0.028	0.112	−0.012	0.012	−1.829	0.383	−4.924	1.530	−9.461	2.933	0.156	0.378	0.362	0.499	1.221	1.427
	HL	−0.038	0.115	−0.015	0.011	−1.453	0.273	−2.099	0.620	0.084	0.636	−0.094	0.281	−1.779	0.420	−2.295	0.661
	OR	−0.047	0.104	−0.022	0.005	−1.286	0.252	−1.863	0.530	−8.959	3.848	0.255	0.469	−0.718	0.327	−0.469	0.275
	MR	−0.019	0.057	−0.022	0.026	−3.473	0.776	−6.520	1.989	−14.982	5.375	−0.307	0.912	−0.305	0.601	0.315	1.537
yahma/ alpaca _cleaned	**NR**	−0.011	0.028	0.001	0.004	−1.273	0.197	−3.821	0.985	−5.540	1.662	0.696	0.401	1.619	0.701	3.503	2.006
	HL	−0.012	0.048	−0.014	0.009	−1.031	0.276	−0.869	0.367	1.186	0.808	0.275	0.266	−0.061	0.491	0.624	1.419
	OR	−0.020	0.037	−0.009	0.007	−1.384	0.273	−1.779	0.514	−15.632	7.442	0.717	0.450	0.003	0.234	−0.164	0.430
	MR	−0.009	0.025	0.002	0.021	−4.016	0.851	−13.299	4.164	−40.767	19.104	0.148	0.592	0.436	0.564	−1.673	1.078

The rows named with 'yahma/alpaca_cleaned' in Tables 5 and 6 show SFT results for four pre-trained models on 'yahma/alpaca_cleaned' dataset, which contains many more samples than the previous two datasets. In the rows named with 'yahma/alpaca_cleaned' in Table 5, nearly all time entries in the Δ rows are negative and are smaller (i.e., more negative) than the values in the corresponding cells of other dataset rows. The detailed timing results are tabulated in Table A3. Table 6 shows the decreases for the average losses and update percentages, as similar to Table 6. The averages in 'Δv_rel_mean' is positive

while the averages in 'Δq_rel_mean' is near zero or negative. That is, LoRA-V exhibits larger perturbations than LoRA-Q to mitigate cross-contamination.

To check that our approach is available to other LLM, we fine-tune a 13-billion pre-trained model(NousResearch/Llama-2-13b-hf). Tables 7 and 8 show SFT results for pre-trained model with 'databricks/databricks-dolly-15k' datasets. Due to limitations in the experimental setup (devices capacity), we perform experiments with a small batch size (with 1), and thus did not show the same reduction in time as the previous 7B models. In Table 7, nearly all time entries in the Δ and Δ (%) rows are negative, especially in mask generation and transmission time. Table 8 shows the decreases of the average losses and differences of update percentages. The averages of update percentage for LoRA-V are remarkably increased while the averages for LoRA-Q are not. Therefore, LoRA-V exhibits larger perturbations than LoRA-Q to mitigate cross-contamination.

Table 7. Detailed timing results for batch generation, per-step, and per-epoch computations for a 13B pre-trained model ('NousResearch/Llama-2-13b-hf') with 'databricks/databricks-dolly-15k' dataset and **Offline(Delay)** setting. Δ, Δ (%), and units (s, ms) follow the same definition as in Table 1. All numbers except packing time are averages. The detailed results are reported in Table A4.

Exp.	Δ	Packing Time (s)	Step Time (s)	Mask gen. (ms)	Mask h2d. (ms)	Mask (ms)	Step Time -Mask (s)	Epoch Time (s)
Online (Fast)	✓ − ×	−2.6398	−0.0007	−0.9947	−0.0927	−1.0874	0.0005	−0.5833
	Δ (%)	−4.2218	−0.0725	−89.3139	−31.7140	−77.3384	0.0590	−0.0796
Offline (Fast)	✓ − ×	−2.3581	−0.0003	−0.9566	0.0100	−0.9467	0.0006	−0.2733
	Δ (%)	−2.5067	−0.0361	−84.2240	4.2319	−68.9914	0.0780	−0.0400

Table 8. Average and standard deviation of loss and parameter-update percentage differences (Δ = ✓ − ×) per step a 13B pre-trained model ('NousResearch/Llama-2-13b-hf') with 'databricks/databricks-dolly-15k' dataset. All parameter-update values are reported as percentages (%).

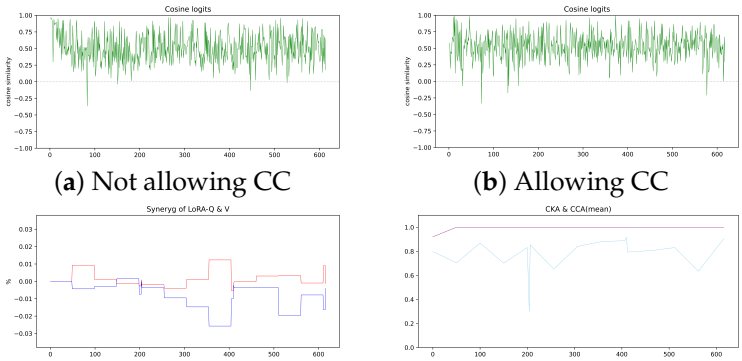
Exp.	Δ Step Loss		Δ Epoch Last Loss		Δq_rel_mean		Δq_p95		Δq_max		Δv_rel_mean		Δv_p95		Δv_max	
	Avg.	Std.	Avg.	Std.	Avg.	Std.	Avg.	Std.	Avg.	Std.	Avg.	Std.	Avg.	Std.	Avg.	Std.
Online(Fast)	1.267	2.814	0.000	0.000	0.483	0.360	2.382	2.437	7.378	3.454	8.883	5.252	20.151	12.563	43.604	17.801
Offline(Fast)	−0.053	0.679	0.000	0.000	1.484	0.788	6.269	3.070	9.388	3.967	3.839	2.572	6.666	4.406	−3.644	−3.644

LoRA introduces an additional $r(d_{in} + d_{out})$ parameters and a proportional amount of computation per projection matrix. This increase is linear in the low-rank dimension r, compared to the base model parameters $O(d_{in} \times d_{out})$, and thus the relative overhead (%) decreases as the model size grows. In this study, the LoRA adaptation is limited to the Q and V projections, resulting in minimal computational and memory increments, which remain negligible even for large-scale models (e.g., 34B, 70B). In contrast, the PAM mechanism scales with $\mathcal{O}(BL^2)$ in both mask construction and transfer buffer size, where B and L denote batch size and sequence length, respectively. As the context length increases (e.g., 8k–16k), PAM can become a dominant source of overhead due to mask generation and host-to-device (H2D) transfer latency. The proposed method removes the PAM pathway entirely, eliminating the $\mathcal{O}(BL^2)$ masking cost. Although the attention operation itself remains $\mathcal{O}(L^2)$, the reduction in PAM-induced overhead leads to a larger absolute performance gain for longer sequences. By discarding the $B \times L \times L$-scale mask buffer, the method also recovers GPU memory capacity, allowing longer sequence lengths or slightly larger micro-batches on the same hardware. This can mitigate the frequent batch-size-one bottleneck observed in large models and improve the effective training throughput. The complementary behavior of LoRA applied to Q/V remains valid as model capacity increases (see Section 3.4). However, quantitative verification on 34B/70B models with extended contexts (8k–16k) is left as future work due to resource constraints.

4.3. Complementary Training Between LoRA-V and LoRA-Q

As analyzed and expected in Sections 3.1 and 3.2, all the training results show that pre-trained models with packing can be fine-tuned well without packing-aware masking by covering the tensor subspace from LoRA-V and LoRA-Q parameters. The time for batch generation, per step, and epoch are decreased since additional PAM stages are ignored. The loss differences between SFT with and without PAM are closed to zero for all experiments. Furthermore, the time overhead factors by applying PAM, as explained in Section 3.4, increase the average values per time column in tables. Reductions for each time seem too small to highlight the benefit of eliminating PAM. However, the implementation for PAM stage such as Algorithms 1 and 3 is not required and the implementation cost for PAM is erased. As expected in Section 3.3, the interactive training movements between LoRA-V and LoRA-Q are shown in the tables and figures for gradient percentages in the experiments (Section 4.2).

However, it is unclear whether LoRA-V and LoRA-Q actually interact during training. We thus examine whether adapting only one of them can attain comparable performance. Under the **Offline(Delay)** setting for 'NousResearch/Llama-2-7b-hf' model and 'databricks/databricks-dolly-15k' dataset, Table 9 and Figure 3 compare training that adapts both LoRA-V and LoRA-Q against training that adapts only one of them. For the LoRA-V-only and LoRA-Q-only experiments, the rank is set to $r = 64$ to attain performance comparable to the original LoRA setup [20]. In Table 9, the Δ rows for the V-only and Q-only settings for same pre-trained model and dataset in Table 9 report differences relative to the traditional "LoRA-V & LoRA-Q" configuration shown in the first row. The experiment setting that mimics worst-case masking slowdowns per Algorithm 3 is named **Delay**, while Fast skips them for optimization.



(**a**) Not allowing CC (**b**) Allowing CC

(**c**) Synergy comparison (**d**) CKA and CCA(mean) (w.CC)

Figure 3. Cosine similarity comparison between LoRA-Q and LoRA-V grad. (**a**,**b**), synergy comparison (**c**) and CKA and CCA (mean) with allowing CC (**d**) for **Offline(Delay)**. (**a**,**b**) show cosine similarity values for all steps with training both parameters when not allowing cross-contamination and when allowing, respectively. (**c**) shows synergy between LoRA-Q and LoRA-V for each 50 steps when not allowing cross-contamination (red) and when allowing (blue). (**d**) represents CKA (skyblue) and CCA (mean, purple) values for five times per epoch when allowing cross-contamination.

Across average time values in Table 9, both single-parameter experiments (LoRA-V-only or LoRA-Q-only) run faster than training both parameters without PAM, while the averages for masking generation and transmission time are similarly decreased. However, for the decreases in loss per step as shown in Table 10, training both parameters shows greater decrease than training only one of them. Furthermore, for the averages of update percentages columns, training two-parameters notably shows positive averages in Δv_rel columns with negative averages in Δq_rel columns, while each training single-parameter shows much less update percentages for the corresponding training parameter comparing

to training two parameters. These observations suggest that adapting both parameters yields complementary, interactive training effects, as explained in Section 3.3.

Table 9. Detailed timing results for batch generation, per-step, and per-epoch computations for training LoRA-Q only or LoRA-V only with **Offline(Delay)** setting. Δ, Δ (%), and units (s, ms) follow the same definition as in Table 1. All numbers except packing time are averages. The detailed values are represented in Table A5.

Trained Param.	Δ	Packing Time (s)	Step Time (s)	Mask gen. (ms)	Mask h2d. (ms)	Mask (ms)	Step Time -Mask (s)	Epoch Time (s)
LoRA-Q & LoRA-V	✓ − ×	−1.3939	−0.0018	−2.2591	−0.3955	−2.6546	0.0009	−0.3600
	Δ (%)	−1.4663	−0.2428	−95.0280	−81.8502	−92.8020	0.1157	−0.2361
LoRA-V -only	✓ − ×	−1.3796	−0.1056	−2.2360	−0.4049	−2.6409	−0.1030	−21.7233
	Δ (%)	−1.4512	−14.2452	−94.0521	−83.7955	−92.3230	−13.9428	−14.2482
LoRA-Q -only	✓ − ×	−1.3674	−0.1024	−2.2163	−0.3516	−2.5678	−0.0998	−21.0633
	Δ (%)	−1.4385	−13.8136	−93.2234	−72.7649	−89.7675	−13.5193	−13.8153

Table 10. Average and standard deviation of loss and parameter-update percentage differences (Δ = ✓ − ×) per step for training LoRA-Q only or LoRA-V only with **Offline(Delay)** setting. All parameter-update values are reported as percentages (%).

Trained Parameters	Δ Step Loss		Δ Epoch Last Loss		Δq_rel_mean		Δq_p95		Δq_max		Δv_rel_mean		Δv_p95		Δv_max	
	Avg.	Std.	Avg.	Std.	Avg.	Std.	Avg.	Std.	Avg.	Std.	Avg.	Std.	Avg.	Std.	Avg.	Std.
LoRA-Q & LoRA-V	−0.045	0.101	−0.004	0.005	−1.351	0.385	−2.668	0.791	−5.935	2.225	0.260	0.269	0.327	0.304	0.654	1.295
LoRA-V-only	−0.031	0.100	0.010	0.008	−5.052	2.161	−7.324	2.850	−10.831	4.522	−0.457	0.186	−0.435	0.247	−0.804	0.417
LoRA-Q-only	0.032	0.080	0.031	0.014	0.597	0.348	−0.371	0.218	−3.561	2.052	−4.540	1.693	−5.501	1.872	−6.092	1.662

To verify that the two parameters learn complementarily not only through simple update percentage trends but also over the entire step, we measured the following three methods. First, at step t, we measure the alignment of the functional effects induced by the query-side and value-side adapters to check the complementary training for both adaptive parameters. Let $y(\theta)$ denote the model outputs (logits) for a fixed mini-batch. We form two infinitesimal "what-if" updates along the current gradients, restricted to each group:

$$\Delta\theta_Q = \epsilon \frac{g_Q}{\|g_Q\|_2}, \quad \Delta\theta_V = \epsilon \frac{g_V}{\|g_V\|_2}, \tag{28}$$

and compute the resulting output changes:

$$\Delta y_Q = y(\theta + \Delta\theta_Q) - y(\theta), \quad \Delta y_V = y(\theta + \Delta\theta_V) - y(\theta). \tag{29}$$

The cosine similarity is

$$\text{cos\_logit} = \frac{\langle \Delta y_Q, \Delta y_V \rangle}{\|\Delta y_Q\|_2 \|\Delta y_V\|_2} \in [-1, 1], \tag{30}$$

where $\langle\ ,\ \rangle$ denotes dot-product. We evaluate in inference mode during all training steps, using FP32 for the dot-product and norms, and skipping steps where either norm $< 10^{-8}$. We optionally report the same metric on the last hidden states instead of logits to reduce memory. If cos_logit ≈ 0, complementary effects acting along largely independent directions. As shown in Figure 3a,b, both experiments show similar cosine similarity distribution (near 0.5) for all steps. This means that two-parameter groups are partial alignment and the cosine similarity is not enough to evaluate interaction.

Second, we quantify the interaction between the query-side and value-side LoRA adapters with a super-additivity score

$$S_{loss} = L_{Q+V} - L_Q - L_V + L_{base} \,, \tag{31}$$

where L_{base} is the loss with both adapters disabled, L_Q and L_V enable only the query or value adapter, and L_{Q+V} enables both. By construction, $S_{loss} < 0$ indicates synergy (the joint improvement exceeds the sum of individual improvements), while $S_{loss} > 0$ suggests redundancy or antagonism. As shown in Figure 3c, across each 50 steps in the **Offline(Delay)** training in first experiment, the baseline condition (not allowing cross-contamination, red line) yields a slightly positive S_{loss} (near-additive with mild redundancy), whereas the proposed approach (allowing cross-contamination, blue line) shifts S_{loss} to a slightly negative value, indicating weak but consistent synergy. This trend persists when summarized over the final training phase (S_{loss}).

Lastly, to quantify the representational relationship between the query-side and value-side LoRA adapters, we employ Centered Kernel Alignment (CKA) and Canonical Correlation Analysis (CCA), two widely used measures for comparing neural representations. CKA [33] measures the alignment of representational subspaces independent of scaling, whereas CCA [34,35] quantifies linear correlation between feature activations. A high CKA or CCA score (closer to 1.0) indicates strong representational alignment, while lower values suggest decorrelation or complementary subspace usage. As shown in Figure 3d, the higher values of both metrics (closer to 1.0) indicate stronger alignment. The resulting mean $\pm$ std scores, CKA $= 0.7828 \pm 0.1434$ and CCA (mean) $= 0.9955 \pm 0.0183$, suggest that LoRA-Q and LoRA-V share an almost identical global subspace (high CCA) but maintain moderate local diversity (moderate CKA). This pattern implies that the two LoRA components learn complementary roles within a shared representational space, effectively attenuating cross-contamination without redundant updates.

Consequently, across training, the cosine similarity between Q- and V-induced updates concentrates around 0.5, indicating partial alignment rather than strict complementarity. Nevertheless, the synergy score S_{loss} when allowing cross-contamination is slightly negative, evidencing weak super-additivity. Furthermore, CKA and CCA scores while allowing cross-contamination indicate complementary attenuation between two adaptive parameters, enabling both adapters yields improvements beyond the sum of their individual effects.

5. Conclusions and Future Work

In this paper, we analyze scaled dot-product attention by decomposing the tensors derived from LoRA parameters and from packing that allows cross-contamination. We show that, in the absence of PAM, the cross-contamination subspace can be attenuated by the learnable subspace from LoRA-Q and LoRA-V parameters, and we then fine-tune the pre-trained models with both online and offline packing. Across experiments, training without PAM achieves losses comparable to those with PAM while reducing overall training time, especially mask generation and transmission stage. Ablations further show that LoRA-Q and LoRA-V learn jointly in a complementary and interactive manner during SFT. Therefore, eliminating PAM removes both implementation complexity and the factors of training-time overhead. Our scaling analysis suggests that the relative efficiency gain of removing PAM increases with longer context lengths L, whereas LoRA's percentage overhead decreases as the model size grows. Comprehensive quantitative evaluations on larger models (34B and 70B) and extended contexts (8k–16k tokens) remain promising directions for future work.

Author Contributions: J.W.S.: Writing—original draft, Writing—review and editing, Investigation, Visualization, Methodology, Conceptualization, Data curation. H.-Y.J.: Supervision, Review, Funding acquisition. All authors have read and agreed to the published version of the manuscript.

Funding: This research was partly funded by the IITP (Institute of Information & Communications Technology Planning & Evaluation)-ITRC (Information Technology Research Center) grant funded by the Korean government (Ministry of Science and ICT) (IITP-2025-RS-2020-II201808) (50%) and grant funded by Korea Planning & Evaluation Institute of Industrial Technology (KEIT) grant funded by the Korean government (MOTIE) (RS-2025-04003004, IS²D (Intelligent Self-evolving Security Dome) (50%).

Data Availability Statement: Publicly available datasets were analyzed in this study. The datasets are available on Hugging Face: 1. databricks/databricks-dolly-15k—https://huggingface.co/datasets/databricks/databricks-dolly-15k (accessed on 1 September 2025); 2. tatsu-lab/alpaca—https://huggingface.co/datasets/tatsu-lab/alpaca (accessed on 1 September 2025); 3. yahma/alpaca-cleane—https://huggingface.co/datasets/yahma/alpaca-cleaned (accessed on 1 September 2025). The pre-trained models used in this study are available on Hugging Face under their original licenses: 4. NousResearch/Llama-2-7b-hf—https://huggingface.co/NousResearch/Llama-2-7b-hf (accessed on 1 September 2025); 5. huggyllama/llama-7b—https://huggingface.co/huggyllama/llama-7b (accessed on 1 September 2025); 6. openlm-research/open_llama_7b—https://huggingface.co/openlm-research/open_llama_7b (accessed on 1 September 2025); 7. mistralai/Mistral-7B-v0.1—https://huggingface.co/mistralai/Mistral-7B-v0.1 (accessed on 1 September 2025).

Conflicts of Interest: The authors declare no conflicts of interest.

Appendix A. Derivation of First-Order Perturbations for LoRA-Q and LoRA-V

For a differentiable vector-valued function $f : \mathbb{R}^n \to \mathbb{R}^m$, the first-order Taylor expansion about x_0 is

$$f(x_0 + \Delta x) = f(x_0) + f^{'}(x_0)\Delta x = f(x_0) + J_f(x_0)\Delta x + R_2 \,, \tag{A1}$$

where J_f is Jacobian for $f(x)$ and R_2 is the second-order terms. Expanding to two variables, Equation (A1) is transformed as follows:

$$f(x_0 + \Delta x, y_0 + \Delta y) = f(x_0, y_0) + J_f(x_0)\Delta x + J_f(y_0)\Delta y \tag{A2}$$

$$= f(x_0, y_0) + (\frac{\partial f}{\partial x})[\Delta x] + (\frac{\partial f}{\partial y})[\Delta y] + R_2 \,. \tag{A3}$$

After moving the first term on the right side to the left side, then

$$f(x_0 + \Delta x, y_0 + \Delta y) - f(x_0, y_0) = (\frac{\partial f}{\partial x})[\Delta x] + (\frac{\partial f}{\partial y})[\Delta y] + R_2 \,. \tag{A4}$$

Substituting (f, x, y) for $(\mathcal{H}_i, S_i, v_i)$, Equation (A4) is rewritten as follows:

$$\mathcal{H}_i(S_{i0} + \Delta S_i, v_{i0} + \Delta v_i) - \mathcal{H}_i(S_{i0}, v_{i0}) = (\frac{\partial \mathcal{H}_i}{\partial S_i})[\Delta S_i] + (\frac{\partial \mathcal{H}_i}{\partial v_i})[\Delta v_i] + R_{2,i} \,. \tag{A5}$$

For LoRA applied only to Q and V, the head output is linear in V and smoothly depends on Q. Since $\Delta S_i = (\Delta q_i^T)k_i/\sqrt{d_k}$ (Equation (16)), a first-order Fréchet expansion at (q_0, v_0) leaves a remainder bounded by

$$\|R_{2,i}\|_F \leq \frac{c_1}{d_k}\|v_{i0}\|_F\|k_i\|_2^2\|\Delta q_i\|_F^2 + \frac{c_2}{\sqrt{d_k}}\|k_i\|_2\|\Delta q_i\|_F\|\Delta v_i\|_F \,, \tag{A6}$$

where $\|\cdot\|_F$ and $\|\cdot\|_2$ denote respectively Frobenius and spectral operator norm. c_1 and c_2 are the constants depending on the curvature of softmax and the norms of k_i, v_{i0}. The first term of right side in Equation (A6) is pure second-order term for Δq_i and second term is mixed term for Δq_i and Δv_i. When only Δv_i is applied, $R_{2,i} = 0$ since $\mathcal{H}_i$ is exactly linear in v_{i0}. Due to $1/\sqrt{d_k}$ scaling from S_i, the second-order term with Δq_i is damped as $1/d_k$. Since scale of mixed term is as small as $O(\|\Delta q_i\|\|\Delta v_i\|/\sqrt{d_k})$, the mixed term can be dropped in the first-order approximation.

Hence, with typical head dimensions ($d \geq 64$) and small LoRA scales, second-order terms are negligible compared to the first-order terms.

$$\mathcal{H}_i(S_{i0} + \Delta S_i, v_{i0} + \Delta v_i) - \mathcal{H}_i(S_{i0}, v_{i0}) \approx (\frac{\partial \mathcal{H}_i}{\partial S_i})[\Delta S_i] + (\frac{\partial \mathcal{H}_i}{\partial v_i})[\Delta v_i] \,. \tag{A7}$$

As defined in Equation (13), $\mathcal{H}_i$ is differentiated with respect to S_i while keeping v fixed at v_0, since the mixed terms are neglected in the first-order Taylor expansion.

Appendix B. Detailed Tables for Training Time Results

"Allow CC" denotes that cross-contamination is allowed (setting without PAM). Δ and Δ (%) follow the same definition as in Table 1. All numbers except packing time are averages; units are seconds (s) or milliseconds (ms). The 'Mask' column is computed as 'mask gen.' + 'mask h2d.' for each step.

Table A1. Detailed timing results for batch generation, per-step, and per-epoch computations for pre-trained models with 'databricks/databricks-dolly-15k' dataset and **Offline(Delay)** setting.

Models	Allow CC	Packing Time (s)	Step Time (s)	Mask gen. (ms)	Mask h2d. (ms)	Mask (ms)	Step Time -Mask (s)	Epoch Time (s)
NR	×	95.0622	0.7413	2.3773	0.4832	2.8605	0.7384	152.4633
	✓	93.6683	0.7395	0.1182	0.0877	0.2059	0.7393	152.1033
	Δ	−1.3939	−0.0018	−2.2591	−0.3955	−2.6546	0.0009	−0.3600
	Δ (%)	−1.4663	−0.2428	−95.0280	−81.8502	−92.8020	0.1157	−0.2361
HL	×	112.5778	0.7217	2.0247	0.4538	2.4785	0.7192	148.4300
	✓	111.4029	0.7241	0.1204	0.1682	0.2886	0.7238	148.9233
	Δ	−1.1749	0.0024	−1.9042	−0.2856	−2.1899	0.0046	0.4933
	Δ (%)	−1.0437	0.3325	−94.0534	−62.9352	−88.3559	0.6382	0.3323
OR	×	74.0801	0.7365	2.3400	0.3996	2.7395	0.7338	140.4333
	✓	72.8015	0.7193	0.1047	0.0781	0.1828	0.7191	137.1533
	Δ	−1.2786	−0.0172	−2.2353	−0.3215	−2.5568	−0.0146	−3.2800
	Δ (%)	−1.7260	−2.3354	−95.5256	−80.4555	−93.3272	−1.9957	−2.3356
MR	×	75.4886	0.8768	1.0961	0.2668	1.3629	0.8754	349.5333
	✓	74.3069	0.8682	0.1080	0.1013	0.2093	0.8680	346.1067
	Δ	−1.1817	−0.0086	−0.9880	−0.1656	−1.1536	−0.0074	−3.4267
	Δ (%)	−1.5654	−0.9808	−90.1469	−62.0315	−84.6430	−0.8506	−0.9803

Table A2. Detailed timing results for batch generation, per-step, and per-epoch computations for pre-trained models with 'tatsu-lab/alpaca' dataset and **Offline(Delay)** experiment setting.

Models	Allow CC	Packing Time (s)	Step Time (s)	Mask gen. (ms)	Mask h2d. (ms)	Mask (ms)	Step Time -Mask (s)	Epoch Time (s)
NR	×	427.7267	0.7496	2.3093	0.3778	2.6870	0.7469	381.2733
	✓	424.4628	0.7346	0.1704	0.1522	0.3227	0.7343	373.6800
	Δ	−3.2639	−0.0149	−2.1388	−0.2255	−2.3644	−0.0126	−7.5933
	Δ (%)	−0.7631	−2.0011	−92.6211	−59.7141	−87.9903	−1.6917	−1.9916
HL	×	374.9585	0.7537	2.5680	0.3756	2.9436	0.7508	383.3700
	✓	371.2932	0.7457	0.1661	0.1158	0.2819	0.7454	379.3267
	Δ	−3.6653	−0.0079	−2.4019	−0.2598	−2.6617	−0.0053	−4.0433
	Δ (%)	−0.9775	−1.0614	−93.5319	−69.1693	−90.4233	−0.7111	−1.0547

Table A2. *Cont.*

Models	Allow CC	Packing Time (s)	Step Time (s)	Mask gen. (ms)	Mask h2d. (ms)	Mask (ms)	Step Time -Mask (s)	Epoch Time (s)
OR	×	357.2428	0.7430	2.2023	0.4577	2.6599	0.7403	349.6967
	✓	354.3143	0.7334	0.1282	0.1501	0.2784	0.7331	345.2100
	Δ	−2.9285	−0.0095	−2.0740	−0.3075	−2.3816	−0.0072	−4.4867
	Δ (%)	−0.8198	−1.2921	−94.1788	−67.2056	−89.5334	−0.9750	−1.2830
MR	×	377.5409	0.8698	1.2790	0.2471	1.5260	0.8683	849.5233
	✓	374.1231	0.8696	0.1125	0.1227	0.2352	0.8694	849.3233
	Δ	−3.4178	−0.0002	−1.1665	−0.1243	−1.2908	0.0011	−0.2000
	Δ (%)	−0.9053	−0.0230	−91.2041	−50.3440	−84.5872	0.1256	−0.0235

Table A3. Detailed timing results for batch generation, per-step and per-epoch computations for pre-trained models with 'yahma/alpaca_cleaned' dataset and **Offline(Delay)** experiment setting.

Models	Allow CC	Packing Time (s)	Step Time (s)	Mask gen. (ms)	Mask h2d. (ms)	Mask (ms)	Step Time -Mask (s)	Epoch Time (s)
NR	×	1081.9873	0.7433	2.1832	0.3702	2.5534	0.7407	841.1333
	✓	1075.0005	0.7395	0.1253	0.0799	0.2052	0.7393	836.8600
	Δ	−6.9868	−0.0038	−2.0580	−0.2902	−2.3482	−0.0015	−4.2733
	Δ (%)	−0.6457	−0.5112	−94.2607	−78.4171	−91.9637	−0.1960	−0.5080
HL	×	1013.1999	0.7645	2.2612	0.3829	2.6441	0.7619	865.1433
	✓	1005.8142	0.7399	0.0857	0.0933	0.1791	0.7397	837.2800
	Δ	−7.3857	−0.0246	−2.1755	−0.2896	−2.4651	−0.0221	−27.8633
	Δ (%)	−0.7290	−3.2178	−96.2100	−75.6333	−93.2264	−2.9054	−3.2207
OR	×	908.6478	0.7459	2.0874	0.4092	2.4966	0.7434	781.4233
	✓	902.3665	0.7379	0.0889	0.1283	0.2172	0.7377	773.1000
	Δ	−6.2812	−0.0079	−1.9985	−0.2809	−2.2794	−0.0057	−8.3233
	Δ (%)	−0.6913	−1.0725	−95.7411	−68.6461	−91.3002	−0.7695	−1.0651
MR	×	943.3118	0.8586	1.1662	0.2702	1.4365	0.8572	1864.6200
	✓	936.4189	0.8529	0.1082	0.1044	0.2126	0.8527	1852.2800
	Δ	−6.8929	−0.0057	−1.0580	−0.1658	−1.2238	−0.0045	−12.3400
	Δ (%)	−0.7307	−0.6639	−90.7220	−61.3620	−85.2001	−0.5222	−0.6618

Table A4. Detailed timing results for batch generation, per-step and per-epoch computations with **Delay** setting. ('NousResearch/Llama-2-13b-hf' model, 'databricks/databricks-dolly-15k' dataset).

Exp.	Allow CC	Packing Time (s)	Step Time (s)	Mask gen. (ms)	Mask h2d. (ms)	Mask (ms)	Step Time -Mask (s)	Epoch Time (s)
Online (Fast)	×	62.5291	0.8278	1.1136	0.2923	1.4059	0.8264	732.3633
	✓	59.8893	0.8272	0.1190	0.1996	0.3186	0.8269	731.7800
	Δ	−2.6398	−0.0007	−0.9947	−0.0927	−1.0874	0.0005	−0.5833
	Δ (%)	−4.2218	−0.0725	−89.3139	−31.7140	−77.3384	0.0590	−0.0796
Offline (Fast)	×	94.0748	0.8306	1.1359	0.2363	1.3722	0.8292	682.4933
	✓	91.7167	0.8303	0.1792	0.2463	0.4255	0.8299	682.2200
	Δ	−2.3581	−0.0003	−0.9566	0.0100	−0.9467	0.0006	−0.2733
	Δ (%)	−2.5067	−0.0361	−84.2240	4.2319	−68.9914	0.0780	−0.0400

Table A5. Detailed timing results for batch generation, per-step and per-epoch computations for training only LoRA-Q or LoRA-V with **Offline(Delay)** setting. ('NousResearch/Llama-2-7b-hf' model, 'databricks/databricks-dolly-15k' dataset).

Trained Param.	Allow CC	Packing Time (s)	Step Time (s)	Mask gen. (ms)	Mask h2d. (ms)	Mask (ms)	Step Time -Mask (s)	Epoch Time (s)
LoRA-Q & LoRA-V	×	95.0622	0.7413	2.3773	0.4832	2.8605	0.7384	152.4633
	✓	93.6683	0.7395	0.1182	0.0877	0.2059	0.7393	152.1033
	Δ	−1.3939	−0.0018	−2.2591	−0.3955	−2.6546	0.0009	−0.3600
	Δ (%)	−1.4663	−0.2428	−95.0280	−81.8502	−92.8020	0.1157	−0.2361
LoRA-V -only	✓	93.6826	0.6357	0.1414	0.0783	0.2196	0.6355	130.7400
	Δ	−1.3796	−0.1056	−2.2360	−0.4049	−2.6409	−0.1030	−21.7233
	Δ (%)	−1.4512	−14.2452	−94.0521	−83.7955	−92.3230	−13.9428	−14.2482
LoRA-Q -only	✓	93.6948	0.6389	0.1611	0.1316	0.2927	0.6386	131.4000
	Δ	−1.3674	−0.1024	−2.2163	−0.3516	−2.5678	−0.0998	−21.0633
	Δ (%)	−1.4385	−13.8136	−93.2234	−72.7649	−89.7675	−13.5193	−13.8153

We additionally conduct experiments with $r = 4$ and $r = 16$ for **Offline(Delay)** setting as mentioned in Section 4.2. Both experiments fine-tunes the same pre-trained model (NousResearch/Llama-2-7b-hf) and dataset (databricks/databricks-dolly-15k).

Table A6. Detailed timing results for batch generation, per-step, and per-epoch computations for training r-rank parameters.

r	Allow CC	Packing Time (s)	Step Time (s)	Mask gen. (ms)	Mask h2d. (ms)	Mask (ms)	Step Time -Mask (s)	Epoch Time (s)
8	×	95.0622	0.7413	2.3773	0.4832	2.8605	0.7384	152.4633
	✓	93.6683	0.7395	0.1182	0.0877	0.2059	0.7393	152.1033
	Δ	−1.3939	−0.0018	−2.2591	−0.3955	−2.6546	0.0009	−0.3600
	Δ (%)	−1.4663	−0.2428	−95.0280	−81.8502	−92.8020	0.1157	−0.2361
4	✓	93.6475	0.7240	0.0845	0.0778	0.1623	0.7238	148.9000
	Δ	−1.4147	−0.0173	−2.2928	−0.4054	−2.6982	−0.0146	−3.5633
	Δ (%)	−1.4881	−2.3337	−96.4455	−83.8990	−94.3262	−1.9774	−2.3372
16	✓	93.6269	0.7251	0.0510	0.1238	0.1748	0.7249	149.1200
	Δ	−1.4353	−0.0163	−2.3263	−0.3594	−2.6857	−0.0135	−3.3433
	Δ (%)	−1.5099	−2.1854	−97.8547	−74.3791	−93.8892	−1.8301	−2.1929

Table A7. Average and standard deviation of loss and parameter-update percentage differences ($\Delta = ✓ - ×$) per step for different r-rank. All parameter-update values are reported as percentages (%).

r	Δ Step Loss		Δ Epoch Last Loss		Δq_rel_mean		Δq_p95		Δq_max		Δv_rel_mean		Δv_p95		Δv_max	
	Avg.	Std.	Avg.	Std.	Avg.	Std.	Avg.	Std.	Avg.	Std.	Avg.	Std.	Avg.	Std.	Avg.	Std.
8	−0.045	0.101	−0.004	0.005	−1.351	0.385	−2.668	0.791	−5.935	2.225	0.260	0.269	0.327	0.304	0.654	1.295
4	−0.041	0.099	−0.002	0.004	−0.518	0.255	−1.525	0.692	−4.897	1.877	1.182	0.511	1.333	0.567	1.045	1.056
16	−0.047	0.100	−0.003	0.005	−1.910	0.574	−3.131	0.875	−6.418	2.363	−0.333	0.147	−0.233	0.179	−0.602	0.469

References

1. Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A.N.; Kaiser, Ł.; Polosukhin, I. Attention is all you need. In Proceedings of the Advances in Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA, 4–9 December 2017.
2. Touvron, H.; Lavril, T.; Izacard, G.; Martinet, X.; Lachaux, M.A.; Lacroix, T.; Rozière, B.; Goyal, N.; Hambro, E.; Azhar, F.; et al. LLaMA: Open and Efficient Foundation Language Models. *arXiv* **2023**, arXiv:2302.13971. [CrossRef]
3. Touvron, H.; Martin, L.; Stone, K.; Albert, P.; Almahairi, A.; Babaei, Y.; Bashlykov, N.; Batra, S.; Bhargava, P.; Bhosale, S.; et al. Llama 2: Open Foundation and Fine-Tuned Chat Models. *arXiv* **2023**, arXiv:2307.09288. [CrossRef]
4. Radford, A.; Narasimhan, K.; Salimans, T.; Sutskever, I. Improving Language Understanding by Generative Pre-Training. *Preprint* **2018**, 1–12.
5. Radford, A.; Wu, J.; Child, R.; Luan, D.; Amodei, D.; Sutskever, I. Language models are unsupervised multitask learners. *OpenAI Blog* **2019**, *1*, 9.
6. Brown, T.B.; Mann, B.; Ryder, N.; Subbiah, M.; Kaplan, J.; Dhariwal, P.; Neelakantan, A.; Shyam, P.; Sastry, G.; Askell, A.; et al. Language Models are Few-Shot Learners. *Adv. Neural Inf. Process. Syst.* **2020**, *33*, 1877–1901.
7. Krell, M.M.; Kosec, M.; Perez, S.P.; Fitzgibbon, A. Efficient Sequence Packing without Cross-contamination: Accelerating Large Language Models without Impacting Performance. *arXiv* **2021**, arXiv:2107.02027. [CrossRef]
8. Ge, H.; Feng, J.; Huang, Q.; Fu, F.; Nie, X.; Zuo, L.; Lin, H.; Cui, B.; Liu, X. ByteScale: Efficient Scaling of LLM Training with a 2048K Context Length on More Than 12,000 GPUs. *arXiv* **2025**, arXiv:2502.21231. [CrossRef]
9. Raffel, C.; Shazeer, N.; Roberts, A.; Lee, K.; Narang, S.; Matena, M.; Zhou, Y.; Li, W.; Liu, P.J. Exploring the limits of transfer learning with a unified text-to-text transformer. *J. Mach. Learn. Res.* **2020**, *21*, 5485–5551.
10. Kosec, M.; Fu, S.; Krell, M.M. Packing: Towards 2x NLP BERT Acceleration. 2021. Available online: https://openreview.net/forum?id=3_MUAtqR0aA (accessed on 1 September 2025).
11. Zheng, Y.; Zhang, R.; Zhang, J.; Ye, Y.; Luo, Z. LlamaFactory: Unified Efficient Fine-Tuning of 100+ Language Models. In Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics, Bangkok, Thailand, 11–16 August 2024; pp. 400–410.

12. Bai, Y.; Lv, X.; Zhang, J.; He, Y.; Qi, J.; Hou, L.; Tang, J.; Dong, Y.; Li, J. LongAlign: A Recipe for Long Context Alignment of Large Language Models. In Proceedings of the Findings of the Association for Computational Linguistics: EMNLP 2024, Miami, FL, USA, 12–16 November 2024; pp. 1376–1395.
13. Wang, S.; Wang, G.; Wang, Y.; Li, J.; Hovy, E.; Guo, C. Packing Analysis: Packing Is More Appropriate for Large Models or Datasets in Supervised Fine-tuning. In Proceedings of the Findings of the Association for Computational Linguistics: ACL 2025, Vienna, Austria, 27 July–1 August 2025; pp. 4953–4967.
14. Yao, Y.; Tan, J.; Liang, K.; Zhang, F.; Niu, Y.; Hu, J.; Gong, R.; Lin, D.; Xu, N. Hierarchical Balance Packing: Towards Efficient Supervised Fine-tuning for Long-Context LLM. *arXiv* **2025**, arXiv:2503.07680. [CrossRef]
15. Staniszewski, K.; Tworkowski, S.; Jaszczur, S.; Zhao, Y.; Michalewski, H.; Kuciński, Ł; Miłoś, P. Structured Packing in LLM Training Improves Long Context Utilization. In Proceedings of the AAAI Conference on Artificial Intelligence, Philadelphia, PA, USA, 25 February–4 March 2025; pp. 25201–25209.
16. Dong, J.; Jiang, L.; Jin, W.; Cheng, L. Threshold Filtering Packing for Supervised Fine-Tuning: Training Related Samples within Packs. In Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies, Albuquerque, NM, USA, 29 April–4 May 2025; pp. 4422–4435.
17. Ding, H.; Wang, Z.; Paolini, G.; Kumar, V.; Deoras, A.; Roth, D.; Soatto, S. Fewer truncations improve language modeling. In Proceedings of the 41st International Conference on Machine Learning, Vienna, Austria, 21–27 July 2024; Volume 439, pp. 11030–11048.
18. Kundu, A.; Lee, R.D.; Wynter, L.; Ganti, R.K.; Mishra, M. Enhancing training efficiency using packing with flash attention. *arXiv* **2024**, arXiv:2407.09105. [CrossRef]
19. Han, I.; Jayaram, R.; Karbasi, A.; Mirrokni, V.; Woodruff, D.P.; Zandieh, A. Hyperattention: Long-context attention in near-linear time. *arXiv* **2023**, arXiv:2310.05869. [CrossRef]
20. Han, E.J.; Shen, Y.; Wallis, P.; Allen-Zhu, Z.; Li, Y.; Wang, S.; Wang, L.; Chen, W. LoRA: Low-Rank Adaptation of Large Language Models. In Proceedings of the International Conference on Learning Representations, Kigali, Rwanda, 1–5 May 2023.
21. Zi, B.; Qi, X.; Wang, L.; Wang, J.; Wong, K.-F.; Zhang, L. Delta-lora: Fine-tuning high-rank parameters with the delta of low-rank matrices. *arXiv* **2023**, arXiv:2309.02411. [CrossRef]
22. Lin, C.; Li, L.; Li, D.; Zou, J.; Xue, W.; Guo, Y. Nora: Nested low-rank adaptation for efficient fine-tuning large models. *arXiv* **2024**, arXiv:2408.10280. [CrossRef]
23. Mao, Y.; Huang, K.; Guan, C.; Bao, G.; Mo, F.; Xu, J. DoRA: Enhancing Parameter-Efficient Fine-Tuning with Dynamic Rank Distribution. In Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics, Bangkok, Thailand, 11–16 August 2024; Volume 1, pp. 11662–11675.
24. Martins, A.; Astudillo, R. From softmax to sparsemax: A sparse model of attention and multi-label classification. In Proceedings of the International Conference on Machine Learning, New York, NY, USA, 19–24 June 2016; pp. 1614–1623.
25. Yin, Q.; He, X.; Zhuang, X.; Zhao, Y.; Yao, J.; Shen, X.; Zhang, Q. StableMask: Refining causal masking in decoder-only transformer. In Proceedings of the 41st International Conference on Machine Learning, Vienna, Austria, 21–27 July 2024; Volume 2354, pp. 57033–57052.
26. Yao, X.; Qian, H.; Hu, X.; Xu, G.; Liu, W.; Luan, J.; Wang, B.; Liu, Y. Theoretical Insights into Fine-Tuning Attention Mechanism: Generalization and Optimization. *arXiv* **2025**, arXiv:2410.02247. [CrossRef]
27. Dettmers, T.; Pagnoni, A.; Holtzman, A.; Zettlemoyer, L. QLORA: Efficient finetuning of quantized LLMs. In Proceedings of the 37th International Conference on Neural Information Processing Systems, New Orleans, LA, USA, 10–16 December 2023; Volume 441, pp. 10088–10115.
28. Conover, M.; Hayes, M.; Mathur, A.; Xie, J.; Wan, J.; Shah, S.; Ghodsi, A.; Wendell, P.; Zaharia, M.; Xin, R. Free Dolly: Introducing the World's First Truly Open Instruction-Tuned LLM. 2023. Available online: https://www.databricks.com/blog/2023/04/12/dolly-first-open-commercially-viable-instruction-tuned-llm (accessed on 1 September 2025).
29. Taori, R.; Gulrajani, I.; Zhang, T.; Dubois, Y.; Li, X.; Guestrin, C.; Liang, P.; Hashimoto, T.B. Stanford Alpaca: An Instruction-Following LLaMA Model. GitHub Repository. 2023. Available online: https://github.com/tatsu-lab/stanford_alpaca (accessed on 1 September 2025).
30. Wolf, T.; Debut, L.; Sanh, V.; Chaumond, J.; Delangue, C.; Moi, A.; Cistac, P.; Rault, T.; Louf, R.; Funtowicz, M.; et al. Transformers: State-of-the-Art Natural Language Processing. In Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations, Online, 16–20 November 2020; pp. 38–45.
31. Jiang, A.Q.; Sablayrolles, A.; Mensch, A.; Bamford, C.; Chaplot, D.S.; de las Casas, D.; Bress, F.; Lengyel, G.; Lample, G.; Saulnier, L.; et al. Mistral 7B. *arXiv* **2023**, arXiv:2310.06825. [CrossRef]
32. Hendrycks, D.; Burns, C.; Basart, S.; Zou, A.; Mazeika, M.; Song, D.; Steinhardt, J. Measuring massive multitask language understanding. *arXiv* **2020**, arXiv:2009.03300. [CrossRef]
33. Kornblith, S.; Norouzi, M.; Lee, H.; Hinton, G. Similarity and matching of neural network representations. *arXiv* **2019**, arXiv:1905.00414. [CrossRef]

34. Raghu, M.; Gilmer, J.; Yosinski, J.; Sohl-Dickstein, J. Svcca: Singular vector canonical correlation analysis for deep learning dynamics and interpretability. In Proceedings of the Advances in Neural Information Processing Systems (30), Long Beach, CA, USA, 4–9 December 2017.
35. Morcos, A.; Raghu, M.; Bengio, S. Insights on representational similarity in neural networks with canonical correlation. In Proceedings of the Advances in Neural Information Processing Systems (31), Montreal, QC, Canada, 3–8 December 2018.

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.