mathematics

MDPI

Article

Analysis and Mitigation of Performance Degradation from Layer Insertions into the Middle of Pre-Trained Language Models

Gyunyeop Kim and Sangwoo Kang *

Department of Computing, Gachon University, Sujeong-gu, Seongnam-si 13120, Republic of Korea; gyop0817@gachon.ac.kr

* Correspondence: swkang@gachon.ac.kr; Tel.: +82-031-750-8669

Abstract

Full fine-tuning of pre-trained models sometimes requires inserting trainable layers into the middle of a pre-trained backbone, but such middle-layer insertion can severely degrade downstream performance. We hypothesize that this degradation arises because conventionally inserted layers, when randomly initialized and combined with output-side activation, perturb intermediate representations before the pre-trained model has adapted. We study this phenomenon across natural language processing and computer vision benchmarks by varying insertion locations, the number of inserted layers, and activation designs. To address this problem, we propose a practical stabilization method for middle-layer insertion under full fine-tuning: a bias-free inserted layer with unit initialization and weight-side activation. This design is intended to remain closer to an identity-like transformation at initialization, thereby reducing initialization-time perturbation rather than claiming exact preservation of the original representations. In the tested DeBERTa-v3, T5-base, and ViT-base settings, the proposed method substantially mitigates the severe degradation caused by naive middle-layer insertion and maintains performance close to the no-added-layer baseline, including settings with up to 24 inserted layers.

Keywords: fine-tuning; transfer learning; weight initialization; activation function; transformer; language model

MSC: 68T50

1. Introduction

Pre-trained transformer models have become a standard foundation for a wide range of downstream tasks in natural language processing and computer vision [1,2]. Models such as BERT [3], RoBERTa [4], and Swin Transformer [5] are typically adapted through fine-tuning, often together with task-specific architectural modifications.

A common practice in full fine-tuning is to attach trainable components outside the pre-trained backbone, such as classification heads or task-specific output layers [3,6,7]. However, some downstream applications may require computation to be introduced not only at the output side but also inside the backbone itself. In such cases, one may wish to insert trainable layers between pre-trained transformer blocks.

Despite its practical relevance, middle-layer insertion is often unstable under full fine-tuning. When a newly inserted layer is placed between well-trained blocks, the next pre-trained block no longer receives the representation distribution it was optimized for during pre-training. Instead, it receives a transformed representation produced by a newly

Check for updates

Academic Editor: Raymond Lee

Received: 25 March 2026
Revised: 16 April 2026
Accepted: 17 April 2026
Published: 20 April 2026

Copyright: © 2026 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.

initialized module. This mismatch can degrade downstream performance and may hinder the stable reuse of pre-trained knowledge.

A likely reason for this instability is the design of the inserted layer at initialization. In conventional settings, the inserted layer is randomly initialized and followed by an output-side activation function. Even when the incoming representation is well formed, such a layer can perturb intermediate vectors before the model has adapted, thereby introducing a source of interference during the early stage of full fine-tuning. As more inserted layers are introduced, this perturbation can accumulate across blocks and make optimization increasingly difficult.

To illustrate the issue, consider a pre-trained transformer model P composed of blocks $\{P_1, P_2, \ldots, P_n\}$. If a randomly initialized inserted layer is placed between P_3 and P_4, then the output of P_3 is no longer passed directly to P_4. Instead, P_4 receives a transformed representation whose distribution may differ substantially from the one seen during pre-training. This observation motivates our central hypothesis: performance degradation from middle-layer insertion is driven primarily by initialization-time perturbation of intermediate representations rather than by increased depth alone.

Activation design can further amplify this problem. When a nonlinearity is applied to the output of an inserted layer, it can alter intermediate vectors even more aggressively, making the inserted module more disruptive before full-model adaptation becomes stable.

Rather than introducing a new parameter-efficient fine-tuning framework or a theoretically distinct function-preserving paradigm, this study focuses on a practical stabilization problem: how to make middle-layer insertion behave more stably under full fine-tuning. In spirit, this setting is related to prior work on function-preserving or identity-style network expansion, but our goal is different: we do not aim to guarantee exact function preservation under architecture morphing, but to reduce initialization-time perturbation when trainable layers are inserted into an already pre-trained transformer backbone and then optimized with full fine-tuning. Specifically, we study whether the degradation caused by naive middle-layer insertion can be mitigated by making the inserted layer less disruptive at initialization. To this end, we propose a bias-free inserted layer with unit initialization and weight-side activation. Accordingly, the contribution of this work is not to claim a new fine-tuning framework, but to show that controlling initialization-time perturbation can substantially mitigate the severe degradation caused by naive middle-layer insertion.

2. Related Works

2.1. Transfer Learning and Pre-Trained Models

Transfer learning reuses knowledge learned from one task or domain to improve performance on another [2]. In computer vision, pre-training on large-scale datasets has long been effective for downstream tasks such as fine-grained image classification, segmentation, and detection [8–10]. The same general principle underlies modern transformer-based models, where large-scale pre-training provides reusable representations that can later be adapted through fine-tuning.

In natural language processing, pre-trained language models are typically learned from large unlabeled corpora using self-supervised objectives. A representative example is masked language modeling, in which some input tokens are masked and the model is trained to reconstruct them [3]. These pre-trained representations have made fine-tuning a standard paradigm for downstream adaptation across a wide range of tasks.

2.2. Fine-Tuning with Architectural Modification

Fine-tuning often combines parameter updating with task-specific architectural modification. A common design is to attach trainable components outside the pre-trained back-

bone, such as classification heads or output-side task layers. For example, SEQSON [11] adds a linear classifier on top of the encoder in abstractive summarization, and DyLex [12] incorporates additional task-specific representations for sequence labeling. These studies show that adding trainable computation can improve task adaptation, but they primarily focus on modifications around the output side or on task-specific adaptation modules.

In contrast, our focus is on a different problem setting: stabilizing trainable layer insertion between pre-trained transformer blocks during full fine-tuning rather than proposing a new parameter-efficient fine-tuning framework or a theoretically distinct adaptation paradigm. In this sense, our method is better understood as a practical stabilization approach for middle-layer insertion, motivated by the need to preserve pre-trained behavior more faithfully at initialization when the backbone architecture itself is modified.

2.3. Function-Preserving Expansion and Identity-Style Stabilization

Prior work has studied how to expand a trained neural network while preserving its original function at initialization. Net2Net [13] introduced function-preserving transformations for deepening or widening a network, and Network Morphism [14] generalized this idea to broader architecture-morphing operations. These studies are related in spirit to our work because they emphasize that newly introduced layers should avoid disrupting already learned behavior at initialization.

At the same time, our setting differs from this literature in an important way. The goal of function-preserving expansion is typically to construct a deeper or wider network that initially reproduces the behavior of an existing trained model. By contrast, our goal is not to guarantee exact function preservation under architecture morphing, but to reduce initialization-time perturbation when trainable layers are inserted into an already pre-trained transformer backbone and then optimized with full fine-tuning. Accordingly, the proposed method is better interpreted as a practical stabilization design for middle-layer insertion under full-model adaptation than as a new function-preserving expansion framework.

More broadly, identity-style parameterizations have also been used to improve optimization stability in deep networks. Identity mappings in residual networks [15] and zero-initialized residual scaling in ReZero [16] both reflect the general principle that keeping newly introduced transformations close to an identity-like path can make optimization more stable. Similarly, parameter-efficient adaptation methods such as LoRA [17] use update parameterizations that start from a near-zero effect on the backbone. Our method is closest to this broader intuition, but differs in that it addresses explicitly inserted layers inside a pre-trained backbone under full fine-tuning, rather than residual gating or low-rank parameter updates in a parameter-efficient setting.

2.4. Weight Initialization and Stability of Inserted Layers

Weight initialization plays a central role in the optimization of deep neural networks. Poor initialization can hinder training by causing unstable gradients or unfavorable optimization dynamics. Widely used initialization schemes include Xavier initialization [18] and He initialization [19], which were designed to support stable signal propagation under standard layer compositions. Xavier initialization sets the variance according to the input and output dimensionalities [18], whereas He initialization is tailored for ReLU-based networks [19,20].

However, the role of initialization changes when a new trainable layer is inserted into the middle of a pre-trained model and the whole model is then optimized jointly. In that setting, the issue is not only whether the inserted layer is trainable, but also whether its initialization perturbs intermediate representations before the model backbone has adapted

to the architectural modification. This perspective motivates our emphasis on initialization and activation design as practical factors that determine how disruptive naive middle-layer insertion becomes at the start of full fine-tuning.

3. Proposed Method

The proposed method aims to insert an additional layer between the middle layers of a pre-trained model. When inserting an additional layer, we design the layer so that it does not act as noise to the pre-trained information during the early stage of full fine-tuning. In a conventional formulation, an additional layer is defined together with an activation function and an initialization scheme. For example, when a fully connected neural network is used as an additional layer, it operates on learnable parameters W and b through the affine transformation $WX + b$, and the activation function is then applied to the output, i.e., $\alpha(WX + b)$. Here, W and b denote the learnable weight and bias parameters, respectively, X is the input, and α is the activation function. In practice, W and b are commonly initialized using methods such as Xavier initialization or He initialization.

The main idea of our method is to reduce the deformation and randomization of intermediate values caused by the initialization and activation design of an additional layer. In many conventional initialization methods, the parameters of the newly added layer are initialized with random values. As a result, even when the input representation is well trained, the output of the additional layer may initially have random characteristics. If such a layer is inserted between the middle layers of a pre-trained model, the next pre-trained layer receives an input distribution that differs from the one seen during pre-training. During the early stage of full fine-tuning, this mismatch can hinder stable adaptation and may lead to the loss of pre-trained information. In addition, when the activation function is applied to the output of the additional layer, the resulting vector can be altered even further, which can make the inserted module more disruptive to the pre-trained representation flow.

To address this issue, we construct an additional layer that is intended to interfere as little as possible with the existing pre-trained information at initialization. Rather than claiming an entirely new fine-tuning framework, the proposed methodology is intended as a practical stabilization method for middle-layer insertion under full fine-tuning. The proposed design aims to allow the additional layer to be trained while reducing unnecessary perturbation to the pre-trained model in the early phase of optimization.

3.1. Position of Additional Layers

In this section, we describe the positions of the additional layers used in our experiments before introducing the detailed formulation of the proposed method. The model structures considered in the experiments are illustrated in Figure 1. In this study, an additional layer is inserted between the middle layers of transformer blocks in one of three ways: (1) only in multi-head self-attention, (2) only after the feed-forward network, and (3) in both positions. These positions were chosen as representative internal insertion points because they perturb the two major computational paths inside a transformer block: the attention branch and the feed-forward branch. In this sense, they serve as stress-test settings for middle-layer insertion, allowing us to examine how perturbations introduced at different internal paths affect full fine-tuning.

In Multi-head Self-attention. The structure with an additional layer inserted in multi-head self-attention is shown in Figure 1a. An additional layer A is positioned after the linear projections that generate Q_i, K_i, and V_i in multi-head self-attention, where i denotes the head index. The inputs Q_i, K_i, and V_i are passed through the additional layer to generate new Q'_i, K'_i, and V'_i, which are then used to perform multi-head self-attention. This insertion point directly perturbs the Q/K/V projection path and therefore can affect both

attention-score formation and value propagation inside the attention branch. The equations are as follows:

$$Q_i = W_i^Q E \tag{1}$$

$$K_i = W_i^K E \tag{2}$$

$$V_i = W_i^V E \tag{3}$$

$$Q'_i = A_i^{Q'}(Q_i) \tag{4}$$

$$K'_i = A_i^{K'}(K_i) \tag{5}$$

$$V'_i = A_i^{V'}(V_i) \tag{6}$$

$$Attention(Q', K', V') = Softmax(\frac{Q'K'^T}{\sqrt{d_{K'}}})V' \tag{7}$$

$$head_i = Attention(Q'_i, K'_i, V'_i) \tag{8}$$

$$MultiHead(Q', K', V') = Concat(head_1, head_2, ..., head_h)W^O \tag{9}$$

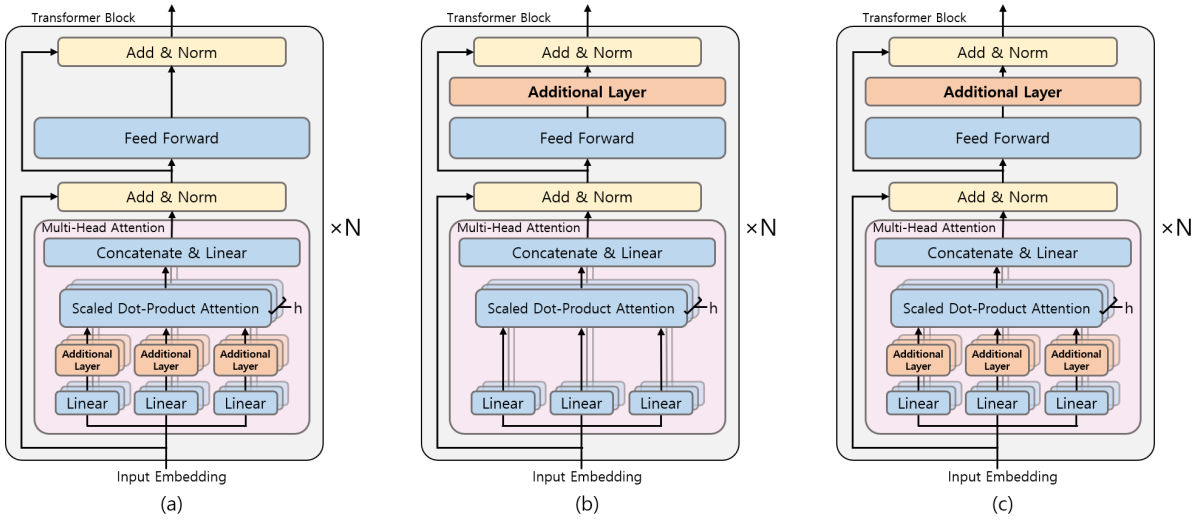


Figure 1. Model structure with an additional layer. (**a**) In multi-head self-attention. (**b**) After feed-forward network. (**c**) Both.

These equations follow the multi-head attention formulation of the Transformer [1]. Here, E denotes the input embedding, and W_i^Q, W_i^K, W_i^V, and W^O are the learnable parameters of multi-head self-attention in the pre-trained model. i denotes the head index, and h denotes the number of heads. In addition, $A_i^{Q'}(\cdot)$, $A_i^{K'}(\cdot)$, and $A_i^{V'}(\cdot)$ denote the additional layers. The detailed structure of A is described in Section 3.2.

After Feed-forward Network. The position of the additional layer after the feed-forward network is shown in Figure 1b. In this setting, the additional layer is inserted after the feed-forward network within the transformer block. This insertion point perturbs the feed-forward branch after attention and can therefore affect the post-attention token-wise channel transformation. The equation is as follows:

$$FFN'(x) = A(max(0, xW_1 + b_1)W_2 + b_2) \tag{10}$$

This equation is based on the Transformer [1]. Specifically, $FFN'(x)$ denotes the feed-forward network with an additional layer A inserted after the original $FFN(x)$ formulation.

Here, W_1, W_2, b_1, and b_2 are the learnable parameters of the pre-trained model, and $A(\cdot)$ denotes the additional layer.

Both. The Both setting inserts an additional layer in both multi-head self-attention and after the feed-forward network. Because this setting simultaneously perturbs both major internal paths of the transformer block, we include it as a stronger stress-test condition for middle-layer insertion. The corresponding structure is shown in Figure 1c.

3.2. Inserted Additional Layer

The proposed method inserts an additional layer in a way that is intended to preserve pre-trained information as much as possible during the early stage of full fine-tuning. Our starting point is the observation that, in a pre-trained model, each layer receives an input distribution that has already been shaped by pre-training and passes a correspondingly well formed output to the next layer. When an additional layer is inserted between such layers, the next pre-trained layer no longer receives the original intermediate representation but instead receives a transformed output produced by the newly inserted module.

The proposed method focuses on two main reasons why this transformed output can become disruptive at initialization: random weight initialization and output-side activation. When the additional layer is initialized with random values, even a well-trained input representation can be mapped to an output with untrained characteristics. If such a layer is inserted between the middle layers of a pre-trained model, the next pre-trained layer receives an input distribution that differs from the one seen during pre-training. During the early stage of full fine-tuning, this mismatch can hinder stable adaptation and may lead to the loss of pre-trained information. In addition, applying an activation function to the output of the additional layer can further change the intermediate representation, making the inserted module more disruptive to the pre-trained representation flow. As the number of additional layers increases, this effect can accumulate across layers, which may further degrade the fine-tuning process. This phenomenon is examined experimentally in Section 4.4.

To address this issue, we construct an additional layer that is intended to interfere as little as possible with the existing pre-trained information at initialization. The overall structure of the proposed methodology is illustrated in Figure 2, which shows the case where an additional layer is inserted in the multi-head self-attention position. Unlike the conventional formulation $\alpha(Wx + b)$, the proposed layer does not apply the activation function to the output vector. Instead, it applies the activation function directly to the weight matrix before multiplication.

Unit Initialization. In this study, the learnable parameters of the additional layer are initialized using a unit tensor. A unit tensor (matrix) has the property of preserving the input under matrix multiplication. If a tensor Y is multiplied by a unit tensor U, then the output remains Y (i.e., $Y \cdot U = Y$). Therefore, we initialize the learnable parameter W^A of the additional layer as a unit tensor. With this initialization, the next pre-trained layer does not initially receive a randomly transformed vector from the inserted layer. This makes it possible to use pre-trained information more faithfully in the early stage of full fine-tuning and reduces the risk of losing pre-trained information during training. The initial weight W_0^A of the additional layer is defined as follows:

$$W_0^A = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \end{bmatrix} \tag{11}$$

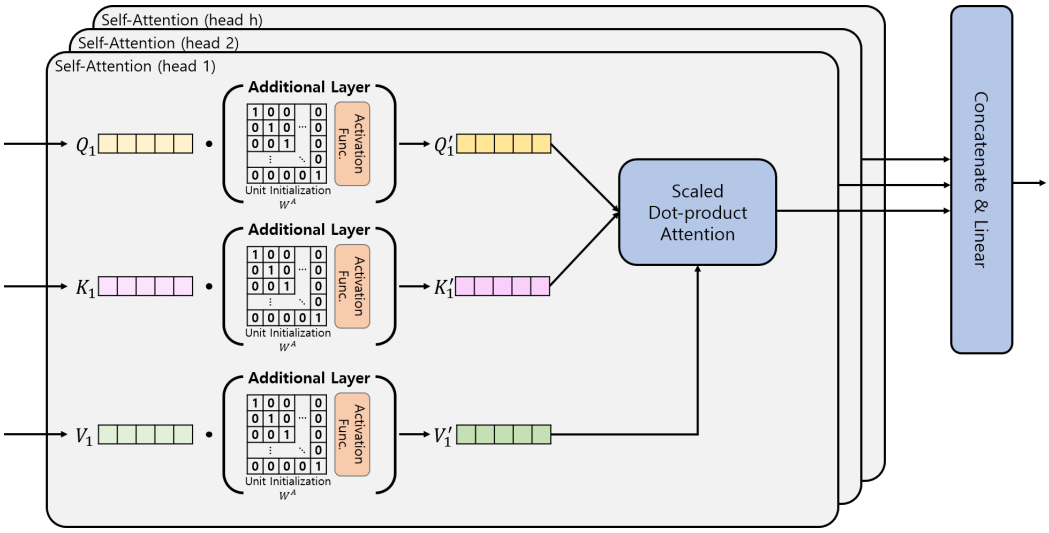


Figure 2. Additional layer structure when the layer position is in multi-head self-attention. The inserted layer A is bias-free, and the activation function is applied to the weight matrix rather than to the output vector.

Weight-side Activation. In the proposed method, the activation function is not applied to the output of the additional layer. Instead, it is applied to the learnable parameter W^A of the additional layer. In our experiments, we used GeLU [21] as the activation function. When GeLU is applied to a unit tensor, the resulting matrix remains close to the original unit tensor, although it is not exactly identical. Because of this, even when the activation function is applied at initialization, the resulting transformation changes only slightly compared with conventional output-side activation. This helps mitigate the extent to which the inserted layer acts as noise during the early stage of full fine-tuning and reduces the risk of disrupting pre-trained information. We emphasize that this does not preserve the identity exactly, because applying GeLU to W^A changes the diagonal entries of the unit tensor. Rather, the proposed parameterization keeps the transformation closer to the identity at initialization and reduces the magnitude of the induced perturbation. The structure of additional layer A is as follows:

$$A(x) = \alpha(W^A)x \tag{12}$$

Here, $x \in \mathbb{R}^{d^x}$ denotes the input vector to the additional layer, where d^x is the input dimension. The parameter $W^A \in \mathbb{R}^{d^x \times d^x}$ is the learnable weight of the additional layer, and $\alpha(\cdot)$ denotes the activation function. In this formulation, $\alpha(\cdot)$ is applied element-wise to the entries of W^A before matrix multiplication, and the inserted layer is bias-free. In implementation, because the input is represented as a row-wise batch tensor, the same operation is computed as $\texttt{torch.matmul}(x, \alpha(W^A)^\top)$, which is equivalent to the formulation above. This clarification distinguishes the proposed weight-side activation from the conventional output-side activation applied after an affine transformation.

We used this bias-free setting consistently in all experiments, including Q/K/V-side insertion and FFN-side insertion. The omission of bias was intended to minimize additive shifts at initialization, so that the inserted layer would perturb the hidden representation as little as possible during the early stage of full fine-tuning. In this sense, the proposed method introduces nonlinearity in a restricted manner in weight space rather than directly transforming the intermediate vector through output-side activation.

In a representative DeBERTa-MRPC setting, we additionally measured the initialization-time relative perturbation $\|A(x) - x\|_2 / \|x\|_2$. The naive Xavier + output GELU insertion showed 1.4106 ± 0.0045, whereas the proposed unit + weight GELU setting showed

0.1587 ± 0.0000, corresponding to an approximately 88.8% reduction relative to the naive insertion. These results support the interpretation that the proposed parameterization remains closer to the identity at initialization and substantially reduces perturbation, although we do not claim exact identity preservation (see Table 1).

Table 1. Summary of the datasets used in the experiments. # denotes the number of examples.

Corpus	Task	# Train	# Dev	# Label	Metrics
		GLUE			
CoLA [22]	Sentence acceptability judgment	8.5 k	1 k	-	Matthews corr.
SST-2 [23]	Sentiment analysis	67.3 k	0.9 k	2	Accuracy
MRPC [24]	Paraphrasing/sentence similarity	3.7 k	0.4 k	2	Accuracy
STS-B [25]	Paraphrasing/sentence similarity	5.6 k	1.5 k	-	Pearson corr.
QNLI [26]	Natural language inference	105 k	5.5 k	2	Accuracy
RTE [26]	Natural language inference	2.5 k	0.3 k	2	Accuracy
WNLI [27]	Natural language inference	0.6 k	0.2 k	2	Accuracy
		Text Generation			
CNN/DM [28]	Text summarization	287 k	11.5 k	-	ROUGE
WMT16 en-ro [29]	Translation	610 k	2 k	-	BLEU
		Image Classification			
ImageNet [30]	Image classification	1.2 m	50 k	1000	Accuracy
CIFAR100 [31]	Image classification	50 k	10 k	100	Accuracy

4. Results

4.1. Dataset

We conducted experiments on both natural language processing (NLP) and computer vision (CV) tasks to evaluate the proposed methodology. Because the proposed method modifies the internal structure of a pre-trained model, we used benchmark datasets that are commonly employed to evaluate the downstream performance of pre-trained models.

For NLP understanding tasks, we used the GLUE benchmark [26] to evaluate transformer-based language models. GLUE is a standard benchmark for measuring natural language understanding performance across several classification and similarity tasks.

To evaluate generation performance in encoder–decoder language models, we used CNN/DM [28] and WMT16 English–Romanian [29]. CNN/DM is used to evaluate abstractive summarization, whereas WMT16 English–Romanian is used to evaluate machine translation performance.

For CV, we used the ImageNet [30] and CIFAR100 [31] image classification datasets to evaluate the effect of the proposed method on pre-trained vision models. For GLUE, we report results on the validation split rather than on the hidden test set, and hyperparameter tuning and checkpoint selection were performed under the same validation protocol. The remaining datasets were also evaluated on their standard validation splits, as summarized in Table 1.

4.2. Comparison Method

Baseline. In this study, we evaluate a methodology that inserts an additional layer into a pre-trained model. For NLP, we experimented with DeBERTaV3$_{large}$ [32] and T5$_{base}$ [33]. For CV, we used ViT$_{base}$ [34]. As the primary reference, we compare each modified model against the corresponding pre-trained model fine-tuned without an additional layer. To ensure a fair comparison, we implemented the no-added-layer baseline ourselves under the same experimental framework. For all main comparisons, hyperparameters were selected

on the validation split, and the reported results correspond to the best hyperparameter setting for each method under a single training run.

We also compare against a naive insertion setting in which an additional layer is inserted without the proposed design. This comparison allows us to test whether the proposed method mitigates the degradation caused by conventional middle-layer insertion. In the original baseline setting, the inserted layer was initialized with Xavier initialization [18] and followed by a GELU [21] activation applied to the output. The additional layer was inserted into the bottom k transformer blocks, where k corresponds to more than 80% of the total number of transformer blocks. In the default naive insertion baseline, the inserted layer is implemented as a bias-free affine transformation with Xavier uniform initialization, followed by an output-side GELU activation. Importantly, the naive insertion, freeze-pre-trained-model, and proposed method settings use the same internal insertion structure and the same number of inserted parameters. Therefore, the direct comparison among these settings is conducted under a matched internal insertion budget.

Freeze Pre-trained Model. The structure used in the proposed method is related to the general idea of inserting an additional layer into the middle of a pre-trained model, which is also seen in adapter-style approaches. Adapter methods [35], including Pfeiffer Adapter [36] and LoRA [17], insert trainable modules into intermediate layers of a pre-trained model. Unlike our setting, however, these methods typically freeze the backbone and learn only the inserted parameters rather than performing full fine-tuning of all model parameters.

In this comparison, we freeze the parameters of the pre-trained model and train only the inserted additional layer. The inserted layer uses Xavier initialization and an output-side ReLU [20] activation. This setting allows us to examine whether the inserted layer can adapt to a frozen backbone without additional adaptation mechanisms. As with the naive insertion and proposed method settings, this comparison uses the same inserted-layer architecture and insertion positions, so that the main differences lie in the training strategy and initialization/activation design rather than in parameter size.

4.3. Main Result

Table 2 reports the GLUE benchmark results for DeBERTaV3$_{large}$ [32]. The naive insertion setting without the proposed method shows substantial performance degradation relative to the no-added-layer baseline. On CoLA, for example, the naive insertion setting drops to 0.00 in the I and A settings and to 12.89 in the B setting, compared with the baseline score of 75.87. Similarly, on STS-B, the naive insertion setting drops to 47.07, 34.13, and 31.30 in the I, A, and B settings, respectively, compared with the baseline score of 92.97. These results indicate that naive middle-layer insertion can make full fine-tuning highly unstable across multiple tasks. Representative worst-case examples make this contrast clearer. For example, on CoLA, the no-added-layer baseline achieved 75.87, whereas the naive insertion setting dropped to 0.00 and the proposed method recovered to 75.85 in the I setting. Likewise, on STS-B, the naive insertion setting dropped to 31.30 in the B setting, whereas the proposed method reached 92.94, close to the baseline score of 92.97. These numerical contrasts provide explicit baseline–naive–proposed references in the main results, while Figure 3 further illustrates how the naive and proposed settings diverge as the number of inserted layers increases.

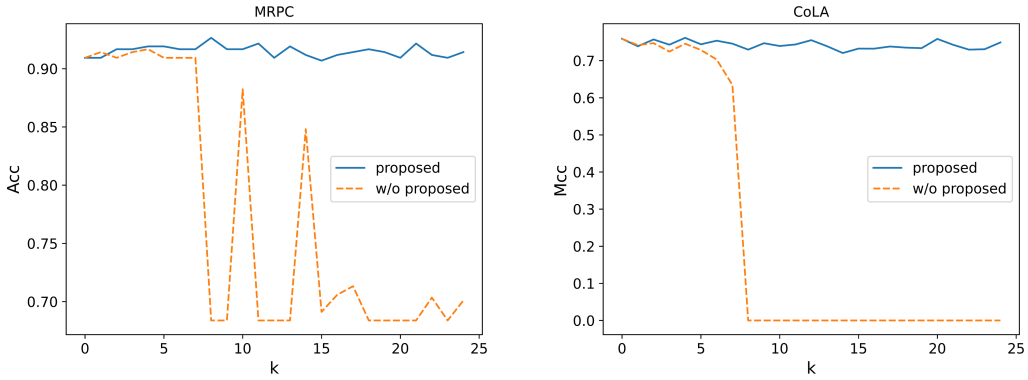


Figure 3. Performance as a function of the number of additional layers with and without the proposed method. In this figure, the additional layers are inserted into the bottom-k transformer blocks of the pre-trained model. This figure compares cumulative inserted-layer settings only ($k > 0$). The no-added-layer baseline corresponds to the $k = 0$ reference without layer insertion and is therefore reported separately in the main result tables rather than as part of the cumulative insertion sweep.

The freeze-pre-trained-model setting fine-tunes only the additional layer while keeping the backbone fixed. In most datasets this setting did not recover baseline-level performance, although it sometimes improved over the naive insertion setting. For example, SST-2 and QNLI show partial recovery relative to naive insertion, but the results remain substantially below the no-added-layer baseline overall. This suggests that simply adapting the inserted layer while freezing the pre-trained backbone is not sufficient to resolve the instability caused by middle-layer insertion.

Table 2. GLUE benchmark results for DeBERTaV3$_{large}$. I, A, and B denote the insertion positions: in multi-head self-attention, after feed-forward network, and both.

Model	COLA Mcc	SST-2 Acc	MRPC Acc	STS-B Corr	QNLI Acc	RTE Acc	WNLI Acc
DeBERTaV3$_{large}$	75.87	95.87	90.93	92.97	93.61	91.34	91.54
Insert Additional Layer without Proposed Method							
DeBERTaV3$_I$	0	50.92	70.10	47.07	50.54	52.70	56.34
DeBERTaV3$_A$	0	50.92	68.38	34.13	50.54	52.70	57.75
DeBERTaV3$_B$	12.89	82.56	71.08	31.30	50.54	56.68	56.34
Freeze Pre-trained Model							
Adapter$_I$	0	89.44	70.83	22.00	64.34	57.76	56.34
Adapter$_A$	5.29	87.50	72.30	80.86	83.31	57.40	60.56
Adapter$_B$	14.23	85.78	71.32	24.74	63.26	55.60	56.34
with Proposed Method							
Proposed$_I$	75.85	95.99	91.67	93.28	93.78	91.70	92.95
Proposed$_A$	73.03	96.10	91.91	92.84	93.74	91.70	95.77
Proposed$_B$	74.28	95.53	92.16	92.94	93.65	91.70	94.36

The results obtained with the proposed method are shown in the with Proposed Method block. Under the same internal insertion size, the proposed method substantially mitigated the severe degradation observed in the naive insertion setting and restored performance to a level that was often close to the no-added-layer baseline. On CoLA, the proposed method remains slightly below the baseline in all three insertion settings, but the gap is small compared with the severe collapse observed under naive insertion. In some tasks, the proposed method slightly exceeded the baseline; however, we do not interpret this as the main evidence of the method. Rather, the main observation is that the proposed method avoids the large degradation observed under naive insertion while using the same

inserted-layer architecture and parameter size. We also conducted experiments on $T5_{base}$, and the corresponding GLUE results are provided in Appendix C.

Table 3 reports the text generation results on CNN/DM [28] and WMT16 English–Romanian [29] using $T5_{base}$ [33]. $T5_{base}$ showed a relatively less catastrophic degradation pattern under naive insertion than $DeBERTaV3_{large}$, although the degradation was still substantial in some settings. Without the proposed method, performance drops are clearly observed on both CNN/DM and WMT16. For example, on WMT16 English–Romanian, the naive *B* setting dropped to 18.30 BLEU, whereas the no-added-layer baseline was 26.45 and the proposed *B* setting achieved 26.72. When using the proposed method, the severe degradation observed under naive insertion was largely mitigated, and the resulting performance remained close to the baseline, with some settings slightly exceeding it. In this paper, we interpret such cases primarily as evidence that the proposed method stabilizes internal insertion under the same insertion size rather than as evidence of consistent gains over the no-added-layer baseline. Because these T5 results are reported from single runs, small differences relative to the baseline should be interpreted with caution.

The proposed method was also evaluated on a pre-trained vision foundation model. Table 4 presents the results on ViT_{base} [34]. Naive insertion without the proposed method caused substantial degradation, although the severity varied across insertion settings. In particular, the *B* setting showed the largest drop on both ImageNet and CIFAR100, whereas the *A* setting remained noticeably higher but still below the no-added-layer baseline. A representative example is the CIFAR100 result in the *B* setting, where naive insertion achieved 54.69, whereas the no-added-layer baseline was 92.81 and the proposed method recovered to 92.77. Under the same internal insertion size, the proposed method substantially mitigated this degradation and restored performance to a level close to the baseline. Although some proposed settings slightly exceeded the baseline, we do not emphasize these cases as the main contribution of the method. Instead, we interpret them conservatively and focus on the fact that the proposed design avoids the large degradation observed under naive insertion. Because these ViT results are also reported from single runs, small improvements over the baseline should not be over-interpreted.

Table 3. Text generation results with $T5_{base}$. $T5^{*}_{base}$ denotes the performance reported in the original T5 paper [33].

Model	CNN/DM ROUGE-1	CNN/DM ROUGE-2	CNN/DM ROUGE-N	WMT16 en-ro BLEU
$T5^{*}_{base}$	42.05	20.34	39.40	28.0
$T5_{base}$	42.60	20.15	39.58	26.45
Insert Additional Layer without Proposed Method				
$T5_{I}$	39.72	18.30	36.74	24.41
$T5_{A}$	41.66	19.39	38.53	24.88
$T5_{B}$	36.84	16.24	33.91	18.30
with Proposed Method				
$Proposed_{I}$	42.44	20.15	39.37	26.69
$Proposed_{A}$	43.20	20.70	40.15	26.79
$Proposed_{B}$	42.21	19.85	39.23	26.72

Table 4. Image classification results with ViT_{base}. ViT^{*}_{base} denotes the performance reported in the original ViT paper [34].

Model	ImageNet Acc	CIFAR100 Acc
ViT^{*}_{base}	83.97	91.67
ViT_{base}	83.99	92.81
Insert Additional Layer without Proposed Method		
ViT_{I}	66.85	61.55
ViT_{A}	81.01	82.18
ViT_{B}	53.72	54.69
with Proposed Method		
Proposed_{I}	84.28	92.62
Proposed_{A}	84.09	92.54
Proposed_{B}	84.22	92.77

4.4. Performance Degradation by Additional Layer

This study assumes that an additional layer inserted between the middle layers of a pre-trained model can act as a source of interference to pre-trained information. To examine this issue empirically, we measured downstream performance as a function of the number of additional layers when the proposed method was not used. In this experiment, each additional layer was initialized with Xavier initialization, and GELU was applied to the output. We considered two cumulative insertion settings: top-k insertion over the total N transformer blocks and bottom-k insertion over the same blocks. For DeBERTaV3_{large}, we have $N = 24$.

Figure 4 shows downstream performance on MRPC as the number of additional layers increases. The overall trend indicates that performance deteriorates as more additional layers are inserted without the proposed design. This result supports the interpretation that naive middle-layer insertion can disrupt pre-trained representations and that the disruption becomes more severe as the number of inserted layers increases. In addition, for the same value of k, bottom-k insertion tends to perform worse than top-k insertion. This suggests that perturbations introduced in earlier transformer blocks may have a larger effect on subsequent processing during full fine-tuning.

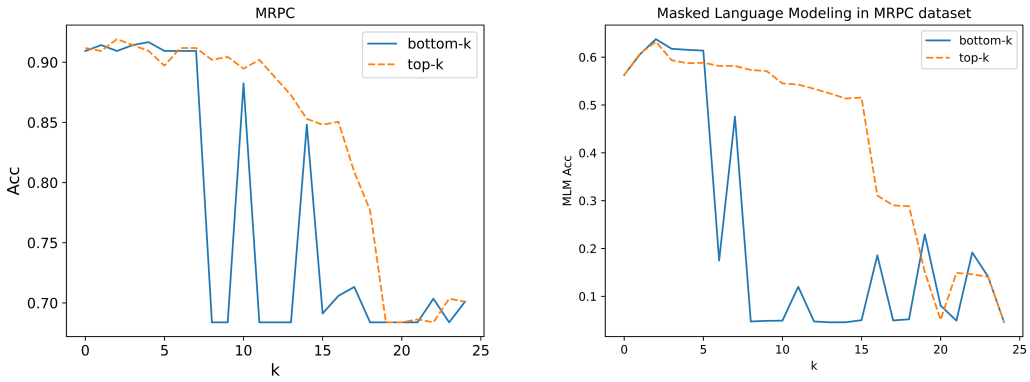


Figure 4. MRPC performance under cumulative inserted-layer settings without the proposed method. Each point corresponds to inserting additional layers into the top-k or bottom-k transformer blocks of DeBERTaV3_{large}, where larger k indicates that more middle layers are cumulatively modified.

We also conducted an auxiliary experiment using the masked language modeling (MLM) objective, which is closely related to the pre-training objective. After MLM training on the MRPC data, MLM accuracy decreased as k increased. This observation is consistent with the interpretation that additional layers can interfere with the reuse of pre-trained

information. To further examine this tendency, we conducted the same experiment on CoLA, as reported in Appendix D.

4.5. Initialization-Time Relative Perturbation

The mechanistic interpretation of this study is that naive middle-layer insertion disrupts intermediate representations at the beginning of fine-tuning, whereas the proposed parameterization keeps the inserted transformation closer to the identity at initialization and thereby reduces the immediate disturbance. To provide direct evidence for this interpretation, we additionally measured the initialization-time relative perturbation of the inserted layer, defined as $\|A(x) - x\|_2 / \|x\|_2$, where x denotes the input to the inserted layer and $A(\cdot)$ is the inserted transformation at initialization. We conducted this diagnostic in a representative fixed DeBERTa-MRPC setting. Under naive Xavier + output GELU insertion, the initialization-time relative perturbation was 1.4106 ± 0.0045, whereas under the proposed unit + weight GELU parameterization, it was reduced to 0.1587 ± 0.0000. This corresponds to an approximately 88.8% reduction relative to the naive insertion. These results provide direct evidence that the proposed parameterization substantially reduces the immediate representation disturbance introduced at initialization. At the same time, this diagnostic is reported for a representative setting at initialization and should therefore be interpreted as supporting evidence for the mechanistic hypothesis rather than as a complete proof of the full noise-accumulation and forgetting process throughout training.

4.6. Expanded Baseline Analysis

To examine whether the effectiveness of the proposed design is specific to the conventional Xavier + output GELU baseline, we additionally compared several inserted-layer variants on representative GLUE tasks, CoLA and MRPC. The expanded baselines include He initialization with output-side ReLU, Xavier initialization without activation, Xavier initialization with weight-side GELU, and zero initialization without activation. Following the reporting convention used in the main result tables, Table 5 reports the best single-run result obtained under validation-based tuning for each configuration. Because the expanded baseline analysis covers multiple initialization and activation variants under a limited computational budget, we use this table to test whether the main tendency depends on a single naive baseline choice, rather than to make statistical claims about small differences across variants. To partially address run-to-run variability, we additionally provide representative fixed-setting multi-seed results on CoLA and MRPC in Appendix E. On CoLA, all naive variants collapsed to 0.00, whereas unit-based variants remained much closer to the baseline, with Unit (No Act.) reaching 72.11 and Unit (Weight GELU) reaching 75.85 compared with the baseline score of 75.87. Because a Matthews correlation of 0.00 indicates no useful correlation between predictions and labels, these CoLA results are consistent with complete performance collapse rather than partial degradation. On MRPC, the naive variants remained in the range of 68.38–70.10, whereas both unit-based variants reached 91.67, close to the baseline score of 90.93. Notably, MRPC is a binary classification task, and 68.38 corresponds to the class-imbalance level of the evaluation split in this setting; thus, results around 68.38 are consistent with near-majority-class prediction and should be interpreted as effective collapse rather than meaningful task learning. These results suggest that the effectiveness of the proposed design is not limited to a single Xavier + GELU baseline, but extends across multiple naive initialization and activation settings.

Table 5. CoLA and MRPC results under expanded baseline configurations.

Method	CoLA (Matthews Corr.)	MRPC (Acc.)
Baseline	75.87	90.93
Naive (Xavier + GELU)	0.00	70.10
Naive (He + ReLU)	0.00	69.61
Xavier (No Act.)	0.00	68.38
Xavier (Weight GELU)	0.00	68.38
Zero (No Act.)	0.00	68.38
Unit (No Act.)	72.11	91.67
Unit (Weight GELU)	75.85	91.67

4.7. Ablation Study

We compared performance with and without the proposed methodology as a function of the number of additional layers. Figure 3 shows the results when additional layers are inserted into the bottom-k transformer blocks. Without the proposed method, performance degrades as the number of inserted layers increases. In contrast, when the proposed method is used, performance remains much more stable as k increases. Figure 3 analyzes cumulative inserted-layer settings only. The no-added-layer baseline is not itself part of this insertion sweep because it corresponds to the case without any inserted layer ($k = 0$). Accordingly, the baseline is reported separately in the main result tables and should be interpreted as the fixed reference score for these cumulative comparisons. These results support the interpretation that the proposed design mitigates the degradation caused by cumulative middle-layer insertion rather than merely improving a single insertion setting. Although some settings show small improvements relative to the no-added-layer baseline, we do not treat such cases as the main contribution of the method.

Table 6 compares the effect of activation placement in the additional layer. In this experiment, the additional layer was inserted at the in multi-head self-attention position, and the weight was initialized with a unit tensor. Applying the activation function to the output caused clear performance degradation, especially relative to the no-activation and weight-side activation settings. These results support the view that output-side activation is more disruptive than weight-side activation in this setting. The no-activation setting also remained relatively stable, but the weight-side activation setting stayed closest to the main proposed configuration.

Table 6. Performance according to activation placement in the additional layer.

Activation Placement	MRPC Acc.	CoLA Matthews Corr.
No Activation	91.67	73.33
Output-side Activation	72.06	65.24
Weight-side Activation	91.67	75.85

Table 7 compares different initialization methods for the additional layer. In this experiment, GeLU was applied to the weight matrix. The conventional Xavier initialization led to substantially lower performance than unit initialization on both MRPC and CoLA. This comparison supports the interpretation that initialization design is a key factor in the degradation caused by naive middle-layer insertion and that unit initialization is much better aligned with stable full fine-tuning in this setting.

Table 7. Performance according to the initialization method.

Initialization Method	MRPC Acc.	CoLA Matthews Corr.
Xavier Initialization	72.55	12.71
Unit Initialization	91.67	75.85

5. Discussion

The experimental results consistently support the main hypothesis of this study: the performance degradation caused by inserting additional layers into the middle of a pre-trained model primarily arises from disruption of pre-trained representations during the early stage of full fine-tuning. When additional layers were inserted without the proposed design, performance dropped substantially across natural language understanding, text generation, and image classification tasks. In contrast, the proposed method substantially mitigated this degradation and restored performance to a level that was often close to the no-added-layer baseline. These findings suggest that the main difficulty is not simply the increase in model depth, but the mismatch introduced by random initialization and output-side activation, which alters intermediate representations before downstream optimization becomes stable. The ablation results further strengthen this interpretation: performance deteriorated as the number of inserted layers increased, the degradation was more severe when layers were inserted into lower transformer blocks, and unit initialization with weight-side activation was more effective than Xavier initialization or output-side activation. Additional comparisons with expanded baselines on CoLA and MRPC further showed that this tendency was not limited to a single Xavier + GELU setting, but remained consistent across several naive initialization and activation variants, including He initialization, zero initialization, and no-activation settings. Moreover, a representative perturbation analysis in the DeBERTa-MRPC setting showed that the proposed unit + weight GELU parameterization substantially reduced initialization-time relative perturbation compared with naive Xavier + output GELU insertion, supporting the interpretation that the proposed method remains closer to the identity at initialization. At the same time, this mechanistic interpretation should be understood as being supported most directly by the representative DeBERTa-MRPC diagnostic reported in this paper rather than as being fully established across all architectures and tasks. In this sense, the perturbation analysis provides direct but still localized evidence for the proposed explanation, while the broader noise-accumulation account remains a plausible interpretation supported by the overall performance trends. Therefore, preserving an identity-like transformation at the beginning of training appears to be important for retaining pre-trained knowledge while still allowing the inserted layers to adapt during fine-tuning.

These findings also have practical implications for the design of transferable architectures. In many downstream applications, it may be desirable to introduce task-specific computation inside a pre-trained backbone rather than only at the output layer. The present results indicate that such mid-layer modification can be feasible if the inserted layer is designed to minimize interference with the existing representation flow. In addition, the fact that freezing the pre-trained model and training only the inserted layer did not sufficiently recover performance suggests that the proposed method should not be interpreted simply as an alternative adapter-style module, but rather as a mechanism for enabling stable full-model adaptation after architectural modification. Accordingly, the contribution of the proposed method is better understood as a practical and empirically effective stabilization mechanism than as a theoretically distinct new fine-tuning framework. At the same time, the results reveal several limitations. Because most of the main results are reported from single runs, small performance differences should be interpreted with caution. To

partially address this issue, we provide fixed-setting multi-seed results on representative DeBERTa-GLUE tasks in Appendix E; however, we do not claim statistical significance for small gains in settings where variance estimates are unavailable. This means that the current evidence is strongest for the representative DeBERTa setting, whereas the T5 and ViT results should be interpreted primarily as supporting the same qualitative tendency under the tested conditions rather than as establishing an equally strong cross-model claim. Similarly, the perturbation analysis is reported for a representative setting and should be interpreted as supporting evidence rather than as a complete mechanistic proof across all models and tasks. The experiments were conducted mainly on transformer-based models and a limited set of NLP and CV benchmarks, and some tasks, such as CoLA, still showed slightly lower performance than the baseline in some settings. Future work should therefore strengthen the empirical basis of these broader claims by extending multi-seed evaluation beyond the representative DeBERTa setting and by testing whether the same perturbation pattern is observed more directly in encoder–decoder and vision architectures. Future work should examine whether the same principle generalizes to larger-scale models, other architectures, other activation functions or parameterizations of inserted layers, and broader fine-tuning settings. It would also be valuable to analyze the phenomenon more directly in representation space, for example by tracking how inserted layers perturb hidden states during optimization.

6. Conclusions

In this study, we proposed a method for full fine-tuning of pre-trained models with additional layers inserted into the middle of the backbone. We experimentally showed that naive middle-layer insertion can lead to substantial performance degradation, and we interpreted this degradation as arising from random initialization and output-side activation, which perturb intermediate representations before the model has adapted. To address this issue, we initialized the inserted-layer weight with a unit tensor and applied the activation function to the weight rather than to the output. We evaluated the proposed method on the GLUE benchmark, text generation, and image classification tasks. The results show that the proposed method substantially mitigated the severe degradation caused by naive middle-layer insertion and, in many settings, maintained performance close to that of the no-added-layer baseline. The current evidence supports this tendency most directly in the representative DeBERTa setting and, under the tested conditions, provides supportive cross-model evidence in T5 and ViT as well, although these latter results should be interpreted more cautiously because they are primarily based on single-run evaluations. Representative perturbation analysis further showed that the proposed parameterization substantially reduced initialization-time perturbation compared with naive insertion, although this evidence should be interpreted as supportive rather than exhaustive. Accordingly, the mechanistic interpretation advanced in this paper is best understood as a plausible explanation supported by representative direct evidence rather than as a fully established account across all architectures and tasks. Taken together, these findings suggest that the main contribution of the proposed method lies in providing a practical way to stabilize middle-layer insertion during full fine-tuning rather than in proposing a theoretically distinct new fine-tuning framework.

Many studies propose diverse deep learning architectures for downstream tasks, and some of these settings may require inserting additional computation into the middle of a pre-trained model. In this study, we showed that stable full fine-tuning remains possible even when an additional layer is inserted into the middle of a pre-trained model, provided that the insertion is designed to reduce unnecessary perturbation at initialization. More broadly, the present results support the practical value of identity-like initialization and restricted

perturbation at insertion time as stabilizing principles for this setting, while further multi-seed validation and broader mechanistic analysis remain important directions for future work. We hope that this perspective will support the development of more diverse fine-tuned architectures that modify pre-trained backbones while preserving stable adaptation.

Author Contributions: Conceptualization, G.K. and S.K.; methodology, G.K.; software, G.K.; validation, G.K. and S.K.; investigation, G.K.; data curation, G.K.; visualization, G.K.; writing—original draft preparation, G.K.; writing—review and editing, G.K. and S.K.; supervision, S.K.; project administration, S.K.; funding acquisition, S.K. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Government of Korea (MSIT) (No. 2022R1A2C1005316) and in part by the Gachon University research fund of 2025 (GCU-202503060001).

Data Availability Statement: The datasets used in this study are publicly available through the Hugging Face Datasets Hub and were loaded using the Hugging Face datasets library. The datasets were derived from the following resources: 1. GLUE, https://huggingface.co/datasets/nyu-mll/glue (accessed on 16 April 2026). 2. CNN/DailyMail, https://huggingface.co/datasets/abisee/cnn_dailymail (accessed on 16 April 2026). 3. WMT16 English–Romanian, https://huggingface.co/datasets/wmt/wmt16 (accessed on 16 April 2026). 4. ImageNet-1K, https://huggingface.co/datasets/ILSVRC/imagenet-1k (accessed on 16 April 2026). 5. CIFAR100, https://huggingface.co/datasets/uoft-cs/cifar100 (accessed on 16 April 2026).

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

Appendix A. Experiment Setup

The hyperparameters used in the experiments are listed in Table A1. The main experiments followed the tuning protocol summarized in Table A1. All experiments were run on a single RTX 4090 or RTX 3090 GPU. Table A2 reports the number of parameters after inserting an additional layer into each pre-trained model. The models and evaluation metrics used in the experiments were implemented using, or adapted from, the Hugging Face codebase [37]. To improve reproducibility, we provide more detailed experimental settings, including model-selection details and additional protocol details for the representative fixed-setting multi-seed supplementary runs. Hyperparameter tuning and checkpoint/model selection were performed on the validation split, and the same validation-based selection rule was applied across all compared settings. Unless otherwise specified, the main result tables report the best single-run result obtained under validation-based hyperparameter tuning for each compared setting. To examine run-to-run variability, we additionally report fixed-setting multi-seed results in Appendix E. In these supplementary runs, all hyperparameters were fixed, and only the random seed was changed.

Table A1. Hyperparameters for fine-tuning DeBERTaV3$_{large}$, T5$_{base}$, and ViT$_{base}$. GLUE benchmarks were fine-tuned on DeBERTaV3$_{large}$ and T5$_{base}$, text generation datasets were fine-tuned on T5$_{base}$, and image classification datasets were fine-tuned on ViT$_{base}$. Additional protocol details used for representative fixed-setting multi-seed runs are provided in Appendix E. # denotes the number of inserted layers.

Hyper-Parameter	GLUE	Text Generation	Image Classification
Warmup Steps	{50, 500}	-	-
Learning Rate	$\{1.5 \times 10^{-5}, 3 \times 10^{-5}, 5 \times 10^{-5}, 1 \times 10^{-4}\}$	$\{1 \times 10^{-3}, 1 \times 10^{-4}\}$	$\{3 \times 10^{-2}, 1 \times 10^{-2}\}$
Batch Size	16 or 8	128	32
Epochs (Steps)	15	15,000 steps	ImageNet: 5/CIFAR100: 30
Optimizer	AdamW	AdamW	SGD
Weight Decay	0.01	0.01	0

Table A1. *Cont.*

Hyper-Parameter	GLUE	Text Generation	Image Classification
AdamW ϵ	1×10^{-6}	1×10^{-6}	-
AdamW β_1, β_2	0.9, 0.999	0.9, 0.999	-
SGD Momentum	-	-	0.9
Image Resolution	-	-	384
Gradient Clipping	1.0	-	1.0
Scheduler	linear	-	cosine
Dropout	0.1	0.1	0.0
Checkpoint Selection Rule	best validation score	best validation score	best validation score
Pre-trained Model	DeBERTaV3$_{large}$, T5$_{base}$	T5$_{base}$	ViT$_{base}$
# inserted layers	DeBERTaV3$_{large}$: {20, 24} T5$_{base}$: {10, 12}	{10, 12}	12

Table A2. Number of parameters after inserting an additional layer into the pre-trained model. + denotes the setting with additional layer insertion.

Additional Layer Position	DeBERTaV3$_{large}$	T5$_{base}$	ViT$_{base}$
No-added-layer baseline	435 M	223 M	86 M
+ In Multi-head Self-Attention	440 M	229 M	88 M
+ After Feed-forward Network	460 M	238 M	93 M
+ Both	465 M	243 M	95 M

Appendix B. Implementation Details

All inserted layers used in this study were bias-free. This design was adopted to minimize additive shifts at initialization and to reduce hidden-state perturbation during the early stage of full fine-tuning. The proposed weight-side activation applies the scalar nonlinearity element-wise to the entries of the weight matrix, i.e., $\alpha(W^A)$ denotes element-wise activation of W^A, not a separate matrix-valued operation. The same bias-free inserted-layer formulation was used consistently for both Q/K/V-side insertion and FFN-side insertion. In implementation, the inserted layer was computed as $\texttt{torch.matmul}(x, \alpha(W^A)^\top)$ for row-wise batch input tensors.

For ViT$_{base}$, we followed the original pre-LN transformer block structure. In the attention-side insertion setting, the inserted layer was applied on the Q/K/V path based on the hidden states after the first LayerNorm in each encoder block. In the FFN-side insertion setting, the inserted layer was applied after the feed-forward branch of the same pre-LN block. The CLS token was processed together with the patch tokens in the same transformer sequence, and no separate branch or special inserted-layer design was used for the CLS token. Unless otherwise specified, the inserted-layer setting was applied to all 12 encoder blocks. For fair comparison, the ViT backbone and the optimizer/scheduler settings were kept identical to those of the no-added-layer baseline.

Appendix C. Experiments with T5$_{base}$ on the GLUE Benchmark

Table A3 reports GLUE results for T5$_{base}$. Overall, the results show a pattern similar to that of DeBERTaV3$_{large}$: naive middle-layer insertion causes substantial degradation, whereas the proposed method largely mitigates this degradation. Compared with DeBERTaV3$_{large}$, the degradation pattern under naive insertion is relatively less catastrophic for T5$_{base}$ in some settings, but it remains substantial overall and should not be interpreted as robustness in an absolute sense. In the naive insertion setting, clear degradation is observed across multiple tasks. For example, STS-B drops from 89.94 in the no-added-layer

baseline to 39.23 in the *B* setting, and QNLI drops from 93.01 to 67.23 in the same setting. The freeze-pre-trained-model setting also fails to recover baseline-level performance.

When the proposed method is used, performance is restored to a level that is often close to the no-added-layer baseline, although some tasks remain slightly below it. CoLA, for example, remains below the baseline in the *I* and *A* settings, whereas the *B* setting slightly exceeds the baseline. As in the main results, we do not interpret such cases as the main evidence of the method. Instead, the key observation is that the proposed method avoids the severe degradation seen under naive insertion while using the same inserted-layer structure and insertion size.

Table A3. GLUE benchmark results with $T5_{base}$. I, A, and B denote the insertion positions: in multi-head self-attention, after feed-forward network, and both. $T5^{*}_{base}$ denotes the performance reported in the original T5 paper [33].

Model	CoLA Mcc	SST-2 Acc	MRPC Acc	STS-B Corr	QNLI Acc	RTE Acc
$T5^{*}_{base}$	51.10	95.20	87.50	89.40	93.70	80.10
$T5_{base}$	61.57	94.95	91.18	89.94	93.01	78.70
Insert Additional Layer without Proposed Method						
$T5_{I}$	19.51	88.07	73.28	80.73	67.38	57.04
$T5_{A}$	14.58	88.76	83.82	84.21	85.45	58.12
$T5_{B}$	16.54	87.96	73.28	39.23	67.23	58.84
Freeze Pre-trained Model						
Adapter_{I}	9.69	88.99	71.57	22.75	63.39	56.32
Adapter_{A}	17.51	88.99	82.84	83.33	84.44	58.12
Adapter_{B}	13.46	86.58	71.57	21.81	64.59	54.51
with Proposed Method						
Proposed_{I}	60.06	94.61	90.93	90.76	92.84	84.12
Proposed_{A}	58.57	94.61	91.67	90.40	92.99	81.23
Proposed_{B}	61.59	94.72	92.89	91.07	92.84	83.03

Appendix D. Performance Degradation as a Function of the Number of Additional Layers in CoLA

Figure A1 shows CoLA performance as a function of the number of additional layers. The overall pattern is similar to that in Figure 4: as the number of additional layers increases, the Matthews correlation coefficient decreases. In addition, for most values of *k*, bottom-*k* insertion performs worse than top-*k* insertion. This tendency is also consistent with the MLM-based auxiliary analysis.

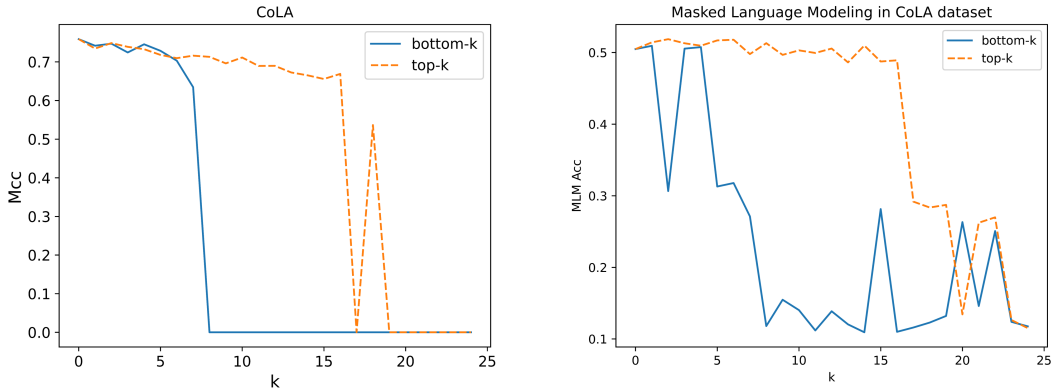


Figure A1. Performance as a function of the number of additional layers in CoLA. In this figure, each additional layer is initialized with Xavier initialization and followed by output-side GELU activation. Top-*k* inserts additional layers into the last *k* transformer blocks, whereas bottom-*k* inserts additional layers into the first *k* transformer blocks.

Appendix E. Multi-Seed Stability on CoLA and MRPC

To complement the single-run results in the main tables, we additionally conducted fixed-setting multi-seed experiments on representative DeBERTa-GLUE tasks using the In multi-head self-attention insertion setting (I). In these supplementary experiments, all hyperparameters were fixed, and only the random seed was changed. Specifically, we used 15 epochs, batch size 8, learning rate 3×10^{-5}, warmup steps 50, AdamW, a linear scheduler, and gradient clipping of 1.0. The inserted layers were applied to all 24 transformer blocks, and the best checkpoint was selected using the same validation metric across seeds. Table A4 reports the mean and standard deviation across seeds on the same percentage scale as the main tables, so that the supplementary results can be compared directly with Table 2. These supplementary results are intended to provide additional evidence about run-to-run stability rather than to replace the main validation-selected single-run reporting protocol used in the main tables.

Table A4. Representative fixed-setting multi-seed results on DeBERTa-GLUE. The insertion setting corresponds to the in multi-head self-attention configuration (I) used in Table 2. We report mean and standard deviation across seeds on the same percentage scale as the main tables.

Task	Setting	Mean	Std
CoLA	DeBERTaV3$_{large}$	73.37	2.18
CoLA	DeBERTaV3$_{I}$	0.00	0.00
CoLA	Proposed$_{I}$	72.52	2.89
MRPC	DeBERTaV3$_{large}$	90.77	0.14
MRPC	DeBERTaV3$_{I}$	68.81	0.86
MRPC	Proposed$_{I}$	91.26	0.51

References

1. Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A.N.; Kaiser, L.U.; Polosukhin, I. Attention is All you Need. In *Proceedings of the Advances in Neural Information Processing Systems, Long Beach, CA, USA, 4–9 December 2017*; Guyon, I., Luxburg, U.V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., Garnett, R., Eds.; Curran Associates, Inc.: New York, NY, USA, 2017; Volume 30.
2. He, K.; Girshick, R.; Dollar, P. Rethinking ImageNet Pre-Training. In Proceedings of the 2019 IEEE/CVF International Conference on Computer Vision (ICCV), Seoul, Republic of Korea, 27 October–2 November 2019; pp. 4917–4926. [CrossRef]
3. Devlin, J.; Chang, M.W.; Lee, K.; Toutanova, K. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the NAACL:2019:1, Minneapolis, MN, USA, 6 June 2019*; Burstein, J., Doran, C., Solorio, T., Eds.; Association for Computational Linguistics: Stroudsburg, PA, USA, 2019; pp. 4171–4186. [CrossRef]
4. Liu, Y.; Ott, M.; Goyal, N.; Du, J.; Joshi, M.; Chen, D.; Levy, O.; Lewis, M.; Zettlemoyer, L.; Stoyanov, V. RoBERTa: A Robustly Optimized BERT Pretraining Approach. *arXiv* **2019**, arXiv:1907.11692. . [CrossRef]
5. Liu, Z.; Lin, Y.; Cao, Y.; Hu, H.; Wei, Y.; Zhang, Z.; Lin, S.; Guo, B. Swin Transformer: Hierarchical Vision Transformer Using Shifted Windows. In Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV), Montreal, QC, Canada, 10–17 October 2021; pp. 10012–10022.
6. Zheng, H.; Shen, L.; Tang, A.; Luo, Y.; Hu, H.; Du, B.; Tao, D. Learn from Model Beyond Fine-Tuning: A Survey. *arXiv* **2023**, arXiv:2310.08184. [CrossRef]
7. Kim, G.; Kang, S. Effective Transfer Learning with Label-Based Discriminative Feature Learning. *Sensors* **2022**, *22*, 2025. [CrossRef]
8. Yuan, Y.; Chen, W.; Yang, Y.; Wang, Z. In Defense of the Triplet Loss Again: Learning Robust Person Re-Identification with Fast Approximated Triplet Loss and Label Distillation. In Proceedings of the 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW), Los Alamitos, CA, USA, 14–19 June 2020; pp. 1454–1463. [CrossRef]
9. Long, J.; Shelhamer, E.; Darrell, T. Fully convolutional networks for semantic segmentation. In Proceedings of the 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Los Alamitos, CA, USA, 7–12 June 2015; pp. 3431–3440. [CrossRef]
10. Girshick, R.; Donahue, J.; Darrell, T.; Malik, J. Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation. In Proceedings of the 2014 IEEE Conference on Computer Vision and Pattern Recognition, Columbus, OH, USA, 24–27 June 2014; pp. 580–587. [CrossRef]

11. Wang, F.; Song, K.; Zhang, H.; Jin, L.; Cho, S.; Yao, W.; Wang, X.; Chen, M.; Yu, D. Salience Allocation as Guidance for Abstractive Summarization. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing, Abu Dhabi, United Arab Emirates, 7–11 December 2022*; Goldberg, Y., Kozareva, Z., Zhang, Y., Eds.; Mohamed bin Zayed, University of Artificial Intelligence: Abu Dhabi, United Arab Emirates, 2022; pp. 6094–6106. [CrossRef]
12. Wang, B.; Zhang, Z.; Xu, K.; Hao, G.Y.; Zhang, Y.; Shang, L.; Li, L.; Chen, X.; Jiang, X.; Liu, Q. DyLex: Incorporating Dynamic Lexicons into BERT for Sequence Labeling. In Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, Online and Punta Cana, Dominican Republic, 7–11 November 2021; pp. 2679–2693. [CrossRef]
13. Chen, T.; Goodfellow, I.J.; Shlens, J. Net2Net: Accelerating Learning via Knowledge Transfer. In Proceedings of the 4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, 2–4 May 2016.
14. Wei, T.; Wang, C.; Rui, Y.; Chen, C.W. Network morphism. In Proceedings of the 33rd International Conference on International Conference on Machine Learning, New York, NY, USA, 19–24 June 2016; Volume 48, pp. 564–572.
15. He, K.; Zhang, X.; Ren, S.; Sun, J. Identity Mappings in Deep Residual Networks. In *Proceedings of the Computer Vision-ECCV 2016—14th European Conference, Amsterdam, The Netherlands, 11–14 October 2016*; Leibe, B., Matas, J., Sebe, N., Welling, M., Eds.; Proceedings, Part IV; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2016; pp. 630–645. [CrossRef]
16. Bachlechner, T.; Majumder, B.P.; Mao, H.H.; Cottrell, G.W.; McAuley, J. ReZero is All You Need: Fast Convergence at Large Depth. *arXiv* **2020**, arXiv:2003.04887. [CrossRef]
17. Hu, 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, Virtual, 25–29 April 2022.
18. Glorot, X.; Bengio, Y. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010*; Teh, Y.W., Titterington, M., Eds.; Proceedings of Machine Learning Research (PMLR): London, UK, 2010; Volume 9, pp. 249–256.
19. He, K.; Zhang, X.; Ren, S.; Sun, J. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. In Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV), Santiago, Chile, 7–13 December 2015; pp. 1026–1034. [CrossRef]
20. Agarap, A.F. Deep learning using rectified linear units (relu). *arXiv* **2018**, arXiv:1803.08375.
21. Hendrycks, D.; Gimpel, K. Gaussian Error Linear Units (GELUs). *arXiv* **2023**, arXiv:1606.08415. [CrossRef]
22. Warstadt, A.; Singh, A.; Bowman, S.R. Neural Network Acceptability Judgments. *arXiv* **2018**, arXiv:1805.12471. [CrossRef]
23. Socher, R.; Perelygin, A.; Wu, J.; Chuang, J.; Manning, C.D.; Ng, A.; Potts, C. Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank. In Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing, Seattle, WA, USA, 18–21 October 2013; pp. 1631–1642.
24. Dolan, W.B.; Brockett, C. Automatically constructing a corpus of sentential paraphrases. In Proceedings of the International Workshop on Paraphrasing, Jeju Island, Republic of Korea, 9–16 October 2005.
25. Cer, D.; Diab, M.; Agirre, E.; Lopez-Gazpio, I.; Specia, L. SemEval-2017 Task 1: Semantic Textual Similarity Multilingual and Crosslingual Focused Evaluation. In *Proceedings of the SEMEVAL:2017:2, Vancouver, BC, Canada, 3–4 August 2017*; Bethard, S., Carpuat, M., Apidianaki, M., Mohammad, S.M., Cer, D., Jurgens, D., Eds.; Association for Computational Linguistics (ACL): Kerrville, TX, USA, 2017; pp. 1–14. [CrossRef]
26. Wang, A.; Singh, A.; Michael, J.; Hill, F.; Levy, O.; Bowman, S. GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding. In Proceedings of the EMNLP:2018:54, Brussels, Belgium, 31 October–1 November 2018; pp. 353–355. [CrossRef]
27. Levesque, H.J.; Davis, E.; Morgenstern, L. The Winograd schema challenge. In Proceedings of the AAAI Spring Symposium: Logical Formalizations of Commonsense Reasoning, Stanford, CA, USA, 21–23 March 2011; Volume 46, p. 47.
28. See, A.; Liu, P.J.; Manning, C.D. Get to the Point: Summarization with Pointer-Generator Networks. In Proceedings of the ACL:2017:1, Vancouver, BC, Canada, 30 July–4 August 2017; pp. 1073–1083. [CrossRef]
29. Bojar, O.R.; Chatterjee, R.; Federmann, C.; Graham, Y.; Haddow, B.; Huck, M.; Jimeno Yepes, A.; Koehn, P.; Logacheva, V.; Monz, C.; et al. Findings of the 2016 Conference on Machine Translation. In Proceedings of the First Conference on Machine Translation, Berlin, Germany, 11–12 August 2016; pp. 131–198.
30. Deng, J.; Dong, W.; Socher, R.; Li, L.-J.; Li, K.; Li, F.-F. Imagenet: A large-scale hierarchical image database. In Proceedings of the 2009 IEEE Conference on Computer Vision and Pattern Recognition, Miami, FL, USA, 20–25 June 2009; pp. 248–255.
31. Krizhevsky, A. *Learning Multiple Layers of Features from Tiny Images*; University of Toronto: Toronto, ON, Canada, 2009; pp. 32–33.
32. He, P.; Gao, J.; Chen, W. DeBERTaV3: Improving DeBERTa using ELECTRA-Style Pre-Training with Gradient-Disentangled Embedding Sharing. *arXiv* **2021**, arXiv:2111.09543. [CrossRef]
33. 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*, 1–67.

34. Dosovitskiy, A.; Beyer, L.; Kolesnikov, A.; Weissenborn, D.; Zhai, X.; Unterthiner, T.; Dehghani, M.; Minderer, M.; Heigold, G.; Gelly, S.; et al. An Image is Worth 16 × 16 Words: Transformers for Image Recognition at Scale. In Proceedings of the Ninth International Conference on Learning Representations, Virtual, 3–7 May 2021.
35. Houlsby, N.; Giurgiu, A.; Jastrzebski, S.; Morrone, B.; de Laroussilhe, Q.; Gesmundo, A.; Attariyan, M.; Gelly, S. Parameter-Efficient Transfer Learning for NLP. *arXiv* **2019**, arXiv:1902.00751. [CrossRef]
36. Pfeiffer, J.; Kamath, A.; Rücklé, A.; Cho, K.; Gurevych, I. AdapterFusion: Non-Destructive Task Composition for Transfer Learning. In Proceedings of the EACL:2021:main, Online, 19–20 April 2021; pp. 487–503. [CrossRef]
37. Wolf, T.; Debut, L.; Sanh, V.; Chaumond, J.; Delangue, C.; Moi, A.; Cistac, P.; Rault, T.; Louf, R.; Funtowicz, M.; et al. HuggingFace's Transformers: State-of-the-art Natural Language Processing. *arXiv* **2020**, arXiv:1910.03771. [CrossRef]

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.