

EBS: Efficient Block-Sparse Framework for Continual Learning of Large Language Models

Zhixin Han¹, Hang Gao², Yinhao Bai³, Yuhua Zhao¹
Bitong Luo¹, Zhiyu Li⁴, Mengting Hu^{1*}, Minlie Huang⁵

¹College of Software, Nankai University

²College of Artificial Intelligence, Tianjin University of Science and Technology

³JD AI Research, Beijing, China

⁴MemTensor (Shanghai) Technology Co., Ltd, Shanghai, China

⁵Tsinghua University, Beijing, China

{zhixinhan, zyh22, luobitong}@mail.nankai.edu.cn

gaohang@tust.edu.cn, yinhaobai68@gmail.com

lizy@memtensor.cn, mthu@nankai.edu.cn, aihuang@tsinghua.edu.cn

Abstract

By continuously acquiring new knowledge, continual learning enables large language models (LLMs) to remain adaptive, accurate, and relevant as the world evolves. In continual learning, tasks arrive one after another and the model learns them sequentially. As more tasks are introduced, existing methods for learning a new task impose increasing computational demands. However, we find that the computational overhead, which gradually increases as the number of tasks grows, can be mitigated. To limit the continual increase in computational costs, we introduce a simple but efficient block-sparse (EBS) framework for continual learning of LLMs. In EBS, we employ a dedicated, task-specific low-rank adaptation (LoRA) block for each task. When training on a new task, only one LoRA block is loaded into GPU memory and involved in computation, while at inference each input activates at most two LoRA blocks. This design ensures that the computational cost remains independent of the number of tasks, allowing EBS to maintain high efficiency even when the sequence of tasks becomes very large. We offer a simple implementation that works across different model sizes and transformer architectures (decoder-only and encoder-decoder) while consistently delivering both high efficiency and strong task performance. Experimental results on two continual learning benchmarks demonstrate that EBS achieves state-of-the-art performance on the T5 and LLaMA 2 models.

1 Introduction

Continual learning (Ring, 1994) allows large language models (LLMs) to continuously acquire new

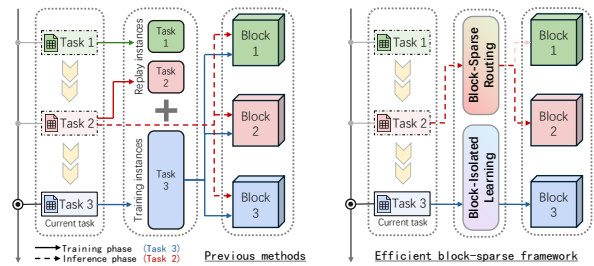


Figure 1: Comparison of EBS and previous methods. Previous methods activate all blocks for training or inference, with introducing experience replay. EBS is replay-free and activates only a sparse subset of blocks.

knowledge, ensuring they remain adaptive and relevant as the world evolves. This paradigm requires the model to *learn tasks sequentially as they arrive over time*. As illustrated in Figure 1, where tasks 1–3 are presented in temporal order, the model can only access and learn from task 3 data after completing training on tasks 1 and 2. A core challenge in this sequential learning process is *catastrophic forgetting* (French, 1999): when learning a new task (e.g., task 3), the model tends to abruptly lose performance on previously learned tasks (e.g., tasks 1 and 2). This occurs primarily because updating the model’s parameters to optimize for the new task often overwrites critical weights needed for older tasks. To counteract this forgetting, continual learning frameworks must be designed not only to learn new tasks effectively but also to rigorously maintain performance on all past tasks.

Existing continual learning mainly presents two research directions. First is to design task-specific parameter blocks, thereby prevent overwriting of weights critical for previous tasks. Recent works explore parameter-efficient tuning methods (blocks) (Ding et al., 2022), including low-rank

*Corresponding author.

adaptation (Hu et al., 2022; Wang et al., 2023a; Zhao et al., 2024) (LoRA) and prompt tuning (Lester et al., 2021; Razdaibiedina et al., 2023), for equipping LLMs with continual learning capabilities (Zhao et al., 2024). These works tend to share a unified architecture (see left side of Figure 1), assigning a dedicated block for each task. During training on a new task, only the current task’s block parameters are updated while all prior blocks remain frozen yet persistently active in forward/backward computations. During inference for any target task, all blocks invariably participate in the computational graph. This operational design ensures that every block engages in all computational phases regardless of task relevance. Consequently, *as the task count increases, the linear accumulation of task-specific blocks induces proportional growth in computational overhead and memory consumption.*

Second is to employ experience replay mechanisms to alleviate catastrophic forgetting. This mechanism strategically stores representative samples (or generated pseudo-samples) from past tasks and revisits them with current task data during training. As illustrated in Figure 1 (left), when learning task 3, subsets of data from tasks 1–2 are incorporated into the training batch. This forces simultaneous learning of new knowledge (task 3) while rehearsing previously acquired skills (tasks 1–2). Widely adopted in methods like SAPT (Zhao et al., 2024) and Recurrent-KIF (Feng et al., 2025), replay effectively preserves historical task performance. However, *as task count increases, the required replay volume grows proportionally, driving up the computational cost.*

To mitigate the growing computational demands of continual learning as the number of tasks increases, we introduce a simple replay-free **Efficient Block-Sparse (EBS)** framework built with LoRA blocks. As shown in the right part of Figure 1, EBS consists of two main components: block-isolated learning and block-sparse routing. With block-isolated learning, only the newly introduced block for the current task is loaded into GPU memory and participates in LoRA fine-tuning; previously learned blocks remain in storage and are not involved in this training phase. Moreover, with the help of block-sparse routing, each inference input activates only one or two LoRA blocks. As depicted in Figure 1, when performing inference on task 2, at most two blocks are enabled for each input instance. Because EBS utilizes at most two

LoRA blocks in both the training and inference phases, we term it “*block-sparse*”. Furthermore, EBS does not employ any replay method to incorporate data (or generated pseudo-data) from previous tasks into the current task’s training. An implementation of EBS is publicly available at <https://github.com/ZhixinHan/EBS>.

In summary, our contributions are as follows:

- We introduce a novel framework, EBS, for continual learning. It consists of two main components: block-isolated learning and block-sparse routing. In block-isolated learning, only one LoRA block is enabled, while in block-sparse routing, at most two LoRA blocks are activated.
- EBS ensures that the computational cost remains independent of the number of tasks, allowing this framework to maintain high efficiency when the sequence of tasks becomes very large.
- Although EBS substantially reduces computational cost (and memory usage during training), it still achieves state-of-the-art performance. Our proposed method is highly versatile and exceptionally easy to deploy. It can be seamlessly applied to LLMs of various parameter scales and is compatible with different transformer architectures, including the decoder-only model and the encoder-decoder model, consistently delivering strong performance.

2 Related Work

Recent efforts on continual learning for LLMs have yielded a series of influential approaches, which not only frame the current research landscape but also constitute the baselines for our comparison. Zhao et al. (2024) sequentially train the LoRA on the task orders, called SeqLoRA. Zhao et al. (2024) replay real samples from old tasks when learning new tasks to avoid forgetting, called Replay. Wang et al. (2022b) propose a method that learns to dynamically prompt (L2P) a pre-trained model to learn tasks sequentially under different task transitions. Qin and Joty (2022) propose a unified framework, LFPT5, for lifelong few-shot language learning based on prompt tuning of T5 (Raffel et al., 2020). Razdaibiedina et al. (2023) introduce progressive prompts (ProgPrompt) for continual learning in language models. Wang et al. (2023b) ap-

ply the parameter isolation strategy. Wang et al. (2023a) propose orthogonal low-rank adaptation (O-LoRA) for continual learning in language models, effectively mitigating catastrophic forgetting while learning new tasks. Zhao et al. (2024) propose a shared attention framework, SAPT, to align the parameter-efficient tuning learning and selection via the shared attentive learning & selection module, adopting two representative parameter-efficient tuning methods: prompt tuning (Lester et al., 2021) and LoRA (Hu et al., 2022). Du et al. (2024) introduce MIGU (magnitude-based gradient updating for continual learning), a rehearsal-free and task-label-free method that only updates the model parameters with large magnitudes of output in language models’ linear layers. Feng et al. (2025) present Recurrent-KIF, a novel continual learning framework for recurrent knowledge identification and fusion, which enables dynamic estimation of parameter importance distributions to enhance knowledge transfer (Thrun, 1995). Unlike prior approaches that keep an ever-growing set of task-specific blocks and/or replay samples involved in computation as tasks accumulate, EBS enforces almost *constant-cost* execution by activating only one LoRA block in training without replay and at most two blocks per input in inference.

In continual learning, beyond the fundamental requirement of preventing catastrophic forgetting, there is a higher-level objective known as *knowledge transfer*. Knowledge transfer, specifically backward transfer (Ke and Liu, 2023), refers to the improvement in performance on previously learned tasks resulting from learning new tasks. Backward transfer not only requires the model to retain knowledge from previously learned tasks, but also aims for the new task to enhance performance on those earlier tasks. If the newly learned task shares certain underlying relationships with some of previous ones, the model may be able to integrate knowledge across tasks to achieve improved performance.

3 Methodology

3.1 Framework Overview

Let $\mathcal{T} = \{T_i\}_{i=1}^{|\mathcal{T}|}$ be a sequence of tasks arriving in chronological order, and let t_i^j represent the j -th text instance of task T_i . As the model encounters tasks sequentially, it should, after finishing training on task T_i , retain strong performance on every one of the first i tasks, including the task just learned. Our framework for continual learning comprises

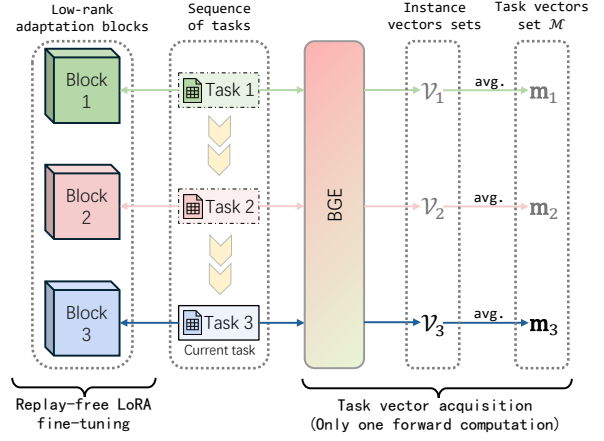


Figure 2: Computation process of block-isolated learning.

two main components: block-isolated learning and block-sparse routing, which respectively reduce computational cost during training and inference, especially when the total number of tasks becomes very large.

3.2 Block-Isolated Learning

As shown in Figure 2, block-isolated learning includes two computation processes: replay-free LoRA fine-tuning and task vector acquisition. In replay-free LoRA fine-tuning, when a new task arrives, we train a dedicated, task-specific LoRA block, while the LoRA blocks for previously learned tasks are neither loaded into GPU memory nor involved in computation. Unlike existing methods, we do not employ any replay method. In task vector acquisition, EBS leverages the entire training set of this new task to derive an embedding vector that characterizes the task.

3.2.1 Replay-Free LoRA Fine-Tuning

When task i arrives, we attach a dedicated, randomly initialized LoRA block and fine-tune only this block on the current task. The base parameters of the LLM remain frozen throughout, and the training process yields a task-specific set of LoRA parameters $\Delta W_i = B_i A_i$ for T_i . During fine-tuning, only the current task’s LoRA block participates in computation, thereby reducing active LoRA computation and preventing adapter-related training GPU memory from accumulating with the number of learned tasks. Furthermore, neither data nor pseudo-data from other tasks participates in computation.

3.2.2 Task Vector Acquisition

In addition to fine-tuning, we obtain a task embedding, i.e., we represent each task with a dense vector. The detailed procedure is described below.

First, we employ BAAI general embedding (BGE) (Xiao et al., 2024), an encoder-only sentence embedding model, to obtain a representation vector¹ $\mathbf{v}_i^j$ for the text instance t_i^j in task T_i . From the BGE model series, we choose the BAAI/bge-small-en-v1.5 checkpoint².

After processing all training instances of task T_i , we collect their representation vectors into the set $\mathcal{V}_i = \{\mathbf{v}_i^j\}_{j=1}^{|\mathcal{V}_i|}$. With this collection in hand, we obtain a task vector $\mathbf{m}_i'$ —identical in dimensionality to an individual sentence vector—by taking the arithmetic mean (Iyyer et al., 2015) of all vectors in $\mathcal{V}_i$:

$$\mathbf{m}_i' = \frac{1}{|\mathcal{V}_i|} \sum_{j=1}^{|\mathcal{V}_i|} \mathbf{v}_i^j \quad (1)$$

$\mathbf{m}_i'$ captures the collective semantics of all the training text samples within task T_i .

BGE model normalizes the output embedding vector to have the magnitude of 1, i.e., $\|\mathbf{v}_i^j\|_2 = 1$. However, the task vector $\mathbf{m}_i'$ produced by Equation (1) generally does not satisfy $\|\mathbf{m}_i'\|_2 = 1$. To align it with a BGE embedding vector, we therefore apply an L_2 -normalization to $\mathbf{m}_i'$, yielding the normalized task vector $\mathbf{m}_i$:

$$\mathbf{m}_i = \frac{\mathbf{m}_i'}{\|\mathbf{m}_i'\|_2} \quad (2)$$

Finally, we store $\mathbf{m}_i$ for use in the inference phase. Since the BGE embeddings are low-dimensional, $\mathbf{m}_i$ is lightweight and adds negligible storage cost. Consequently, together with the vectors saved from the previous $i - 1$ tasks, we now possess a stored collection of task embeddings:

$$\mathcal{M} = \{\mathbf{m}_k\}_{k=1}^i \quad (3)$$

EBS uses a single shared 33.4M-parameter BGE encoder for task-vector acquisition. Because this encoder is shared rather than replicated, it contributes the same 33.4M parameters regardless of the LLM backbone size or the number of learned

¹Different from more commonly used mean pooling, BGE is trained to use the last hidden state of [CLS] as the sentence embedding.

²BGE v1.5 is released in three parameter scales—small, base, and large. In our preliminary experiments, the smallest “small” variant already achieved satisfactory results, while the larger models delivered only marginal gains.

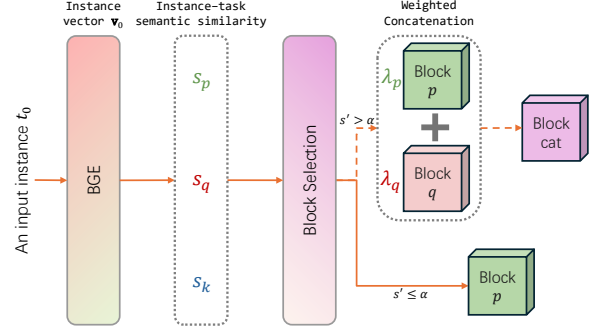


Figure 3: Computation process of block-sparse routing.

tasks. Constructing $\mathbf{m}_i$ requires encoding each training instance once with BGE, typically in batches. This is a one-time operation per task and requires no backward propagation or parameter updates.

3.3 Block-Sparse Routing

In a continual learning protocol, once the model finishes training on task T_i , it should be evaluated on all i tasks encountered so far to measure performance. However, at inference time, the model generally has no prior knowledge of which task an incoming instance belongs to. Although we can circumvent the issue by explicitly supplying the task label during evaluation, such an operation is unrealistic. Real-world systems must infer the task identity from the input alone without any additional information. Accordingly, following (Zhao et al., 2024), we evaluate EBS without disclosing the task label. Because the task affiliation of the current inference instance is unknown, we denote the input instance as t_0 .

To foster knowledge transfer, as shown in Figure 3, in block-sparse routing, EBS automatically assesses the relationships among tasks given t_0 , deciding whether to use only one LoRA block or to aggregate LoRA blocks from multiple tasks for inference. To ensure the efficiency of EBS and to mitigate potential noise introduced by involving too many blocks, we constrain EBS to aggregate at most two LoRA blocks.

3.3.1 Instance-Task Semantic Similarity

We feed t_0 into the BGE model to obtain its embedding vector $\mathbf{v}_0$. Next, the model retrieves the stored task vector set $\mathcal{M}$ and measures the similarity (expressed as a single scalar score) between each task vector $\mathbf{m}_k \in \mathcal{M}$ and $\mathbf{v}_0$. In line with BGE’s practice for measuring semantic similarity, we compute

their dot product³ to obtain the similarity set:

$$\mathcal{S} = \{s_k = \mathbf{m}_k^\top \mathbf{v}_0\}_{k=1}^i \quad (4)$$

By definition, a larger s_k implies that $\mathbf{m}_k$ is more semantically similar to $\mathbf{v}_0$.

3.3.2 Block Selection

After getting the similarity set $\mathcal{S}$, we extract its two largest similarity scores, denoted s_p and s_q , where $s_p \geq s_q$. These two scores identify the associated task vectors $\mathbf{m}_p$ and $\mathbf{m}_q$, enabling us to pinpoint T_p and T_q as the two tasks most semantically related to the current input instance. We then measure how similar these two tasks are by taking the dot product of their vectors:

$$s' = \mathbf{m}_p^\top \mathbf{m}_q \quad (5)$$

A larger s' implies that task T_p is more semantically similar to task T_q .

In the inference phase, the model can choose between two distinct computational paths, depending on the input instance. If s' is high, using both LoRA parameters, ΔW_p and ΔW_q , can promote knowledge transfer, because the model simultaneously draws on the information it learned from two closely related tasks. If, however, the two tasks are not closely related, simultaneously applying two LoRA blocks may introduce noise. In that case, we rely solely on ΔW_p . This strategy affords substantial flexibility: even within a same task, different input instances may trigger different inference routes. Although EBS stores i LoRA blocks, at most two participate in the forward computation for each input, which bounds active LoRA computation. In our current implementation, all i blocks remain GPU-resident during inference, so total inference GPU memory still increases with the number of tasks. By contrast, the strong baseline SAPT-LoRA (see Table 1 for its results) activates all LoRA blocks indiscriminately.

To quantify task relatedness, we introduce a manually set threshold α . If $s' > \alpha$, the two tasks are considered related; otherwise, they are treated as unrelated. Clearly, α is a hyperparameter. Because the similarity distribution of the BGE model is about in the interval $[0.6, 1]$ ⁴, we set α accordingly within this range.

³Because both the task vector $\mathbf{m}_k$ and the input text embedding $\mathbf{v}_0$ have been already normalized to have the magnitude of 1, using dot product and cos similarity (Salton et al., 1975) will have the same result.

⁴Refer to the BAAI/bge-small-en-v1.5’s documentation provided on the Hugging Face website.

If s' falls within the interval $(\alpha, 1]$, we need to activate two LoRA blocks for inference. To facilitate this, we first normalize s' proportionally from its original interval $(\alpha, 1]$ into the normalized interval $(0, 1]$. Mathematically, this normalization procedure is:

$$\hat{s} = \frac{s' - \alpha}{1 - \alpha} \quad (6)$$

Here, the normalized $\hat{s}$ now resides strictly within the interval $(0, 1]$. This ensures a well-defined, standardized measure of the similarity between two tasks, suitable for the subsequent computation.

Similarly, the instance–task semantic similarities s_p and s_q can also be normalized. Because both values fall in the $[0.6, 1]$ interval⁵, we apply the same normalizing computation as in Equation (6) to obtain their rescaled counterparts:

$$\hat{s}_p = \frac{s_p - 0.6}{1 - 0.6} \quad (7)$$

$$\hat{s}_q = \frac{s_q - 0.6}{1 - 0.6} \quad (8)$$

Consequently, the normalized scores $\hat{s}_p$ and $\hat{s}_q$ are also guaranteed to lie in the interval $[0, 1]$.

3.3.3 Weighted Concatenation

When aggregating two LoRA blocks, ΔW_p and ΔW_q , we can assign different weights to each block according to their relative importance. Assigning separate weights, denoted as λ_p and λ_q , enables EBS to better capture and leverage the varying relevance of each LoRA block to the current input instance t_0 . To enforce a normalized weighting scheme, we define the weights to sum to one: $\lambda_p + \lambda_q = 1$.

The question is how to determine λ_p and λ_q . A naive choice is to apply the softmax (Bridle, 1989) to the two normalized scores $\hat{s}_p$ and $\hat{s}_q$ to obtain these two weights. Yet this ignores the normalized task–task semantic similarity $\hat{s}$. Intuitively:

- If $\hat{s}$ is large, tasks T_p and T_q are more closely related; the knowledge in ΔW_p and ΔW_q is likely jointly useful. We therefore prefer the two weights not to diverge too much, so as to better balance the importance of each block.
- If $\hat{s}$ is small, the two tasks are more weakly related; one LoRA block is probably more relevant to the current input instance than the other. In this case, we aim to create a sharper

⁵Note that $s' > \alpha$ does not guarantee $s_p > \alpha$ or $s_q > \alpha$.

contrast between the weights, assigning even greater emphasis to the block most relevant to the instance.

To reflect this intuition, we use $\hat{s}$ as the temperature in the softmax: a higher temperature makes the distribution flatter (weights closer), while a lower temperature makes it sharper (weights further apart). As a result, the weighting coefficients λ_p and λ_q are computed as follows:

$$\lambda_p = \frac{\exp\left(\frac{\hat{s}_p}{\hat{s}}\right)}{\exp\left(\frac{\hat{s}_p}{\hat{s}}\right) + \exp\left(\frac{\hat{s}_q}{\hat{s}}\right)} \quad (9)$$

$$\lambda_q = \frac{\exp\left(\frac{\hat{s}_q}{\hat{s}}\right)}{\exp\left(\frac{\hat{s}_p}{\hat{s}}\right) + \exp\left(\frac{\hat{s}_q}{\hat{s}}\right)} \quad (10)$$

To avoid complexity and keep the computation as simple as possible, we adopt a straightforward, intuitive strategy. We directly perform a weighted concatenation (Prabhakar et al., 2025) of the two LoRA blocks, $\Delta W_p = B_p A_p$ and $\Delta W_q = B_q A_q$:

$$B_{\text{cat}} = [\lambda_p B_p, \lambda_q B_q] \quad (11)$$

$$A_{\text{cat}} = \begin{bmatrix} A_p \\ A_q \end{bmatrix} \quad (12)$$

$$\Delta W_{\text{cat}} = B_{\text{cat}} A_{\text{cat}} \quad (13)$$

The two LoRA blocks are aggregated into a single block, ΔW_{cat} , which can then be directly applied to the input instance t_0 . This not only preserves computational efficiency (it avoids utilizing many LoRA blocks at the same time), but also promotes effective knowledge transfer.

4 Experiments

4.1 Datasets

Following previous studies, we use two widely adopted benchmarks: SuperNI and Long Sequence⁶ to evaluate EBS. To ensure a fair comparison, we follow (Zhao et al., 2024) by selecting the same 15 tasks and using the same instance sampling strategy for each benchmark. And for each benchmark, we evaluate on the two task orders specified in (Zhao et al., 2024), and all reported results are averaged over two orders. Furthermore, we also ensure consistency in the evaluation metrics used for every task.

⁶Detailed statistics and descriptions of these two benchmarks are provided in (Wang et al., 2022a) and (Razdaibiedina et al., 2023), respectively.

Methods	SuperNI			Long Sequence		
	AP $\uparrow$	FR $\downarrow$	BT $\uparrow$	AP $\uparrow$	FR $\downarrow$	BT $\uparrow$
SeqLoRA	6.43	33.39	-30.94	9.72	78.61	-73.37
Replay	35.37	16.92	-15.79	71.28	13.05	-12.18
L2P	12.73	11.87	-7.95	57.98	22.49	-16.63
LFPT5	34.37	15.80	-14.47	67.01	13.89	-12.80
ProgPrompt	3.34	35.57	-33.18	7.98	71.55	-66.71
EPI	-	-	-	75.15	1.61	-1.42
O-LoRA	25.89	26.37	-24.59	69.24	7.00	-4.05
SAPT-Prompt	41.11	1.32	-0.65	79.14	1.68	-1.48
SAPT-LoRA	<u>51.54</u>	<u>0.91</u>	<u>-0.57</u>	<u>82.02</u>	<u>1.50</u>	<u>-1.25</u>
MIGU	-	-	-	76.3	-	-
Recurrent-KIF	43.3	-	-8.4	-	-	-
EBS	52.85	0.04	-0.04	82.44	0.31	-0.24

Table 1: Results of different methods on the T5 model (encoder-decoder architecture). All baseline results are taken from (Zhao et al., 2024), except for those of MIGU and Recurrent-KIF. The results for MIGU and Recurrent-KIF are both taken from their respective papers. $\uparrow$ indicates that higher values are better, while $\downarrow$ indicates that lower values are better. The best results are bold-typed and the second best ones are underlined.

4.2 Implementation Details

Our experiments are conducted on an NVIDIA GPU. We use the AdamW optimizer (Loshchilov and Hutter, 2019) and select the best-performing checkpoint during training. In training, we employ BF16 precision, while FP32 is used for inference. The threshold α is set to 0.9.

4.3 Evaluation Metrics

Following previous studies, we assess the performance of EBS using Average Performance (AP) (Chaudhry et al., 2018), Forgetting Rate (FR) (Chaudhry et al., 2018), and Backward Transfer (BT) (Ke and Liu, 2023).

AP denotes the average performance across all tasks evaluated after training on the last task. FR measures the extent to which knowledge acquired from previous tasks has been forgotten. BT measures the improvement in performance on previously learned tasks resulting from learning new tasks, and we emphasize that BT should be considered the central evaluation metric for assessing the knowledge transfer capability of our model. Note that BT may be negative.

4.4 Main Results

We conduct experiments on the T5 model, i.e., google-t5/t5-large. Table 1 presents the comparative results. Our model surpasses the strong baseline, SAPT-LoRA, across all metrics. This

Methods	SuperNI	Long Sequence
	AP $\uparrow$	AP $\uparrow$
Replay	39.48	71.43
LFPT5	38.71	70.31
EPI	-	72.27
O-LoRA	24.26	54.95
SAPT-Prompt	47.39	77.62
SAPT-LoRA	<u>56.23</u>	<u>81.75</u>
EBS	56.56	84.86

Table 2: Results of different methods on the LLaMA model (decoder-only architecture). All baseline results are taken from (Zhao et al., 2024). The best results are bold-typed and the second best ones are underlined.

Metric	SAPT-LoRA	EBS
Training time	1h 58m 20s	38m 13s
Training memory	19445MiB	7991MiB
Inference time	56m 37s	24m 5s
FLOPs/token	1G	824M

Table 3: Efficiency comparison of SAPT-LoRA and EBS on the SuperNI benchmark. To ensure a fair comparison, we keep all variables that could potentially affect the results consistent across experiments. Notably, the reported FLOPs/token is an empirical measurement obtained after the model had reached stable operation, rather than a theoretical estimate.

demonstrates that our model has successfully achieved state-of-the-art performance.

We conduct experiments on the LLaMA model, i.e., meta-llama/Llama-2-7b-hf (Touvron et al., 2023). Table 2 presents the comparative results. Our model surpasses the strong baseline, SAPT-LoRA. This demonstrates that our model has successfully achieved state-of-the-art performance.

4.5 Efficiency Performance

As mentioned above, EBS substantially reduces computational cost (and GPU memory usage in training), especially when the number of tasks becomes very large. To evaluate efficiency, we compare EBS against the strong baseline SAPT-LoRA on the SuperNI benchmark using the T5 model, i.e., google-t5/t5-large. Table 3 summarizes the comparison results in terms of training time, training GPU memory, inference time, and FLOPs/token. Here, *training time* refers to the time required to train the model on the last task. *Training memory* denotes the average GPU memory consumption in training on the last task. *Inference time* is measured as the total time needed to perform inference on all tasks after the entire continual learning process. *FLOPs/token* represents

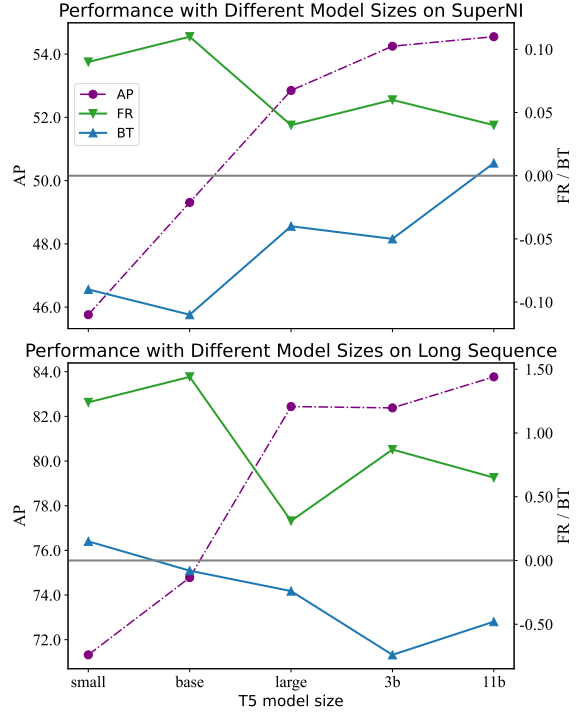


Figure 4: Performance with different T5 model sizes on two benchmarks.

the average number of floating-point operations per token, calculated over the test sets of all tasks after completing the continual learning process over the entire benchmark. The results clearly show that our method is efficient, achieving markedly lower training time, inference time, memory consumption, and computational cost compared to SAPT-LoRA.

Task-vector acquisition is not included in the training times reported in Table 3 and incurs a separate one-time preprocessing cost. We profile this cost on an NVIDIA GeForce RTX 5060 Ti under a 16 GiB GPU-memory limit. Averaged over all 30 tasks from SuperNI and Long Sequence, task-vector acquisition takes 6.67 s per task. The measurement includes text formatting, tokenization, dynamic padding, transferring the padded input tensors from CPU memory to GPU memory, encoder forward passes, pooling, normalization, transferring the resulting embeddings from GPU memory back to CPU memory, and concatenating the returned embeddings on the CPU. It excludes model and dataset loading and disk I/O.

4.6 Scale to Different Model Sizes

Figure 4 reports our method’s performance on SuperNI (top) and Long Sequence (bottom) as we scale the T5 model (google-t5/t5-large) through five sizes. The results show that Aver-

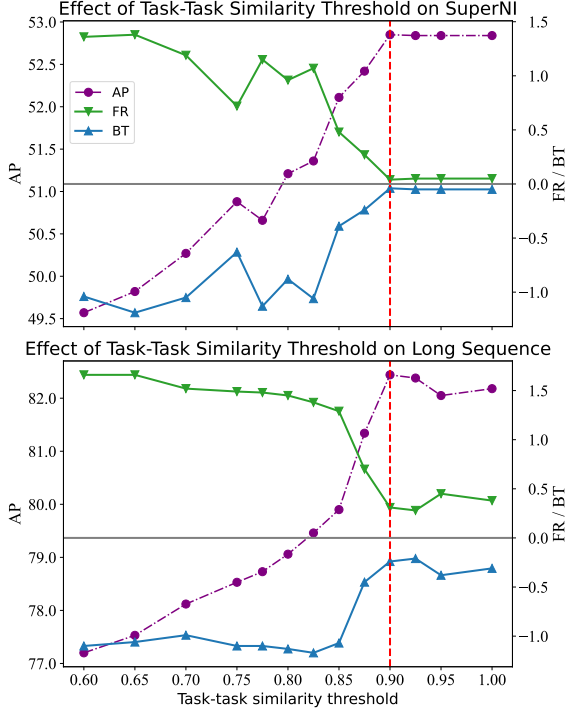


Figure 5: Effect of the threshold α on two benchmarks.

age Performance increases with larger model sizes, whereas the Forgetting Rate shows a downward trend.

4.7 Effect of Threshold α

Figure 5 presents the effects of different thresholds on performance of the T5 model. As we mentioned above, α is a hyperparameter. The figure demonstrates that, for both benchmarks, the optimal threshold is 0.90. The fact that different benchmarks share the same optimal threshold may be attributed to the use of the BGE model without any further training. Since the parameters of the BGE model are not updated for either benchmark, the optimal threshold of 0.90 is likely determined by the characteristics of the BGE model itself, rather than by the specifics of the individual benchmarks. This observation warrants further investigation.

When α is set to 0.60, the two LoRA blocks chosen by block-sparse routing are always regarded as related. Consequently, EBS consistently performs concatenation on both blocks. Experimental results indicate that the overall performance deteriorates considerably. This degradation is likely caused by the interference that arises from forcibly combining two unrelated blocks.

When α is set to 1.00, the two LoRA blocks chosen by block-sparse routing are always regarded

Methods	SuperNI			Long Sequence		
	AP $\uparrow$	FR $\downarrow$	BT $\uparrow$	AP $\uparrow$	FR $\downarrow$	BT $\uparrow$
GM	52.87	0.03	-0.03	82.18	0.42	-0.35
MP	46.11	0.14	0.03	51.69	1.05	<u>-0.35</u>
CM	52.90	0.01	<u>-0.01</u>	82.18	0.42	<u>-0.35</u>
PC1	17.21	7.47	<u>-7.36</u>	14.52	28.11	-26.06
RS	50.65	0.78	-0.78	64.53	4.82	-4.75
AM	52.85	0.04	-0.04	82.44	0.31	-0.24

Table 4: Results under different task vector acquisition methods. The best results are bold-typed and the second best ones are underlined.

Methods	SuperNI			Long Sequence		
	AP $\uparrow$	FR $\downarrow$	BT $\uparrow$	AP $\uparrow$	FR $\downarrow$	BT $\uparrow$
<u>Linear</u>	52.37	0.30	-0.25	81.98	0.56	-0.42
TIES	37.66	2.06	-1.78	53.10	5.36	-4.97
DARE TIES	6.96	3.04	-3.00	9.57	5.54	-5.54
DARE linear	7.50	2.92	-2.88	10.10	5.47	-5.47
Magnitude prune	51.97	0.39	-0.34	81.85	0.66	-0.52
Concatenation	52.85	0.04	-0.04	82.44	0.31	-0.24

Table 5: Results under different LoRA block aggregation strategies. The best results are bold-typed and the second best ones are underlined.

as unrelated. Consequently, EBS always chooses only one block for inference. Experimental results demonstrate that if knowledge transfer is entirely omitted, the performance remains suboptimal.

4.8 Effect of Vector Acquisition Method

As mentioned previously, we obtain a task vector $\mathbf{m}'_i$ by taking the arithmetic mean of all vectors in $\mathcal{V}_i$. This naturally leads to the question of whether there exists a more effective approach for computing $\mathbf{m}'_i$ from $\mathcal{V}_i$. To address this, we carried out additional experiments with the T5 model. Table 4 summarizes the differences in the results under different computation strategies. In the table, GM refers to geometric median (Unnam and Reddy, 2020), MP to max-pooling over the vector dimension, CM to coordinate-wise median, PC1 to the first principal component (Jolliffe and Cadima, 2016), RS to randomly sample a vector from $\mathcal{V}_i$, and AM to our arithmetic mean. Experimental results demonstrate that GM, CM, and our AM all yield excellent results.

4.9 Effect of LoRA Aggregation Strategy

We directly perform the concatenation of the two LoRA blocks. As for the effectiveness of other LoRA block aggregation strategies, we carry out a comprehensive study. Besides the concatena-

tion adopted in our work, we also examine linear (Prabhakar et al., 2025), TIES (Yadav et al., 2023), DARE TIES (Yu et al., 2024), DARE linear (Yu et al., 2024), and magnitude prune (Mangrulkar and Paul, 2024) as alternative aggregation strategies. Table 5 reports the differences in the performance on the T5 model of EBS when applying various aggregation strategies. Results demonstrate that, although our method is quite simple, it achieves the best.

4.10 Reliability of Block-Sparse Routing

A potential concern of the proposed block-sparse routing is that it may fail to activate the LoRA block corresponding to the true underlying task, which could lead to performance degradation.

To evaluate this concern, we empirically assess whether the LoRA block corresponding to the correct task is included in the block(s) activated by our routing procedure. On SuperNI, the activated block set always contains the correct LoRA block (100% inclusion). On Long Sequence, only 0.41% of test instances fail to include the correct LoRA block in the activated set, indicating that routing errors are rare in practice.

We further assess the routing accuracy when restricting inference to the single most similar LoRA block (top-1). Under this stricter setting, the routing accuracy is 99.75% on SuperNI and 92.85% on Long Sequence. These results suggest that, although routing is learned-free and purely similarity-based, it reliably retrieves the correct LoRA block in most cases.

5 Conclusion

In this paper, we propose EBS for continual learning of LLMs, limiting the continual increase in computational cost. EBS comprises block-isolated learning, where only one LoRA block is enabled, and block-sparse routing, where at most two LoRA blocks are activated. Our framework offers a simple implementation that works across different model sizes and transformer architectures (decoder-only and encoder-decoder) while consistently delivering strong performance.

Limitations

Since most baseline methods are evaluated on the T5 and LLaMA model families, we follow the same experimental setup. Due to the lack of comparable baselines under a consistent setting, we do not in-

clude results on the latest language models in our paper.

After learning i tasks, EBS retains i LoRA blocks, so adapter storage grows linearly. Our current implementation loads all learned LoRA blocks at process initialization and keeps them GPU-resident during inference; total inference GPU memory therefore also grows with i , although at most two blocks participate in each LLM forward pass. A tiered GPU-CPU-storage cache could reduce GPU residency at the cost of block-loading, cache-management, and latency overhead, but such a system is not implemented or evaluated in this work.

As discussed in Section 4.7, the optimal threshold remains the same (0.90) on both benchmarks. Based on this, we infer that the optimal threshold is likely determined by the characteristics of the BGE model itself, rather than by the specifics of the individual benchmarks. This observation warrants further investigation.

Our current framework is inherently limited to activating at most two LoRA blocks. While activating additional blocks at inference time could be beneficial for some instances, our present design does not support selecting more than two blocks. Extending the framework to enable the activation of more than two LoRA blocks is therefore a natural direction for future work.

Acknowledgments

This work is supported by the National Natural Science Foundation of China (No.62406151, No.62607054).

References

- John S. Bridle. 1989. [Training stochastic model recognition algorithms as networks can lead to maximum mutual information estimation of parameters](#). In *Advances in Neural Information Processing Systems*, volume 2, pages 211–217. Morgan Kaufmann.
- Arslan Chaudhry, Puneet K. Dokania, Thalaiyasingam Ajanthan, and Philip H. S. Torr. 2018. Riemannian walk for incremental learning: Understanding forgetting and intransigence. In *Computer Vision – ECCV 2018*, pages 556–572, Cham. Springer International Publishing.
- Ning Ding, Yujia Qin, Guang Yang, Fuchao Wei, Zonghan Yang, Yusheng Su, Shengding Hu, Yulin Chen, Chi-Min Chan, Weize Chen, Jing Yi, Weilin Zhao, Xiaozhi Wang, Zhiyuan Liu, Hai-Tao Zheng, Jianfei Chen, Yang Liu, Jie Tang, Juanzi Li, and Maosong

- Sun. 2022. [Delta tuning: A comprehensive study of parameter efficient methods for pre-trained language models](#). *Preprint*, arXiv:2203.06904.
- Wenyu Du, Shuang Cheng, Tongxu Luo, Zihan Qiu, Zeyu Huang, Ka Chun Cheung, Reynold Cheng, and Jie Fu. 2024. [Unlocking continual learning abilities in language models](#). *Preprint*, arXiv:2406.17245.
- Yujie Feng, Xujia Wang, Zexin Lu, Shenghong Fu, Guangyuan Shi, Yongxin Xu, Yasha Wang, Philip S. Yu, Xu Chu, and Xiao-Ming Wu. 2025. [Recurrent knowledge identification and fusion for language model continual learning](#). In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 27396–27413, Vienna, Austria. Association for Computational Linguistics.
- Robert M. French. 1999. [Catastrophic forgetting in connectionist networks](#). *Trends in Cognitive Sciences*, 3(4):128–135.
- Edward J Hu, yelong shen, Phillip Wallis, Zeyuan Allen-Zhu, Yanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. [LoRA: Low-rank adaptation of large language models](#). In *International Conference on Learning Representations*.
- Mohit Iyyer, Varun Manjunatha, Jordan Boyd-Graber, and Hal Daumé III. 2015. [Deep unordered composition rivals syntactic methods for text classification](#). In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1681–1691, Beijing, China. Association for Computational Linguistics.
- Ian T. Jolliffe and Jorge Cadima. 2016. [Principal component analysis: a review and recent developments](#). *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 374(2065):20150202.
- Zixuan Ke and Bing Liu. 2023. [Continual learning of natural language processing tasks: A survey](#). *Preprint*, arXiv:2211.12701.
- Brian Lester, Rami Al-Rfou, and Noah Constant. 2021. [The power of scale for parameter-efficient prompt tuning](#). In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 3045–3059, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.
- Ilya Loshchilov and Frank Hutter. 2019. [Decoupled weight decay regularization](#). In *International Conference on Learning Representations*.
- Sourab Mangrulkar and Sayak Paul. 2024. [PEFT welcomes new merging methods](#). Hugging Face Blog.
- Akshara Prabhakar, Yanzhi Li, Karthik Narasimhan, Sham Kakade, Eran Malach, and Samy Jelassi. 2025. [LoRA soups: Merging LoRAs for practical skill composition tasks](#). In *Proceedings of the 31st International Conference on Computational Linguistics: Industry Track*, pages 644–655, Abu Dhabi, UAE. Association for Computational Linguistics.
- Chengwei Qin and Shafiq Joty. 2022. [LFPT5: A unified framework for lifelong few-shot language learning based on prompt tuning of t5](#). In *International Conference on Learning Representations*.
- Qwen Team. 2026. [Qwen3.5: Towards native multi-modal agents](#).
- Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. [Exploring the limits of transfer learning with a unified text-to-text transformer](#). *Journal of Machine Learning Research*, 21(140):1–67.
- Anastasia Razdaibiedina, Yuning Mao, Rui Hou, Madian Khabisa, Mike Lewis, and Amjad Almahairi. 2023. [Progressive prompts: Continual learning for language models](#). In *The Eleventh International Conference on Learning Representations*.
- Mark Bishop Ring. 1994. [Continual Learning in Reinforcement Environments](#). Ph.D. thesis, The University of Texas at Austin.
- G. Salton, A. Wong, and C. S. Yang. 1975. [A vector space model for automatic indexing](#). *Commun. ACM*, 18(11):613–620.
- Sebastian Thrun. 1995. [Is learning the n-th thing any easier than learning the first?](#) In *Advances in Neural Information Processing Systems*, volume 8, pages 640–646. MIT Press.
- Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, and 49 others. 2023. [Llama 2: Open foundation and fine-tuned chat models](#). *Preprint*, arXiv:2307.09288.
- Narendra Babu Unnam and P. Krishna Reddy. 2020. [A document representation framework with interpretable features using pre-trained word embeddings](#). *International Journal of Data Science and Analytics*, 10(1):49–64.
- Xiao Wang, Tianze Chen, Qiming Ge, Han Xia, Rong Bao, Rui Zheng, Qi Zhang, Tao Gui, and Xuanjing Huang. 2023a. [Orthogonal subspace learning for language model continual learning](#). *Preprint*, arXiv:2310.14152.
- Yizhong Wang, Swaroop Mishra, Pegah Alipoormolabashi, Yeganeh Kordi, Amirreza Mirzaei, Atharva

Naik, Arjun Ashok, Arut Selvan Dhanasekaran, Anjana Arunkumar, David Stap, Eshaan Pathak, Gianis Karamanolakis, Haizhi Lai, Ishan Purohit, Ishani Mondal, Jacob Anderson, Kirby Kuznia, Krithika Doshi, Kuntal Kumar Pal, and 16 others. 2022a. [Super-NaturalInstructions: Generalization via declarative instructions on 1600+ NLP tasks](#). In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 5085–5109, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Zhicheng Wang, Yufang Liu, Tao Ji, Xiaoling Wang, Yuanbin Wu, Congcong Jiang, Ye Chao, Zhencong Han, Ling Wang, Xu Shao, and Wenqiu Zeng. 2023b. [Rehearsal-free continual language learning via efficient parameter isolation](#). In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 10933–10946, Toronto, Canada. Association for Computational Linguistics.

Zifeng Wang, Zizhao Zhang, Chen-Yu Lee, Han Zhang, Ruoxi Sun, Xiaoqi Ren, Guolong Su, Vincent Perot, Jennifer Dy, and Tomas Pfister. 2022b. Learning to prompt for continual learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 139–149.

Shitao Xiao, Zheng Liu, Peitian Zhang, Niklas Muenighoff, Defu Lian, and Jian-Yun Nie. 2024. [C-pack: Packed resources for general chinese embeddings](#). In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR '24*, page 641–649, New York, NY, USA. Association for Computing Machinery.

Prateek Yadav, Derek Tam, Leshem Choshen, Colin Raffel, and Mohit Bansal. 2023. [Ties-merging: Resolving interference when merging models](#). In *Advances in Neural Information Processing Systems*, volume 36, pages 7093–7115. Curran Associates, Inc.

Le Yu, Bowen Yu, Haiyang Yu, Fei Huang, and Yongbin Li. 2024. [Language models are super mario: Absorbing abilities from homologous models as a free lunch](#). In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 57755–57775. PMLR.

Weixiang Zhao, Shilong Wang, Yulin Hu, Yanyan Zhao, Bing Qin, Xuanyu Zhang, Qing Yang, Dongliang Xu, and Wanxiang Che. 2024. [SAPT: A shared attention framework for parameter-efficient continual learning of large language models](#). In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 11641–11661, Bangkok, Thailand. Association for Computational Linguistics.

A Small-Sample Setting

Benchmark	AP↑	FR↓	BT↑
SuperNI (full)	52.85	0.04	-0.04
SuperNI (2%)	50.61	0.24	-0.24
Long Sequence (full)	82.44	0.31	-0.24
Long Sequence (2%)	82.38	0.39	-0.32

Table 6: Results of different sample sizes on the T5 model.

We want to know whether our method remains effective when only a small amount of data is available to derive the task vectors. To assess this, we randomly sample 2% of the original training set for each task to derive task vectors, while keeping all other settings unchanged. We conduct experiments with the google-t5/t5-large. The results are shown in Table 6. Despite the extremely small sample size, performance does not degrade substantially on both benchmarks.

B Experiments on Qwen3.5-2B-Base

Methods	SuperNI	Long Sequence
	AP↑	AP↑
SAPT-LoRA	56.25	81.90
EBS	57.33	85.73

Table 7: Results of SAPT-LoRA and EBS on the Qwen3.5-2B-Base model.

To evaluate whether our approach scales to current architectures, we replace the base model with Qwen/Qwen3.5-2B-Base (Qwen Team, 2026), a recent model from the Qwen LLM family. As shown in Table 7, the results show that EBS also outperforms the strong baseline SAPT-LoRA on this base model.