Adaptation Approaches for Nearest Neighbor Language Models

Rishabh Bhardwaj[1]* **George Polovets**[2] **Monica Sunkara**[2]
[1]Singapore University of Technology and Design, Singapore
[2]AWS AI Labs
rishabhbhardwaj15@gmail.com
{polovg, sunkaral}@amazon.com

Abstract

Semi-parametric Nearest Neighbor Language Models (*k*NN-LMs) have produced impressive gains over purely parametric LMs, by leveraging large-scale neighborhood retrieval over external memory datastores. However, there has been little investigation into adapting such models for new domains. This work attempts to fill that gap and suggests the following approaches for adapting *k*NN-LMs — 1) adapting the underlying LM (using Adapters), 2) expanding neighborhood retrieval over an additional adaptation datastore, and 3) adapting the weights (scores) of retrieved neighbors using a learned Rescorer module. We study each adaptation strategy separately, as well as the combined performance improvement through ablation experiments and an extensive set of evaluations run over seven adaptation domains. Our combined adaptation approach consistently outperforms purely parametric adaptation and zero-shot (*k*NN-LM) baselines that construct datastores from the adaptation data. On average, we see perplexity improvements of 17.1% and 16% for these respective baselines, across domains.

1 Introduction

Natural Language Processing (NLP) has observed large performance improvements with recent advancements in neural Language Models (LMs). These models have enabled learning rich, semantic text representations (Mikolov et al., 2010; Bengio et al., 2000) that have facilitated a wide range of downstream language tasks (Radford et al., 2018, 2019). For the task of next-word prediction, parametric LMs utilize the rich contextual text representations as input to a classifier (output layer), which produces a distribution over the possible next words.

In contrast to parametric LMs, *k*-Nearest Neighbor LMs (*k*NN-LMs) are semi-parametric models that maintain an external memory (i.e. **datastore**) (Khandelwal et al., 2019). This datastore is composed of key-value pairs, where the keys are contextual embeddings created from passing text data through an LM, and the values are the respective next-word labels. The datastore can be used to retrieve *k*-nearest neighbors for the current context. The retrieved values induce a probability distribution over the next word, which is combined with the LM probabilities.

This mixture of probabilities has produced impressive gains over probabilities obtained from purely parametric LMs and has been shown to generate even larger improvements with the in-

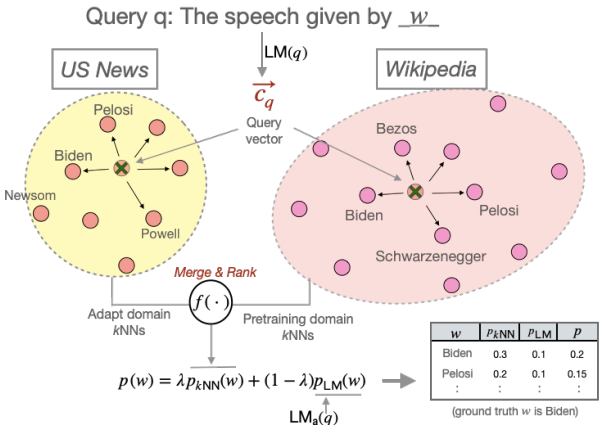

Figure 1: An illustration of the proposed *k*NN-LM adaptation approach. The current context is used as a query (q) for nearest-neighbor retrieval. The context is passed through the LM to obtain the query vector representation $\vec{c}_q$, which is then used to retrieve nearest neighbors from a large pretraining datastore and a smaller adaptation datastore (displayed in pink and yellow, respectively). The function $f(\cdot)$ represents merging of datastores (Merge), followed by rescoring (Rank) of the retrieved neighbors to obtain $p_{k\text{NN}}$. The probability distribution over the candidate next words is computed by the mixture of probabilities $p_{k\text{NN}}$ and p_{LM}, where p_{LM} denotes probabilities obtained from domain-adapted LM.

*Work done during an internship at AWS AI Labs.

Findings of the Association for Computational Linguistics: ACL 2023, pages 1135–1146
July 9-14, 2023

crease in the scale of the datastore (Khandelwal et al., 2019; Yogatama et al., 2021; He et al., 2021). While the dependency on a large-scale datastore is easy to satisfy when developing general-purpose pretrained models, it is challenging to develop effective *k*NN-LMs when it comes to specialized domains. This is due to the scarcity of domain-specific data, limiting the size of the corresponding datastore.

We posit that large, general-purpose datastores, referred to as the **pretraining datastore**, contain a significant amount of relevant information which can still be applied to specialized domains. This information can be leveraged through nearest-neighbor retrieval and should prove especially useful in situations where there is an insufficient amount of domain-specific data to generate an effective standalone datastore.

Unlike parametric neural architectures which can employ gradient-based finetuning for domain adaptation, it is less obvious how to adapt *k*NN-LMs primarily because of the non-parametric nature of datastores. One simple approach would be to reconstruct the datastore, using domain-adapted LM representations. However, this comes at the cost of incurring a large memory footprint for each adaptation domain. In this work, we instead choose to focus on adaptation strategies that are parameter and memory efficient. Given the complementary nature of the parametric and non-parametric components in a *k*NN-LM, we pursue adaptation strategies separately for each component and analyze their impact on the *k*NN-LM system's adaptation performance.

1. Adaptation of the parametric LM: Given that we constrain ourselves to parameter-efficient adaptation techniques, we utilize Adapters (Houlsby et al., 2019) for finetuning the parametric LM because of their competitive performance with full model finetuning (Hou et al., 2022; Liu et al., 2022). We also investigate the impact of adapting the parametric component on the quality of retrieved neighbors from the pretraining datastore.

2. Adaptation of the non-parametric *k*NN: As a memory-efficient alternative to reconstructing the pretraining datastore with domain-adapted representations, we formulate *k*NN adaptation as learning a domain-specific neighborhood scoring function (i.e. a **Rescorer**). This proposed Rescorer is trained to assign optimal weights to each retrieved neighbor for a given domain. We also consider expanding our neighborhood retrieval to include an additional datastore referred to as the **adaptation datastore**, created purely from the target domain. Relative to the pretraining datastore, the addition of the adaptation datastore further increases the memory footprint by an incremental amount.

In line with previous works, we focus our experiments solely on the core Language Modeling task of next-word prediction (Khandelwal et al., 2019; Yogatama et al., 2021). Results on seven adaptation domains ranging from science and books, to conversational text, demonstrate that our component-level strategies consistently improve over respective parametric and semi-parametric baselines, and produce even better results when combined together. Specifically, we find that adaptation of the parametric component increases recall of ground-truth labels found in the retrieved neighbors. We also confirm that the large-scale pretraining datastore contains relevant information for adaptation domains, via its performance edge over models that exclude it. Finally, we observe that expanding the nearest neighbor search to include elements from the adaptation datastore contributes to the best overall performing strategy. Figure 1 demonstrates the overall approach using Wikipedia and US News as example pretraining and adaptation domains, respectively.

2 kNN-LMs

For a context c_t defined by the sequence of words $(w_1, \ldots, w_{t-1})$, the causal language modeling task aims to model a probability distribution over the next word[1] w_t. Let $p_{\text{LM}}(w_t|c_t)$ and $p_{k\text{NN}}(w_t|c_t)$ be the probability masses computed by the LM and *k*NN components, respectively. Details on how $p_{k\text{NN}}(w_t|c_t)$ is computed and combined with $p_{\text{LM}}(w_t|c_t)$ to produce the final *k*NN-LM predictions, are outlined in the following sections.

Datastore creation: Given a source domain training set $\mathcal{X}_s$, let $c_i = (w_1, \ldots, w_{t-1})$ be a sequence in $\mathcal{X}_s$. The datastore is defined as a set of D_s tuples $\{(\vec{c}_i, w_i)\}_{i=1}^{D_s}$, where the key $\vec{c}_i \in \mathbb{R}^{d_h}$ denotes the contextual representation of c_i, produced by the

[1]We use "token" and "word" interchangeably.

LM and value w_i denotes the next word label in the sequence.

***k*-Nearest neighbor retrieval:** During inference, we obtain a query vector $\vec{c_q} \in \mathbb{R}^{d_h}$ for *k*NN retrieval by producing the contextual LM representation for the current sequence of tokens c_q. The neighborhood of $\vec{c_q}$ is constructed by retrieving its k nearest instances from the datastore. Let $\mathcal{D}(\cdot) : \mathbb{R}^{2d_h} \rightarrow \mathbb{R}$ refer to the distance measure[2]. The *k*-nearest neighbors of $\vec{c_q}$ can be obtained by:

$$\mathcal{K} \coloneqq \arg\min_{\text{k}} \{\mathcal{D}(\vec{c_q}, \vec{c_i})\}_{i \in [D_s]} \tag{1}$$

where k in the subscript denotes indices in $[D_s]$=$\{1, \ldots, D_s\}$ which corresponds to k smallest distances. The score (weight) s_i of a neighbor key $\vec{c_i}$ is defined as:

$$s_i \coloneqq ||\vec{c_q} - \vec{c_i}||^2, i \in \mathcal{K} \tag{2}$$

Thus, the *k*NN probability of the next word can be obtained via:

$$p_{k\text{NN}}(w_t|c_t) \propto \sum_{i \in \mathcal{K}} \mathbb{1}[w_i{=}w_t] \exp(-s_i). \tag{3}$$

Unifying *k*NN and LM: The probability distribution of the *k*NN-LM system can be obtained by interpolating the component probabilities

$$\begin{aligned} &p_{k\text{NN-LM}}(w_t|c_t) = \\ &\lambda\, p_{k\text{NN}}(w_t|c_t) + (1-\lambda)\, p_{\text{LM}}(w_t|c_t) \end{aligned} \tag{4}$$

where $\lambda \in [0, 1]$.

Since each probability distribution lies on a simplex spanning the token vocabulary, performing a convex combination of the two maintains a valid probability distribution.

3 *k*NN-LM Adaptation

3.1 Retrieval Quality Metrics

Beyond tracking LM perplexity improvement, we also introduce two simple metrics to measure the relevance and quality of retrieved neighborhoods. For neighborhood relevance, we define Recall as the fraction of times a ground-truth word is in the retrieved set of neighbors. For neighborhood quality, we denote Precision as the fraction of times the *k*NN assigns more probability to the ground truth token than the LM. We define:

$$\text{Precision} = \sum_{t=1}^{N} \frac{\mathbb{1}[p_{\text{LM}}(w_t^*|c_t) < p_{k\text{NN}}(w_t^*|c_t)]}{N},$$

$$\text{Recall} = \sum_{t=1}^{N} \frac{\mathbb{1}[w_t^* \in \mathcal{K}_t]}{N}.$$

where $\mathcal{K}_t \coloneqq \{\vec{w_i} : i \in [\mathcal{K}]\}$, w_t^* is the ground truth next word for the context $\vec{c_t}$, and N is the total number of words in the dataset.

3.2 Parametric LM Adaptation

We follow a parameter-efficient adaptation approach by keeping the pretrained LM fixed and learning Adapter modules, attached to all the model layers (Houlsby et al., 2019). Henceforth, we use $\mathbf{LM_a}$ to denote the domain-adapted LM.

While Adapter-based models have shown performance on par with model fine-tuning on various adaptation tasks across domains (Pfeiffer et al., 2020), the impact of LM adaptation in a semi-parametric setting remains unclear. Given our constraint to keep the pretraining datastore keys static, updates to the parametric LM could create a mismatch between contextual representations used for querying and result in meaningless retrieved neighborhoods. We posit that Adapter-based LMs do not suffer from this because they preserve the metric space induced by the LM contextual encodings[3]. Adapters tune representations in the original space such that they are more relevant to the adaptation domain.

> *Hypothesis-1* : LM adaptation with Adapters not only assists the parametric models to perform better (↓ perplexity), but also improves the quality of neighborhoods retrieved from the pretraining datastore (↑ Recall).

3.3 *k*NN Adaptation

Given that we choose to keep the memory footprint of our adaptation approaches small (relative to the pretraining footprint), we fix the pretraining datastore representations (and thus the Recall of the retrieved neighborhoods) and instead focus on improving the Precision. This leads to our second hypothesis:

[2]In practice large-scale datastores utilize approximate search methods for retrieval, detailed further in Section 4.1.

[3]This is due to Adapters keeping the pretrained LM weight matrices frozen, thus preserving the coordinate space that is projected onto, when extracting contextual representations.

Hypothesis-2 : Using squared L2 distance between the query and neighbor key vectors is not the optimal neighbor scoring scheme. Instead, a more optimal scoring function can be learned for each domain.

We propose learning a domain-optimized scoring function (a Rescorer) that learns to assign more weight to retrieved neighbors containing the ground-truth label. We discuss the setup and architecture for the Rescorer in more detail subsequently.

Rescorer Formulation: Given a query vector $\vec{c_q}$ obtained from LM_a, we retrieve a large set of neighbors $\mathcal{K}$. Each retrieved neighbor tuple $(\vec{c_i}, w_i) \in \mathcal{K}$ is passed through a neural module to obtain a domain-optimized score s_i^r. Let $f^r(\cdot) : \mathbb{R}^{d_r} \rightarrow \mathbb{R}$ denote the Rescorer function. Its input is a set of three vectors: query $\vec{c_q}$, neighbor key vector $\vec{c_i}$, token embedding of the neighbor value w_i, as well as six features $\vec{x_i} = \{x_1, \ldots, x_6\}$ obtained from the pairwise dot products and pairwise euclidean distances between these three vectors[4]. The total input dimension is; $d_r = 3d_h + 6$ where d_h is the dimension of the LM contextual representation. The final neighbor score s_i' can be computed as[5]:

$$s_i^r = f^r([\vec{c_i}, \vec{c_q}, \vec{w_i}, \vec{x_i}]) \quad (5)$$

$$s_i' = s_i^r - s_i \quad (6)$$

Rescorer Architecture: We employ a three-layer fully-connected network for the Rescorer architecture. The input vectors are first layer-normalized and concatenated. They are then passed through two ReLU-activated dense layers with a skip connection $\mathbb{R}^{d_r} \rightarrow \mathbb{R}^{128} \rightarrow \mathbb{R}^{128}$ and a final dense (regression) layer $\mathbb{R}^{128} \rightarrow \mathbb{R}^1$ to generate the neighbor's score. The overall Rescorer workflow is shown in Figure 2.

Rescorer Training: We train the Rescorer to discriminate neighbors containing the ground truth as their values, by employing Contrastive Learning. We construct positive examples for retrieved neighbor tuples $(\vec{c_i}, w_i)$ if w_i corresponds to the correct ground-truth next word, otherwise they are treated as negatives. We collect contextual embeddings

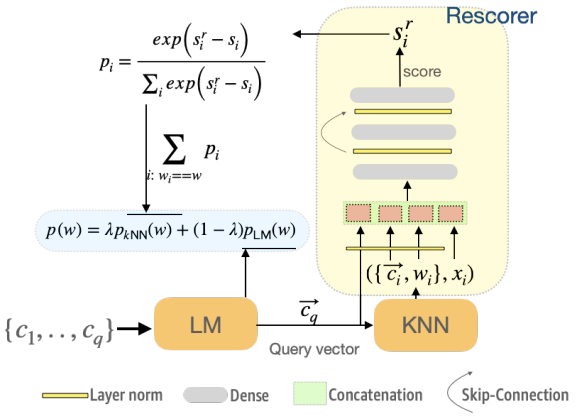

Figure 2: *k*NN-LM Rescorer workflow. Neighbors extracted using the query contextual embedding are passed to a fully-connected Rescorer network. The scores output from the network are used to produce adapted nearest-neighbor probabilities p_{kNN}, which are used in the final p_w calculation.

for one million tokens from the adaptation domain training split[6] $\{w_1, \ldots, w_{1M}\}$ along with their nearest neighbors. Contrastive training examples are discarded if the ground-truth word is not found in the neighborhood values. From each neighborhood, the highest-scored (distance-based) positive neighbor is selected and 10 negative neighbors are randomly sampled. Contrastive Loss (Oord et al., 2018) is used to learn the Rescorer parameters and is defined as:

$$\mathcal{L} = -\log \frac{\exp\left(\frac{s_p^r}{\tau}\right)}{\exp\left(\frac{s_p^r}{\tau}\right) + \sum_n \exp\left(\frac{s_n^r}{\tau}\right)} \quad (7)$$

where s_p^r and s_n^r denote the Rescorer scores assigned to the positive and negative examples, respectively, and τ is a temperature hyperparameter.

3.4 Merging *k*NNs

While regenerating the pretraining datastore using an adapted LM (Section 3.2) is generally a very memory-intensive procedure, creating a separate datastore purely from adaptation training data is expected to increase the memory footprint by a relatively small amount[7]. With the availability of both pretraining and adaptation datastores, a natural extension is to utilize both during neighborhood retrieval. We extract the nearest neighbors independently from the pretraining

[4]We find that using these extra features produces the best quality Rescorer.

[5]We empirically observed that combining the learned and distance-based scores produces the best results.

[6]If the training set has less than one million tokens, we utilize all of its tokens.

[7]In our experimental setup, this amounts to 1-10% relative increase in memory footprint

datastore $\mathcal{K}_w$ and adaptation datastore $\mathcal{K}_a$ and merge them to create $\mathcal{K}_a \cup \mathcal{K}_w$.

3.5 Adaptation of *k*NN-LMs

We summarize the overall adaptation strategy outlined in prior sections as follows:

1. Updating the parametric LM using lightweight Adapter modules.
2. Merging the retrieved neighbors from the pretraining and adaptation datastores into a single neighborhood.
3. Training a Rescorer with Contrastive Loss, to learn domain-optimal scores for retrieved neighbors.

In the following results sections, we confirm the validity of Hypothesis-1 and Hypothesis-2, as well as the efficacy of our simple neighborhood merging scheme through ablation studies. We also investigate the benefit of our collective adaptation strategy on modeling downstream domains.

4 Experiments

4.1 Experimental Setup

For all of our experiments, we utilize the off-the-shelf GPT-2 (Radford et al., 2019) model from Huggingface Transformers (Wolf et al., 2019), as the pretrained LM. This model contains 117 million parameters with a vocabulary size of 50,257 word units, and directly matches the decoder-only configuration used in Khandelwal et al. (2019). For the adaptation setting, Adapter modules are added to each layer of the pretrained GPT-2 resulting in 0.7% extra parameters during finetuning. Training the Rescorer also amounts to learning an incremental 320K parameters, or roughly 0.3% additional parameters relative to that of GPT-2. The Rescorer and Adapters are trained (separately) using AdamW optimizer with learning rates of 0.001 and 0.0001, respectively and a weight decay of 0.01. For sampling positive and negative examples during Rescorer training, we utilize a liberally sized neighborhood of size k=1000. Logging is performed every 200 iterations and early stopping is performed if there is no validation performance improvement for up to three logging steps.

The pretraining datastore is constructed from running GPT-2 on 1 billion tokens sampled from Wikipedia[8] (i.e. $\mathcal{K}_w$) and any adaptation datastores are constructed from up to 100 million tokens taken from the training split of adaptation domains (i.e. $\mathcal{K}_a$). We select seven datasets across multiple domains to evaluate the performance of our adaptation strategies: XSum (Narayan et al., 2018) and XL-Sum (Hasan et al., 2021) covering the news domain; SciQ (Johannes Welbl, 2017) and arXiv (Cohan et al., 2018) for the science domain; BookSum (Kryscinski et al., 2021) for the literature domain, SAMSum (Gliwa et al., 2019) for the conversational domain, and GovReport (Huang et al., 2021) for the government domain. For any summary-related datasets, we only utilize the original document text for our purposes and exclude summary ground-truth text. Table 1 provides a breakdown of the resulting adaptation datastore sizes.

Xsum	SciQ	arXiv	BookSum	SAMSum	XLSum	GovSum
99.5	1.0	77.3	4.5	0.4	100.0	10.5

Table 1: Adaptation datastore size (in millions of entries).

For nearest neighbor retrieval, we use FAISS - a library designed for fast similarity search in high dimensional space (Johnson et al., 2019). Similar to Khandelwal et al. (2019), we observe that L2-based FAISS search obtains better results than the inner-product, so we adopt this setting for our work as well. For all experiments, we perform hyperparameter search over k where k$\in\{1, 2, 4, \ldots, 512, 1000\}$ and the *k*NN interpolation parameter $\lambda\in\{0.01, 0.02, 0.04, \ldots, 0.98\}$.

4.2 Models Used for Evaluation

Because our work is the first to explore the intersection of LM adaptation with semi-parametric LMs, we use relevant baselines from both individual methodologies to track our modeling improvements. We provide the pretraining **(w)*k*NN** and adaptation **(a)*k*NN** neighborhood retrieval perplexities for reference[9], to illustrate relevance of the pretraining domain to target domains and relationship between retrieval quality and datastore size. For the LM adaptation baseline, we compare against the performance of parametric finetuning with Adapters $\mathbf{LM}_a$. For the semi-parametric LM base-

[8](Foundation)–https://huggingface.co/datasets/wikipedia

[9](w)*k*NN is obtained by putting $\lambda = 0.9999$ in Equation (4) to tackle cases where ground truth is not present in the retrieved neighborhood.

lines, we use two types of zero-shot evaluations of the kNN-LM. One applies zero-shot evaluation using the pretrained datastore **(w)*k*NN-LM** and the other evaluates using a datastore constructed out of the adaptation domain training data **(a)*k*NN-LM**. The latter strategy, also presented in Khandelwal et al. (2019), to the best of our knowledge, is the only other work that utilizes adaptation data with kNN-LMs.

Beyond these models, we perform extensive experimentation with different combinations of datastores to use for retrieval (Wikipedia - **(w)**, Adaptation training split - **(a)**, Both - **(w+a)**), types of parametric LMs (Pretrained LM - **LM**, Adapted LM - **LM**$_a$), and usage of Rescorers (Rescorer used - ***k*NN**$_r$, No Rescorer used - ***k*NN**). These combinations provide precise ablations of our adaptation component improvements and their interactions with one another.

5 Results and Discussions

5.1 Hypothesis Pilot Studies

We first motivate our larger experimental effort with pilot studies that test out Hypothesis-1, Hypothesis-2, and the neighborhood merging strategy. These studies are run on a subset of 100K test tokens taken from each adaptation domain. In our initial pilot experiments, we find that using k=1000 neighbors produces the best results. Any adaptation strategy requiring gradient-based optimization is performed on the respective adaptation training splits.

Evaluating Hypothesis-1: To test this hypothesis, we measure the impact of LM adaptation on retrieval quality from the pretraining , by observing changes to the kNN's Recall value. Table 2 demonstrates that adaptation of the parametric LM (LM$_a$) improves not only perplexity, but also retrieval Recall (retrieved neighbors using LM$_a$ are denoted by kNN*, while neighbors retrieved with the pretrained LM are denoted by kNN). This appears to support our hypothesis that techniques like Adapters, which preserve the LM representation space, can also benefit the retrieval component.

Evaluating Hypothesis-2: To test whether Rescorer-generated scores improve over purely distance-based scores, we contrast the resulting Precision of both types of scoring methods. Table 3 shows that the domain-adapted scores produced by the Rescorer yield significantly higher neighborhood Precision on average than those using purely L2-based scoring. This applies for neighbors retrieved from the pretraining datastore (w)kNN$_r$, as well as from datastores constructed from adaptation domain samples (a)kNN$_r$. This suggests that the Rescorer can act as a general-purpose improvement over the standard kNN-LM setup, regardless of whether neighbors are retrieved from in-domain or out-of-domain datastores. The improvement in Precision also confirms the efficacy of Contrastive Learning in producing a Rescorer that can discriminate between neighbors containing the ground-truth token from those that don't.

Domain	Perplexity (↓)				Recall (↑)	
	LM	LM$_a$	kNN	kNN*	kNN	kNN*
XSum	22.45	**18.95**	83.95	**74.67**	88.72	**89.29**
SciQ	22.15	**16.10**	46.86	**38.64**	92.53	**93.26**
arXiv	56.83	**24.97**	513.44	**270.11**	77.54	**79.89**
BookSum	21.15	**20.45**	64.92	**62.15**	90.14	**90.34**
SAMSum	46.86	**32.25**	298.08	**228.99**	96.36	**96.64**
XL-Sum	24.87	**21.84**	100.92	**89.65**	87.98	**88.60**
GovReport	19.31	**14.72**	83.62	**66.91**	88.55	**89.47**

Table 2: Hypothesis -1 pilot study. Adapted LM representations improve both perplexity and retrieval Recall.

	Precision (↑)			
Domain	(w)kNN	(w)kNN$_r$	(a)kNN	(a)kNN$_r$
XSum	29.6	**44.9**	45.9	**59.8**
SciQ	33.9	**48.2**	45.8	**53.0**
arXiv	25.6	**38.2**	52.8	**65.4**
BookSum	33.1	**54.7**	33.7	**50.1**
SAMSum	25.9	**27.7**	37.0	**38.6**
XL-Sum	29.9	**46.7**	43.9	**58.5**
GovReport	25.7	**42.6**	43.7	**55.9**

Table 3: Hypothesis-2 pilot study. Applying the Rescorer leads to Precision improvements in neighbors retrieved from both pretraining and adaptation datastores.

Effectiveness of Neighborhood Merging To test the effectiveness of the simple neighborhood merging strategy, we contrast the Recall of merged neighborhoods to those of standalone neighborhoods from each individual datastore. In this study, we keep the total number of retrieved neighbors fixed and empirically find that retrieving 500 nearest neighbors from each datastore in the merging strategy works best. The results of this study (Table 4) show that the combined set of neighbors $\mathcal{K}_a \cup \mathcal{K}_w$ has a better Recall value than either individual neighborhood. Due to this observed Recall improvement, we use this simple merging tech-

	Recall (↑)		
Domain	(w)*k*NN	(a)*k*NN	(w+a)*k*NN
XSum	89.3	92.7	**93.1**
SciQ	92.5	91.5	**94.7**
arXiv	79.9	91.9	**92.1**
BookSum	90.3	86.8	**91.5**
SAMSum	84.3	85.7	**88.9**
XL-Sum	88.6	92.1	**92.5**
GovReport	89.5	92.5	**93.6**

Table 4: Recall improvement from merging *k*NNs. Merged neighborhoods consistently obtain higher Recall than those obtained from the individual datastores.

nique in our overall adaptation strategy. When training a Rescorer on these merged neighborhoods, we pass an additional binary input feature to inform the model on which datastore a particular neighbor comes from.

5.2 Domain Adaptation Evaluations

Table 5 compares the perplexities of the various models evaluated on the seven adaptation test sets. First we note that while the adapted LM yields the expected perplexity reductions over the pretrained LM (LM_a < LM), we observe that zero-shot evaluation of the pretrained *k*NN-LM also performs better than the pretrained LM ((w)*k*NN-LM < LM). This continues to confirm the capacity of the pretraining datastore to retrieve relevant neighbors for downstream domains. We also find that in a majority of the cases, zero-shot evaluation of a *k*NN-LM constructed over the adaptation datastore, outperforms parametric adaptation ((a)*k*NN-LM < LM_a). This corroborates the finding from Khandelwal et al. (2019), where utilizing data for neighborhood retrieval can outperform using it for LM training.

The results further support our Hypothesis-1, namely that parametric LM adaptation improvement is compounded when used in the *k*NN-LM setting (e.g. $(w)k\text{NN}_r$ < LM_a < (w)*k*NN-LM_a). They also add support for Hypothesis-2 where the Rescorer acts as a general-purpose improvement to *k*NN-LM models (by noting that $k\text{NN}_r$-based models outperform respective *k*NN-based models). We observe that merging neighborhoods from both datastores also provides some small perplexity gains. Overall, our combined adaptation approach (last row of Table 5) produces an average of 17.1% and 16% perplexity improvement over the parametric adaptation $\mathbf{LM}_a$ and semi-parametric baselines **(a)*k*NN-LM** respectively.

Pretraining-datastore under low-resource adaptation. We analyze the impact on Recall when combining neighbors from the pretraining and adaptation datastores in a low-resource adaptation setting (which is a common scenario). We utilize the Xsum dataset (containing nearly 100M training tokens), to analyze the impact of merging retrieved neighborhoods for different sizes of the adaptation datastore. In Figure 3-a), we observe that the Recall of retrieved neighbors significantly decreases as the adaptation datastore size decreases (green, $\mathcal{K}_a$). However, the merged neighborhood Recall enjoys a relatively flat curve (blue, $\mathcal{K}_a \cup \mathcal{K}_w$). This suggests that the pretraining datastore acts as an important buffer in maintaining high-quality neighborhoods for low-resource adaptation scenarios. A complementary study to consider for the low-resource setting is the impact of the size of the pretraining datastore on the merged retrieval Recall. In this set of experiments, we fix the size of the adaptation datastore to be 100K. From Figure 3-b), we observe that Recall monotonically increases with the size of the pretraining datastore and may continue to improve even after the pretraining datastore exceeds 1 billion tokens. Thus, scaling the pretraining datastore can lead to improved retrieval quality on downstream domains.

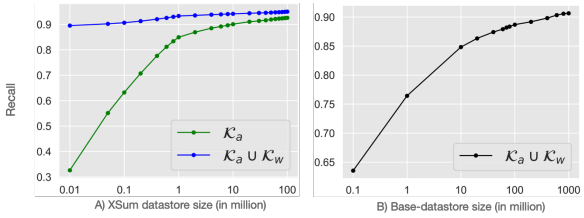

Figure 3: a) Change in Recall of merged neighborhoods compared against the size of the adaptation datastore; b) change in Recall of merged neighborhoods compared against the size of the pretraining datastore.

Which LM representations are better for datastore construction? An important question to consider, is which representations from GPT-2 are most useful in constructing the datastore. To investigate this, we experiment with using different layers from GPT-2 in constructing a Wikipedia-based datastore. To increase the throughput experimentation, we use a smaller-sized datastore of size 10 million. We consider the output of the penultimate Transformer block as well as the following layers from the last Transformer block in our analysis:

	Configuration						Perplexity (↓)						
	Setting	**LM**	**LMa**	**(w)KNN**	**(a)KNN**	**rescore**	**XSum**	**SciQ**	**arXiv**	**BookSum**	**SAMSum**	**XL-Sum**	**GovReport**
Baseline	LM (only)	✓					22.45	22.15	56.83	21.15	46.86	24.87	19.32
	LM_a (only)		✓				18.95	16.09	24.97	20.45	32.26	21.84	14.72
	(w)*k*NN (only)			✓			83.96	46.86	513.45	64.92	298.08	100.92	83.62
	(a)*k*NN (only)				✓		38.38	57.82	87.58	109.87	229.89	47.75	40.39
Baseline (2019)	(w)*k*NN-LM	✓		✓			21.64	19.19	53.03	20.50	46.27	24.03	18.99
	(a)*k*NN-LM	✓			✓		17.01	14.71	24.38	20.60	39.99	19.39	14.87
Ours	(w)*k*NN-LM_a		✓	✓			18.42	14.62	24.42	19.72	31.94	21.22	14.47
	(w)*k*NN_r-LM	✓		✓		✓	21.32	18.5	51.89	20.09	46.20	23.68	18.81
	(w)*k*NN_r-LM_a	✓	✓	✓		✓	18.23	14.22	24.19	19.35	31.92	20.98	14.36
	(a)*k*NN-LM_a		✓		✓		15.30	12.88	17.81	20.22	31.48	18.12	13.08
	(a)*k*NN_r-LM_a		✓		✓	✓	14.85	12.72	17.49	20.09	31.47	17.72	12.87
	(w+a)*k*NN-LM_a		✓	✓	✓		15.20	12.15	17.85	19.72	31.20	17.99	13.01
	(w+a)*k*NN_r-LM_a		✓	✓	✓	✓	**14.71**	**11.95**	**17.47**	**19.42**	**31.18**	**17.53**	**12.79**

Table 5: Performance of different *k*NN-LM configurations. (w)*k*NN, (a)*k*NN, and (a+w)*k*NN denote neighborhood search from pretraining datastore, adaptation datastore, and equal contribution from both, respectively. LM and LM_a denote standard GPT-2 and domain-adapted GPT-2.

first layer norm (LN1), output of Multi-Headed Attention (MHA), second layer norm (LN2), output of final feed-forward layer (FFN). Thus, each datastore differs only in its key vector representations $\vec{c}_q$ for a given context c_q. The *k*NN-LM probability is computed as per Equation (4) where k is set to 1000 and λ is a hyperparameter tuned via grid search in $\lambda \in \{0.01, 0.02, 0.04, \ldots, 0.98\}$. Evaluation is performed on 100K test tokens obtained from unseen Wikipedia documents.

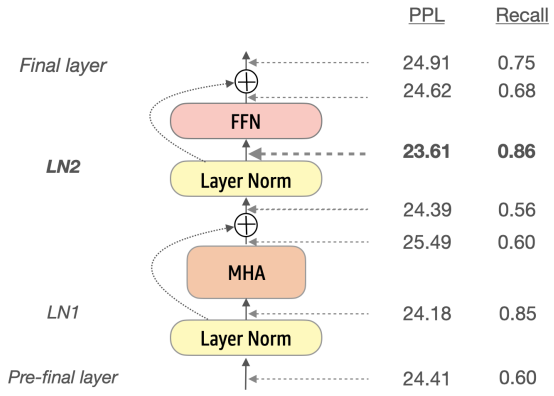

Figure 4: The figure shows the internals of GPT-2 final layer. Perplexity (PPL) and Recall score of different context vector ($\vec{c}_q$) candidate representations for datastore construction.

As shown in Figure 4, we observe that using the output of the LN2 layer creates the best representation space for the datastore keys and produces the best test perplexity of 23.61 and highest Recall of 0.86. We also observe that the best λ returned for an LN2-based *k*NN-LM is 0.1, which is the highest among context representation candidates considered.

Computational cost. We compare our computational overhead with respect to the standard *k*NN-LM proposed by Khandelwal et al. (2019). During inference, an Adapter increases the inference time of GPT-2 by about 1.2 milliseconds per token. The Rescorer takes about 60 milliseconds per token to score 1000 neighbors. We run the parametric model on a single GPU[10] *k*NN and the Rescorer on CPU.

6 Related work

Our proposed work investigates the intersection of techniques used for parametric Language Model adaptation with semi-parametric systems (*k*NN-LMs). Therefore we discuss the related works in each of these areas and contrast our respective contributions.

Parametric LM Adaptation Popularization of Large-Scale Pretrained Language Models (PLMs) has necessitated research into parameter-efficient adaptation methods, to avoid maintaining large models for each domain. Many parameter-efficient methods keep the pretrained LM parameters frozen and learn additional layers during adaptation (Houlsby et al., 2019; Ben-Zaken et al., 2022), or modify the parameters of existing layers (Hu et al., 2022; Hou et al., 2022). This work explores how applying such techniques (namely Adapters) can improve the semi-parametric LM adaptation performance.

Semi-Parametric KNN-LMs Previous works have motivated that scaling the datastore for large-scale retrieval acts as a complimentary path to scaling data used for LM training (Khandelwal et al., 2019; Borgeaud et al., 2022; Khandelwal et al., 2021). However, adaptation approaches of these

[10]Tesla V100-SXM2-16GB

semi-parametric systems beyond zero-shot evaluation (Khandelwal et al., 2019; Khandelwal et al., 2021) have not been explored up until this work.

To improve the quality of retrieval-enhanced methods, neighborhood Rescorer techniques have been employed for other domains such as Q&A (Glass et al., 2022) and information retrieval (Nogueira and Cho, 2019). In contrast, this work explores applications of Rescorer techniques for the Language Modeling task and considers them for lightweight adaptation of semi-parametric LMs.

7 Conclusion

We proposed a multi-pronged strategy for adapting kNN-LM systems. Through our studies, we demonstrated that a general-purpose pretraining datastore contains relevant information, which can be utilized for downstream domains. We showed that parametric and non-parametric adaptation methods complement each other and that using the complete semi-parametric adaptation strategy outperforms adapting just one of the kNN-LM components. Our methods could further be extended by noting that the Recall of retrieved neighborhoods is often imperfect. Thus, a gate could be learned to predict whether kNN retrieval should be triggered. While our study focused on the Language Modeling task, our approach could be applied towards other NLP tasks such as text generation and translation.

8 Acknowledgement

The authors express their gratitude to Kyu Han and Shiva Sundaram for their continuous support throughout this work. They are also appreciative of Omid Sadjadi, Sundararajan Srinivasan, and Zejiang Hou for providing valuable feedback on the preliminary draft.

References

Elad Ben-Zaken, Shauli Ravfogel, and Yoav Goldberg. 2022. Bitfit: Simple parameter-efficient fine-tuning for transformer-based masked language-models. *ArXiv*, abs/2106.10199.

Yoshua Bengio, Réjean Ducharme, and Pascal Vincent. 2000. A neural probabilistic language model. *Advances in neural information processing systems*, 13.

Sebastian Borgeaud, Arthur Mensch, Jordan Hoffmann, Trevor Cai, Eliza Rutherford, Katie Millican, George van den Driessche, Jean-Baptiste Lespiau, Bogdan Damoc, Aidan Clark, Diego de Las Casas, Aurelia Guy, Jacob Menick, Roman Ring, T. W. Hennigan, Saffron Huang, Lorenzo Maggiore, Chris Jones, Albin Cassirer, Andy Brock, Michela Paganini, Geoffrey Irving, Oriol Vinyals, Simon Osindero, Karen Simonyan, Jack W. Rae, Erich Elsen, and L. Sifre. 2022. Improving language models by retrieving from trillions of tokens. In *ICML*.

Arman Cohan, Franck Dernoncourt, Doo Soon Kim, Trung Bui, Seokhwan Kim, Walter Chang, and Nazli Goharian. 2018. A discourse-aware attention model for abstractive summarization of long documents. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 615–621, New Orleans, Louisiana. Association for Computational Linguistics.

Wikimedia Foundation. Wikimedia downloads.

Michael R. Glass, Gaetano Rossiello, Md. Faisal Mahbub Chowdhury, Ankita Rajaram Naik, Pengshan Cai, and A. Gliozzo. 2022. Re2g: Retrieve, rerank, generate. *ArXiv*, abs/2207.06300.

Bogdan Gliwa, Iwona Mochol, Maciej Biesek, and Aleksander Wawer. 2019. SAMSum corpus: A human-annotated dialogue dataset for abstractive summarization. In *Proceedings of the 2nd Workshop on New Frontiers in Summarization*, pages 70–79, Hong Kong, China. Association for Computational Linguistics.

Tahmid Hasan, Abhik Bhattacharjee, Md. Saiful Islam, Kazi Mubasshir, Yuan-Fang Li, Yong-Bin Kang, M. Sohel Rahman, and Rifat Shahriyar. 2021. XL-sum: Large-scale multilingual abstractive summarization for 44 languages. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pages 4693–4703, Online. Association for Computational Linguistics.

Junxian He, Graham Neubig, and Taylor Berg-Kirkpatrick. 2021. Efficient nearest neighbor language models. *arXiv preprint arXiv:2109.04212*.

Zejiang Hou, Julian Salazar, and George Polovets. 2022. Meta-learning the difference: Preparing large language models for efficient adaptation. *ArXiv*, abs/2207.03509.

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

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, and Weizhu Chen. 2022. Lora: Low-rank adaptation of large language models. *ArXiv*, abs/2106.09685.

Luyang Huang, Shuyang Cao, Nikolaus Parulian, Heng Ji, and Lu Wang. 2021. Efficient attentions for long document summarization.

Matt Gardner Johannes Welbl, Nelson F. Liu. 2017. Crowdsourcing multiple choice science questions.

Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2019. Billion-scale similarity search with gpus. *IEEE Transactions on Big Data*, 7(3):535–547.

Urvashi Khandelwal, Angela Fan, Dan Jurafsky, Luke Zettlemoyer, and Mike Lewis. 2021. Nearest neighbor machine translation. *ArXiv*, abs/2010.00710.

Urvashi Khandelwal, Omer Levy, Dan Jurafsky, Luke Zettlemoyer, and Mike Lewis. 2019. Generalization through memorization: Nearest neighbor language models. *arXiv preprint arXiv:1911.00172*.

Wojciech Kryscinski, Nazneen Fatema Rajani, Divyansh Agarwal, Caiming Xiong, and Dragomir R Radev. 2021. Booksum: A collection of datasets for long-form narrative summarization.

Haokun Liu, Derek Tam, Mohammed Muqeeth, Jay Mohta, Tenghao Huang, Mohit Bansal, and Colin Raffel. 2022. Few-shot parameter-efficient fine-tuning is better and cheaper than in-context learning. *ArXiv*, abs/2205.05638.

Tomas Mikolov, Martin Karafiát, Lukas Burget, Jan Cernockỳ, and Sanjeev Khudanpur. 2010. Recurrent neural network based language model. In *Interspeech*, volume 2, pages 1045–1048. Makuhari.

Shashi Narayan, Shay B. Cohen, and Mirella Lapata. 2018. Don't give me the details, just the summary! topic-aware convolutional neural networks for extreme summarization. *ArXiv*, abs/1808.08745.

Rodrigo Nogueira and Kyunghyun Cho. 2019. Passage re-ranking with bert. *ArXiv*, abs/1901.04085.

Aaron van den Oord, Yazhe Li, and Oriol Vinyals. 2018. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*.

Jonas Pfeiffer, Andreas Rücklé, Clifton Poth, Aishwarya Kamath, Ivan Vulić, Sebastian Ruder, Kyunghyun Cho, and Iryna Gurevych. 2020. Adapterhub: A framework for adapting transformers. *arXiv preprint arXiv:2007.07779*.

Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. 2018. Improving language understanding by generative pre-training.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, and Jamie Brew. 2019. Huggingface's transformers: State-of-the-art natural language processing. *ArXiv*, abs/1910.03771.

Dani Yogatama, Cyprien de Masson d'Autume, and Lingpeng Kong. 2021. Adaptive semiparametric language models. *Transactions of the Association for Computational Linguistics*, 9:362–373.

ACL 2023 Responsible NLP Checklist

A For every submission:

☐ A1. Did you describe the limitations of your work?
Left blank.

☐ A2. Did you discuss any potential risks of your work?
Left blank.

☐ A3. Do the abstract and introduction summarize the paper's main claims?
Left blank.

☐ A4. Have you used AI writing assistants when working on this paper?
Left blank.

B ☐ Did you use or create scientific artifacts?

Left blank.

☐ B1. Did you cite the creators of artifacts you used?
Left blank.

☐ B2. Did you discuss the license or terms for use and / or distribution of any artifacts?
Left blank.

☐ B3. Did you discuss if your use of existing artifact(s) was consistent with their intended use, provided that it was specified? For the artifacts you create, do you specify intended use and whether that is compatible with the original access conditions (in particular, derivatives of data accessed for research purposes should not be used outside of research contexts)?
Left blank.

☐ B4. Did you discuss the steps taken to check whether the data that was collected / used contains any information that names or uniquely identifies individual people or offensive content, and the steps taken to protect / anonymize it?
Left blank.

☐ B5. Did you provide documentation of the artifacts, e.g., coverage of domains, languages, and linguistic phenomena, demographic groups represented, etc.?
Left blank.

☐ B6. Did you report relevant statistics like the number of examples, details of train / test / dev splits, etc. for the data that you used / created? Even for commonly-used benchmark datasets, include the number of examples in train / validation / test splits, as these provide necessary context for a reader to understand experimental results. For example, small differences in accuracy on large test sets may be significant, while on small test sets they may not be.
Left blank.

C ☐ Did you run computational experiments?

Left blank.

☐ C1. Did you report the number of parameters in the models used, the total computational budget (e.g., GPU hours), and computing infrastructure used?
Left blank.

The Responsible NLP Checklist used at ACL 2023 is adopted from NAACL 2022, with the addition of a question on AI writing assistance.

☐ C2. Did you discuss the experimental setup, including hyperparameter search and best-found hyperparameter values?
Left blank.

☐ C3. Did you report descriptive statistics about your results (e.g., error bars around results, summary statistics from sets of experiments), and is it transparent whether you are reporting the max, mean, etc. or just a single run?
Left blank.

☐ C4. If you used existing packages (e.g., for preprocessing, for normalization, or for evaluation), did you report the implementation, model, and parameter settings used (e.g., NLTK, Spacy, ROUGE, etc.)?
Left blank.

D ☐ Did you use human annotators (e.g., crowdworkers) or research with human participants?

Left blank.

☐ D1. Did you report the full text of instructions given to participants, including e.g., screenshots, disclaimers of any risks to participants or annotators, etc.?
Left blank.

☐ D2. Did you report information about how you recruited (e.g., crowdsourcing platform, students) and paid participants, and discuss if such payment is adequate given the participants' demographic (e.g., country of residence)?
Left blank.

☐ D3. Did you discuss whether and how consent was obtained from people whose data you're using/curating? For example, if you collected data via crowdsourcing, did your instructions to crowdworkers explain how the data would be used?
Left blank.

☐ D4. Was the data collection protocol approved (or determined exempt) by an ethics review board?
Left blank.

☐ D5. Did you report the basic demographic and geographic characteristics of the annotator population that is the source of the data?
Left blank.