Efficient Continual Learning for Small Language Models with a Discrete Key-Value Bottleneck

Andor Diera
Ulm University
andor.diera@uni-ulm.de

Lukas Galke
University of Southern Denmark
galke@imada.sdu.dk

Fabian Karl
Ulm University
fabian.karl@uni-ulm.de

Ansgar Scherp
Ulm University
ansgar.scherp@uni-ulm.de

Abstract

Continual learning remains a challenge across various natural language processing (NLP) tasks, as models updated with new training data often risk catastrophic forgetting of previously acquired knowledge. We introduce a discrete key-value bottleneck (DKVB) for encoder-only language models, enabling efficient continual learning through localized updates. Inspired by a discrete key-value bottleneck in vision, we consider new and NLP-specific challenges. We compare different bottleneck architectures for NLP and introduce a new, task-independent initialization technique for the discrete keys. We evaluate our DKVB for NLP in four continual learning scenarios and show that it alleviates catastrophic forgetting. Our experiments demonstrate that the proposed approach achieves competitive performance compared to popular continual learning methods while incurring lower computational costs. Furthermore, we show that DKVB remains effective even in challenging single-head continual learning scenarios where no task ID is provided.[1]

1 Introduction

Large language models are receiving increasing attention from the public due to their impressive zero-shot and few-shot abilities in a wide range of tasks (Brown et al., 2020). Yet, for easier tasks where there is enough training data for supervised fine-tuning, e. g., text classification, using smaller encoder-only language models is still preferable due to their often superior performance and lower computational requirements (Yuan et al., 2023; Yu et al., 2023; Qorib et al., 2024; Li et al., 2025). Compared to large general-purpose models, fine-tuned networks lack general portability to new conditions and have limited generalization beyond their training distribution (Luo et al., 2023). For many target applications in natural language processing (NLP), training and test data can have a difference in the underlying distribution (Hupkes et al., 2023), and in the case of continual learning, the input distribution can change over time (Wang et al., 2024). To mitigate these challenges, different changes to model architectures and training regimens have been proposed (Biesialska et al., 2020; Ke and Liu, 2022; Wang et al., 2024). While many of these methods improve continual learning, they often require task-specific modules and computationally demanding extensions to the base model (Ke et al., 2021; Buzzega et al., 2020; Momeni et al., 2025).

In this work, we propose an adaptation of the Discrete Key-Value Bottleneck (DKVB) architecture (Träuble et al., 2023) to the field of NLP. Discretization techniques can improve generalization in neural networks without introducing new task-specific parameters, regularization functions, or memory buffers (Liu et al., 2021, 2023; Träuble et al., 2023). More specifically, the DKVB architecture has shown strong performance in low-resource, class incremental learning scenarios for computer vision. This is due to local, context-dependent updates on learnable discrete key-value pairs that prevent catastrophic forgetting in the models.

To address the challenges of adapting DKVB to NLP, we begin by analyzing how different variants of the discrete key-value bottleneck interact with pre-trained encoder-only language models in standard learning scenarios. In doing so, we tackle key challenges such as the high dimensionality of text representations, the choice of pooling strategies, and the design of an effective decoder head. Subsequently, we take the best-performing DKVB configurations and evaluate their performance in continual learning scenarios. Finally, we show that given a dictionary of discrete keys optimized on a general-purpose corpus, DKVB achieves similar effectiveness compared to leading continual learning approaches while requiring less training time. The main contributions of our paper are:

[1]Source code available at: github.com/drndr/dkvb_nlp

- We analyze different optimization techniques and architectures of a DKVB in NLP using BERT, RoBERTa, and DistillBERT.
- We compare our DKVB for NLP to baseline methods in continual learning scenarios, i. e., domain incremental, class incremental, and task-type incremental learning.
- We demonstrate that the DKVB alleviates catastrophic forgetting and is more efficient than most continual learning methods.

2 Related Work

2.1 Continual Learning

Sequentially learning multiple tasks remains a significant challenge in the field of deep learning. Standard neural networks trained on a new task tend to forget most of the knowledge tied to tasks they have previously learned, leading to the phenomenon commonly labeled as *catastrophic forgetting* (McCloskey and Cohen, 1989; Van de Ven and Tolias, 2019). On the other hand, leveraging knowledge learned from old tasks to improve performance on new tasks, known as *knowledge transfer*, is a highly sought-after capability in NLP (Ke and Liu, 2022). Since re-training a model from scratch is often expensive, various methods for continual learning have been proposed to handle these challenges. Existing approaches in continual learning can be categorized into five distinct families: regularization-based, optimization-based, replay-based, architecture-based, and instruction-based, with the latter being specific to large language models (Biesialska et al., 2020; Ke and Liu, 2022; Wang et al., 2024; Shi et al., 2024). A detailed description of these approaches can be found in Appendix A.

2.2 Discrete Representation Learning

Employing discrete variables in deep learning is challenging, as indicated by the prevalence of continuous latent variables in most research methods, even when the underlying modality inherently involves discrete elements (e. g., text data). Van Den Oord et al. (2017) were the first to show the viability of large-scale discrete neural representation learning through the use of vector quantization. Their Vector Quantized-Variational Autoencoder (VQ-VAE) model utilizes a discrete latent space and thus avoids the "posterior collapse" problem common in many VAE models when the decoder ignores the latent space of the encoder and relies solely on the autoregressive properties of the input samples (Goyal et al., 2017). Subsequently, their methodologies have been widely employed in various applications, including audio (Borsos et al., 2023), videos (Yan et al., 2021), and anomaly detection (Marimont and Tarroni, 2021). More recently, discretization has been utilized for machine unlearning (Shah et al., 2023) and to improve disentangled representation learning (Noh et al., 2023) and robustness (Liu et al., 2021, 2023; Träuble et al., 2023). Discretization methods with bottlenecks have been shown to improve generalization in reinforcement learning (Liu et al., 2021, 2023), visual reasoning (Liu et al., 2023), and vision-based continual learning (Träuble et al., 2023).

3 A Discrete Key-Value Bottleneck for Encoder-only Language Models

The DKVB architecture as described in Träuble et al. (2023) is fundamentally model and task-agnostic, but so far has been only studied in the field of computer vision. The use of DKVB in language models poses new challenges, including the (i) sequential nature of the input data, the (ii) high dimensionality of the encoded representations, and the (iii) difference in commonly used pooling techniques between vision and language models. Below, we describe DKVB's base architecture, the key initialization process, and our proposed architectural adaptations and pre-experiments for finding the most suitable architectural variant.

3.1 Base Architecture and Key Initialization

The DKVB architecture follows three steps: encode input, process via a discrete bottleneck, and decode. Figure 1 show an overview of the architecture.

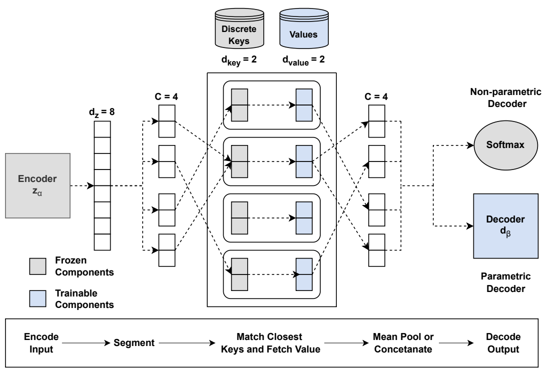


Figure 1: The base Discrete Key-Value Bottleneck.

In the first step, an encoder model projects input vector x into a lower dimensional vector $z \in \mathbb{R}^{m_z}$. This is followed by pooling (if needed) and partitioning z into C separate heads of dimension d_{key}. Each head possesses a unique discrete key-value codebook of size K, where the keys are initialized before training and are mapped to randomly initialized trainable value codes. In the second step, each head is first quantized by fetching the closest key (based on L2 distance) from the corresponding head's codebook. Subsequently, the corresponding value code of dimension d_{value} is retrieved for each head. Note the size of the bottleneck with respect to trainable parameter scales with the number of heads and codebook size. In the last step, the values are passed to a decoder to produce the final output. The decoder can be either parametric (with trainable weights) or non-parametric (by applying the softmax function to the mean pooled value codes).

The discrete keys of the bottleneck are initialized before training. Due to the 1-1 mapping between the keys and value codes, there is no gradient back-propagation between the values and keys. To ensure that the keys are broadly distributed in the feature space and have good representational power for given downstream tasks, they are first randomly initialized and then modified by using the encoded input samples as the basis for applying exponential moving average (EMA) updates (Van Den Oord et al., 2017). Alternatively, the keys can be initialized on input data different from the one in training, albeit with some decrease in downstream task performance (Träuble et al., 2023). After initialization, the keys are frozen and are not influenced by later changes in the input distribution shifts.

3.2 Architecture Adaptations for NLP

We introduce an adaptation of the DKVB architecture for the specific challenges in natural language processing. As argued above, these challenges are related to the high dimensionality of the data, pooling techniques, and decoding. We conduct pre-experiments with different architectures to find the most suitable bottleneck architecture variants and consider the following NLP-specific challenges:

Dimensionality While natural language has an inherently discrete symbolic representation, text embeddings encode these discrete symbols into a continuous latent space (Muennighoff et al., 2023). This results in a high dimensional output $z \in \mathbb{R}^{t \times h}$, where t is the token dimension (i.e., the number of tokens in the fixed length input sequence) and h is the hidden dimension. Previous experiments with DKVB were conducted on low dimensional image data that has been pooled before forwarding output $z \in \mathbb{R}^{h}$ to the bottleneck (Träuble et al., 2023). To address this difference, we design model variants with pooling applied before or after the bottleneck. Similarly, we experiment with creating the heads by partitioning hidden dimension h and token dimension t separately.

Pooling Type Most modern convolutional networks in computer vision utilize max pooling as pooling operation (He et al., 2015). Max pooling retains the most important features in images but is less commonly used in NLP due to the loss of sequential information. The two most commonly used pooling techniques in NLP are mean pooling and pooling based on a special token (CLS). In mean pooling, the contextualized token embeddings are averaged out, while the CLS pooling utilizes a special token optimized to represent the whole sequence (Devlin et al., 2018). We include both variants in our architecture search.

Decoding Decoders with adjustable weights offer more expressiveness than non-parametric decoders but are more sensitive to changes in the training conditions (Ostapenko et al., 2022). For simple tasks where linear mapping is sufficient, using just a softmax function as a non-parametric decoder might be appropriate. However, for many NLP tasks, it is crucial to capture complex patterns in the encoded representations (Wang et al., 2018). We include both approaches in our experiments. For the parametric decoder, we concatenate the value codes and feed them into a simple linear layer preceded by a dropout layer. In the non-parametric version, we apply mean pooling on the values and apply a softmax function on the pooled representation.

3.3 Analyzing DKVB Variants for NLP

We analyze different variants of the DKVB architecture in encoder-only language models. For the pre-experiments, we use two popular text classification datasets. The R8 dataset, which is a subset of the R21578 news dataset (Lewis, 1997) with 8 classes, and the Twenty Newsgroup (20ng) (Lang, 1995) which contains documents categorized into 20 newsgroups. We apply the standard train-test split for both datasets, as used in (Galke and Scherp,

Table 1: Accuracy and standard deviation (in subscript) of the different DKVB architecture variants on the R8 and 20ng datasets in a non-continual, standard learning setup, averaged over 5 runs.

		Dataset: R8				Dataset: 20ng			
		Pooling Before		Pooling After		Pooling Before		Pooling After	
Decoder	Segmentation	CLS	Mean	CLS	Mean	CLS	Mean	CLS	Mean
Parametric	hidden	$69.87_{1.07}$	$91.54_{1.19}$	$88.55_{1.05}$	$\mathbf{96.04}_{0.26}$	$19.26_{2.42}$	$53.62_{1.00}$	$48.35_{0.69}$	$\mathbf{77.83}_{0.89}$
	token	-	-	$88.06_{1.00}$	$95.20_{1.21}$	-	-	$44.65_{0.86}$	$69.33_{0.96}$
Non Parametric	hidden	$66.61_{0.29}$	$92.18_{0.36}$	$88.53_{0.22}$	$94.24_{0.39}$	$21.09_{1.31}$	$55.93_{0.75}$	$52.03_{0.24}$	$73.51_{0.20}$
	token	-	-	$64.39_{0.20}$	$73.70_{0.20}$	-	-	$10.95_{0.18}$	$15.26_{1.28}$
BERT (frozen) w/o DKVB		$95.94_{0.18}$				$72.11_{0.49}$			
BERT w/o DKVB		$98.00_{0.34}$				$84.06_{0.53}$			

2022). We first use a frozen BERT model as the pre-trained encoder for DKVB and perform a hyper-parameter search on the number of epochs, batch size, and learning rates.

We report the performance of the best configurations. For learning rates, we found it is beneficial to have a high learning rate for the values layer. Additionally, in the case of the parametric decoder setup, a lower learning rate is applied to the decoder. We use a key dimension of 12 and the number of key-value pairs of $4,096$ for the discrete bottleneck parameters as in (Träuble et al., 2023). Key initialization is done before training for three epochs with an EMA decay of 0.2. Alongside the different variants for the DKVB, we list the results of a fine-tuned BERT and a frozen BERT with a fine-tuned linear classifier on top for reference. This we consider as the upper bounds.

The test performance of the different architecture configurations can be seen in Table 1. The gap between the best-performing DKVB architecture and the fully fine-tuned BERT model is 2% on R8 and 7% on 20ng. The frozen BERT model achieved the same performance on R8 but attained 5% lower accuracy on 20ng compared to the best-performing DKVB variant. Overall the best performance was obtained by using the parametric decoder, applying mean pooling after the bottleneck, and using the hidden dimension as the base of the segmentation. To investigate the performance on other encoder-only language models, we experimented with RoBERTa (Liu et al., 2019) and DistilBERT (Sanh et al., 2019), and found the optimal bottleneck architecture to be the same (see Appendix C).

4 Continual Learning Settings

The goal of continual learning (CL) is to sequentially learn a function $f : X_k \to Y_k$ for all tasks k in sequence K. Each task k has a training set $M_k = \{(x_i, y_i, d_i, t)\}_{i=1}^{N_k}$, where $x_i \in X_k$ is a training sample, $y_i \subseteq Y_k$ is a set of class labels, $d_i \subseteq D_k$ is the corresponding domain set (e. g., legal documents, movie reviews, news articles) of the sample, $t \in T_k$ is the task-type of the training set (e. g., sentiment analysis, topical classification, natural language inference etc), and N_k is the number of samples in task k. To evaluate the DKVB architecture, we define three different incremental learning setups based on these components.

In the **Domain Incremental Learning (DIL)** setting, the task type and class labels are assumed to be consistent across all tasks. The domain of the input changes between tasks, with each task having a set of non-overlapping domains $D_k \cap D_{k'} = \emptyset$. A common DIL task-type in NLP is sentiment classification, where all tasks have the same class labels (i. e., positive, negative, neutral), but include samples from different source domains.

In the **Class Incremental Learning (CIL)** setting, each task has a set of non-overlapping classes $Y_k \cap Y_{k'} = \emptyset$. During testing, any previously learned class may be presented. CIL is generally considered the most challenging incremental learning scenario (Ke and Liu, 2022; Träuble et al., 2023). Apart from catastrophic forgetting and knowledge transfer, this setting includes the added complication of inter-task class separation (Kim et al., 2022). Inter-task class separation requires learning decision boundaries between the new task's classes and the classes from previous tasks without access to data from those previous tasks.

The main challenge in the **Task-type Incremental Learning (TIL)** setting lies in the varying task-types. While it is possible that the tasks also have non-overlapping input domains and class labels in these settings, what differentiates TIL from other incremental learning scenarios are the disjoint task-types in each task $t_k \neq t_{k'}$. This task-type is not identical to the type of objective function used in training (i. e., classification loss or regression loss); rather, it defines the downstream task of the model, such as topical classification, sentiment analysis, or measuring semantic similarity. The scenario of using one model to learn different task-types has also been heavily researched in the field of multi-task learning (Crawshaw, 2020). The dominant approach in TIL is using a multi-head configuration with a separate head (or output layer) for each task. Since this decreases the probability of catastrophic forgetting, the main challenge in TIL is bi-directional knowledge transfer (Ke and Liu, 2022).

5 Experimental Setup

In our continual learning experiments, we compare the performance of DKVB to other CL methods in the three settings described above, namely TIL, DIL, and CIL. In addition, we adapt the challenging single-head CIL setup from Trauble et al. (Träuble et al., 2023) to topical text classification. We take the best-performing bottleneck architectures from the pre-experiments and apply the same bottleneck parameters and hyperparameters. We use accuracy as the primary evaluation metric in all our experiments and present the average performance and standard deviation over five runs. These runs involve random initialization and randomized task sequence order. Additionally, we report the average per epoch runtime of each method. Details about the implementation, hyperparameters, and bottleneck parameters can be found in the Appendix B.

Datasets For the main experiments, we use three datasets, two of which have also been used in Ke et al. (2021). We use the Document Sentiment Classification (DSC) dataset in the DIL setting. It consists of 10 subsets of product reviews with a positive or negative sentiment label. Each subset constitutes a separate task with $4,000$ training, 500 validation, and 500 test samples. Since the tasks are similar and only differ in the product domain, this dataset is used to evaluate knowledge transfer. In the CIL setup, we use the earlier described 20ng dataset (Lang, 1995). Similarly to Ke et al. (2021), we create a sequence of 10 tasks consisting of two classes each. This setup is mainly used to test the models' abilities to overcome catastrophic forgetting. For the TIL setup, we create a sequence of tasks by combining four tasks from the GLUE benchmark (Wang et al., 2018). This dataset, which we call 4GLUE, includes four different task-types: The RTE dataset is used for testing natural language inference, the MRPC is used for measuring semantic textual similarity, the SST-2 is a popular dataset for sentiment analysis, and the QQP dataset which is used for natural language inference and question answering.

For the single-head CIL experiments, we use two different versions of the R21578 news dataset (Lewis, 1997), R8 (includes 8 classes) and R52 (with 52 classes). Due to the R21578 dataset's highly skewed class frequency distribution, we simulate a low-resource training scenario and include only 100 samples from each class in both datasets. On R8, we divide the dataset into 8 increments, with one class for each increment. On R52, we create 26 increments, each with two random classes.

Procedure We follow the CL evaluation procedure of De Lange et al. (2021). A model is trained sequentially on all tasks and is evaluated by averaging the test performance of each task recorded after the final training increment. This results in each task in the sequence being a binary classification problem. In the multi-head configurations (for CIL and TIL), we use a separate decoder for each task and provide the task ID during training and evaluation. To further investigate the continual learning capabilities of the DKVB, we implement the single-head CIL setup of Träuble et al. (2023). Compared to the multi-head CIL task, this setup is considered to be more challenging and lacks explicit task boundaries. For its evaluation, the models are tested on the whole test data after each increment, including previously unseen classes.

Baselines We use the best-performing methods reported in Ke et al. (2021), selecting one from each CL approach (cf. Section 2.1). From regularization-based methods, we choose **EWC** (Serra et al., 2018), a common baseline with strong performance in many CL studies. **DER++** (Buzzega et al., 2020) belongs to the replay-based methods and uses distilled knowledge from past experiences to guide the incremental training process. **OWM** (Zeng et al., 2019) is an optimization-based approach that constrains the gradient updates to a direction orthogonal to the input space of previously trained tasks. Lastly, **CTR** (Ke et al., 2021) is an architecture-based approach that utilizes capsule networks to

Table 2: Average Accuracy and Macro F1 scores of the tasks in the three continual learning scenarios, using the average and standard deviation (in subscript) of 5 runs with randomized sequence orders.

CL Method	**Model**	**DIL (DSC)** Acc	F1	**CIL (20NG)** Acc	F1	**TIL (4GLUE)** Acc	F1
NCL	BERT	$88.50_{0.63}$	$87.83_{0.64}$	$53.95_{0.49}$	$38.94_{0.53}$	$62.12_{0.58}$	$58.29_{0.60}$
NCL	BERT (frozen)	$87.42_{2.23}$	$86.58_{2.17}$	$96.35_{1.03}$	$96.31_{1.22}$	$71.90_{0.11}$	$\mathbf{69.10}_{0.12}$
NCL	Adapter-BERT	$88.71_{2.20}$	$\mathbf{88.10}_{2.24}$	$65.61_{9.34}$	$58.62_{11.63}$	$68.74_{0.30}$	$63.86_{0.53}$
DER++	BERT (frozen)	$84.30_{1.54}$	$82.85_{1.94}$	$59.68_{9.23}$	$47.62_{13.74}$	$70.64_{3.27}$	$68.68_{3.89}$
EWC	BERT (frozen)	$86.21_{4.89}$	$85.53_{4.98}$	$\mathbf{96.80}_{0.20}$	$\mathbf{96.80}_{0.20}$	$66.54_{9.74}$	$58.26_{14.84}$
OWM	BERT (frozen)	$86.06_{2.63}$	$85.28_{2.66}$	$88.80_{0.28}$	$88.16_{0.30}$	$67.54_{2.11}$	$61.90_{2.57}$
CTR	Adapter-BERT	$\mathbf{88.73}_{0.35}$	$87.98_{0.37}$	$95.53_{0.14}$	$95.52_{0.16}$	$\mathbf{72.71}_{0.19}$	$66.42_{0.78}$
DKVB-NP Incremental	BERT (frozen)	$80.99_{2.07}$	$79.58_{1.91}$	$59.67_{1.59}$	$54.58_{1.66}$	$58.12_{2.89}$	$50.14_{1.90}$
DKVB-NP Oracle	BERT (frozen)	$\mathbf{83.93}_{1.11}$	$\mathbf{81.98}_{1.74}$	$\mathbf{97.06}_{0.22}$	$95.84_{0.95}$	$\mathbf{69.65}_{0.34}$	$\mathbf{68.92}_{0.38}$
DKVB-NP Generic	BERT (frozen)	$82.12_{0.20}$	$80.97_{0.09}$	$96.30_{0.07}$	$\mathbf{96.27}_{0.10}$	$68.79_{0.51}$	$65.37_{0.03}$
DKVB-P Incremental	BERT (frozen)	$74.09_{4.88}$	$68.01_{5.10}$	$57.81_{2.00}$	$52.89_{2.77}$	$58.77_{3.23}$	$51.02_{1.81}$
DKVB-P Oracle	BERT (frozen)	$81.18_{0.61}$	$80.47_{0.52}$	$95.22_{0.44}$	$95.09_{0.25}$	$58.65_{1.43}$	$51.81_{1.53}$
DKVB-P Generic	BERT (frozen)	$71.71_{1.30}$	$57.75_{0.95}$	$92.76_{0.88}$	$92.73_{0.89}$	$61.40_{0.57}$	$54.76_{0.42}$

prevent catastrophic forgetting and facilitate knowledge transfer. We also include three baselines without any additional forgetting or knowledge transfer handling, noted as *naive continual learning* (**NCL**).

For DKVB we take the best-performing architectures from Section 3.3, and include both the parametric (**DKVB-P**) and non-parametric (**DKVB-NP**) variants. We experiment with three different strategies for key initialization. In the first two strategies, we use the training data for initializing the keys: in the incremental setup, the keys are optimized in a continual fashion before each task using only the training data of the given increment (denoted as **Incremental**), while in the full initialization setup, the keys are initialized once before training, using the full training input distribution (denoted as **Oracle**). In the third setup, we use a cross-domain corpus different from the training data to create general-purpose keys (denoted as **Generic**). For this, we use a small version of the English Wikipedia dump[2], which is commonly included in pre-training datasets. In all three setups, we use an EMA decay of 0.2. For the Incremental and Oracle setups, the key initialization is set to three epochs, while for the Generic we use one epoch.

All CL methods (except CTR) are applied to a frozen BERT model and have a single-head configuration without any task-ID information for the DIL scenario and a multi-head configuration with task-ID provision on the CIL and TIL scenarios. CTR is based on an Adapter-BERT (Houlsby et al., 2019) backbone and requires a multi-head setup and task-ID information for its dynamic architecture in all scenarios. For the single-head CIL experiments, we include the naive baselines and the replay-based DER++ method. The rest of the CL baseline methods either require explicit task boundaries for optimal performance (OWM, EWC) or only work in a multi-head configuration (CTR).

[2]https://huggingface.co/datasets/wikipedia

6 Results

Main Experiments The results of the main experiments are shown in Table 2. In the DIL setting the difference in accuracy between the baseline methods is low, with CTR having the highest score of 88.73%. The performance of the DKVB variants in this scenario is below the baselines. In the CIL setting, there is a substantial variation between model performance, with half of the CL methods achieving over 90% accuracy, while BERT NCL, DER++, and the incremental DKVB variants have an accuracy score below 60%. The best result on the CIL dataset was achieved with the non-parametric DKVB Oracle (97.06%) followed by EWC (96.80%) and BERT frozen NCL (96.35%). In the TIL scenario, the highest accuracy scores were achieved with BERT frozen NCL (71.90%) and CTR (72.71%). Within the DKVB variants the best performance was consistently seen with the non-parametric Oracle variant, closely followed by the non-parametric Generic variant. On the CIL and TIL scenarios both of these methods outperformed most of the baselines. Additional measures of the backward transfer performances can be found in Appendix Section C.

Runtime We measure the average epoch runtimes for each model to compare the computational costs of the different methods. The results can be found in Table 3. Among the evaluated methods, DKVB achieves runtime closest to NCL with a frozen BERT, where training is limited to optimizing a parametric decoder. While the regularization-based EWC and the optimization-based OWM methods also achieve a runtime comparable to the NCL frozen BERT model, adding replay in DER++ and dynamic architecture in CTR substantially increases runtime. The key initialization process scales with the number of samples, but the overall computational cost of DKVB remains lower than most continual learning methods since initialization is done once before training and involves just a forward pass. The average runtime of key initialization is shown in Table 4

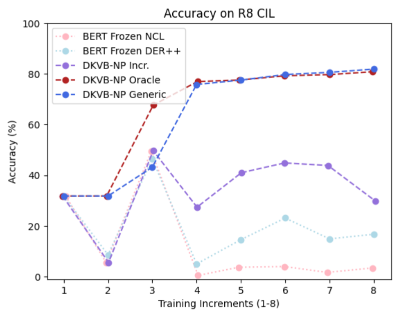
Table 3: Per-epoch training runtimes (in seconds), averaged over a single run. Standard deviations are shown as subscripts.

CL Method	Model	DIL (DSC)	CIL (20NG)	TIL (4GLUE)
NCL	BERT	$20.6_{3.0}$	$8.9_{0.0}$	$482.3_{659.4}$
NCL	BERT (frozen)	$4.4_{0.6}$	$1.9_{0.0}$	$105.5_{144.2}$
NCL	Adapter-BERT	$24.1_{3.4}$	$10.4_{0.0}$	$566.2_{772.7}$
DER++	BERT (frozen)	$26.2_{0.9}$	$7.4_{2.5}$	$249.7_{361.2}$
EWC	BERT (frozen)	$8.0_{0.08}$	$2.3_{0.2}$	$129.0_{176.6}$
OWM	BERT (frozen)	$6.7_{0.3}$	$2.0_{0.1}$	$108.9_{148.6}$
CTR	Adapter-BERT	$487.1_{0.4}$	$195.2_{0.1}$	$3011.7_{0.0}$
DKVB-NP	BERT (frozen)	$4.67_{0.6}$	$2.00_{0.0}$	$109.35_{149.2}$
DKVB-P	BERT (frozen)	$4.88_{0.7}$	$2.07_{0.0}$	$114.38_{156.5}$

Table 4: Per-epoch key initialization runtimes (in seconds) and corresponding sample sizes. Standard deviations are shown as subscripts.

Key Initialization	DIL (DSC)	CIL (20NG)	TIL (4GLUE)
Incremental	$4.7_{0.6}$ (n=4 000)	$1.9_{0.0}$ (n=1 600)	$111.6_{152.5}$ (n=87 470)
Oracle	$46.9_{0.5}$ (n=40 000)	$19.5_{0.1}$ (n=16 000)	$535.22_{1.9}$ (n=349 881)
Generic	$469.0_{2.1}$ (n=205 328)	$469.0_{2.1}$ (n=205 328)	$469.0_{2.1}$ (n=205 328)

Single-head Class Incremental Learning The single-head class incremental learning results are shown in Figure 2. The highest accuracy scores are 81.17% on R8 and 47.78% on R52. Both scores were achieved with the non-parametric DKVB variant using the Generic and Oracle key initialization, respectively. On both datasets, the non-DKVB models, which included the BERT frozen NCL and DER++, displayed sharp drops in performance between increments, indicating the occurrence of catastrophic forgetting and overfitting on the current training increment. DER++ showcased better performance than the naive baseline but still underperformed the Oracle and Generic variants, with a final accuracy score of 16.70% on R8 and 35.75% on R52. The detailed results with additional models (BERT NCL, EWC) can be found in Appendix Section C.



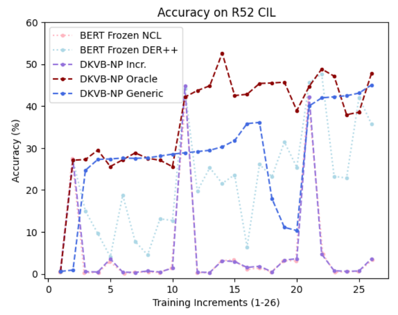


Figure 2: Progressive test accuracy scores in the single-head class increment learning setup, averaged over 5 runs with fixed sequence order

7 Discussion

Key Insights Our experiments show that fine-tuning encoder-only language models with an optimal discrete key-value bottleneck architecture achieves comparable results to partial fine-tuning in standard learning scenarios, but greatly benefits CL, both in terms of performance and efficiency. The best key initialization is obtained by unsupervised access to the full input feature distribution, but utilizing a general-purpose corpus for key initialization is also a viable option for NLP tasks. Below, we discuss these key insights.

Architectural Variants We found that employing pooling before the bottleneck has a substantial negative effect on the model performance (see Section 3.3). This suggests that in contrast to lower dimensional vision tasks (Träuble et al., 2023), it is necessary to retain the full dimensionality of the text encodings. Similarly, CLS pooling is inferior compared to mean pooling across all setups. Segmenting on the token dimension only worked in the case of parametric decoding, indicating that the DKVB module's output yields better representational power if the segmentation happens on the hidden dimension. An additional linear layer after the DKVB module, acting as a parametric decoder, can compensate for encodings with weaker representational power. However, a non-parametric decoder produces comparable results in most configurations.

Continual Learning In the CL experiments reported in Section 5, the non-parametric DKVB variants achieved comparable results to other CL methods and maintained runtimes on par with the NCL frozen BERT variant, but only when using pre-initialized keys.

When using incremental key initialization, performance was consistently subpar, indicating that DKVB requires access to a general-purpose corpus or the full input distribution to achieve competitive performance. While having access to the full data distribution may be unrealistic in practice, our experiments show that this is not needed in NLP. Rather, when initializing the keys using a general-purpose corpus, we obtain results that are close to the Oracle setup.

The largest performance drop between the DKVB variants and other CL methods was seen in the DIL setting. This suggests that DKVB's strength in preventing catastrophic forgetting through distinct key-value bindings becomes its weakness in DIL, as this compartmentalization restricts the model's ability to transfer knowledge across different domains. Notably, NCL methods achieve similar results as the CL methods in this DIL setup, indicating that pre-trained language models without a bottleneck are already well-suited for domain incremental learning.

In the CIL and TIL tasks, only the frozen BERT NCL variant showcased performance comparable to that of the CL methods. The strong performance of frozen BERT in these experiments suggests that if task-ID is available during testing, a simple multi-head configuration with a frozen encoder is often sufficient. Experiments in the single-head CIL setup have shown to be more challenging. As the models are tested on the full test set after each increment, ideally, they should exhibit a progressive increase in accuracy. But when no task-ID is provided and decoding is done with a single head, most models overfit and suffer catastrophic forgetting between increments, with DKVB being the only model to demonstrate improved CL capability in this scenario. This suggests that DKVB's unique architecture effectively maintains knowledge across tasks without needing task-specific heads.

8 Conclusion

The discrete key-value bottleneck offers an efficient approach to continual learning. It enables context-dependent updates in the model without explicit parameter isolation or dynamically expanding the architecture. Considering the special challenges of continual learning with text embeddings, we analyzed twelve architectural variants of the bottleneck. The best variants apply mean pooling after the bottleneck and utilize the hidden dimension of the encoded input representation to create the bottleneck heads.

We conducted a comprehensive evaluation across different continual learning settings in NLP, i. e., domain-incremental learning, class-incremental learning, and task-type incremental learning, and showed that with proper key initialization, the discrete key-value bottleneck offers consistent improvement in most settings and is comparable to dedicated continual learning methods from the literature. Moreover, we showed that it can be used even in the most challenging single-head continual learning scenarios when no task-ID is provided.

9 Limitations

Our study focuses solely on encoder-only language models. While this raises questions about whether our results could generalize to other model architectures, our choice was motivated by their preference for supervised fine-tuning scenarios where the balance between performance and computational efficiency is crucial. Our experiments were also limited to fine-tuning for classification-based downstream tasks. Consequently, it remains to be investigated whether our results extend to other NLP tasks, such as semantic search, entity extraction, or machine translation.

References

Magdalena Biesialska, Katarzyna Biesialska, and Marta R Costa-jussà. 2020. Continual lifelong learning in natural language processing: A survey. In *Proceedings of the 28th International Conference on Computational Linguistics*, pages 6523–6541.

Zalán Borsos, Raphaël Marinier, Damien Vincent, Eugene Kharitonov, Olivier Pietquin, Matt Sharifi, Dominik Roblek, Olivier Teboul, David Grangier, Marco Tagliasacchi, et al. 2023. AudioLM: a language modeling approach to audio generation. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 31:2523–2533.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

Pietro Buzzega, Matteo Boschini, Angelo Porrello, Davide Abati, and Simone Calderara. 2020. Dark experience for general continual learning: a strong, simple baseline. *Advances in neural information processing systems*.

Arslan Chaudhry, Marc'Aurelio Ranzato, Marcus Rohrbach, and Mohamed Elhoseiny. 2018. Efficient lifelong learning with a-gem. In *International Conference on Learning Representations*.

Michael Crawshaw. 2020. Multi-task learning with deep neural networks: A survey. *arXiv preprint arXiv:2009.09796*.

Matthias De Lange, Rahaf Aljundi, Marc Masana, Sarah Parisot, Xu Jia, Aleš Leonardis, Gregory Slabaugh, and Tinne Tuytelaars. 2021. A continual learning survey: Defying forgetting in classification tasks. *IEEE transactions on pattern analysis and machine intelligence*, 44(7):3366–3385.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Lukas Galke and Ansgar Scherp. 2022. Bag-of-words vs. graph vs. sequence in text classification: Questioning the necessity of text-graphs and the surprising strength of a wide mlp. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 4038–4051.

Anirudh Goyal, Alessandro Sordoni, Marc-Alexandre Côté, Nan Rosemary Ke, and Yoshua Bengio. 2017. Z-forcing: Training stochastic recurrent networks. *Advances in neural information processing systems*, 30.

Yiduo Guo, Wenpeng Hu, Dongyan Zhao, and Bing Liu. 2022. Adaptive orthogonal projection for batch and online continual learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 6783–6791.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Spatial pyramid pooling in deep convolutional networks for visual recognition. *IEEE transactions on pattern analysis and machine intelligence*, 37(9):1904–1916.

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*, volume 97, pages 2790–2799.

Ching-Yi Hung, Cheng-Hao Tu, Cheng-En Wu, Chien-Hung Chen, Yi-Ming Chan, and Chu-Song Chen. 2019. Compacting, picking and growing for unforgetting continual learning. *Advances in Neural Information Processing Systems*, 32.

Dieuwke Hupkes, Mario Giulianelli, Verna Dankers, Mikel Artetxe, Yanai Elazar, Tiago Pimentel, Christos Christodoulopoulos, Karim Lasri, Naomi Saphra, Arabella Sinclair, et al. 2023. A taxonomy and review of generalization research in NLP. *Nature Machine Intelligence*.

Khurram Javed and Martha White. 2019. Meta-learning representations for continual learning. *Advances in neural information processing systems*, 32.

Zixuan Ke and Bing Liu. 2022. Continual learning of natural language processing tasks: A survey. *arXiv preprint arXiv:2211.12701*.

Zixuan Ke, Bing Liu, Nianzu Ma, Hu Xu, and Lei Shu. 2021. Achieving forgetting prevention and knowledge transfer in continual learning. *Advances in Neural Information Processing Systems*, 34:22443–22456.

Gyuhak Kim, Changnan Xiao, Tatsuya Konishi, Zixuan Ke, and Bing Liu. 2022. A theoretical study on solving continual learning. *Advances in neural information processing systems*, 35:5065–5079.

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

Ken Lang. 1995. Newsweeder: Learning to filter netnews. In *International Conference on Machine Learning*, pages 331–339.

David Lewis. 1997. Reuters-21578 Text Categorization Collection. UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C52G6M.

Haitao Li, Qingyao Ai, Jia Chen, Qian Dong, Zhijing Wu, and Yiqun Liu. 2025. Blade: Enhancing black-box large language models with small domain-specific models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pages 24422–24430.

Dianbo Liu, Alex Lamb, Xu Ji, Pascal Junior Tikeng Notsawo, Michael Mozer, Yoshua Bengio, and Kenji Kawaguchi. 2023. Adaptive discrete communication bottlenecks with dynamic vector quantization for heterogeneous representational coarseness. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 8825–8833.

Dianbo Liu, Alex M Lamb, Kenji Kawaguchi, Goyal Anirudh, Chen Sun, Michael C Mozer, and Yoshua Bengio. 2021. Discrete-valued neural communication. *Advances in Neural Information Processing Systems*, 34:2109–2121.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. RoBERTa: A robustly optimized BERT pretraining approach. *arXiv preprint arXiv:1907.11692*.

David Lopez-Paz and Marc'Aurelio Ranzato. 2017. Gradient episodic memory for continual learning. *Advances in neural information processing systems*, 30.

Yun Luo, Zhen Yang, Xuefeng Bai, Fandong Meng, Jie Zhou, and Yue Zhang. 2023. Investigating forgetting in pre-trained representations through continual learning. *arXiv preprint arXiv:2305.05968*.

Sergio Naval Marimont and Giacomo Tarroni. 2021. Anomaly detection through latent space restoration using vector quantized variational autoencoders. In *2021 IEEE 18th International Symposium on Biomedical Imaging (ISBI)*, pages 1764–1767. IEEE.

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

Seyed Iman Mirzadeh, Mehrdad Farajtabar, Dilan Gorur, Razvan Pascanu, and Hassan Ghasemzadeh. 2020. Linear mode connectivity in multitask and continual learning. *arXiv preprint arXiv:2010.04495*.

Saleh Momeni, Sahisnu Mazumder, and Bing Liu. 2025. Continual learning using a kernel-based method over foundation models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pages 19528–19536.

Niklas Muennighoff, Nouamane Tazi, Loic Magne, and Nils Reimers. 2023. Mteb: Massive text embedding benchmark. In *Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics*, pages 2014–2037.

Haechan Noh, Sangeek Hyun, Woojin Jeong, Hanshin Lim, and Jae-Pil Heo. 2023. Disentangled representation learning for unsupervised neural quantization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12001–12010.

Oleksiy Ostapenko, Timothee Lesort, Pau Rodriguez, Md Rifat Arefin, Arthur Douillard, Irina Rish, and Laurent Charlin. 2022. Continual learning with foundation models: An empirical study of latent replay. In *Conference on Lifelong Learning Agents*, pages 60–91. PMLR.

Muhammad Reza Qorib, Geonsik Moon, and Hwee Tou Ng. 2024. Are decoder-only language models better than encoder-only language models in understanding word meaning? In *Annual Meeting of the Association for Computational Linguistics*.

Anastasia Razdaibiedina, Yuning Mao, Rui Hou, Madian Khabsa, Mike Lewis, and Amjad Almahairi. 2023. Progressive prompts: Continual learning for language models. In *The Eleventh International Conference on Learning Representations*.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2019. DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*.

Thomas Scialom, Tuhin Chakrabarty, and Smaranda Muresan. 2022. Fine-tuned language models are continual learners. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 6107–6122.

Joan Serra, Didac Suris, Marius Miron, and Alexandros Karatzoglou. 2018. Overcoming catastrophic forgetting with hard attention to the task. In *International conference on machine learning*, pages 4548–4557. PMLR.

Vedant Shah, Frederik Träuble, Ashish Malik, Hugo Larochelle, Michael Mozer, Sanjeev Arora, Yoshua Bengio, and Anirudh Goyal. 2023. Unlearning via sparse representations. *arXiv preprint arXiv:2311.15268*.

Haizhou Shi, Zihao Xu, Hengyi Wang, Weiyi Qin, Wenyuan Wang, Yibin Wang, Zifeng Wang, Sayna Ebrahimi, and Hao Wang. 2024. Continual learning of large language models: A comprehensive survey. *ACM Computing Surveys*.

Hanul Shin, Jung Kwon Lee, Jaehong Kim, and Jiwon Kim. 2017. Continual learning with deep generative replay. *Advances in neural information processing systems*, 30.

Frederik Träuble, Anirudh Goyal, Nasim Rahaman, Michael Curtis Mozer, Kenji Kawaguchi, Yoshua Bengio, and Bernhard Schölkopf. 2023. Discrete key-value bottleneck. In *International Conference on Machine Learning*, pages 34431–34455. PMLR.

Gido M Van de Ven and Andreas S Tolias. 2019. Three scenarios for continual learning. *arXiv preprint arXiv:1904.07734*.

Aaron Van Den Oord, Oriol Vinyals, et al. 2017. Neural discrete representation learning. *Advances in neural information processing systems*, 30.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. 2018. Glue: A multi-task benchmark and analysis platform for natural language understanding. *EMNLP 2018*, page 353.

Liyuan Wang, Xingxing Zhang, Hang Su, and Jun Zhu. 2024. A comprehensive survey of continual learning: theory, method and application. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

Mitchell Wortsman, Vivek Ramanujan, Rosanne Liu, Aniruddha Kembhavi, Mohammad Rastegari, Jason Yosinski, and Ali Farhadi. 2020. Supermasks in superposition. *Advances in Neural Information Processing Systems*, 33:15173–15184.

Wilson Yan, Yunzhi Zhang, Pieter Abbeel, and Aravind Srinivas. 2021. VideoGPT: Video generation using VQ-VAE and transformers. *arXiv preprint arXiv:2104.10157*.

Wenpeng Yin, Jia Li, and Caiming Xiong. 2022. Contintin: Continual learning from task instructions. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3062–3072.

Hao Yu, Zachary Yang, Kellin Pelrine, Jean Francois Godbout, and Reihaneh Rabbany. 2023. Open, closed, or small language models for text classification? *arXiv preprint arXiv:2308.10092*.

Lifan Yuan, Yangyi Chen, Ganqu Cui, Hongcheng Gao, Fangyuan Zou, Xingyi Cheng, Heng Ji, Zhiyuan Liu, and Maosong Sun. 2023. Revisiting out-of-distribution robustness in nlp: Benchmarks, analysis, and llms evaluations. *Advances in Neural Information Processing Systems*, 36:58478–58507.

Guanxiong Zeng, Yang Chen, Bo Cui, and Shan Yu. 2019. Continual learning of context-dependent processing in neural networks. *Nature Machine Intelligence*, 1(8):364–372.

Supplementary Materials

A Extended Related Work

Regularization-based methods This family of methods involves incorporating explicit regularization terms to maintain a balance between the old and new tasks. This is usually done by adding penalty or regularization to the loss function to prevent large changes to parameters deemed important for old tasks (Wang et al., 2024). A popular method in this family is Elastic Weight Consideration (EWC), which calculates the importance of parameters with the Fisher information matrix, and applies smaller updates to weights deemed critical for earlier tasks (Kirkpatrick et al., 2017).

Replay-based methods These methods either retain a subset of training examples from previous tasks in memory such as A-GEM (Chaudhry et al., 2018), or learn to generate pseudo samples from previous tasks, like in DGR (Shin et al., 2017). These samples are then incorporated into the training regimen of new tasks. While this can alleviate catasthropical forgetting, the size of a memory buffer is limited, which can potentially affect generalizability (Wang et al., 2024).

Optimization-based methods Explicitly manipulating the optimization process is another way to tackle the challenges of continual learning. Gradient-projection methods ensure that gradient updates happen exclusively in the orthogonal direction to the gradients of an old tasks, thereby preventing any impact on weights important for old tasks (Zeng et al., 2019; Guo et al., 2022). Meta learning strategies and methods focusing on obtaining flat minima in the loss landscape can be also utilized in continual learning (Javed and White, 2019; Mirzadeh et al., 2020).

Architecture-based methods Methods in this family can be generally divided into *parameter isolation* and, *dynamic architecture* approaches, depending on whether the model architecture is fixed or not (Wang et al., 2024). Models such as SupSup (Wortsman et al., 2020) and HAT (Serra et al., 2018) optimize a binary mask to selectively choose dedicated parameters for each task and fall under the parameter isolation category. Other methods dynamically expand the model with new parameters to increase capacity for learning new tasks (Ke et al., 2021; Hung et al., 2019).

Instruction-based methods This family is unique to the field of NLP. These methods are based on task-specific instructions given to encoder-decoder or decoder only language models when a new task is encountered. While some methods in this family show promising knowledge transfer capabilities (Scialom et al., 2022; Yin et al., 2022; Razdaibiedina et al., 2023), without explicit fine-tuning they are mostly limited by the knowledge acquired in the pre-training phase.

B Extended Experimental Setup

Implementation For all model backbones in our experiments, we use the BERT-base model from Huggingface[3] and use cross-entropy loss as our objective function. We base our discrete-key-value bottleneck implementation on the *vector-quantize-pytorch*[4] package. In the pre-experiments, we truncate each input sample to 256 tokens. For the main continual learning experiments, we rely on the *PyContinual*[5] framework and reuse their implementations and hyperparameters on the baseline methods. To remain comparable to other studies using the *PyContinual* framework, we kept the default pre-processing steps, used a maximum token length of 128, and applied the default convolutional decoder of the baseline models. For the single-head class incremental learning experiments we use a fixed randomized sequence order when creating the increments, and used a token length of 256. The source code for our experiments alongside the models can be found at github.com/drndr/dkvb_nlp.

Optimization As part of our pre-experiments, we also conducted a hyperparameter search and a sensitivity analysis on the bottleneck parameters. Outside of the selected hyperparameters and bottleneck parameters, all other configurations remained fixed during the search. Our experiments use the BERT-base architecture with a hidden size of 768. For the optimizer, we chose AdamW with a weight decay of 0.01. The dropout rate for the parametric decoder was set to 0.1. For the reference fully fine-tuned BERT numbers we reused the hyperparameters reported in (Galke and Scherp, 2022), for the frozen BERT variant we relied on the parametric DKVB variant hyperparameters with mean pooling. During fine-tuning, we carefully optimized the models on both datasets using grid-search-based manual tuning. A search space for the selected hyperparameters was defined, specifically we chose the batch size from $\{8, 16, 32\}$, the number of epochs from $\{5, 10\}$, the learning rate for the values layer from $\{1e\text{-}1, 1e\text{-}2, 1e\text{-}3\}$, and the decoder learning rate from $\{1e\text{-}3, 1e\text{-}4, 1e\text{-}5\}$. The best performing (based on the validation loss) configurations for each architecture variant can be seen in Table 5. The hyperparameters were reused for the continual learning main experiments.

In the single-head class incremental learning experiments we conducted an additional manual hyperparameter search for the DKVB variants and found a batch size of 16 with a global learning rate of $1e - 2$ to be the best performing. For EWC we set the lambda parameter to 5,000. In the DER++ model we used a memory buffer size of 256 and set the sampling reate in each increment to 16.

B.1 Bottleneck Parameters

In our experiments, we rely on the optimal bottleneck parameter analysis of (Träuble et al., 2023). Additionally, we also conduct a small sensitivity study for the discrete key dimension and number of key-value pairs on the R8 dataset. For this, we use the DKVB-NP model variant from the pre-experiments and keep everything fixed, changing only these two bottleneck parameters. For the base hyperparameters, we reuse the best-performing configurations.

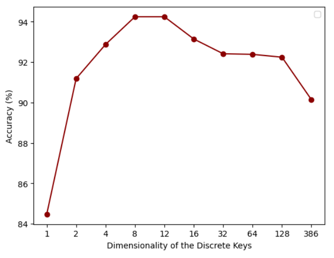
Key dimension The number of dimensions of the discrete keys strongly influences the utility of the bottleneck. This can be explained as follows. Keys that have too few dimensions increase the chance of unintended key sharing between inputs from different distributions, while discrete keys with too high dimensionality can lead to insufficient coverage of the embedding space. Similarly to (Träuble et al., 2023) we found the optimal key dimension to be between 8 to 12. The results of this analysis are depicted in Figure 3.

Number of key-value pairs The number of key-value pairs determines the size of the discretized representational space. In accordance with the analysis of (Träuble et al., 2023), we found that increasing the number of key-value pairs leads to a performance increase. Eventually further increments no longer yield substantial improvements in performance. Note that increasing this parameter also leads to increased model size and increases the

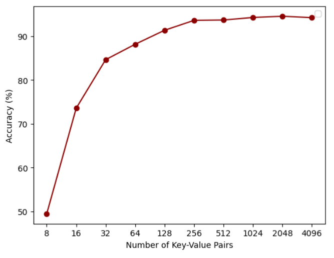
[3]https://huggingface.co/bert-base-uncased
[4]https://github.com/lucidrains/vector-quantize-pytorch
[5]https://github.com/ZixuanKe/PyContinual

Table 5: Best hyperparameter configuration for each architecture variant based on validation loss

Model	Segmentation	Pooling	(A) Dataset: R8 #Epoch	(A) Dataset: R8 Batch size	(A) Dataset: R8 Values LR	(A) Dataset: R8 Decoder LR	(B) Dataset: 20ng #Epoch	(B) Dataset: 20ng Batch size	(B) Dataset: 20ng Values LR	(B) Dataset: 20ng Decoder LR
DKVB-P	hidden	Before CLS	5	32	$1e$-2	$1e$-4	10	16	$1e$-1	$1e$-3
DKVB-P	hidden	Before Mean	5	32	$1e$-1	$1e$-4	10	16	$1e$-2	$1e$-4
DKVB-P	hidden	After CLS	10	16	$1e$-2	$1e$-4	5	16	$1e$-2	$1e$-4
DKVB-P	hidden	After Mean	10	16	$1e$-2	$1e$-3	5	16	$1e$-2	$1e$-3
DKVB-P	token	After CLS	10	16	$1e$-2	$1e$-3	10	16	$1e$-2	$1e$-3
DKVB-P	token	After Mean	10	16	$1e$-2	$1e$-3	10	16	$1e$-2	$1e$-3
DKVB-NP	hidden	Before CLS	5	32	$1e$-1	-	5	32	$1e$-1	-
DKVB-NP	hidden	Before Mean	5	32	$1e$-1	-	10	32	$1e$-1	-
DKVB-NP	hidden	After CLS	10	16	$1e$-1	-	5	16	$1e$-1	-
DKVB-NP	hidden	After Mean	10	32	$1e$-1	-	10	16	$1e$-1	-
DKVB-NP	token	After CLS	10	32	$1e$-1	-	10	32	$1e$-2	-
DKVB-NP	token	After Mean	10	32	$1e$-1	-	10	32	$1e$-2	-



(a)



(b)

Figure 3: Assessing the sensitivity of bottleneck parameters in regards of test accuracy: (a) Dimensionality of discrete key (b) Number of key-value pairs

computational costs of key initialization as well. The results of this analysis are depicted in Figure 3.

C Extended Results

Architectural Variants Results obtained using the RoBERTa (Table 6) and DistilBERT (Table 7) models demonstrate comparable performance patterns to those observed with BERT. Interestingly, DistilBERT produced slightly better accuracy on most architecture variants compared to the other two models. However the highest performance was consistently seen with mean pooling after the bottleneck on all three models. DistilBERT's improved performance can likely be attributed to differences in its tokenization and pooling implementation compared to other models.

Continual Learning Experiments In the field of continual learning additional metrics are often used to measure the performance over incremental learning. Two often used metrics are Forward Transfer (FWT) and Backward Transfer (BWT) (Lopez-Paz and Ranzato, 2017). BWT refers to how learning a new task affects performance on a previously learned task. It can be positive, when learning a new task improves performance on the earlier task, or negative, when it worsens it. Severe negative backward transfer is often called catastrophic forgetting. FWT describes how learning a new task influences performance on a future task. Since pre-trained language models already posses high transfer learning capabilities (Brown et al., 2020), its difficult to isolate the effect of learning specific task on future performance. Therefore we focus on BWT which is formally defined as:

$$\text{BWT} = \frac{1}{T-1}\sum_{i=1}^{T-1}(R_{T,i} - R_{i,i}) \qquad (1)$$

where $R \in \mathbb{R}^{T\times T}$ is the results matrix of an incremental learning scenario with T tasks, where each entry $R_{i,j}$ being the test accuracy on task j after training on task i (Lopez-Paz and Ranzato, 2017).We report the BWT numbers on the three continual learning scenarios in Table 8.

Table 6: Accuracy and standard deviation (in subscript) of the different DKVB architecture variants with RoBERTa on the R8 and 20ng datasets in a non-continual, standard learning setup, averaged over 5 runs.

		Dataset: R8				**Dataset: 20ng**			
		Pooling Before		**Pooling After**		**Pooling Before**		**Pooling After**	
Decoder	**Segmentation**	**CLS**	**Mean**	**CLS**	**Mean**	**CLS**	**Mean**	**CLS**	**Mean**
Parametric	hidden	$49.48_{0.05}$	$91.36_{0.40}$	$91.73_{0.81}$	$\mathbf{94.25}_{0.26}$	$51.05_{0.43}$	$56.63_{0.39}$	$52.18_{1.11}$	$\mathbf{75.08}_{0.21}$
	token	-	-	$90.02_{1.05}$	$94.05_{0.31}$	-	-	$19.86_{1.03}$	$27.30_{0.92}$
Non Parametric	hidden	$49.45_{0.02}$	$92.05_{0.29}$	$74.53_{1.74}$	$93.04_{0.20}$	$56.83_{0.35}$	$60.42_{0.35}$	$53.10_{1.37}$	$70.33_{0.78}$
	token	-	-	$58.74_{0.92}$	$66.33_{0.74}$	-	-	$9.89_{0.66}$	$12.51_{1.67}$
RoBERTa (frozen) w/o DKVB				$94.29_{0.17}$				$69.42_{0.30}$	
RoBERTa w/o DKVB				$97.54_{0.51}$				$83.36_{0.30}$	

Table 7: Accuracy and standard deviation (in subscript) of the different DKVB architecture variants with DistilBERT on the R8 and 20ng datasets in a non-continual, standard learning setup, averaged over 5 runs.

		Dataset: R8				**Dataset: 20ng**			
		Pooling Before		**Pooling After**		**Pooling Before**		**Pooling After**	
Decoder	**Segmentation**	**CLS**	**Mean**	**CLS**	**Mean**	**CLS**	**Mean**	**CLS**	**Mean**
Parametric	hidden	$90.22_{0.30}$	$92.17_{0.25}$	$90.46_{0.44}$	$\mathbf{96.09}_{0.21}$	$56.26_{0.53}$	$60.24_{0.20}$	$60.26_{0.31}$	$\mathbf{79.79}_{0.49}$
	token	-	-	$89.24_{0.93}$	$94.78_{0.86}$	-	-	$45.45_{1.13}$	$68.06_{0.90}$
Non Parametric	hidden	$89.79_{0.24}$	$92.02_{0.38}$	$90.92_{0.22}$	$95.09_{0.27}$	$60.73_{0.48}$	$59.98_{0.30}$	$61.43_{0.50}$	$75.11_{0.44}$
	token	-	-	$66.37_{0.49}$	$72.65_{0.24}$	-	-	$12.04_{0.51}$	$18.70_{1.03}$
DistilBERT (frozen) w/o DKVB				$94.62_{0.16}$				$68.56_{0.38}$	
DistilBERT w/o DKVB				$97.83_{0.24}$				$83.84_{0.23}$	

Table 8: Average Backward Transfer (BWT) scores on the three continual learning scenarios

CL Method	**Model**	**DIL (DSC)**	**CIL (20NG)**	**TIL (4GLUE)**
NCL	BERT	0.29	−29.77	−20.00
NCL	BERT (frozen)	−0.10	−0.38	−6.70
NCL	Adapter-BERT	0.39	−20.01	−16.05
DER++	BERT (frozen)	−0.58	−27.11	−7.05
EWC	BERT (frozen)	0.06	**-0.27**	−10.84
OWM	BERT (frozen)	0.26	−15.44	−8.23
CTR	Adapter-BERT	**0.49**	−0.50	**-6.19**
DKVB-NP Incremental	BERT (frozen)	−3.06	−27.73	−21.30
DKVB-NP Oracle	BERT (frozen)	**-0.88**	**-0.12**	**-7.22**
DKVB-NP Generic	BERT (frozen)	−1.17	−0.29	−7.97
DKVB-P Incremental	BERT (frozen)	−4.88	−29.05	−20.99
DKVB-P Oracle	BERT (frozen)	−1.02	−0.41	−20.56
DKVB-P Generic	BERT (frozen)	−6.24	−4.94	−16.00

Table 9: Mean accuracy scores of single-head class incremental learning experiments on R8, averaged over 5 runs with fixed sequence order

Increment	# Test Samples	BERT	BERT-frozen	BERT-frozen DER++	BERT-frozen EWC	DKVB-NP Incremental	DKVB-NP Oracle	DKVB-NP Wiki
1.	1596	31.79	31.79	31.79	31.79	31.79	31.79	31.79
2.	253	5.52	5.52	8.60	31.79	5.57	31.80	31.79
3.	2840	49.47	49.47	46.47	12.65	49.70	67.70	43.19
4.	41	0.45	0.45	4.99	49.61	27.40	76.98	75.83
5.	190	3.70	3.70	14.57	49.56	41.02	77.57	77.56
6.	206	3.97	3.97	23.11	3.70	44.86	79.26	79.76
7.	108	1.64	1.64	14.89	3,74	43.85	79.72	80.61
8.	251	3.42	3.42	16.71	2.64	29.83	80.86	81.90

Table 10: Mean accuracy scores of single-head class incremental learning experiments on R52, averaged over 5 runs with fixed sequence order

Increment	# Test Samples	BERT	BERT-frozen	BERT-frozen DER++	BERT-frozen EWC	DKVB-NP Incremental	DKVB-NP Oracle	DKVB-NP Wiki
1.	45	0.70	0.50	0.56	0.50	0.73	0.54	0.60
2.	1600	26.20	27.10	26.07	0.50	27.10	27.10	0.92
3.	52	0.46	0.23	14.99	0.85	0.54	27.29	24.71
4.	29	0.35	0.35	9.60	3.69	0.46	29.51	27.30
5.	321	2.92	2.95	4.08	2.57	3.58	25.60	27.41
6.	37	0.35	0.07	18.76	0.35	0.42	27.22	27.63
7.	17	0.35	0.42	7.73	0.35	0.35	28.85	27.55
8.	44	0.54	0.46	4.53	0.35	0.70	27.57	27.67
9.	28	0.50	0.35	13.13	0.42	0.46	27.10	28.10
10.	110	1.40	1.40	12.73	0.35	1.40	25.58	28.56
11.	3046	42.17	42.17	44.18	0.35	44.82	42.17	28.87
12.	16	0.97	0.35	19.78	0.70	0.42	43.71	29.17
13.	10	0.15	0.23	25.31	0.97	0.35	44.85	29.48
14.	193	3.15	3.15	21.58	0.42	3.15	52.64	30.30
15.	213	3.38	3.38	23.62	3.38	2.95	42.52	31.80
16.	154	1.09	1.09	6.43	3.38	1.55	42.83	35.83
17.	145	1.40	1.40	26.11	1.47	1.83	45.40	36.12
18.	32	0.50	0.50	23.19	0.50	0.50	45.52	17.95
19.	203	3.15	3.15	31.43	1.83	3.30	45.71	11.10
20.	227	3.15	3.15	25.42	3.15	3.62	38.94	10.34
21.	2948	42.17	42.17	45.67	3.15	42.25	44.74	40.07
22.	255	4.71	4.71	47.63	42.25	4.71	48.84	41.96
23.	59	0.58	0.38	23.27	42.17	0.77	47.15	42.23
24.	48	0.58	0.58	22.85	0.58	0.62	37.96	42.48
25.	59	0.58	0.58	42.04	0.58	0.70	38.55	43.11
26.	243	3.38	3.38	35.75	0.58	3.62	47.78	45.04