Stacked LoRA: Isolated Low-Rank Adaptation for Lifelong Knowledge Management

Heramb Vivek Patil and Vaishnavee Kiran Sanam and Dr. Minakshi Pradeep Atre
PVG's College of Engineering and Technology, Pune
herambpatil2004@gmail.com, vaishnavee.gcloud@gmail.com, mpatre29@gmail.com

Abstract

Continual learning (CL) presents a significant challenge for large pre-trained models, primarily due to catastrophic forgetting and the high computational cost of sequential knowledge updating. Parameter-Efficient Transfer Learning (PETL) methods offer reduced computational burdens but often struggle to effectively mitigate forgetting. This paper introduces Stacked Low-Rank Adaptation (SLoRA), a novel parameter-efficient approach that leverages the additive composition of task-specific, frozen low-rank adapters to enable modular continual learning with inherent support for explicit knowledge modification. SLoRA was evaluated on vision benchmarks, BERT-base, and the 1-billion-parameter Llama-3.2-1B model. Experiments demonstrated that SLoRA almost completely eliminated catastrophic forgetting, achieving a final average accuracy of 92.75% on Llama-3.2-1B while perfectly preserving prior task performance. Furthermore, SLoRA is computationally efficient, enabling up to a 15x training speed-up over full fine-tuning with 99.7% fewer trainable parameters per update. SLoRA offers a compelling balance of forgetting mitigation, parameter efficiency, and modularity, representing a promising direction for developing adaptable and efficient lifelong knowledgeable foundation models.

1 Introduction

The capability to learn from a stream of tasks, incrementally acquiring new knowledge, without forgetting prior knowledge is a central goal of Continual Learning (CL), a necessity for NLP systems deployed in dynamic environments. However, catastrophic forgetting, whereby performance on earlier tasks (representing previously acquired knowledge) is degraded upon learning new ones (acquiring new knowledge), remains a fundamental obstacle (Zeng et al., 2024).

The challenge is amplified by large pretrained models (LPMs) like BERT and Llama, which demand substantial resources for retraining on new tasks. This full fine-tuning approach is often computationally prohibitive and environmentally costly (Patterson et al., 2021). Moreover, it leads to catastrophic forgetting, where performance on earlier tasks is severely degraded upon learning new ones, effectively erasing previously acquired knowledge (Zeng et al., 2024). Parameter-efficient transfer learning (PEFT) approaches address the computational cost by training only a small number of additional parameters per task, making them suitable for CL settings. Techniques including Adapters (Houlsby et al., 2019), Prompt Tuning (Lester et al., 2021), and LoRA (Hu et al., 2022) have shown promise. However, as noted by Coleman et al. (2025), preventing parameter interference during sequential updates remains an open challenge; a naive combination of PEFT and CL often fails as modules still share parameter spaces, leading to interference.

Interference, leading to the corruption of previously acquired knowledge, hampers traditional PEFT techniques when modules are shared across tasks (He et al., 2021; Wang et al., 2023). While assigning isolated modules to each task prevents forgetting, this strategy leads to unbounded growth in parameters for storing this modular knowledge and lacks a mechanism for explicit knowledge unlearning. Prior works investigating routing (Zhang et al., 2023), mixture-of-experts (Feng et al., 2024), or orthogonal subspace projection (Wang et al., 2023) to manage knowledge interactions often introduce additional complexity or depend on known task identity at inference for knowledge retrieval.

Recent works like InfLoRA (Liang and Li, 2024) and SD-LoRA (Wu et al., 2025) also address cumulative LoRA usage, but with different goals. These methods target the task-agnostic Class-Incremental Learning (CIL) scenario, requiring them to merge

Proceedings of the IJCNLP-AACL 2025 Student Research Workshop, pages 36–46

or blend knowledge into a single model, which forfeits the ability to unlearn. InfLoRA permanently merges adapters, while SD-LoRA retrains all adapter "magnitudes" at each step, breaking parameter isolation.

To address these challenges, we introduce Stacked Low-Rank Adaptation (SLoRA), a novel parameter-efficient approach for the Task-Incremental Learning (TIL) setting. SLoRA provides strong knowledge retention with inherent modularity by additively composing strictly frozen, task-specific low-rank adapters. This architectural isolation is simpler than algorithmic orthogonality and, crucially, enables explicit knowledge modification (i.e., unlearning) by deactivating adapters, a feature not possible with merging or blending approaches. Our evaluations on vision and NLP benchmarks demonstrate SLoRA's effectiveness in mitigating catastrophic knowledge loss while maintaining a competitive parameter footprint. This work lays a strong foundation for adaptable lifelong knowledgeable foundation models.

2 Related Work

Continual Learning (CL) addresses the challenge of learning from a sequence of tasks, incrementally updating models with new knowledge, without forgetting previous knowledge. A fundamental obstacle in CL is catastrophic forgetting (loss of prior knowledge), where adaptation to new tasks (acquisition of new knowledge) degrades performance on earlier ones (Zeng et al., 2024). In NLP, large pre-trained Transformer models require efficient adaptation to new tasks; updating all parameters per task is prohibitively expensive when aiming for efficient knowledge updates. Parameter-Efficient Transfer Learning (PETL) methods tackle this by fine-tuning only a small subset of parameters, yielding benefits in compute, storage, and modularity for injecting new knowledge (Houlsby et al., 2019; He et al., 2021).

Adapter Modules insert small bottleneck layers into each Transformer block, training only these new parameters. The original Adapter approach (Houlsby et al., 2019) demonstrated near full-fine-tuning performance on GLUE while adding only ~3.6% parameters per task. However, naively adding new adapters per task leads to linear growth in parameters for storing task-specific knowledge and can increase inference latency.

LoRA (Low-Rank Adaptation) freezes the original weights and injects trainable low-rank decomposition matrices into each layer, reducing trainable parameters by orders of magnitude and incurring no extra inference cost once merged (Hu et al., 2022). Its performance is sensitive to the chosen rank but matches full fine-tuning quality in many settings for single-task knowledge adaptation.

Prompt-Based Methods, including Prefix-Tuning (Li and Liang, 2021) and Prompt-Tuning (Lester et al., 2021), optimize continuous prefix vectors or soft prompt tokens prepended to inputs, tuning as little as 0.1% of parameters. These methods can be very parameter-efficient for accessing specific knowledge representations but need careful prompt design and may vary in effectiveness across tasks.

While PETL methods excel in single-task adaptation (knowledge injection), applying them to CL (continuous knowledge updating) brings new challenges. As our results for LoRA-Cont (Section 4.4) confirm, a naive sequential application of LoRA fails, suffering severe catastrophic forgetting. This highlights that a dedicated architecture is required. A recent survey specifically on Parameter-Efficient Continual Fine-Tuning highlights the open questions at the intersection of CL and PETL (Coleman et al., 2025).

Several works extend LoRA for CL by enforcing desirable properties in adapter parameters to manage knowledge interactions: **O-LoRA** encourages orthogonality among low-rank adapters for different tasks to reduce interference, effectively eliminating forgetting (preserving knowledge) with only marginal extra parameters (Wang et al., 2023). **C-LoRA** introduces a learnable routing matrix that dynamically allocates subspaces for previous and new tasks, achieving scalable continual adaptation for managing knowledge subspaces without maintaing separate adapters per task (Zhang et al., 2025).

Modular Adapter Approaches allocate task-specific parameters for encapsulating knowledge and freeze them thereafter. While this isolates task knowledge and prevents forgetting (knowledge loss), it leads to parameter counts growing linearly with the number of tasks. **AdapterFusion** combines multiple frozen adapters representing task knowledge by learning a fusion layer that integrates their outputs non-destructively, leveraging cross-task knowledge transfer at the cost of extra composition parameters (Pfeiffer et al., 2020).

Beyond single-method strategies, a growing body of work explores compositional PEFT mod-

ules for CL and multi-task learning by combining knowledge adaptations: **ReLoRA** periodically merges low-rank updates back into the model and reinitializes adapters during training, effectively increasing representational capacity and improving convergence speed (Lialin et al., 2023). **LoraHub** dynamically composes multiple pre-trained LoRA modules for few-shot generalization on unseen tasks, requiring no additional parameters or gradients at inference for knowledge retrieval and composition (Huang et al., 2023). **Task Arithmetic** treats each adapter update as a vector in weight space and performs linear operations (addition, subtraction) to combine task knowledge, enabling straightforward module composition (Zhang et al., 2023). **Mixture-of-LoRAs** (MoA) trains multiple domain experts via LoRA and uses an explicit routing mechanism to select and combine experts per input, blending Mixture-of-Experts principles with LoRA's efficiency for expert-based knowledge retrieval (Feng et al., 2024).

Distinctions from LoRA-based CIL Methods. Our work is related to other LoRA-based CL methods like InfLoRA (Liang and Li, 2024) and SD-LoRA (Wu et al., 2025), but SLoRA is fundamentally different in its problem setting, mechanism, and capabilities.

- **Problem Setting:** InfLoRA and SD-LoRA are designed for Class-Incremental Learning (CIL), which requires a single model to operate without task identity. SLoRA is designed for the Task-Incremental Learning (TIL) setting, where task identity is known at inference.
- **Mechanism:** To achieve its task-agnostic goal, InfLoRA uses permanent merging (losing modularity) and SD-LoRA uses collaborative blending (retraining all adapter magnitudes, breaking isolation). SLoRA uses strict architectural isolation by freezing all past adapters.
- **Capability:** SLoRA's TIL design and isolation mechanism provide a unique capability the CIL methods cannot: explicit knowledge unlearning. A task can be removed simply by deactivating its adapter, which is impossible in models that merge or blend parameters.

Despite these advancements, key trade-offs remain between stability (retaining acquired knowledge), plasticity (acquiring new knowledge) and parameter growth. Our proposed Stacked Low-Rank Adaptation (SLoRA) addresses these by stacking individually trained and frozen low-rank adapters additively, ensuring clear parameter isolation (for knowledge encapsulation), straightforward composition (including unlearning), and inherently modular knowledge management.

3 Methodology

Continual Learning (CL) aims to train models sequentially on new tasks, incrementally updating their knowledge, without forgetting previous knowledge. A key challenge is catastrophic forgetting (knowledge loss) in large pre-trained models, necessitating parameter-efficient adaptation for knowledge acquisition. Stacked Low-Rank Adaptation (SLoRA) is proposed as a novel method for parameter-efficient CL that mitigates forgetting through additive composition of task-specific low-rank adapters (representing task-specific knowledge adaptations).

3.1 SLoRA Method

SLoRA builds on Low-Rank Adaptation (LoRA), which adapts pre-trained weights W_0 by adding a low-rank update $\Delta W = \frac{\alpha}{r} BA$, where $A \in \mathbb{R}^{r \times d_{in}}$, $B \in \mathbb{R}^{d_{out} \times r}$, $r \ll \min(d_{in}, d_{out})$. LoRA trains only A and B, keeping W_0 frozen (Hu et al., 2022). SLoRA extends this by applying additively multiple low-rank task-specific updates. After training in T tasks (0-indexed), the effective weight $W^{(T-1)}$ is the sum of W_0, a base update ΔW_{base}, and stack updates $(T-1)$ $\Delta W_{stack,t}$:

$$W^{(T-1)} = W_0 + \Delta W_{base} + \sum_{t=1}^{T-1} \Delta W_{stack,t}$$

where $\Delta W_{base} = \frac{\alpha_{base}}{r_{base}} B_{base} A_{base}$ and $\Delta W_{stack,t} = \frac{\alpha_{stack}}{r_{stack}} B_{stack,t} A_{stack,t}$ for task t. This parallel and additive composition is depicted in Figure 1.

The training is sequential. For the base task (Task 0), a base LoRA adapter (A_{base}, B_{base}) is attached and trained with W_0 frozen. For each subsequent task $t > 0$ (representing the acquisition of new knowledge), a *new* stack adapter $(A_{stack,t}, B_{stack,t})$ is initialized and added. Crucially, W_0, the base adapter, and *all previously trained stack adapters* are held frozen. Only the newly added stack adapter and the task classifier are trained on the data of task t. This strict parameter isolation prevents interference and protects previously acquired knowledge.

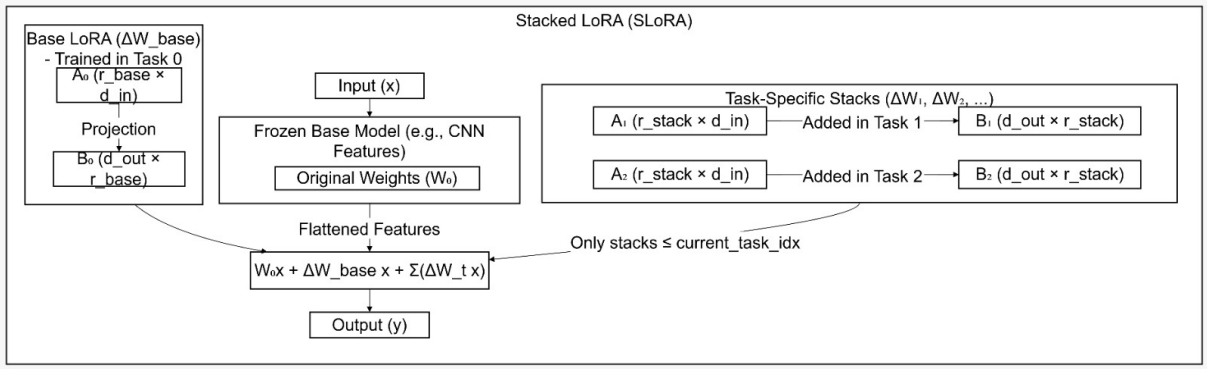

Figure 1: SLoRA architecture: Additive composition of task-specific LoRA adapters in parallel with the base weight. The base model (W_0) and all previously trained adapters (ΔW_{base}, $\Delta W_{stack,1}$) are frozen. Only the new adapter for the current task ($\Delta W_{stack,2}$) is trained. Inference on a task k is performed by summing adapters up to k.

At inference time, to evaluate performance on Task k (0-indexed), the effective weight matrix $W^{(k)}$ is formed by summing W_0, the base adapter, and all stack adapters up to task k: $W^{(k)} = W_0 + \Delta W_{base} + \sum_{t=1}^{k} \Delta W_{stack,t}$. This "selective activation" uses only task-relevant knowledge adaptations. A direct benefit of this modular and additive structure is explicit knowledge modification: Task k is "unlearned" by excluding its stack adapter from the summation during inference (e.g., by adjusting a task index variable), requiring no additional training.

3.2 Experimental Setup

Experiments were conducted on Permuted-MNIST, Split-CIFAR100, and sequential NLP tasks using BERT-base-uncased, across 3 random seeds. The baselines included full fine-tuning (FT), elastic weight consolidation (EWC) (Kirkpatrick et al., 2017), standard continual LoRA (LoRA-Cont), and independent LoRA adapters per task (LoRA-Ind). The implementation used PyTorch and Avalanche (Lomonaco et al., 2021).

For Permuted-MNIST (5 tasks), an MLP with two linear layers was used as the base model, adapted with SLoRA. Training used 2 epochs/task and batch size 64. LoRA (Cont and Ind) used rank 8, alpha 16. SLoRA used base rank 8, alpha 16, stack rank 4, alpha 16. The learning rates were 1e-3 for all methods.

For Split-CIFAR100 (10 tasks), a SimpleCNN with frozen convolutional layers and a two-linear-layer classifier was used. The classifier linear layers were adapted. Training used 50 epochs/task, batch size 64. LoRA (Cont/Ind) used rank 8, alpha 16. SLoRA used base rank 16, alpha 32, stack rank 8, alpha 16. LRs were 1e-3 for all methods. EWC used lambda 1000.

For Sequential NLP Tasks (4 tasks), a frozen BERT-base-uncased model was adapted in its linear layers. Tasks were SST-2, TREC, Yelp Polarity, and Amazon Polarity, using a 10000-example subset per task. Training used 15 epochs/task, batch size 16, max length 128. LoRA (Cont) used rank 8. SLoRA used base rank 8, stack rank 4. LRs were 1e-3 for all methods.

Evaluation after training each task involved measuring accuracy on all tasks seen so far. SLoRA was evaluated using selective activation based on the task index. LoRA-Ind performance was measured by loading the saved task-specific adapter parameters.

To assess scalability on modern LLMs, we conducted further experiments on the `meta-llama/Llama-3.2-1B` model. We used the same sequence of four NLP tasks (SST-2, TREC, Yelp Polarity, Amazon Polarity) with 10,000 examples per task. SLoRA was applied to the linear layers of the attention and feed-forward networks. The base adapter was configured with a rank (r_{base}) of 8 and alpha of 16. Subsequent task-specific stack adapters used a rank (r_{stack}) of 4 and alpha of 8. The model was trained for one epoch per task with a batch size of 4 using the AdamW optimizer.

4 Results

This section presents the empirical evaluation of SLoRA against Full Fine-Tuning (FT), Elastic Weight Consolidation (EWC) (Kirkpatrick et al., 2017), standard continual LoRA (LoRA-Cont), and task-independent LoRA (LoRA-Ind). We first evaluate on standard vision benchmarks (Permuted-

MNIST, Split-CIFAR100) for robust comparison against prior CL literature. We then validate SLoRA's scalability and efficiency on large-scale, real-world NLP tasks using BERT-base and the 1-billion-parameter Llama-3.2-1B model.

Performance is assessed based on forgetting mitigation (knowledge retention), overall accuracy, and parameter efficiency. Experiments were conducted using a single NVIDIA T4 GPU for Permuted-MNIST and a single NVIDIA P100 GPU for Split-CIFAR100 and BERT-base experiments. Implementation was done using PyTorch and Avalanche (Lomonaco et al., 2021). Results for Permuted-MNIST and Split-CIFAR100 are reported as mean accuracy ± standard deviation over 3 random seeds. LLM results are from a single seed due to compute constraints and are interpreted as preliminary; EWC and LoRA-Ind results were not available for BERT.

4.1 Overall Findings Summary

Across diverse domains, SLoRA consistently demonstrates effectiveness in mitigating catastrophic forgetting (knowledge loss). Methods training shared parameters (FT, LoRA-Cont) show significant forgetting. SLoRA, by employing additive, task-specific frozen adapters (representing isolated knowledge adaptations), effectively preserves performance on prior tasks comparable to methods like EWC and LoRA-Ind. SLoRA maintains a competitive parameter footprint, scaling linearly with tasks but more efficiently than full fine-tuning.

4.2 Permuted-MNIST Results (5-Task Sequence)

The Permuted-MNIST benchmark evaluates forgetting on a 5-task sequence. To specifically illustrate forgetting on the first task over time, Table 1 shows the performance on Task 1 after training each subsequent task. Table 7 (in Appendix) summarizes the mean accuracy ± standard deviation on each task after training on all 5 tasks.

Table 1: Accuracy (%) on Permuted-MNIST Task 1 after Training Sequential Tasks (Mean ± SD over 3 Seeds)

Method	After Task 1	After Task 2	After Task 3	After Task 4
FT	97.22 ± 0.25	92.64 ± 2.01	78.34 ± 0.82	66.36 ± 11.42
EWC	97.22 ± 0.25	94.51 ± 0.14	81.88 ± 4.09	71.93 ± 7.66
LoRA-Cont	97.22 ± 0.25	92.23 ± 2.01	79.70 ± 5.69	67.25 ± 5.68
SLoRA	97.22 ± 0.25	**92.82 ± 1.22**	**86.30 ± 2.64**	**72.62 ± 8.69**

Table 1 clearly shows the severe forgetting experienced by FT and LoRA-Cont. SLoRA exhibits better retention of Task 1 knowledge. Table 7 (Appendix) confirms SLoRA achieves the highest overall average accuracy.

4.3 Split-CIFAR100 Results (10-Task Sequence)

Experiments were conducted on Split-CIFAR100 (10 classes/task). Table 8 (in Appendix) presents a concise summary, and Figure 2 plots the average accuracy on tasks seen so far.

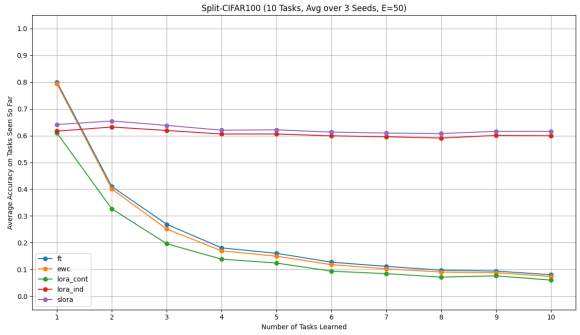

Figure 2: Average accuracy on tasks seen so far on Split-CIFAR100 after training each sequential task (Averaged over 3 Seeds, E=50). SLoRA and LoRA-Ind show near-zero forgetting, while FT, EWC, and LoRA-Cont suffer catastrophic forgetting.

Figure 2 and Table 8 (Appendix) clearly show that FT, EWC, and LoRA-Cont suffer severe catastrophic forgetting. In contrast, LoRA-Ind and SLoRA demonstrate significantly better knowledge retention, maintaining high accuracy on previously learned tasks. SLoRA achieves comparable forgetting mitigation to LoRA-Ind with a slightly higher final average accuracy.

4.4 Sequential NLP Tasks (BERT-base) Results (4-Task Sequence)

The performance of SLoRA was evaluated on a sequence of 4 real-world NLP classification tasks using a frozen BERT-base-uncased model. To validate that a naive PEFT-CL combination fails, we explicitly benchmark against LoRA-Cont.

As shown in Table 2, both FT and the naive LoRA-Cont baseline suffer from severe catastrophic forgetting on BERT-base. This demonstrates that simply using a PEFT method is insufficient. SLoRA's isolated stacked architecture, however, demonstrates remarkable stability, with performance on Tasks 1, 2, and 3 remaining virtually unchanged. SLoRA achieves the highest overall average accuracy (88.33%). On a single NVIDIA

Table 2: Accuracy (%) on Sequential BERT-base Tasks after Training on Task 4 (Single Seed: 42)

Method	Task 1 (SST-2)	Task 2 (TREC)	Task 3 (Yelp)	Task 4 (Amazon)	Avg. Accuracy
FT	60.80	4.26	59.70	53.30	44.51
LoRA-Cont	85.30	74.47	91.50	92.10	85.84
SLoRA	**89.90**	**93.62**	**84.20**	85.60	**88.33**

Table 3: Parameter Efficiency Comparison for BERT-base on 4 Tasks

Method	Trainable Params per Task	Total Unique Params (after 4 Tasks)	Parameter Growth
Full FT	109.5M (same every task)	109.5M	Constant
LoRA-Cont(r=8)	1.35M	1.35M	Constant
LoRA-Ind (r=8)	1.35M	∼5.4M	Linear ($T\times$)
SLoRA (B_r=8, St_r=4)	1.34M (Task 0), 0.67M (T1–T3)	3.35M	Linear ($T\times$)

Note: FT fine-tunes the full model per task. LoRA-Continual updates a shared adapter. LoRA-Independent uses separate adapters per task. SLoRA uses a shared base adapter (Task 0) and stack adapters for subsequent tasks.

P100, SLoRA also converges in only 1.5 minutes per task, an 8x speed-up over FT (12.0 minutes).

The parameter efficiency is detailed in Table 3. SLoRA scales linearly with tasks but requires fewer total parameters than LoRA-Ind due to its smaller stack rank configuration.

4.5 Scalability and Efficiency Analysis on Llama-3.2-1B

To validate SLoRA's performance and efficiency on contemporary large-scale models, we evaluated it on the 1-billion-parameter Llama-3.2-1B. The results confirm that SLoRA's architecture effectively scales, preventing catastrophic forgetting while offering significant computational advantages.

As shown in Table 4, SLoRA achieves a high final average accuracy of 92.75% across the four sequential tasks. Performance on prior tasks remained unchanged after training on subsequent tasks, demonstrating near-zero catastrophic forgetting and validating the knowledge isolation provided by the frozen, additive adapters.

Table 5 presents a quantitative analysis of SLoRA's efficiency. By updating only 2.8 million parameters per task (~0.3% of the model), SLoRA achieves a 15x reduction in training time. This efficiency also translates to an estimated 93% reduction in CO_2e emissions per update. The modular architecture inherently supports unlearning, a capability computationally impractical for monolithically fine-tuned models.

4.6 Hyperparameter Tuning Insights

Targeted ablation experiments on 5-task Permuted-MNIST (single seed: 43) provided insights into SLoRA hyperparameters. Investigating stack rank (r_{stack}) with fixed base rank ($r_{base} = 8, \alpha_{base} = 16$) revealed a clear parameter efficiency vs. performance trade-off. Decreasing r_{stack} from 8 to 1 linearly reduced parameters but led to moderate-to-significant drops in final average accuracy (0.9556 down to 0.7681). Crucially, regardless of r_{stack}, performance on Task 1 after training later tasks remained consistently high (∼0.9520), demonstrating that stack rank variation did not cause forgetting of isolated knowledge. This supports the robustness of SLoRA's freezing mechanism. Varying stack α_{stack} (8, 16, 32 with $r_{stack} = 8$) resulted in only marginal changes in final average accuracy (∼0.955). An ablation without a base adapter (`SLoRA_NoBase`) showed performance (0.9554 final average accuracy) very close to the configuration with a base adapter (0.9556), suggesting stacks build effectively on the frozen W_0 even without a dedicated base LoRA.

Table 4: SLoRA Performance on Llama-3.2-1B across 4 Sequential NLP Tasks. Accuracy on each task was measured after all four tasks were trained.

Task Evaluated	**Final Acc. (%)**
Task 1 (SST-2)	94.30
Task 2 (TREC)	85.11
Task 3 (Yelp Pol.)	96.20
Task 4 (Amazon Pol.)	95.40
Final Avg. Accuracy	**92.75**

Table 5: Computational Efficiency and Architectural Comparison on Llama-3.2-1B.

Metric	Full Fine-Tuning (FT)	SLoRA (Proposed Method)
Performance Degradation (Forgetting)	Severe (Observed on BERT-base)	**Negligible (Near-zero forgetting)**
Trainable Parameters / Update	~1 Billion	**2.8 Million (99.7% fewer)**
Training Time / Update (est.)	~60 minutes	**~4 minutes (15x Speed-up)**
Estimated CO_2e / Update (kg)[a]	~0.163 kg	**~0.011 kg (93% Reduction)**
Architectural Property: Unlearning	Impractical (Requires full retraining)	**Inherent (Deactivate adapter)**

[a]CO_2e emissions estimated for a single task update on an NVIDIA RTX A5000 GPU (230W TDP), using India's average grid intensity of 0.708 kg CO_2e/kWh. FT time is an estimate based on observed speed-up.

5 Discussion

The experimental results demonstrate SLoRA's effectiveness in mitigating catastrophic forgetting, with the findings on Llama-3.2-1B (Section 4.5) providing strong evidence of its scalability. While methods with shared parameters (FT, LoRA-Cont) show significant performance degradation on prior tasks, SLoRA's design of freezing and additively composing adapters ensures that previously acquired knowledge is preserved. This is a direct consequence of parameter isolation, where each task-specific adaptation is encapsulated within a distinct, immutable module.

The analysis in Table 5 highlights a crucial trade-off in continual learning: the balance between performance, parameter count, and computational cost. SLoRA offers a compelling solution by drastically reducing the number of trainable parameters per task update (99.7% fewer than FT for Llama-3.2-1B). This leads to substantial improvements in training speed (Table 6) and energy efficiency, making sequential model updates feasible. While SLoRA's total parameter count grows linearly, the storage overhead for each adapter is minimal compared to storing separate model checkpoints.

We then consider parameter efficiency, a key factor for scalability. PEFT methods, including LoRA-Cont, LoRA-Ind, and SLoRA, require substantially fewer trainable parameters per task step than FT or EWC. While LoRA-Cont has minimal storage, it suffers severe forgetting (Table 2). Both LoRA-Ind and SLoRA scale unique parameter storage linearly with tasks. SLoRA requires fewer parameters than LoRA-Ind in practice, thanks to its use of smaller stack ranks per task, while still achieving comparable or better forgetting mitigation (Table 3). This demonstrates a favorable parameter-performance trade-off.

Architecturally, SLoRA's design offers additional benefits beyond performance and efficiency for knowledge management. A significant advantage is the inherent support for explicit task unlearning (knowledge modification): removing a task's frozen stack from the additive summation during inference effectively unlearns the corresponding knowledge, with no need for retraining. This capability positions SLoRA as a direct and practical approach to the problem of knowledge editing in foundation models, allowing for the targeted removal of outdated or incorrect information. Selective activation also allows for tailored inference by summing relevant stacks (enabling flexible knowledge retrieval).

Hyperparameter tuning experiments (Section 4.6) confirmed that once the core parameter isolation and additive composition are correctly implemented, SLoRA's forgetting mitigation property is robust to variations in stack size and scaling.

In summary, SLoRA provides a compelling parameter-efficient continual learning (knowledge updating) approach for the task-incremental setting. It effectively prevents catastrophic forgetting (knowledge loss) through the additive composition of task-specific, frozen low-rank adapters (representing knowledge adaptations), while also offering architectural simplicity, flexible parameter control (for knowledge representations), and native support for modular knowledge management, including task unlearning (knowledge modification) and selective inference (knowledge retrieval).

6 Conclusion

This work introduced Stacked Low-Rank Adaptation (SLoRA), a parameter-efficient method that addresses catastrophic forgetting in continual learning through the additive composition of task-specific, frozen low-rank adapters. Empirical evaluations on vision benchmarks, BERT-base, and the 1-billion-parameter Llama-3.2-1B demonstrated SLoRA's ability to nearly eliminate forgetting while offering significant computational advantages, including up

to a 15x training speed-up compared to full fine-tuning. Its modular architecture inherently supports critical functionalities like explicit task unlearning by deactivating adapters.

While SLoRA presents a robust solution for the task-incremental setting, key limitations include the linear growth of parameters with tasks and the reliance on task identity at inference. Future work should focus on mitigating parameter growth through techniques like adapter pruning or merging. A primary research direction is the development of task-agnostic inference mechanisms. This could involve implementing a dynamic routing module, potentially using learned steering vectors, to automatically select and combine the appropriate adapter stacks based on the input's semantic content. Such advancements would move towards creating truly autonomous and efficient lifelong learning systems. SLoRA provides a strong foundation for building adaptable, scalable, and manageable foundation models.

7 Limitations and Future Work

While SLoRA demonstrates significant advantages, we identify several limitations that present avenues for future research:

- **Linear Parameter Growth:** The total number of parameters scales linearly with the number of tasks. Although the adapters are parameter-efficient, this growth could become a storage bottleneck in scenarios involving an extremely large sequence of tasks.
- **Inference Latency Overhead:** Unlike standard LoRA adapters that can be merged into the base model to eliminate latency, SLoRA's parallel structure requires real-time summation of adapter outputs. This introduces a minor computational overhead during the forward pass that scales with the number of active adapters.
- **Reliance on Task Identity:** The current inference strategy requires explicit task identity to activate the corresponding adapter stack. This assumption limits its direct application in task-agnostic or online continual learning settings. This reliance, however, is a deliberate design trade-off that enables SLoRA's strict parameter isolation and its unique capability for explicit knowledge unlearning, which is not possible in task-agnostic methods that merge or blend parameters.
- **Scope of Evaluation:** Our experiments were conducted on task-incremental benchmarks. The method's generalization to more challenging CL paradigms, such as class-incremental or domain-incremental learning, remains to be validated.
- **Experimental Rigor on LLMs:** Due to computational constraints, the results for larger models (BERT-base, Llama-3.2-1B) are based on single-seed runs. Multi-seed experiments are necessary to fully establish the statistical significance and robustness of SLoRA's performance at scale.
- **Hyperparameter Sensitivity:** While the core mechanism is robust, the optimal rank (r) and scaling factor (α) for base and stack adapters may vary across different models and task types. This work does not establish a comprehensive guideline for hyperparameter selection.

References

Eric Nuertey Coleman, Luigi Quarantiello, Ziyue Liu, Qinwen Yang, Samrat Mukherjee, Julio Hurtado, and Vincenzo Lomonaco. 2025. Parameter-efficient continual fine-tuning: A survey. *arXiv preprint arXiv:2504.13822*.

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

Junxian He, Chunting Zhou, Xuezhe Ma, Taylor Berg-Kirkpatrick, and Graham Neubig. 2021. Towards a unified view of parameter-efficient transfer learning. *arXiv preprint arXiv:2110.04366*.

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. 2019. Parameter-efficient transfer learning for nlp. In *International conference on machine learning*, pages 2790–2799. PMLR.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, and 1 others. 2022. Lora: Low-rank adaptation of large language models. *ICLR*, 1(2):3.

Chengsong Huang, Qian Liu, Bill Yuchen Lin, Tianyu Pang, Chao Du, and Min Lin. 2023. Lorahub: Efficient cross-task generalization via dynamic lora composition. *arXiv preprint arXiv:2307.13269*.

James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, and 1 others. 2017. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 114(13):3521–3526.

Brian Lester, Rami Al-Rfou, and Noah Constant. 2021. The power of scale for parameter-efficient prompt tuning. *arXiv preprint arXiv:2104.08691*.

Xiang Lisa Li and Percy Liang. 2021. Prefix-tuning: Optimizing continuous prompts for generation. *arXiv preprint arXiv:2101.00190*.

Vladislav Lialin, Namrata Shivagunde, Sherin Muckatira, and Anna Rumshisky. 2023. Relora: High-rank training through low-rank updates. *arXiv preprint arXiv:2307.05695*.

Yan-Shuo Liang and Wu-Jun Li. 2024. Inflora: Interference-free low-rank adaptation for continual learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 23638–23647.

Vincenzo Lomonaco, Lorenzo Pellegrini, Andrea Cossu, Antonio Carta, Gabriele Graffieti, Tyler L Hayes, Matthias De Lange, Marc Masana, Jary Pomponi, Gido M Van de Ven, and 1 others. 2021. Avalanche: an end-to-end library for continual learning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 3600–3610.

David Patterson, Joseph Gonzalez, Quoc Le, Chen Liang, Lluis-Miquel Munguia, Daniel Rothchild, David So, Maud Texier, and Jeff Dean. 2021. Carbon emissions and large neural network training. *arXiv preprint arXiv:2104.10350*.

Jonas Pfeiffer, Aishwarya Kamath, Andreas Rücklé, Kyunghyun Cho, and Iryna Gurevych. 2020. Adapterfusion: Non-destructive task composition for transfer learning. *arXiv preprint arXiv:2005.00247*.

Xiao Wang, Tianze Chen, Qiming Ge, Han Xia, Rong Bao, Rui Zheng, Qi Zhang, Tao Gui, and Xuanjing Huang. 2023. Orthogonal subspace learning for language model continual learning. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 10658–10671, Singapore. Association for Computational Linguistics.

Yichen Wu, Hongming Piao, Long-Kai Huang, Renzhen Wang, Wanhua Li, Hanspeter Pfister, Deyu Meng, Kede Ma, and Ying Wei. 2025. Sd-lora: Scalable decoupled low-rank adaptation for class incremental learning. *arXiv preprint arXiv:2501.13198*.

Min Zeng, Haiqin Yang, Wei Xue, Qifeng Liu, and Yike Guo. 2024. Dirichlet continual learning: Tackling catastrophic forgetting in NLP. In *The 40th Conference on Uncertainty in Artificial Intelligence*.

Jinghan Zhang, Junteng Liu, Junxian He, and 1 others. 2023. Composing parameter-efficient modules with arithmetic operation. *Advances in Neural Information Processing Systems*, 36:12589–12610.

Xin Zhang, Liang Bai, Xian Yang, and Jiye Liang. 2025. C-lora: Continual low-rank adaptation for pre-trained models. *arXiv preprint arXiv:2502.17920*.

Appendix

This appendix contains supplementary materials and additional details not included in the main body of the paper due to space constraints.

A Additional Experimental Details

This section provides additional details regarding the experimental setup and base model architectures used in this study. Detailed hyperparameters for each method and benchmark are provided within Section 3.2 in the main body of the paper.

A.1 Base Model Architectures

The specific base model architectures used for each benchmark are detailed below:

- **Permuted-MNIST:** A simple two-layer MLP was used as the base network. It consisted of a linear layer mapping the flattened 28×28 input (784 features) to 256 hidden units, followed by a ReLU activation. A second linear layer mapped the 256 hidden units to 10 output units (one for each digit class).

- **Split-CIFAR100:** A SimpleCNN architecture was employed. It included three convolutional layers for feature extraction, each with 3x3 kernels, ReLU activation, and followed by 2x2 max pooling. The classifier head, where PEFT methods were applied, contained two linear layers: the first mapping the flattened output of the convolutional layers to 512 hidden units (with ReLU), and the second mapping 512 units to 100 output units (for CIFAR-100 classes). The convolutional layers were kept frozen.

- **BERT-base-uncased:** The standard frozen `bert-base-uncased` model from the Hugging Face Transformers library was used as the base for NLP tasks. PEFT methods were applied to the linear layers within the attention and feed-forward networks of the Transformer blocks.

B Training Procedure Pseudocode

Algorithm 1: SLoRA Sequential Training Procedure

Input: Pre-trained model with `SLoRALinear` layers M, Task sequence $\mathcal{T} = \{Task_0, Task_1, \ldots, Task_{T-1}\}$, hyperparameters $r_{base}, \alpha_{base}, r_{stack}, \alpha_{stack}$

Output: Trained SLoRA model with task-specific adapters

1. **State:** Freeze W_0 in all `SLoRALinear` layers of M.
2. **Train Base Task ($Task_0$):**
3. **For** each `SLoRALinear` layer L in M **do**
4. **State:** Initialize Base LoRA adapter (A_{base}, B_{base}) in L with r_{base}, α_{base}.
5. **State:** Set A_{base}, B_{base} in L to be trainable.
6. **State:** Freeze all other adapters in L (initially none).
7. **End For**
8. **State:** Configure optimizer to train trainable parameters in M and $Task_0$ classifier.
9. **State:** Train M on $Task_0$ data.
10. **For** each `SLoRALinear` layer L in M **do**
11. **State:** Freeze (A_{base}, B_{base}) in L.
12. **End For**
13. **Train Subsequent Tasks ($Task_t$ for $t = 1, \ldots, T-1$):**
14. **For** each t from 1 to $T-1$ **do**
15. **For** each `SLoRALinear` layer L in M **do**
16. **State:** Initialize a *new* Stack adapter $(A_{stack,t}, B_{stack,t})$ in L with $r_{stack}, \alpha_{stack}$.
17. **State:** Set $(A_{stack,t}, B_{stack,t})$ in L to be trainable.
18. **State:** Ensure W_0, Base LoRA, and all previously added Stacks ($< t$) in L are frozen.
19. **End For**
20. **State:** Configure optimizer to train trainable parameters in M and $Task_t$ classifier.
21. **State:** Train M on $Task_t$ data.
22. **For** each `SLoRALinear` layer L in M **do**
23. **State:** Freeze $(A_{stack,t}, B_{stack,t})$ in L.
24. **End For**
25. **End For**

C Additional Result Tables

Table 6: Training Time on BERT-base (batch size 16, single NVIDIA P100, averaged over 3 runs (in minutes)

Method	Mean	Std Dev	Speed-up
Full Fine-Tuning	12.0	0.3	1.0×
SLoRA	**1.5**	**0.1**	**8.0**×

Table 7: Accuracy (%) on Permuted-MNIST Tasks after Training on Task 5 (Mean ± SD over 3 Seeds)

Method	Task 1	Task 2	Task 3	Task 4	Task 5	Avg. Accuracy
FT	51.99 ± 6.73	85.78 ± 2.80	88.96 ± 3.76	92.06 ± 1.42	**97.26 ± 0.17**	83.21 ± 2.49
EWC	54.14 ± 0.90	**86.65 ± 1.85**	87.01 ± 4.54	**95.05 ± 0.46**	97.22 ± 0.02	83.87 ± 1.31
LoRA-Cont	50.42 ± 3.86	79.44 ± 4.84	86.26 ± 2.57	93.29 ± 0.77	96.78 ± 0.31	81.29 ± 1.48
SLoRA	**56.97 ± 7.82**	84.11 ± 1.74	**87.61 ± 0.71**	91.70 ± 1.35	97.08 ± 0.42	**83.49 ± 1.35**

Table 8: Split-CIFAR100 Mean Accuracy (%) on Initial vs. After Final Task (Averaged over 3 seeds, $E = 50$)

Method	Task Accuracy: Initial → After Final										Final Avg. Acc.
	T0	T1	T2	T3	T4	T5	T6	T7	T8	T9	
FT	80→10	82→12	81→11	72→10	80→10	76→10	78→10	78→10	85→10	80→80	8.0
EWC	79→15	80→15	75→15	67→15	74→15	70→15	71→15	71→15	78→15	72→72	7.2
LoRA-Cont	61→12	65→12	59→12	55→12	62→12	56→12	59→12	57→12	69→12	60→60	6.0
LoRA-Ind	62→60	65→63	59→58	57→55	61→60	56→55	57→56	56→55	68→67	59→59	59.6
SLoRA	**64→64**	**67→67**	**61→61**	**57→57**	**63→63**	**57→57**	**59→59**	**59→59**	**68→68**	61→61	**61.8**