COLA: Compressed Latent Adapter Memory for Lifelong Learning

Jaya Krishna Mandivarapu [1]

Abstract

Continual learning (CL) for large language models remains challenging due to catastrophic forgetting and the prohibitive cost of retraining large models as new tasks arrive. Existing approaches either rely on rehearsal buffers, store task-specific parameters that grow linearly with tasks, or restrict model plasticity. We propose COLA (Compressed Latent Adapter Memory), a rehearsal-free continual learning framework that stores task knowledge in a compact latent space rather than explicit model parameters. COLA trains lightweight LoRA adapters for each task and compresses the adapter weights using a contractive autoencoder into low-dimensional latent codes. For task-agnostic inference, COLA retrieves the most relevant latent adapter code using input-conditioned task keys and loads the reconstructed adapter into the frozen backbone. Across intent classification and task-oriented dialogue benchmarks, COLA outperforms strong rehearsal-free baselines, approaches joint multi-task training performance, and substantially reduces per-task storage. These results show that latent adapter memory is an effective mechanism for rehearsal-free continual learning in language models.

1. Introduction

Lifelong or continual learning (CL) is one of the most challenging problems in the field of AI, posing a significant hurdle in the quest for artificial general intelligence (AGI) (Goodfellow et al., 2013; Kemker et al., 2018). In this paradigm, a single system must continually learn new tasks without forgetting previously learned tasks, while also using accumulated knowledge to improve learning on future tasks. The primary challenges in continual learning are: (1) *preventing catastrophic forgetting*—where previously acquired knowledge is lost after learning new tasks (McCloskey & Cohen, 1989; Ratcliff, 1990) and (2) *enabling forward transfer*, where prior knowledge accelerates learning on new tasks; (3) *promoting backward transfer*, in which mastering new tasks enhances performance on previously learned tasks.

Large language models (LLMs) have demonstrated strong capabilities across a wide range of knowledge-intensive language tasks, including open-domain question answering, slot filling, and dialogue systems. These capabilities are largely stored in model parameters learned during pretraining and task-specific fine-tuning. However, adapting LLMs continuously to new tasks remains difficult: full-model retraining is computationally expensive, while sequential fine-tuning can overwrite previously learned knowledge and lead to catastrophic forgetting.

Task-oriented dialogue systems provide a practical example of this challenge. Such systems power modern assistants and customer-support agents, and are often expected to support new domains, intents, APIs, and user workflows over time. Traditional task-oriented dialogue systems follow a modular pipeline consisting of Natural Language Understanding (NLU), Dialogue State Tracking (DST), Dialogue Policy (DP), and Natural Language Generation (NLG). Although training these modules separately can yield strong task-specific performance, recent work has increasingly adopted end-to-end LLM-based dialogue models. These monolithic models require substantial annotated data and share parameters across tasks, making continual adaptation to new domains costly and susceptible to forgetting.

Existing continual learning methods typically rely on rehearsal, regularization, or model expansion. Rehearsal-based methods store and replay previous examples, which may be impractical due to privacy or storage constraints. Regularization methods reduce forgetting by constraining parameter updates, but can limit plasticity. Expansion-based methods avoid interference by allocating task-specific parameters, but their storage grows with the number of tasks.

We propose **COLA** (**Compressed Latent Adapter Memory**), a rehearsal-free continual learning framework for language models. COLA trains lightweight LoRA adapters while keeping the backbone frozen, compresses each adapter into a compact latent code using a contractive autoencoder,

[1]Microsoft. Correspondence to: Jaya Krishna Mandivarapu <jmandivarapu@microsoft.com>.

Proceedings of the 43rd International Conference on Machine Learning, Seoul, South Korea. PMLR 306, 2026. Copyright 2026 by the author(s).

and reconstructs adapters on demand. For task-agnostic inference, COLA retrieves the most relevant latent code using input-conditioned task keys, avoiding stored examples and expensive scoring over all adapters. Across continual text-classification and task-oriented dialogue benchmarks, COLA outperforms strong rehearsal-free baselines, approaches joint multi-task training performance, and substantially reduces per-task storage compared with storing full adapters.

2. Methodology

2.1. Problem Formulation

We consider a sequence of tasks $\mathcal{T} = \{T_1, T_2, \ldots, T_K\}$. Each task T_i is associated with a dataset $D_i = \{(x_j, y_j)\}_{j=1}^{n_i}$. The learner observes tasks sequentially. After completing training on task T_i, the dataset D_i is no longer accessible. Multiple passes over $\mathcal{D}_i$ are permitted while it is resident in memory, but no element of $\mathcal{D}_i$ may be retained when the learner moves on to $\mathcal{D}_{i+1}$. During evaluation, the task identity is unknown, and the model must select or infer the appropriate task-specific knowledge without using stored examples. The goal is to maximize final average performance:

$$A_K = \frac{1}{K} \sum_{i=1}^{K} a_{i,K} \tag{1}$$

where $a_{i,K}$ denotes performance on task T_i after learning all K tasks. COLA addresses this objective by storing each task as a compact latent representation of its adapter weights rather than storing full adapters or rehearsal data.

We focus on a task-incremental setting with task-aware training but task-agnostic evaluation: (1) tasks arrive sequentially; (2) only the current task data is available during training; (3) task identity is available during training but unavailable at evaluation; and (4) no previous task examples may be stored or replayed.

In addressing this constrained scenario, prior approaches typically fall into rehearsal, regularization, and expansion-based methods. *Regularization* approaches aim for a *single* shared parameter set that is nudged to remain useful for old tasks, and *expansion-based* methods allocate a fresh parameter to each new task either by freeing old parameters or by adding a completely new set of parameters. The former methods offer storage efficiency at the cost of significant performance degradation over time, whereas the latter approaches achieve higher performance but suffer from linearly growing parameter storage requirements. **Our contributions.** We introduce **COLA**, a rehearsal-free framework that: a) Attaches lightweight LoRA adapters to a frozen backbone, isolating task-specific knowledge at negligible cost. b) Compresses adapters into compact latent codes via a contractive autoencoder (CAE), resulting in compact per-task memory growth. c) Reconstructs and reuses adapters on demand, enabling modular inference and warm-start transfer across tasks.

2.2. LoRA Adapters for Parameter-Efficient Continual Learning

Large language models (LLMs) now contain billions of parameters, making full fine-tuning prohibitive in both memory and compute. Fine-tuning a large language model (LLM) which contains billions of parameters as a backbone for every new task is unfeasible for continual learning scenarios. Instead, we employ **LoRA** (low-rank adaptation) adapter as our task-specific module of choice. LoRA posits that the *update* required to steer a pre-trained weight matrix toward a new task occupies a low-dimensional subspace. **LoRA**—Low-Rank Adaptation—addresses this by extending the network by injecting *small, trainable rank-decomposition matrices* into frozen projection layers of the Transformer (Hu et al., 2021). When compared to traditional complete model fine-tuning, LoRA reduces the overall trainable parameters by up to four orders of magnitude and cuts GPU memory by a factor of three while achieving similar task accuracy on GPT-2 (Radford et al., 2019), RoBERTa (Liu et al., 2019), and GPT-3 (Brown et al., 2020) benchmarks.

Low-rank update: Let's consider $\mathbf{W}_0, \mathbf{W}_1, \mathbf{W}_2 ... \mathbf{W}_n$ be the weights of each layer of n-layer LLM model. For a given layer $\mathbf{W}_0 \in \mathbb{R}^{d\times k}$ be a frozen projection matrix (e.g. the query or value matrix inside self-attention). Rather than learning a full update $\Delta\mathbf{W} \in \mathbb{R}^{d\times k}$, LoRA constrains it to a rank-r factorisation:

$$\mathbf{W} = \mathbf{W}_0 + \Delta\mathbf{W} \quad \text{with} \quad \Delta\mathbf{W} = \mathbf{B}\,\mathbf{A}, \tag{2}$$

where $\mathbf{B} \in \mathbb{R}^{d\times r}$ and $\mathbf{A} \in \mathbb{R}^{r\times k}$, and the rank satisfies $r \ll \min(d, k)$. During training, *only* $\mathbf{A}$ and $\mathbf{B}$ are updated, while $\mathbf{W}_0$ remains fixed. **Forward pass:** Given a hidden vector $\mathbf{x}$, a LoRA-augmented linear layer computes

$$\mathbf{h} = \mathbf{W}_0\,\mathbf{x} + \mathbf{B}\big(\mathbf{A}\mathbf{x}\big), \tag{3}$$

adding at most $r(d+k)$ trainable parameters— typically $<2\%$ of the layer's original size.

Why LoRA suits continual learning.

(a) *Tiny footprint.* Typically using small rank values achieves the task accuracy required on par with the full fine-tuning of the model. This helps in keeping the memory, compute overhead negligible. Specifically, $<2\%$ additional parameters per task, to train dozens of tasks on a single GPU.

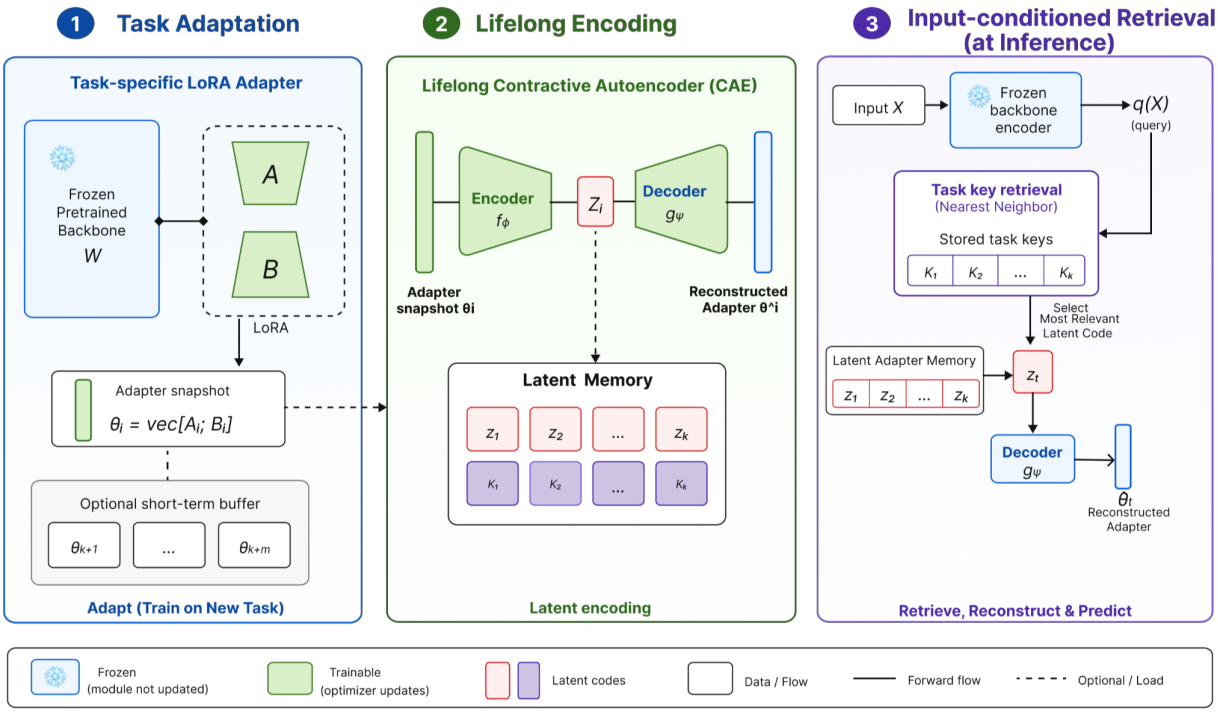

Figure 1. Overview of COLA. (1) For each incoming task, COLA trains a task-specific LoRA adapter while keeping the backbone frozen and vectorizes the learned adapter as an adapter snapshot. (2) A lifelong contractive autoencoder compresses the adapter snapshot into a compact latent code, which is stored together with a task key in latent adapter memory. (3) At inference time, when task identity is unavailable, the input is mapped to a query representation, the nearest task key is retrieved, and the corresponding latent code is decoded to reconstruct the adapter for prediction.

(b) *Task Modularity.* An independent LoRA adapter is allocated per task, preventing interference without touching the backbone. This leads to acquisition of new knowledge without interference with existing knowledge. Recent work shows task-specific LoRA weights can be merged, hot-swapped, re-used at inference time for different use cases.

(c) *Compatibility with compression.* LoRA architecture perfectly suits the proposed approach of injecting the adapter into the latent space of the autoencoder. After training on a task we encode the pair $\{\mathbf{A}, \mathbf{B}\}$ with our conditional auto-encoder, store only the latent code, and discard the full low-rank weights. Thus LoRA serves as a lightweight *front-end* for task adaptation, while CAE provides long-term storage.

2.3. Framework Overview

Figure 1 sketches our **COLA framework** for lifelong learning without catastrophic forgetting. As shown in the fig each new task is handled in three stages—*adapt*, *compress*, and *replay on demand*(during inference)—all while keeping the backbone frozen and storing *no raw data* from previous tasks.

Stage 1: Task adaptation. When a new task T_i arrives with dataset $D_i = \{(X^{(j)}, y^{(j)})\}_{j=1}^{n_i}$, we attach a rank-$r$ LoRA adapter to the frozen backbone and train only the adapter parameters on D_i. After adaptation, the learned low-rank matrices $(\mathbf{A}_i, \mathbf{B}_i)$ are vectorized into an adapter snapshot $\boldsymbol{\theta}_i = \mathrm{vec}[\mathbf{A}_i; \mathbf{B}_i]$.

Stage 2: Lifelong encoding. After training the LoRA adapter for task T_i, we compress its adapter snapshot $\boldsymbol{\theta}_i = \mathrm{vec}[\mathbf{A}_i; \mathbf{B}_i]$ into a compact latent representation using a Contractive Autoencoder (CAE). Let f_ϕ denote the encoder and g_ψ denote the decoder. The encoder maps the adapter snapshot to a latent code $\mathbf{z}_i \in \mathbb{R}^m$, and the decoder reconstructs the adapter weights:

$$\mathbf{z}_i = f_\phi(\boldsymbol{\theta}_i), \quad \hat{\boldsymbol{\theta}}_i = g_\psi(\mathbf{z}_i), \tag{4}$$

where $m \ll |\boldsymbol{\theta}_i|$.

The CAE is optimized with a reconstruction loss and a contractive regularization term, encouraging a smooth latent manifold over task-specific LoRA updates. This allows COLA to represent each task adapter using a compact latent code instead of storing the full adapter parameter vector. After encoding task T_i, COLA stores the latent adapter code $\mathbf{z}_i$ and discards the full adapter snapshot $\boldsymbol{\theta}_i$. The adapter can later be reconstructed on demand by decoding $\mathbf{z}_i$ with the shared decoder g_ψ.

Importantly, the encoder f_ϕ is used only during training or update time to encode newly trained adapter snapshots. It is not required during inference. Thus, deployment does not need to store the encoder; it only requires the shared decoder g_ψ, the latent adapter codes $\{\mathbf{z}_i\}_{i=1}^K$, and the task retrieval keys described in Stage 3. This keeps inference-time memory compact while still allowing COLA to reconstruct task-specific adapters when needed.

For future tasks, a newly trained adapter snapshot can be encoded either by temporarily using the encoder f_ϕ during the update phase or by optimizing a new latent code for a few epochs while keeping the decoder g_ψ fixed. After this update, only the resulting latent code is retained and the full adapter snapshot is discarded. In practice, adapter snapshots may be encoded immediately after task training or temporarily accumulated in a small buffer and batch-encoded before the raw weight tensors are purged.

Stage 3: Input-conditioned adapter retrieval at inference. In a *hard* continual-learning setup, task identity is not given at test time. If the task identity is known, we reconstruct the corresponding adapter from its latent code $\mathbf{z}_i$ via $g_\psi(\mathbf{z}_i)$ and run inference directly. When task identity is unknown, COLA performs input-conditioned retrieval over the latent adapter memory.

During training on task T_i, while the current task data D_i is still available, we compute a compact task key $\mathbf{k}_i$ using the frozen backbone encoder:

$$\mathbf{k}_i = \text{normalize}\left(\frac{1}{|D_i|}\sum_{(X,Y)\in D_i} h_\Theta(X)\right) \tag{5}$$

where $h_\Theta(X)$ denotes a pooled representation of input X from the frozen backbone. After training task T_i, COLA stores only the task key $\mathbf{k}_i$, the compressed latent adapter code $\mathbf{z}_i$, and the shared decoder g_ψ; no training examples from D_i are retained.

At inference, for a test input X, we compute its query representation

$$\mathbf{q}(X) = \text{normalize}(h_\Theta(X)).$$

We then retrieve the most relevant adapter by nearest-neighbor search in the task key space:

$$\hat{t} = \arg\max_{t\in\{1,\dots,K\}} \text{sim}\big(\mathbf{q}(X), \mathbf{k}_t\big),$$

where $\text{sim}(\cdot,\cdot)$ denotes cosine similarity. The selected latent code $\mathbf{z}_{\hat{t}}$ is decoded into adapter weights

$$\hat{\boldsymbol{\theta}}_{\hat{t}} = g_\psi(\mathbf{z}_{\hat{t}}),$$

which are then loaded into the frozen backbone for prediction.

This retrieval mechanism does not require task labels, stored examples, or gold output sequences at test time. Unlike perplexity-based routing over all adapters, which requires K model scoring passes, input-conditioned retrieval uses a lightweight similarity search over task keys and decodes only the selected adapter. In large task memories, the retrieval step can be implemented using approximate nearest-neighbor search, reducing the routing overhead while keeping the adapter memory compact.

Optional warm-start via latent adapter reuse. When a new task T_i arrives, COLA can reuse knowledge from previously learned tasks before training the new adapter. We first compute a task-level query representation from the current task's available training inputs using Eq 5. We then retrieve the most similar previously learned task:

$$\hat{t} = \arg\max_{t<i} \text{sim}\big(\mathbf{q}_i, \mathbf{k}_t\big).$$

The corresponding latent code $\mathbf{z}_{\hat{t}}$ is decoded to reconstruct the previous adapter weights using the decoder in Eq 4. We initialize the new task's LoRA matrices $(\mathbf{A}_i, \mathbf{B}_i)$ from $\hat{\boldsymbol{\theta}}_{\hat{t}}$ rather than from random initialization. This provides a task-relevant warm start, improving convergence when the new task is related to previously learned tasks. If the maximum similarity score is below a predefined threshold, COLA falls back to random LoRA initialization to avoid negative transfer.

2.4. Contractive Autoencoder Objective

COLA is not tied to a specific autoencoder architecture, but in practice we use a contractive autoencoder (CAE) because accurate reconstruction of adapter weights is critical for preserving downstream task performance. Given an adapter snapshot $\boldsymbol{\theta}_i$ and its reconstruction $\hat{\boldsymbol{\theta}}_i = g_\psi(f_\phi(\boldsymbol{\theta}_i))$, we optimize:

$$\mathcal{L}_{\text{CAE}}(\boldsymbol{\theta}_i) = \Big(1 - \cos(\boldsymbol{\theta}_i, \hat{\boldsymbol{\theta}}_i)\Big) + \lambda \left\|J_{f_\phi}(\boldsymbol{\theta}_i)\right\|_F^2, \tag{6}$$

where the first term encourages high-fidelity reconstruction of the original adapter weights and the second term penalizes sensitivity of the encoder representation to small perturbations in the input. The Jacobian penalty is:

$$\left\|J_{f_\phi}(\boldsymbol{\theta}_i)\right\|_F^2 = \sum_{a,b}\left(\frac{\partial h_b(\boldsymbol{\theta}_i)}{\partial \theta_{i,a}}\right)^2, \tag{7}$$

where $h_b(\boldsymbol{\theta}_i)$ denotes the b-th hidden activation of the encoder. This regularization encourages nearby adapter snapshots to map to nearby latent codes, yielding a smoother latent adapter memory. Unless otherwise specified, we set $\lambda = 10^{-4}$ in our experiments.

3. Experiments

Table 1 presents a comprehensive evaluation of COLA on two continual learning (CL) benchmarks using the T5-large model. We follow the standard CL evaluation protocol established in prior work (Wang et al., 2023a), where tasks are learned sequentially and performance is measured as the final accuracy on each task after training on the full task sequence.

Task Orders. As shown in Table 1 to analyze sensitivity to task arrival, we evaluate multiple task orderings as shown by other baselines like O-LoRA (Wang et al., 2023a). Orders 1–3 correspond to the standard 4-task CL benchmark consisting of DBpedia (entity typing), Amazon (product reviews), Yahoo Answers (question topic classification), and AG News (news categorization). These orders represent permutations of the same tasks, allowing assessment of robustness to task sequence variation. Orders 4–6 follow the large-scale CL benchmark introduced by (Razdaibiedina et al., 2023), extending evaluation to 15 tasks spanning diverse natural language understanding domains, including MNLI, CB, WiC, COPA, QQP, BoolQA, RTE, IMDB, Yelp, Amazon, SST-2, DBpedia, AG News, MultiRC, and Yahoo. These longer sequences provide a more challenging setting for knowledge retention and interference mitigation. This task ordering does not adversely affect COLA, as the method is designed to be domain- and order-agnostic. Consequently, we observe consistent accuracies across different task sequences. We additionally report results for MULTI, which denotes joint multi-task training with simultaneous access to all task data. MULTI serves as an approximate upper bound representing an oracle setting without continual learning constraints.

To validate our approach, we evaluate COLA across multiple tasks, including intent classification, dialogue state tracking (DST), natural language generation (NLG), and end-to-end response generation (E2E). Experiments are conducted on four benchmarks—TaskMaster-2019 (TM19) (Byrne et al., 2019), TaskMaster-2020 (TM20) (Byrne et al., 2019), Schema-Guided Dialogue (SGD) (Rastogi et al., 2020), and MultiWOZ (Zang et al., 2020; Eric et al., 2019)—following the methodology in Section 2. We also include a combined NLU dataset that aggregates and relabels three large corpora: CLINC150 (Larson et al., 2019), TM20, and SGD, creating 37 unique tasks; results are shown in Table 2.

Architecture Details and Parameter Chunking We adopt T5-large (770M parameters) as the backbone model and apply LoRA (r=8, α=32, dropout=0.1) to the query and value projections across all transformer layers, resulting in 2,359,296 trainable parameters per task. After task-specific fine-tuning, the LoRA parameters are flattened into a single vector and partitioned into fixed-size chunks of 100,000 values, yielding 24 chunks per task (with the final chunk zero-padded). Each chunk is independently compressed using a shared Contractive Autoencoder (CAE) with architecture $100{,}000 \rightarrow d \rightarrow 100{,}000$, where d denotes the latent dimension (e.g., d=50). All layers use Tanh activations without bias terms. The contractive penalty encourages nearby latent representations for related task adapters, which can support warm-start transfer and future latent-space retrieval mechanisms. With d=50, each task adapter is compressed from 2,359,296 parameters to $24 \times 50 = 1{,}200$ latent codes, corresponding to an effective compression ratio of approximately 1966×. This results in roughly a 2,000× reduction in storage per task compared to storing full adapter parameters.

Metrics For regular tasks in Table 1 we report the accuracy and for *Intent Recognition* in Table 2: We measure the precision of intent prediction by comparing the predicted intent of the model with the gold standard label. For *Dialogue State Tracking (DST)* performance is measured using Joint Goal Accuracy (JGA), defined as the percentage of dialogue turns where all slot-value pairs match the gold state. For *Natural Language Generation (NLG)*: Quality is assessed with the BLEU metric (Papineni et al., 2002) along with the slot error rate (SER (Madotto et al., 2020)), calculated as the fraction of required slot values that are missing or incorrect in the generated response.

Baselines We compare our method across various existing approaches.

- **Adapter-Based**: Different LoRA-based methods, including SeqFT (de Masson D'Autume et al., 2019), O-LoRA (Wang et al., 2023a), Adapter-CL (Madotto et al., 2020), and Adapter-style (Madotto et al., 2020), insert lightweight adapter modules into the base model; each task trains its own adapter, enabling modular task-specific learning without overwriting shared model weights.
- **Replay (GEM) (Chaudhry et al., 2018b)**: fine-tunes the entire model while storing and replaying real samples from prior tasks via a memory buffer; uses gradient projection to avoid catastrophic forgetting.
- **EWC (Kirkpatrick et al., 2017)**: fine-tunes the model with an additional regularization term that penalizes changes to parameters important for previous tasks, based on the Fisher Information Matrix.
- **L2 Regularization (Hoerl & Kennard, 1970)**: constrains the model by penalizing deviation from previously learned weights using a simple L2 loss, encouraging retention of earlier knowledge.
- **Vanilla FT**: sequentially fine-tunes the entire model on each task without any regularization or memory, typically resulting in catastrophic forgetting. Other methods such as (Li & Hoiem, 2017) constrain the shared representation layer to remain close to its pre-task state during new task learning, while (Qin & Joty, 2021) continuously train a soft prompt that jointly learns to solve the tasks and gen-

erate synthetic training samples, which are subsequently used for experience replay.

3.1. Results

Continual Learning Performance. Table 1 summarizes results across six task orderings. On the standard 4-task benchmark (Orders 1–3), COLA achieves a stable average accuracy of 79.6%, outperforming all continual learning baselines. The strongest prior method, O-LoRA, reaches 75.8% on average, placing COLA ahead by +3.8 percentage points. Notably, COLA exhibits identical performance across all task permutations, whereas competing methods show measurable order sensitivity. This invariance suggests that COLA effectively mitigates task interference and forgetting. Compared with other representative approaches, COLA provides consistent gains over LFPT5 (+6.9), IncLoRA (+13.2), Replay (+21.8), and EWC (+29.3). Sequential fine-tuning baselines (SeqFT, SeqLoRA) perform poorly, confirming the severity of catastrophic forgetting in the absence of explicit CL mechanisms.

Scaling to Longer Task Sequences. Under the 15-task setting (Orders 4–6), COLA maintains strong performance with an average accuracy of 76.9%. While most baselines degrade substantially as the number of tasks increases, COLA preserves a small gap relative to the MULTI upper bound (77.5%). Across both benchmarks, MULTI—trained jointly on all task data—establishes an approximate upper bound of 80.3% (4-task) and 77.5% (15-task). COLA approaches this oracle setting closely despite operating under strict continual learning constraints, including sequential task arrival, rehearsal-free updates, and single-pass training. These results indicate that compressing adapters into a shared latent space can approximate joint training performance without requiring simultaneous access to all datasets.

Evaluation on a Distinct Downstream Benchmark. To evaluate generality beyond classification, we also test COLA on task-oriented dialogue benchmarks using Intent Accuracy, Joint Goal Accuracy (JGA), Slot Error Rate (EER), and BLEU. Unlike Table 1, these metrics evaluate structured prediction and generation rather than simple classification accuracy. As shown in Table 2, COLA achieves the strongest Intent Accuracy and JGA among comparable continual learning baselines while remaining competitive on EER and BLEU. This suggests that latent adapter compression preserves task-specific behavior across both classification and dialogue settings.

4. Ablation Studies

4.1. Impact of Reconstruction Quality on Performance

We evaluated the average task performance relative to changes in model parameters generated by the auto encoder, quantified using reconstruction score. As shown in Fig 3 results reveal a strong correlation: as reconstruction similarity score decreases, performance tends to decrease. The baseline accuracy is shown as a dotted red line, marks the point where the model performance closely matches that of the original. Therefore, we use a threshold of 0.99 as the stopping criterion in our following experiments, unless otherwise specified.

	Intent	DST		NLG
Method	**Acc↑**	**JGA↑**	**EER↓**	**BLEU↑**
VANILLA	5.12	8.91	50.76	7.48
L2	4.81	7.21	60.88	5.8
EWC	4.85	9.56	61.33	5.56
AGEM	39.14	12.73	67.99	4.64
LAMOL	12.59	8.45	71.12	3.2
REPLAY	82.38	30.32	**16.12**	**17.9**
COLA(Ours)	**89.87**	**34.86**	32.58	16.26
MULTI	93.45	47.3	11.46	22.51

Figure 2. E2E results in terms of Intent accuracy, Joint Goal Accuracy (JGA), Slot Error Rate (EER), and BLEU.

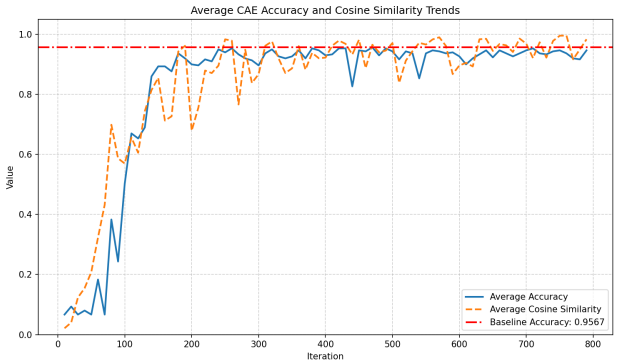

Figure 3. Accuracy vs. Reconstruction Score.

[1]Some results are reproduced by rerunning the official O-LoRA experiments: https://github.com/cmnfriend/O-LoRA.

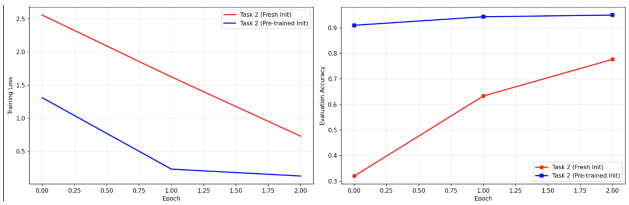

Figure 4. Compared to fresh initialization (red), initializing a new task from the best prior adapter (blue) yields faster convergence (lower training loss) and higher final accuracy

Table 1. Summary of results on two standard continual learning (CL) benchmarks using the T5-large model. Reported values denote the averaged accuracy after training on the final task. As COLA is task independent we indicate the same avg accuracy across Orders 1,2,3 and Orders 4,5,6 [1]

	Standard CL Benchmark				Large Number of Tasks			
Method	Order-1	Order-2	Order-3	Avg	Order-4	Order-5	Order-6	Avg
SeqFT	18.9	24.9	41.7	28.5	7.4	7.4	7.5	7.4
SeqLoRA	44.6	32.7	53.7	43.7	2.3	0.6	1.9	1.6
Replay	55.2	56.9	61.3	57.8	55.0	54.6	53.1	54.2
EWC	48.7	47.7	54.5	50.3	45.3	44.5	45.6	45.1
LwF	54.4	53.1	49.6	52.3	50.1	43.1	47.4	46.9
LFPT5	67.6	72.6	77.9	72.7	70.4	68.2	69.1	69.2
IncLoRA	66.0	64.9	68.3	66.4	63.3	58.5	61.7	61.2
O-LoRA	75.4	75.7	76.3	75.8	72.3	64.8	71.6	69.6
COLA(Ours)	**79.6**	**79.6**	**79.6**	**79.6**	**76.9**	**76.9**	**76.9**	**76.9**
MULTI	80.3	80.3	80.3	80.3	77.5	77.5	77.5	77.5

4.2. Continual Learning Transfer Validation

To validate the effectiveness of our continual learning approach, we conducted a controlled experiment examining knowledge transfer between sequential tasks using the CLINC-OOS intent classification dataset. The experiment consisted of three phases: (1) training a RoBERTa-base model with LoRA adapters on Task 1 (achieving 95.33% accuracy), (2) training a fresh model on Task 2 from scratch, and (3) initializing Task 2 training with the pre-trained Task 1 model weights. Our results demonstrate significant evidence for positive knowledge transfer in continual learning scenarios. The model pre-trained on Task 1 achieved 95.00% accuracy on Task 2, representing a substantial 17.33 percentage point improvement over the fresh model trained from scratch (77.67% accuracy). This improvement indicates that the learned representations from Task 1 contain transferable knowledge that accelerates learning and enhances performance on related downstream tasks. In addition to superior final performance, the transferred model demonstrated faster convergence, reaching high accuracy within the same training budget. These results indicate that representations learned from Task 1 capture transferable knowledge that improves learning efficiency on related tasks, supporting the hypothesis that parameter-efficient continual learning enables effective knowledge reuse in practical NLP settings.

4.3. Why Contractive Autoencoders? and Latent Dimension Ablation

To justify the choice of a Contractive Autoencoder (CAE) as the compression backbone, we compare against two alternative compression strategies: a Standard Autoencoder (without the Jacobian penalty) and a Variational Autoencoder (VAE) with reparameterization trick (β=0.001). As shown in Table 3, VAE catastrophically fails, producing 0% downstream accuracy. The KL divergence term, even with a small β, pushes latent representations toward a unit Gaussian prior, destroying the precise reconstruction fidelity required for adapter parameters where even minor perturbations can corrupt model behavior. While the Standard AE achieves comparable single-task accuracy to CAE, the contractive penalty in CAE regularizes the learned manifold, which becomes critical in the multi-task continual learning setting where latent codes from different task adapters must remain separable for accurate retrieval. We also evaluated PCA via truncated SVD; however, it did not yield effective compression, as it requires storing approximately the same number of values as the original adapter parameters ($\sim$2.36M), making it impractical for the retrieval-based continual learning framework where compact latent representations are essential. The latent dimension study in Table 2 shows that very small latent spaces (25) introduce performance degradation, while moderate sizes (50 to 100) fully recover baseline accuracy. Increasing the dimension further provides no benefit and slightly reduces performance. The compression method ablation demonstrates that CAE preserves accuracy with minimal loss, whereas a standard AE incurs noticeable degradation and VAE fails to reconstruct meaningful task representations.

Table 2. Latent dimension ablation (T5-large, Order-1).

Latent Dim	Avg Acc (%)	Avg Gap
Baseline	80.3	–
25	77.6	-2.6%
50	79.6	-0.7%
100	79.7	-0.6%

Table 3. Compression method ablation on DBpedia (T5-large). Baseline (uncompressed LoRA): 98.77%. Latent dim = 50.

Method	Accuracy (%)	Δ Acc (%)
CAE (Ours)	98.74	-0.03
Standard AE	97.75	-1.02
VAE	56.50	-42.27

4.4. Parameter and Memory Overhead in Continual Learning Approaches

We also compare continual learning (CL) strategies in terms of storage overhead during both training and deployment. The comparison includes replay-based, regularization-based, and modular adaptation approaches. Our method, COLA, stands out as a highly efficient solution. During training, it requires storing only the decoder parameters ($n\phi$) in addition to the shared base model (Θ), avoiding the need to retain large memory buffers or multiple full model copies. At deployment time, COLA stores the shared CAE decoder g_ψ, one latent adapter code $\mathbf{z}_i$ per task, and one task retrieval key $\mathbf{k}_i$ per task. Therefore, the deployment storage is:

$$|g_\psi| + K(|z| + |k|),$$

where K is the number of tasks, $|z|$ is the latent code size, and $|k|$ is the task key size. In contrast, adapter-based continual learning methods store: $K|\mu|$ where $|\mu|$ is the full adapter size. Since $|z| + |k| \ll |\mu|$, COLA substantially reduces per-task memory growth while still supporting task-agnostic retrieval.

5. Related Work

Continual Learning a.k.a Lifelong Learning aims to solve the problem of achieving good performance on alot of numbers of tasks that are arriving sequentially without catastrophic forgetting (Kirkpatrick et al., 2017; McCloskey & Cohen, 1989). Based on the nature of the tasks, Continual learning can be broadly divided into three categories: Class-Incremental Learning (CIL), Domain-Incremental Learning (DIL), Task Incremental Learning (TIL).Depending on how each task's knowledge is stored and subsequently leveraged as new tasks arrive, task-incremental methods can be divided into three categories. 1) *Regularization-based* : where regularization based constraints are introduced on model parameters by modifying the loss function to overcome catastrophic forgetting while learning new tasks. Some of the methods include EWC ..etc (Chaudhry et al., 2018a; Aljundi et al., 2018; Kirkpatrick et al., 2017) 2. *Reply-Based methods* where data from previous tasks either in the form of exemplars or buffer or complete dataset are explicitly stored and revisited to learn new things. Key appraoches include Experience Replay(Mnih et al., 2013), iCarl (Rebuffi et al., 2016) and some othe methods include (Arani et al., 2022; Caccia et al., 2021; Nguyen et al., 2018; Bonicelli et al., 2022; Sarfraz et al., 2023; Lopez-Paz & Ranzato, 2017; Wang et al., 2023b; Liang & Li, 2023). 3.*Architecture-based Networks* where model architecture is evolute as the model training goes on while learning new task. This happens either expanding the network to include new knowledge (Mallya & Lazebnik, 2018; Rusu et al., 2016; Yoon et al., 2018; Masse et al., 2018; Serra et al., 2018; Hung et al., 2019; Mandivarapu et al., 2020; Camp et al., 2020) Several recent studies have shown that fine-tuning only a small portion of a model's parameters can match the performance of full-model updates (Karimi Mahabadi et al., 2021; Lester et al., 2021; Li & Liang, 2021). Although most of this work targets single-task learning, a few efforts have begun to apply parameter-efficient approaches within continual learning. For example,(Madotto et al., 2020; Wang et al., 2023a) Utilize adapters to learn tasks over time, either by updating a subset of weights for each task or by introducing new adapters for incoming tasks. Adapter-based continual learning methods learn tasks by updating small task-specific modules or by introducing new adapters over time (Madotto et al., 2020). These methods reduce interference with the shared backbone, but often require storing full adapters for each task or relying on fixed adapter capacity. When task identity is unavailable, modular methods also need a routing mechanism; prior work has used input reconstruction error for expert selection (Aljundi et al., 2017), perplexity-based adapter selection (Madotto et al., 2020), and key-query retrieval over prompt pools (Wang et al., 2022). COLA follows this task-agnostic retrieval view, but retrieves compressed latent adapter codes and reconstructs LoRA weights on demand instead of storing full adapters.

6. Conclusion

In this work, we introduced COLA, a rehearsal-free continual learning framework for large language models that stores task knowledge as compressed latent representations of adapter weights. By combining parameter-efficient LoRA adapters with contractive autoencoder compression, COLA maintains task performance while dramatically reducing storage requirements. Our experiments across classification and dialogue benchmarks demonstrate that latent adapter representations provide a scalable mechanism for continual learning without replay buffers or growing model architectures. Future work will explore more efficient adapter retrieval mechanisms and scaling COLA to larger numbers of tasks.

References

Aljundi, R., Chakravarty, P., and Tuytelaars, T. Expert gate: Lifelong learning with a network of experts. In *Proceedings of the IEEE Conference on Computer Vision*

and Pattern Recognition, pp. 3366–3375, 2017.

Aljundi, R., Babiloni, F., Elhoseiny, M., Rohrbach, M., and Tuytelaars, T. Memory aware synapses: Learning what (not) to forget. In *Proceedings of the European conference on computer vision (ECCV)*, pp. 139–154, 2018.

Arani, E., Sarfraz, F., and Zonooz, B. Learning fast, learning slow: A general continual learning method based on complementary learning system. *arXiv preprint arXiv:2201.12604*, 2022.

Bonicelli, L., Boschini, M., Porrello, A., Spampinato, C., and Calderara, S. On the effectiveness of lipschitz-driven rehearsal in continual learning. *Advances in Neural Information Processing Systems*, 35:31886–31901, 2022.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33: 1877–1901, 2020.

Byrne, B., Krishnamoorthi, K., Sankar, C., Neelakantan, A., Duckworth, D., Yavuz, S., Goodrich, B., Dubey, A., Cedilnik, A., and Kim, K.-Y. Taskmaster-1: Toward a realistic and diverse dialog dataset. *arXiv preprint arXiv:1909.05358*, 2019.

Caccia, L., Aljundi, R., Asadi, N., Tuytelaars, T., Pineau, J., and Belilovsky, E. New insights on reducing abrupt representation change in online continual learning. *arXiv preprint arXiv:2104.05025*, 2021.

Camp, B., Mandivarapu, J. K., and Estrada, R. Continual learning with deep artificial neurons, 2020. URL https://arxiv.org/abs/2011.07035.

Chaudhry, A., Dokania, P. K., Ajanthan, T., and Torr, P. H. Riemannian walk for incremental learning: Understanding forgetting and intransigence. In *Proceedings of the European conference on computer vision (ECCV)*, pp. 532–547, 2018a.

Chaudhry, A., Ranzato, M., Rohrbach, M., and Elhoseiny, M. Efficient lifelong learning with a-gem. *arXiv preprint arXiv:1812.00420*, 2018b.

de Masson D'Autume, C., Ruder, S., Kong, L., and Yogatama, D. Episodic memory in lifelong language learning. *Advances in Neural Information Processing Systems*, 32, 2019.

Eric, M., Goel, R., Paul, S., Kumar, A., Sethi, A., Ku, P., Goyal, A. K., Agarwal, S., Gao, S., and Hakkani-Tur, D. Multiwoz 2.1: A consolidated multi-domain dialogue dataset with state corrections and state tracking baselines. *arXiv preprint arXiv:1907.01669*, 2019.

Goodfellow, I. J., Mirza, M., Xiao, D., Courville, A., and Bengio, Y. An empirical investigation of catastrophic forgetting in gradient-based neural networks. *arXiv preprint arXiv:1312.6211*, 2013.

Hoerl, A. E. and Kennard, R. W. Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, 12(1):55–67, 1970.

Hu, E. J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., and Chen, W. Lora: Low-rank adaptation of large language models, 2021. URL https://arxiv.org/abs/2106.09685.

Hung, C.-Y., Tu, C.-H., Wu, C.-E., Chen, C.-H., Chan, Y.-M., and Chen, C.-S. Compacting, picking and growing for unforgetting continual learning. *Advances in neural information processing systems*, 32, 2019.

Karimi Mahabadi, R., Henderson, J., and Ruder, S. Compacter: Efficient low-rank hypercomplex adapter layers. *Advances in Neural Information Processing Systems*, 34: 1022–1035, 2021.

Kemker, R., McClure, M., Abitino, A., Hayes, T., and Kanan, C. Measuring catastrophic forgetting in neural networks. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

Kirkpatrick, J., Pascanu, R., Rabinowitz, N., Veness, J., Desjardins, G., Rusu, A. A., Milan, K., Quan, J., Ramalho, T., Grabska-Barwinska, A., et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 114(13):3521–3526, 2017.

Larson, S., Mahendran, A., Peper, J. J., Clarke, C., Lee, A., Hill, P., Kummerfeld, J. K., Leach, K., Laurenzano, M. A., Tang, L., et al. An evaluation dataset for intent classification and out-of-scope prediction. *arXiv preprint arXiv:1909.02027*, 2019.

Lester, B., Al-Rfou, R., and Constant, N. The power of scale for parameter-efficient prompt tuning. *arXiv preprint arXiv:2104.08691*, 2021.

Li, X. L. and Liang, P. Prefix-tuning: Optimizing continuous prompts for generation. *arXiv preprint arXiv:2101.00190*, 2021.

Li, Z. and Hoiem, D. Learning without forgetting. *IEEE transactions on pattern analysis and machine intelligence*, 40(12):2935–2947, 2017.

Liang, Y.-S. and Li, W.-J. Loss decoupling for task-agnostic continual learning. *Advances in Neural Information Processing Systems*, 36:11151–11167, 2023.

Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., and Stoyanov, V. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

Lopez-Paz, D. and Ranzato, M. Gradient episodic memory for continuum learning. In *NIPS*, 2017.

Madotto, A., Lin, Z., Zhou, Z., Moon, S., Crook, P., Liu, B., Yu, Z., Cho, E., and Wang, Z. Continual learning in task-oriented dialogue systems. *arXiv preprint arXiv:2012.15504*, 2020.

Mallya, A. and Lazebnik, S. Packnet: Adding multiple tasks to a single network by iterative pruning. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pp. 7765–7773, 2018.

Mandivarapu, J. K., Camp, B., and Estrada, R. Self-net: Lifelong learning via continual self-modeling. *Frontiers in Artificial Intelligence*, Volume 3 - 2020, 2020. ISSN 2624-8212. doi: 10.3389/frai.2020.00019. URL https://www.frontiersin.org/journals/artificial-intelligence/articles/10.3389/frai.2020.00019.

Masse, N. Y., Grant, G. D., and Freedman, D. J. Alleviating catastrophic forgetting using context-dependent gating and synaptic stabilization. *CoRR*, abs/1802.01569, 2018. URL http://arxiv.org/abs/1802.01569.

McCloskey, M. and Cohen, N. J. Catastrophic interference in connectionist networks: The sequential learning problem. In *Psychology of learning and motivation*, volume 24, pp. 109–165. Elsevier, 1989.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. Playing Atari with Deep Reinforcement Learning. *ArXiv e-prints*, December 2013.

Nguyen, C. V., Li, Y., Bui, T. D., and Turner, R. E. Variational continual learning. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=BkQqq0gRb.

Papineni, K., Roukos, S., Ward, T., and Zhu, W.-J. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, pp. 311–318, 2002.

Qin, C. and Joty, S. Lfpt5: A unified framework for lifelong few-shot language learning based on prompt tuning of t5. *arXiv preprint arXiv:2110.07298*, 2021.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I., et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

Rastogi, A., Zang, X., Sunkara, S., Gupta, R., and Khaitan, P. Towards scalable multi-domain conversational agents: The schema-guided dialogue dataset. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pp. 8689–8696, 2020.

Ratcliff, R. Connectionist models of recognition memory: constraints imposed by learning and forgetting functions. *Psychological review*, 97(2):285, 1990.

Razdaibiedina, A., Mao, Y., Hou, R., Khabsa, M., Lewis, M., and Almahairi, A. Progressive prompts: Continual learning for language models. *arXiv preprint arXiv:2301.12314*, 2023.

Rebuffi, S., Kolesnikov, A., and Lampert, C. H. icarl: Incremental classifier and representation learning. *CoRR*, abs/1611.07725, 2016. URL http://arxiv.org/abs/1611.07725.

Rusu, A. A., Rabinowitz, N. C., Desjardins, G., Soyer, H., Kirkpatrick, J., Kavukcuoglu, K., Pascanu, R., and Hadsell, R. Progressive Neural Networks. *ArXiv e-prints*, June 2016.

Sarfraz, F., Arani, E., and Zonooz, B. Error sensitivity modulation based experience replay: Mitigating abrupt representation drift in continual learning. *arXiv preprint arXiv:2302.11344*, 2023.

Serra, J., Suris, D., Miron, M., and Karatzoglou, A. Overcoming catastrophic forgetting with hard attention to the task. In *International conference on machine learning*, pp. 4548–4557. PMLR, 2018.

Wang, X., Chen, T., Ge, Q., Xia, H., Bao, R., Zheng, R., Zhang, Q., Gui, T., and Huang, X. Orthogonal subspace learning for language model continual learning, 2023a. URL https://arxiv.org/abs/2310.14152.

Wang, Z., Zhang, Z., Ebrahimi, S., Sun, R., Zhang, H., Lee, C.-Y., Ren, X., Su, G., Perot, V., Dy, J., and Pfister, T. Learning to prompt for continual learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 139–149, 2022.

Wang, Z., Zhan, Z., Gong, Y., Shao, Y., Ioannidis, S., Wang, Y., and Dy, J. Dualhsic: Hsic-bottleneck and alignment for continual learning. In *International Conference on Machine Learning*, pp. 36578–36592. PMLR, 2023b.

Yoon, J., Yang, E., Lee, J., and Hwang, S. J. Lifelong learning with dynamically expandable networks. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=Sk7KsfW0-.

Zang, X., Rastogi, A., Sunkara, S., Gupta, R., Zhang, J., and Chen, J. Multiwoz 2.2: A dialogue dataset with additional annotation corrections and state tracking baselines. *arXiv preprint arXiv:2007.12720*, 2020.

A. Appendix

A.1. Datasets

We evaluate COLA across several task settings, including intent classification, dialogue state tracking (DST), natural language generation (NLG), and end-to-end response generation (E2E). Our experiments use four task-oriented dialogue benchmarks: TaskMaster-2019 (TM19) (Byrne et al., 2019), TaskMaster-2020 (TM20) (Byrne et al., 2019), Schema-Guided Dialogue (SGD) (Rastogi et al., 2020), and MultiWOZ (Zang et al., 2020; Eric et al., 2019), following the methodology described in Section 2. We also use CLINC150 (Larson et al., 2019) for intent classification.

CLINC150. CLINC150 is a multi-domain intent classification dataset containing 23,700 utterances across 150 intent classes and 10 domains. It is commonly used to evaluate intent classification and out-of-scope detection.

MultiWOZ. MultiWOZ is a large-scale multi-domain task-oriented dialogue dataset collected using a Wizard-of-Oz setup. It contains multi-turn dialogues across domains such as restaurants, hotels, attractions, trains, and taxis. We use corrected versions of MultiWOZ that address annotation errors and inconsistencies in earlier releases (Zang et al., 2020; Eric et al., 2019).

TaskMaster. TaskMaster contains spoken and written task-oriented dialogues across multiple domains. TaskMaster-2019 (TM19) includes six domains, such as restaurants, food ordering, and movies. TaskMaster-2020 (TM20) contains movie-ticketing dialogues covering user goals such as selecting a theater, movie, date, time, and number of tickets, as well as opting out of a transaction.

Schema-Guided Dialogue (SGD). SGD contains over 20K annotated task-oriented dialogue examples spanning more than 20 domains, including banks, events, media, and calendar services. The dialogues capture interactions between users and virtual assistants over service/API schemas.

We additionally construct a combined natural language understanding (NLU) benchmark by aggregating and relabeling examples from CLINC150, TM20, SGD, and MultiWOZ. This combined benchmark enables evaluation across a broader set of domains, intents, and dialogue scenarios.

Table 4. Comparison of storage overhead across continual learning methods.

Method	Train Overhead	Deploy Overhead	Notes
COLA	$\Theta + \lvert\phi\rvert + \lvert\psi\rvert$	$\lvert g_\psi\rvert + K(\lvert z\rvert + \lvert k\rvert)$	Latent adapter memory
Replay / GEM	$\Theta + n\lvert M\rvert$	$\Theta + n\lvert M\rvert$	High memory
Adapter-style	$\Theta + n\lvert\mu\rvert$	$n\lvert\mu\rvert$	Modular
EWC	Θ	Θ	Forgetting?
L2 Reg.	Θ	Θ	Forgetting?
Vanilla	Θ	Θ	No memory

Name	Train	Valid	Test	Dom.	Intent	Turns
TM19	4,403	551	553	6	112	19.97
TM20	13,839	1,731	1,734	7	128	16.92
MWoZ	7,906	1,000	1,000	5	15	13.93
SGD	5,278	761	1,531	19	43	14.71
CLINIC150	10,525	3,080	5,470	-	10	-
Total	41,951	7,123	10,288	37	290	16.23

Table 5. Main datasets statistics.