Attribute Injection for Pretrained Language Models: A New Benchmark and An Efficient Method

Reinald Kim Amplayo*
University of Edinburgh
reinald.kim@ed.ac.uk

Kang Min Yoo **Sang-Woo Lee**
NAVER AI Lab, NAVER Clova AI
{kangmin.yoo,sang.woo.lee}@navercorp.com

Abstract

Metadata attributes (e.g., user and product IDs from reviews) can be incorporated as additional inputs to neural-based NLP models, by expanding the architecture of the models to improve performance. However, recent models rely on pretrained language models (PLMs), in which previously used techniques for attribute injection are either nontrivial or cost-ineffective. In this paper, we introduce a benchmark for evaluating attribute injection models, which comprises eight datasets across a diverse range of tasks and domains and six synthetically sparsified ones. We also propose a lightweight and memory-efficient method to inject attributes into PLMs. We extend *adapters*, i.e. tiny plug-in feed-forward modules, to include attributes both independently of or jointly with the text. We use approximation techniques to parameterize the model efficiently for domains with large attribute vocabularies, and training mechanisms to handle multi-labeled and sparse attributes. Extensive experiments and analyses show that our method outperforms previous attribute injection methods and achieves state-of-the-art performance on all datasets.

1 Introduction

Neural-based NLP models are powered by large-scale textual datasets, which are mostly crawled from the web (Denoyer and Gallinari, 2006; Sandhaus, 2008; Zhu et al., 2015; Ni et al., 2019; Raffel et al., 2020). Web texts usually are attached with *metadata*, i.e. attributes that describe the texts. For example, product reviews have user and product IDs, as well as their ratings, while research papers on arXiv have author lists and research areas as metadata attributes (see Figure 1). While most of the recent models disregard them and focus more on ungrounded language understanding (understanding language on its own, e.g., GLUE; Wang et al., 2018, *inter alia*), prior work has shown that incorporating these attributes into our model increases not just its performance but also its interpretability and customizability (Tang et al., 2015; Chen et al., 2016; Kim et al., 2019). This work explores the task of *attribute injection* (Amplayo, 2019), which aims to use attributes to improve the performance of NLP models effectively.

Yelp Review
Text: My boyfriend's fav. place and the stein of beers are priced pretty good. Game nights get super packed so go early to save a seat. Kitchen closes at midnight which is too early when your buzz kicks in around 1am.
Attributes:
– User: `n6LeAoIuDR3NfIBEsmL_zg`
– Product: `7TMf1NuuAdvhG7IojZSKnw`
Paper Abstract
Text: We present new and improved fixed-parameter algorithms for computing maximum agreement forests (MAFs) of pairs of rooted binary phylogenetic trees. The size of such a forest for two trees corresponds to their subtree prune-and-regraft distance and, if the agreement forest is acyclic, to their hybridization number ...
Attributes:
– Authors: `Chris Whidden`, `Robert G. Beiko`, `Norbert Zeh`
– Research Areas: `q-bio.PE`, `cs.DS`

Figure 1: Examples of a Yelp review and an arXiv paper abstract and their corresponding attributes. Texts in `typewriter` font are attribute labels.

Previous methods for attribute injection (Tang et al., 2015; Zhu et al., 2015; Chen et al., 2016; Ma et al., 2017; Dou, 2017; Amplayo et al., 2018; Wu et al., 2018; Long et al., 2018; Kim et al., 2019; Amplayo, 2019) involve two steps: (1) designing an architecture that accepts both texts and attributes, and (2) training the model from scratch using task-specific datasets. However, these methods of modifying different modules of the model can be non-trivial when applied to pretrained language models (PLMs; Devlin et al., 2019; Liu et al., 2019; Qiu et al., 2020). The use of PLMs disallows designing new and specialized architectures. Zhang et al.

* Work done while Reinald was at NAVER AI Lab. He is now at Google Research.

Proceedings of the 29th International Conference on Computational Linguistics, pages 1051–1064
October 12–17, 2022.

(2021b) append large and deep layers of attribute-specific Transformers to the end of PLMs, which cannot leverage the potential of attributes in the intermediate layers of PLMs and also scales poorly to multiple tasks and domains. Finally, more recent work on language model customization and controllability makes use of textual prompts (Brown et al., 2020; Schick and Schütze, 2021), specialized tokens (Fan et al., 2018; Keskar et al., 2019), and additional neural modules (Perez et al., 2018; Wang et al., 2019; Lauscher et al., 2020; Liu et al., 2021) to introduce additional contexts, such as style, topic, and end task. Unfortunately, these techniques do not generalize to all kinds of attributes, such as those that are non-textual (e.g., user IDs that are not text-translatable), multi-labeled (e.g., multiple authors of a paper), and with large vocabularies (e.g., thousands of products available).

Our contribution in this paper is two-fold. Firstly, we introduce a new benchmark to evaluate models for attribute injection. The benchmark consists of eight datasets which include three newly collated ones from different tasks and domains, and six synthetically sparsified datasets which are specifically created to evaluate models in sparse settings. These datasets span from diverse tasks, such as sentiment classification, spoiler detection, and message type classification, and contain attributes that have different properties (sparse vs. non-sparse, single-labeled vs. multi-labeled, etc.). Experiments on the benchmark show that our method outperforms previous approaches, as well as competitive baselines that fully fine-tune the pretrained language model. We also conduct extensive analyses to show that our method is robust to sparse and cold-start attributes and that it is modular with attribute-specific modules transferrable to other tasks using the same attributes.

Secondly, we propose a lightweight and memory-efficient method that is specifically suited to inject attributes into PLMs, which can be non-textual, multi-labeled, and have large vocabularies. We make use of adapters (Houlsby et al., 2019), i.e. feed-forward modules inserted between layers of PLMs that are tiny in size, and extend them such that attributes are injected as additional inputs to the model. We introduce two kinds of injection methods, which either incorporate attributes independently of or jointly with the text representation. A naive implementation of the latter would substantially increase the parameters, especially when the attribute vocabulary is large, thus we use ideas from low-rank matrix approximations as well as parameterized hypercomplex multiplications (Zhang et al., 2021a; Mahabadi et al., 2021) to significantly decrease the fine-tuned parameters by up to 192× for a default base-sized BERT (Devlin et al., 2019) setting. We also use two mechanisms, attribute dropout and post-aggregation, to handle attribute sparsity and multi-labeled attributes, respectively. Our use of adapters enables us to parameter-efficiently train our model, i.e. by freezing pretrained weights and only updating new parameters at training time. We make our code and datasets publicly available.[1]

2 Related Work

Prior to the neural network and deep learning era, traditional methods for NLP have relied on feature sets as input to machine learning models. These feature sets include metadata attributes such as author lists and publication venue of research papers (Rosen-Zvi et al., 2004; Joorabchi and Mahdi, 2011; Kim et al., 2017), topics of sentences (Ramage et al., 2009; Liu and Forss, 2014; Zhao and Mao, 2017), as well as spatial (Yang et al., 2017) and temporal (Fukuhara et al., 2007) metadata attributes found in tweets. Attributes are mostly used in the area of sentiment classification (Gao et al., 2013), where most of the time textual data includes freely available user and product attributes. These methods rely on manually curated features that would represent the semantics of user and product information.

Deep neural networks gave rise to better representation learning (Bengio et al., 2013; Mikolov et al., 2013), which allows us to learn from scratch semantic representation of attributes in the form of dense vectors (Tang et al., 2015). The design of how to represent attributes has evolved from using attribute-specific word and document embeddings (Tang et al., 2015) and attention pooling weights (Chen et al., 2016; Ma et al., 2017; Amplayo et al., 2018; Wu et al., 2018), to more complicated architectures such as memory networks (Dou, 2017; Long et al., 2018) and importance matrices (Amplayo, 2019). These designs are model- and domain-dependent and can be non-trivial to apply to other models and datasets. Our proposed method, on the other hand, works well on any pretrained language model which are mostly based on Trans-

[1] https://github.com/rktamplayo/Injector

former (Vaswani et al., 2017; Devlin et al., 2019). Zhang et al. (2021b) used six layers of attribute-injected Transformers where attributes are used as input to the self-attention module, which is costly in terms of memory, i.e., equivalent to adding 50% of the original BERT-base parameters. Attributes are useful in the intermediate layers when learning the semantics of the input text, which this model cannot leverage.

Our work is closely related to recent literature on controlled text generation, where most of the work use either specialized control tokens concatenated with the input text (Sennrich et al., 2016; Kikuchi et al., 2016; Ficler and Goldberg, 2017; Fan et al., 2018; Keskar et al., 2019), or textual prompts that instruct the model on what to generate (Brown et al., 2020; Schick and Schütze, 2021; Gao et al., 2021; Zhao et al., 2021). While these methods have been successfully applied to pretrained language models, the attributes used to control the text are limited to those that are text-translatable (e.g., topics such as "technology") and those with limited vocabulary (e.g., "positive" or "negative" sentiment).

Methods to efficiently fine-tune pretrained models have been explored since their inception (Dathathri et al., 2019; Li and Liang, 2021; He et al., 2022). One of the more popular methods is the use of adapters (Houlsby et al., 2019), where tiny trainable feed-forward modules are inserted and pretrained weights are frozen during training. Most of the prior work on adapters focuses on either improving their effectiveness and efficiency (Mao et al., 2021; Mahabadi et al., 2021; He et al., 2022) or applying it to domain adaptation and transfer learning tasks (Pfeiffer et al., 2021; He et al., 2021; Cooper Stickland et al., 2021). Our work extends the use of adapters to additionally accept attributes as input, which is closely related to Perez et al. (2018) and Lauscher et al. (2020), where the adapter accepts textual questions and commonsense knowledge as input, respectively. However, unlike previous work where the additional input can be transformed into natural language tokens, attributes are usually not the case (e.g., see user and product IDs in Figure 1), which poses new challenges in efficiency when applied to adapters.

3 Attribute Injection Benchmark

Problem Setting Let $x = \{x_i\}_{i=1}^{N}$ denote the input text of N tokens, y is a task-specific output, and $p(y|x)$ is a discriminative model that predicts y given x. Suppose there exists a set of non-textual and categorical attributes $z = \{z_j\}_{j=1}^{M}$ that describe text x (e.g., user and product IDs of product reviews). These attributes can be multi-labeled, i.e. $z_j = [z_j^{(k)}]$ (e.g., multiple authors of a research paper) and use a finite yet possibly large vocabulary $\mathcal{Z}_j$, i.e. $z_j \subseteq \mathcal{Z}_j$. The task of attribute injection aims to build a model $q(y|x, z)$ that additionally incorporates z as input such that the gain in task performance between p and q is maximized. In our setting, p is a pretrained language model (PLM) fine-tuned to the task, while q is a PLM that also takes z as additional input.

The Datasets To evaluate attribute injection models on a wide variety of tasks and datasets from different domains, we introduce ATTRIB, a collection of benchmark datasets to evaluate attribute injection methods. ATTRIB consists of a total of 14 datasets. The first eight datasets are collected from different domains with a diverse set of tasks spanning from sentiment classification to spoiler detection. Three of the eight datasets are newly introduced in this paper. Table 1 reports the dataset statistics, which shows the different characteristics (i.e., size of training data, input text length, number of attributes, sparsity, and multi-labelity of attributes) of each dataset.

The first three datasets (**Y2013**, **Y2014**, and **IMDB**; Tang et al., 2015) are sentiment classification datasets with user and product attributes from two different sources (Yelp and IMDB). **AAPR** (Yang et al., 2018) is a dataset for classifying whether an arXiv paper is accepted to a conference or not, with two attributes, lists of authors and research areas. **POLMED** (Kim et al., 2019) is a classification dataset in the political domain, where the goal is to classify the message type of a tweet, with four attributes, politician, media source, audience, and political bias.

We also introduce three new attribute injection datasets (see Appendix for details on dataset collection). **FOOD** is a dataset in which given a recipe from Food.com and three attributes, user and lists of ingredients and tags, we are tasked to predict the estimated number of minutes it takes to make the food, rounded down to the tens. **GOOD** is a spoiler detection dataset where we classify whether a book review from Goodreads contains a spoiler or not, with three attributes: user, book, and rating. And finally, **BEER** is a multi-aspect rating predic-

Dataset	Y2013	Y2014	IMDB	AAPR	POLMED	FOOD	GOOD	BEER
#Train	62.5K	183.0K	67.4K	33.5K	4.5K	162.4K	714.7K	1.5M
#Dev	7.8K	22.7K	8.4K	2.0K	–	20.3K	10.0K	10.0K
#Test	8.7K	25.4K	9.1K	2.0K	0.5K	20.3K	10.0K	10.0K
#Words/Input	210(166)	218(175)	425(278)	97(36)	38(62)	101(65)	132(72)	133(56)
#Classes	5	5	10	2	9	16	2	4×9
#Attributes	2	2	2	2	4	3	3	3
#Attr. Vocab	3.3K	9.0K	2.9K	51.6K	0.5K	40.5K	43.8K	98.0K
%Sparse	0.0%	0.0%	0.0%	97.8%	63.8%	80.0%	34.4%	75.3%
Multi-label?				✓		✓		

Table 1: ATTRIB datasets statistics. The second block reports new datasets introduced in this paper. BEER is a multi-task dataset, with nine classes for each of four given aspects. %Sparse is the percentage of attributes with less than 10 training examples. Multi-label attributes include lists of authors and research areas for AAPR, and lists of ingredients and tags for FOOD. The kinds of attributes available and further analyses can be found in the Appendix.

tion dataset in which given a review and three attributes, user, beer, and overall rating, we are tasked to predict the ratings of four *aspects*, i.e. properties that influence user satisfaction: appearance, aroma, palate, and taste.

Additionally, we collated six synthetically sparsified datasets that were created specifically to evaluate attribute injection methods in cold-start environments[2]. Specifically, Amplayo et al. (2018) downsampled the Y2013 and IMDB training datasets such that the attributes are sparser than the original. There are three levels of sparsity: 20%, 50%, and 80%, where $x\%$ means that $x\%$ of attributes are cold-start and thus are unseen in the training examples. Statistics of these datasets are shown in the Appendix.

4 Modeling Approach

Our proposed method, which we call INJECTORS, can be summarized as follows. We extend adapters (Houlsby et al., 2019), which are tiny feed-forward neural networks plugged into pretrained language models, such that they also accept attributes z as input. Attributes z can be represented as additional bias parameters or as perturbations to the weight matrix parameter of the adapter, motivated by how attributes are used to classify texts. We decrease the number of parameters exponentially using low-rank matrix approximations and parameterized hypercomplex multiplications (Zhang et al., 2021a). Finally, we introduce two training mechanisms, attribute dropout and post-aggregation, to mitigate problems regarding attribute sparsity and multi-label properties. Figure 2 illustrates an overview of INJECTORS.

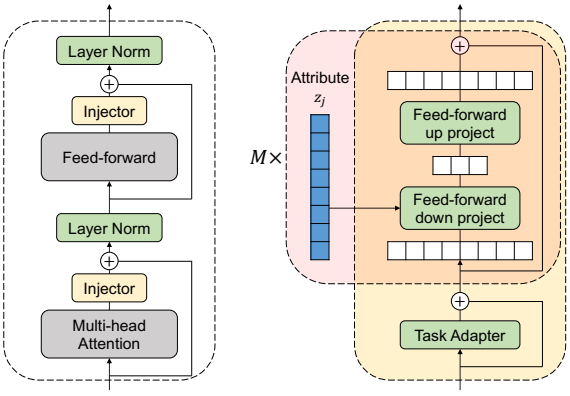

Figure 2: Architecture of the INJECTOR module when integrated into one block of a Transformer model (see left of figure). INJECTOR starts with a task-specific adapter, followed by M attribute-specific adapters, one for each attribute given in the task (see right of figure). Green-colored modules are trained and fine-tuned, while gray-colored modules are fixed.

Preliminary: Adapters We first briefly describe adapters. Let $\mathbf{h} \in \mathbb{R}^{d_h}$ be the output hidden vector from a multi-head attention or feed-forward layer in a Transformer block. An adapter layer is basically two feed-forward networks that projects $\mathbf{h}$ into vector $\mathbf{h}' \in \mathbb{R}^{d_h}$ with a much smaller dimension $d_a \ll d_h$:

$$\begin{aligned}\mathbf{h}' &= \text{Adapt}(\mathbf{h}) \\ &= \text{FFNet}_{up}(f(\text{FFNet}_{down}(\mathbf{h}))) + \mathbf{h} \end{aligned} \quad (1)$$

where $\text{FFNet}(\mathbf{x}) = \mathbf{W}\mathbf{x} + \mathbf{b}$, $\mathbf{W}$ and $\mathbf{b}$ are learned weight and bias parameters of FFNet, $f(\cdot)$ is a non-linear function, and the addition represents a residual layer.

[2]In this paper, we distinguish the definition of *sparsity* and *cold-start* attributes as the former referring to attributes with *few* training examples, and the latter referring to those with *zero* training examples.

Task-specific Adapter INJECTORS start with a task-specific adapter that uses Equation 1 to transform the previous hidden vector $\mathbf{h}$ to $\mathbf{h}'$. The use of a separate task-specific adapter is essential to make our method modularizable and learned attributes on one task transferrable to another. We show extensive analyses of the modularity of our method in the later sections.

Attribute-specific Adapters Attributes z are injected through attribute-specific adapters, where they are used in two different ways. Firstly, they are used as *bias* parameters independent of the text representation. This is motivated by the fact that attributes can have a prior disposition regardless of what is written in the text. For example, a user may tend to give lower review ratings than average. Secondly, they are also used as *weight* parameters. This allows our method to jointly model attributes with the text representation. This is motivated by how attributes can change the semantics of the text. For example, one user may like very sweet food while another user may dislike it, thus the use of the word *sweet* in the text may mean differently to them.

More formally, for each attribute $z_j \in z$, we obtain its embedding $\mathbf{z}_j$ from a learned embedding matrix and sequentially transform the previously attribute-injected vector $\mathbf{h}_{z_{j-1}}$ to attribute-injected vector $\mathbf{h}_{z_j}$ using the following equation:

$$\begin{aligned}\mathbf{h}_{z_j} &= \text{AttrAdapt}(\mathbf{h}_{z_{j-1}}, \mathbf{z}_j) && (2)\\ &= \text{FFNet}_{up}(f(\mathbf{W}_{z_j}\mathbf{h}_{z_{j-1}} + \mathbf{b}_{z_j})) \\ &\quad + \mathbf{h}_{z_{j-1}} && (3)\end{aligned}$$

where $\mathbf{h}_{z_0} = \mathbf{h}'$ from the output of task-specific adapter. Unlike standard adapters, the attribute-specific weight matrix $\mathbf{W}_{z_j} \in \mathbb{R}^{d_h \times d_a}$ and bias parameter $\mathbf{b}_{z_j} \in \mathbb{R}^{d_a}$ of the down-project feed-forward network are not learned from scratch, but instead are calculated as follows.

The calculation of the bias parameter $\mathbf{b}_{z_j}$ is trivial; we perform a linear transformation of the attribute embedding $\mathbf{z}_j$:

$$\mathbf{b}_{z_j} = g_{bias}(\mathbf{z}_j) + \mathbf{c}_{bias} \quad (4)$$

where $g_{bias} \in \mathbb{R}^{d_z} \mapsto \mathbb{R}^{d_a}$ is a linear projection, $\mathbf{c}_{bias} \in \mathbb{R}^{d_a}$ is a learned vector, and d_z is the attribute embedding size.

We also define $\mathbf{W}_{z_j}$ as:

$$\mathbf{W}_{z_j} = g_{weight}(\mathbf{z}_j) + \mathbf{C}_{weight} \quad (5)$$

where $\mathbf{C}_{weight} \in \mathbb{R}^{d_h \times d_a}$ is a learned matrix. The function g_{weight}, however, cannot be defined similarly as a linear projection. This would require a tensor parameter of size $d_z \times d_h \times d_a$ to linearly project $\mathbf{z}_j$ to $\mathbf{W}_j$. Considering the fact that we may have multiple attributes for each domain, the number of parameters would not scale well and makes the model very large and difficult to train. Inspired by Mahabadi et al. (2021), we use ideas from low-rank matrix decomposition and parameterized hypercomplex multiplications (PHMs; Zhang et al., 2021a) to substantially decrease the number of parameters. Figure 3 shows an illustrative overview.

More specifically, we first transform attribute embedding $\mathbf{z}_j$ into vectors in hypercomplex space with O dimensions, i.e.:

$$\hat{\mathbf{z}}_j = [\sigma_1(\mathbf{z}_j), ..., \sigma_O(\mathbf{z}_j)] \in \mathbb{H}^{d_z} \quad (6)$$

where $\sigma_o(\cdot) \in \mathbb{R}^{d_z} \mapsto \mathbb{R}^{d_a}$ is a linear projection in the oth dimension. A hypercomplex vector with O dimensions is basically a set of vectors with one real vector and $O-1$ "imaginary" vectors.[3]

For each dimension o, we first define a small rank-one matrix $\mathbf{S}_o \in \mathbb{R}^{d_a \times d_h/O^2}$ as an outer product between $\hat{\mathbf{z}}_{j,o}$ and a learned vector $\mathbf{s}_o \in \mathbb{R}^{d_h/O^2}$:

$$\mathbf{S}_o = \hat{\mathbf{z}}_{j,o}\mathbf{s}_o^\top \quad (7)$$

and then define a large matrix $\hat{\mathbf{W}}_{j,o} \in \mathbb{R}^{d_h \times d_a}$ as the Kronecker product, denoted by $\otimes$ between two matrices $\mathbf{S}_o$ and a learned matrix $\mathbf{A}_o \in \mathbb{R}^{O \times O}$, followed by a reshape and the hyperbolic tangent function:

$$\hat{\mathbf{W}}_o = \text{Reshape}(\tanh(\mathbf{S}_o \otimes \mathbf{A}_o)) \quad (8)$$

Finally, we add the large matrices $\hat{\mathbf{W}}_o$ of each dimension. To sum up, we define $g_{weight}(\mathbf{z}_j)$ as:

$$g_{weight}(\mathbf{z}_j) = \sum_{o=0}^{O} \text{Reshape}(\tanh(\sigma_o(\mathbf{z}_j)\mathbf{s}_o^\top \otimes \mathbf{A}_o)) \quad (9)$$

Low-rank (Eq. 7) and PHMs (Eqs. 8-9) are both necessary to achieve a high performance with decreased parameters. While the low-rank method in itself reduces the most parameters, it also reduces the expressive power of the model since it outputs rank-one matrices. PHMs mitigate this by performing a sum of Kronecker products, increasing the

[3] Following Tay et al. (2019) and Zhang et al. (2021a), we remove the imaginary units of these vectors to easily perform operations on them, thus these vectors are also in the real space.

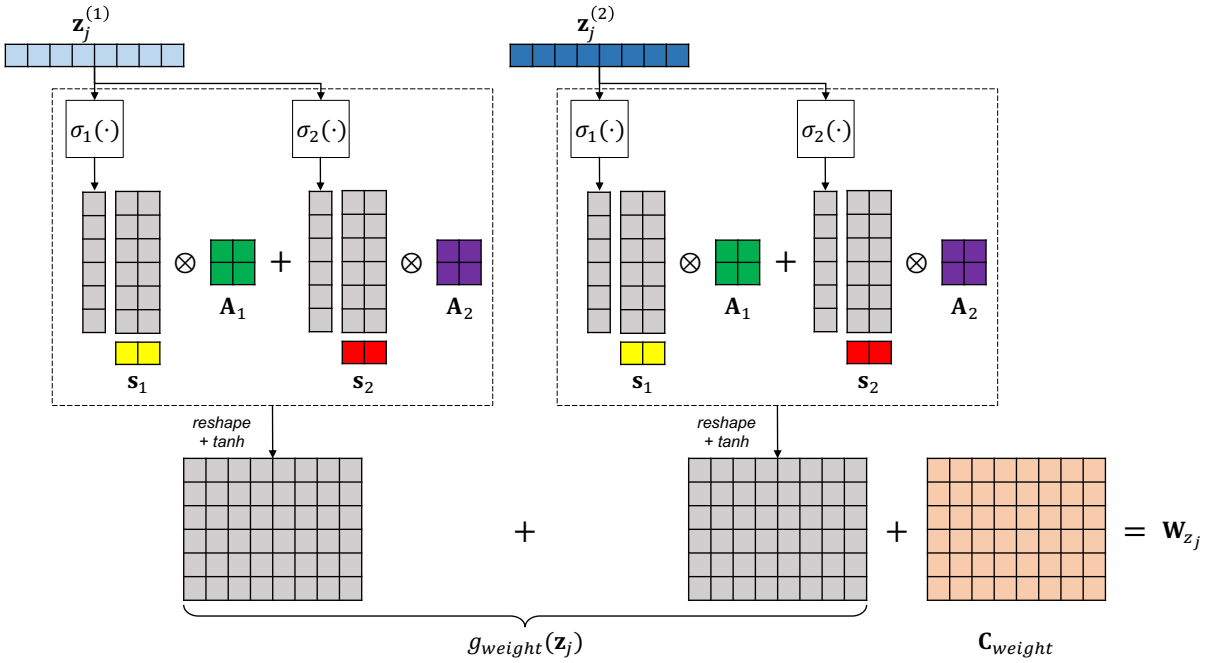

Figure 3: An illustration of how attribute embedding $\mathbf{z}_j$ is transformed into weight matrix $\mathbf{W}_{z_j}$. The colored tensors are learned parameters, while the gray ones are derived. By using a set of tiny parameters $\mathbf{A}_o$ and $\mathbf{s}_o$, we are able to obtain large matrices. When there are multiple labels for attribute z_j, we process them separately and aggregate the resulting large matrices.

rank of the matrix to potentially at most O^2. Finally, this process effectively reduces the number of parameters from $\mathcal{O}(d_z * d_h * d_a)$ to $\mathcal{O}(d_z * d_a)$, since the parameters in σ_o dominate the other parameters (see Appendix for a detailed parameter analysis).

Attribute Dropout and Post-Aggregation For cases where attributes are sparse and multi-labeled, we use the following mechanisms. Firstly, we add a dropout mechanism that randomly masks out attributes from training instances with a rate r_{drop}. This replicates how instances at test time would look, where some attributes are not found in the vocabulary.

Secondly, when there are more than one labels of an attribute, instead of aggregating them first before processing, as in Kim et al. (2019), we perform aggregation post hoc for $\mathbf{W}_{z_j}$), i.e.:

$$\mathbf{b}_{z_j} = \sum\nolimits_k g_{bias}(\mathbf{z}_j^{(k)}) + \mathbf{c}_{bias} \quad (10)$$

$$\mathbf{W}_{z_j} = \sum\nolimits_k g_{weight}(\mathbf{z}_j^{(k)}) + \mathbf{C}_{weight} \quad (11)$$

Aggregating attribute embeddings reduces their individual representation power, while our post-aggregation mechanism preserves this since a sum of non-linear transformations is injective (Xu et al., 2019).

5 Experimental Setup

Training Configuration For the PLMs, we used weights and settings of `bert-base-uncased` (Devlin et al., 2019), available in the HuggingFace library (Wolf et al., 2020). We set the dimensions of all parameters as follows: $d_z = d_h = 768$, $d_a = 64$, and $O = 4$. Using this setting and our parameter-saving method, we are able to decrease the parameters by $192\times$ the naive method. To handle long input texts and fit them into the 512 token limit of PLMs, we truncate them by concatenating the first and last 250 tokens, following Zhang et al. (2021b). We set both the general and attribute dropout rates to 0.2 and the batch size to 8. We used Adam with weight decay (Loshchilov and Hutter, 2019) to optimize our models with a learning rate of $3e-5$ and 200K training steps, with the first 20K steps used to warm-up training linearly.

To train our models, we added a logistic classifier that transforms the `[CLS]` token into logits. The weights here are updated during training. We then used a cross-entropy loss to train the models on all datasets except for Goodreads and Beeradvocate. The Goodreads dataset is very imbalanced towards the negative class (i.e., not a spoiler). We thus put more importance on detecting the spoiler class and used a weighted cross-entropy loss with

0.5 weight on the negative class and 1.0 weight on the positive class. For Beeradvocate, we treat the task as a multi-task problem, where each aspect rating prediction is a separate task. Thus, we used multiple classifiers, one for each aspect, and aggregate the losses from all classifiers by averaging. For PolMed where there is no available development set, we performed a 10-fold cross-validation, following Kim et al. (2019).

Comparison Systems We compared our method with several approaches, including the following no-attribute baselines: (1) **BERT-base ($\mathcal{B}$)**: The base model used in our experiments; and (2) $\mathcal{B}$ **+ ADAPTERS**: Extra tiny parameters are added to the base model and are used for training instead of the full model.

Baselines with attributes injected include the following, where we use the same base model $\mathcal{B}$ for all baselines for ease of comparison: (3) $\mathcal{B}$ **+ TOKENS**: the attributes are used as special control tokens prepended in front of the input. The `[CLS]` token is then passed to the logistic classifier; (4) $\mathcal{B}$ **+ UPA** (Chen et al., 2016): attributes are used as additional bias vectors when calculating the weights of the attention pooling module; (5) $\mathcal{B}$ **+ CHIM** (Amplayo, 2019): attributes are used as importance matrices multiplied to the weight matrix of the logistic classifier; and (6) $\mathcal{B}$ **+ MAA** and $\mathcal{B}$ **+ TINYMAA** (Zhang et al., 2021b): attributes are used to modify the parameters used for calculating query/key/value in the self-attention. To match the parameters of other baselines, we implemented a tiny version with a single Transformer layer and smaller dimensions; and (7-8) $\mathcal{B}$. All models were trained on a single GeForce GTX 1080Ti GPU, except for $\mathcal{B}$ + MAA, in which four GPUs were required to train on the same setting for all datasets.

Finally, we also included in our comparisons a version of our model using the **RoBERTa-base** ($\mathcal{R}$; Liu et al., 2019) configuration ($\mathcal{R}$ **+ INJECTORS**).

6 Results

We evaluated system outputs with accuracy for all datasets except GOOD, where we used F1-score. For brevity in BEER, we took the average of the accuracy of all sub-tasks. Our results are summarized in Table 2. Token-based injection performs worse than the base model on four datasets, which shows that the method is not effective for attribute injection. Overall, among the attribute injected systems, INJECTORS outperforms the other baselines on all datasets, even when BERT parameters are frozen[4]. The difference in performance is especially significant in GOOD, where INJECTORS performs 9.68 points better than the second model. Finally, We also see an increase in performance when applying INJECTORS to RoBERTa, showing that our method can be applied effectively to better-pretrained models. We also conducted ablation studies, detailed in the Appendix, showing the contributions of the different components in the proposed model.

Ablation Studies We present in Table 3 various ablation studies, which assess the contribution of different model components. Our experiments confirm that the use of both bias and weight injection as well as the addition of task adapter improve performance. Interestingly, some datasets prefer one injection type over the other. GOOD dataset, for example, prefers bias injection, i.e., using attributes as prior and independent of the text (e.g., the tendency of the user to write spoilers). Moreover, our training mechanisms also increase the performance of the model. This is especially true for post-aggregation on the FOOD dataset since two of its attributes are multi-labeled (ingredients and tags). Finally, we show that the model variant without one of the parameter-saving methods either performs worse or does not run at all.

On Cold-Start Attributes We checked the performance of the models when trained with synthetic *cold-start attributes* using the Y2013-COLD and IMDB-COLD datasets. We compared the performance of BERT-base, CHIM, TINYMAA, MAA, and INJECTORS. Table 4 shows their performance along with HCSC (Amplayo et al., 2018), which is a hybrid of BiLSTM and CNN with a UPA-style attribute injection method (Chen et al., 2016) and is extended to perform well on cold-start scenarios. As can be seen, our method still performs the best on these datasets, while TINYMAA is consistently worse than the base model when attributes are 80% sparse. MAA improves over TINYMAA with the cost of significant increase in parameters. All BERT-based methods perform better than HCSC on both datasets. Overall, the improvements are the smallest when the dataset is the sparsest (80% sparsified data). This is expected since datasets are smaller and include cold-start attributes.

[4] We did not see an improvement over our final model when fine-tuning *all* parameters of $\mathcal{B}$ + INJECTORS.

Model	Y2013	Y2014	IMDB	AAPR	PolMed	Food	Good	Beer	TP
BERT-base ($\mathcal{B}$)	67.97	68.07	52.45	63.70	41.82	41.89	41.98	50.48	1.00×
$\mathcal{B}$ + Adapters	66.47	67.44	52.69	62.85	44.24	42.02	48.92	50.71	0.02×
$\mathcal{B}$ + Tokens	*67.87*	*67.98*	52.68	64.85	42.63	*41.23*	44.79	*50.25*	1.00×
$\mathcal{B}$ + UPA	68.38	68.82	55.76	64.40	42.83	43.97	43.96	51.98	1.01×
$\mathcal{B}$ + CHIM	68.71	68.56	54.31	65.30	43.64	43.35	43.58	52.29	1.01×
$\mathcal{B}$ + TinyMAA	68.03	68.57	55.76	65.00	44.65	43.63	44.65	54.58	1.01×
$\mathcal{B}$ + MAA (our impl.)	70.15	70.51	56.98	65.15	43.43	43.33	48.10	55.04	1.50×
$\mathcal{B}$ + MAA (original)	70.3	71.4	57.3	–	–	–	–	–	1.50×
$\mathcal{B}$ + Injectors	**70.86**	**71.69***	**58.90***	**67.10***	**47.27***	**45.01**	**57.78***	**57.87***	0.10×
RoBERTa-base ($\mathcal{R}$)	68.01	69.11	53.31	61.50	42.42	41.82	44.12	52.86	1.12×
$\mathcal{R}$ + Injectors	73.00	73.50	60.32	67.30	46.26	44.19	60.73	58.20	0.10×

Table 2: Performance (F1-score on Good, Accuracy otherwise) of competing methods on eight datasets, along with the percentage of trained parameters (TP; excluding embeddings). Attribute injected PLMs that perform worse than the base model $\mathcal{B}$ are *italicized*. Among $\mathcal{B}$-based models, best systems are in **bold** and the second-best are underlined. Asterisk (*) signifies a significant difference between our model and the second-best model (paired bootstrap resampling; $p < 0.05$).

Model	Y2013	Y2014	IMDB	AAPR	PolMed	Food	Good	Beer
$\mathcal{B}$ + Injectors	70.86	71.69	58.90	67.10	47.27	45.01	57.78	57.87
– bias injection	70.33	71.24	58.54	66.45	46.67	44.86	57.06	57.67
– weight injection	70.51	71.27	58.55	66.85	45.66	44.74	57.61	57.29
– task adapter	69.21	69.68	57.25	65.55	46.89	44.30	56.57	57.28
– attribute drop	69.29	70.94	57.69	65.55	46.33	44.48	56.62	57.41
– post-aggregation	70.76	71.35	58.63	64.42	47.27	43.30	57.78	57.69
– low-rank	OOM	OOM	OOM	OOM	OOM	OOM	OOM	OOM
– PHM	68.27	69.05	56.74	65.15	46.16	43.89	55.99	56.51

Table 3: Performance of Injectors and versions thereof without some of our proposed components (second block), training mechanisms (third block), and parameter-saving methods (fourth block). OOM denotes the model does not run on our experimental setup due to out of memory error.

	Y2013-cold		
Model	20%	50%	80%
BERT-base ($\mathcal{B}$)	65.24	63.88	58.23
$\mathcal{B}$ + CHIM	65.85	64.12	58.56
$\mathcal{B}$ + TinyMAA	66.26	64.20	57.68
$\mathcal{B}$ + MAA	67.03	64.22	58.48
$\mathcal{B}$ + Injectors	**67.89**	**64.32**	**59.23**
HCSC	63.6	60.8	53.8
	IMDB-cold		
Model	20%	50%	80%
BERT-base ($\mathcal{B}$)	50.24	47.77	41.37
$\mathcal{B}$ + CHIM	51.08	47.81	41.22
$\mathcal{B}$ + TinyMAA	52.45	48.27	40.78
$\mathcal{B}$ + MAA	53.52	49.25	41.61
$\mathcal{B}$ + Injectors	**55.62**	**49.87**	**41.78**
HCSC	50.5	45.6	36.8

Table 4: Performance on the Y2013-cold and IMDB-cold. Best systems are shown in bold. Models perform worse than $\mathcal{B}$ are colored red.

Model	A	RPT→A
$\mathcal{B}$ + MAA	53.65	51.27 (−4.43%)
$\mathcal{B}$ + Injectors	55.58	56.23 (+1.17%)
Model	R	APT→R
$\mathcal{B}$ + MAA	51.79	50.57 (−2.36%)
$\mathcal{B}$ + Injectors	53.62	55.48 (+3.47%)
Model	P	ART→P
$\mathcal{B}$ + MAA	52.73	50.84 (−3.57%)
$\mathcal{B}$ + Injectors	55.44	55.63 (+0.34%)
Model	T	ARP→T
$\mathcal{B}$ + MAA	58.25	56.08 (−3.73%)
$\mathcal{B}$ + Injectors	59.39	60.27 (+1.48%)

Table 5: Performance of models on single-task Beer (A: appearance, R: aroma, P: palette, T: taste). The arrow (→) indicates that the attribute-specific adapters of the model in the right-hand side (e.g., A) are initialized using parameters of the left-hand side model (e.g., RPT) and are frozen during training.

On Model Modularity Adapters allow us to compose multiple modules with different functionalities, possibly trained from different models. For example, when predicting the review rating of a specific user on a *new aspect* of a product, we may want to use the learned representation of a user on a previous model (i.e., a review rating prediction model focused on previously known aspects of the product). This is especially crucial when there is fewer data for such new tasks[5]. Since Injectors are basically a sequence of adapters, we expect that modular composition across different models

[5]We leave the exploration of attribute transfer to completely new tasks in future work due to the absence of multi-task datasets with common attributes.

is also effective in our setting. In this section, we verify this using the following experiment.

We use the BEER dataset, which is a multi-task aspect rating prediction dataset with four different aspects. We divide the dataset into two subsets: (1) a single-task *target* dataset and (2) a 3-task *source* dataset. We train the model using the source dataset, obtaining attribute-specific parameters. We then transfer these parameters when training the model using the target dataset, and only fine-tune the parameters of the task adapter and the classifier.

We split the training dataset into four parts, one for each aspect, to ensure that there are *no overlapping training examples* between source and target datasets. For each aspect, we treated it as the target and combined the three non-target datasets as the source dataset. We experimented with MAA and INJECTORS, and report the results in Table 5. When compared to the same model trained directly on the target task (second column), our model is able to achieve a small improvement even when the attribute-specific parameters are fixed and learned from a different task (third column). On the other hand, MAA shows a decrease in performance in all cases.

7 Conclusions

We considered the use of attributes as additional contexts when fine-tuning pretrained language models. We proposed the INJECTOR module, an extension of adapters that also accepts attributes as input. Our method considers two kinds of injection strategies, uses parameter-saving techniques and introduces training mechanisms to account for sparse and multi-labeled attributes. We also introduced ATTRIB, a collection of 14 datasets to evaluate attribute injection methods. Experiments on this benchmark of various classification tasks showed that our method improves substantially over previous methods. Finally, we conducted extensive analyses on how INJECTORS handle attribute sparsity and verify their modularity. In the future, we plan to apply our methods to real-world data where there are millions of attributes. We also plan to explore the use of attribute injection methods for text generation tasks, i.e. injecting attributes when generating texts. Our code and the ATTRIB benchmark will be publicly available online at https://github.com/rktamplayo/Injector.

Acknowledgments

We would like to thank Jaewook Kang and other members of the NAVER AI Lab for their insightful comments. Reinald is supported by a Google PhD Fellowship.

References

Reinald Kim Amplayo. 2019. Rethinking attribute representation and injection for sentiment classification. In *EMNLP-IJCNLP*, pages 5602–5613.

Reinald Kim Amplayo, Jihyeok Kim, Sua Sung, and Seung-won Hwang. 2018. Cold-start aware user and product attention for sentiment classification. In *ACL*, pages 2535–2544.

Yoshua Bengio, Aaron C. Courville, and Pascal Vincent. 2013. Representation learning: A review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35:1798–1828.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, J. Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, T. Henighan, R. Child, A. Ramesh, Daniel M. Ziegler, Jeff Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. In *NeurIPS*.

Huimin Chen, Maosong Sun, Cunchao Tu, Yankai Lin, and Zhiyuan Liu. 2016. Neural sentiment classification with user and product attention. In *EMNLP*, pages 1650–1659.

Asa Cooper Stickland, Alexandre Berard, and Vassilina Nikoulina. 2021. Multilingual domain adaptation for NMT: Decoupling language and domain information with adapters. In *Proceedings of the Sixth Conference on Machine Translation*, pages 578–598, Online. Association for Computational Linguistics.

Sumanth Dathathri, Andrea Madotto, Janice Lan, Jane Hung, Eric Frank, Piero Molino, Jason Yosinski, and Rosanne Liu. 2019. Plug and play language models: A simple approach to controlled text generation. *arXiv preprint arXiv:1912.02164*.

Ludovic Denoyer and P. Gallinari. 2006. The wikipedia xml corpus. In *INEX*.

J. Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. In *NAACL*.

Zi-Yi Dou. 2017. Capturing user and product information for document level sentiment analysis with deep memory network. In *EMNLP*, pages 521–526.

Angela Fan, David Grangier, and Michael Auli. 2018. Controllable abstractive summarization. In *NMT@ACL*.

Jessica Ficler and Y. Goldberg. 2017. Controlling linguistic style aspects in neural language generation. *ArXiv*, abs/1707.02633.

T. Fukuhara, H. Nakagawa, and T. Nishida. 2007. Understanding sentiment of people from news articles: Temporal sentiment analysis of social events. In *ICWSM*.

Tianyu Gao, Adam Fisch, and Danqi Chen. 2021. Making pre-trained language models better few-shot learners. In *ACL-IJCNLP*.

W. Gao, Naoki Yoshinaga, Nobuhiro Kaji, and M. Kitsuregawa. 2013. Modeling user leniency and product popularity for sentiment classification. In *IJCNLP*.

Junxian He, Chunting Zhou, Xuezhe Ma, Taylor Berg-Kirkpatrick, and Graham Neubig. 2022. Towards a unified view of parameter-efficient transfer learning. In *International Conference on Learning Representations*.

Ruidan He, Linlin Liu, Hai Ye, Qingyu Tan, Bosheng Ding, Liying Cheng, Jiawei Low, Lidong Bing, and Luo Si. 2021. On the effectiveness of adapter-based tuning for pretrained language model adaptation. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 2208–2222, Online. Association for Computational Linguistics.

N. Houlsby, A. Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin de Laroussilhe, Andrea Gesmundo, Mona Attariyan, and S. Gelly. 2019. Parameter-efficient transfer learning for nlp. In *ICML*.

Arash Joorabchi and A. Mahdi. 2011. An unsupervised approach to automatic classification of scientific literature utilizing bibliographic metadata. *Journal of Information Science*, 37:499 – 514.

Nitish Shirish Keskar, Bryan McCann, Lav R Varshney, Caiming Xiong, and Richard Socher. 2019. Ctrl: A conditional transformer language model for controllable generation. *arXiv preprint arXiv:1909.05858*.

Yuta Kikuchi, Graham Neubig, Ryohei Sasano, Hiroya Takamura, and M. Okumura. 2016. Controlling output length in neural encoder-decoders. In *EMNLP*.

Jihyeok Kim, Reinald Kim Amplayo, Kyungjae Lee, Sua Sung, Minji Seo, and Seung-won Hwang. 2019. Categorical metadata representation for customized text classification. *TACL*, 7:201–215.

Jooyeon Kim, Dongwoo Kim, and Alice H. Oh. 2017. Joint modeling of topics, citations, and topical authority in academic corpora. *Transactions of the Association for Computational Linguistics*, 5:191–204.

Anne Lauscher, Olga Majewska, Leonardo F. R. Ribeiro, Iryna Gurevych, Nikolai Rozanov, and Goran Glavaš. 2020. Common sense or world knowledge? investigating adapter-based knowledge injection into pretrained transformers. In *Proceedings of Deep Learning Inside Out (DeeLIO): The First Workshop on Knowledge Extraction and Integration for Deep Learning Architectures*, pages 43–49, Online. Association for Computational Linguistics.

Xiang Lisa Li and Percy Liang. 2021. Prefix-tuning: Optimizing continuous prompts for generation. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 4582–4597, Online. Association for Computational Linguistics.

Shuhua Liu and Thomas Forss. 2014. Web content classification based on topic and sentiment analysis of text. In *KDIR*.

Xiao Liu, Yanan Zheng, Zhengxiao Du, Ming Ding, Yujie Qian, Zhilin Yang, and Jie Tang. 2021. Gpt understands, too. *arXiv:2103.10385*.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, M. Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *ArXiv*, abs/1907.11692.

Yunfei Long, Mingyu Ma, Qin Lu, Rong Xiang, and Chu-Ren Huang. 2018. Dual memory network model for biased product review classification. In *WASSA@EMNLP*, pages 140–148.

I. Loshchilov and F. Hutter. 2019. Decoupled weight decay regularization. In *ICLR*.

Dehong Ma, Sujian Li, Xiaodong Zhang, Houfeng Wang, and Xu Sun. 2017. Cascading multiway attentions for document-level sentiment classification. In *IJCNLP*, pages 634–643.

Rabeeh Karimi Mahabadi, James Henderson, and Sebastian Ruder. 2021. Compacter: Efficient low-rank hypercomplex adapter layers. *ArXiv*, abs/2106.04647.

Bodhisattwa Prasad Majumder, Shuyang Li, Jianmo Ni, and Julian McAuley. 2019. Generating personalized recipes from historical user preferences. In *EMNLP/IJCNLP*.

Yuning Mao, Lambert Mathias, Rui Hou, Amjad Almahairi, Hao Ma, Jiawei Han, Wen-tau Yih, and Madian Khabsa. 2021. Unipelt: A unified framework for parameter-efficient language model tuning. *arXiv preprint arXiv:2110.07577*.

Julian McAuley, J. Leskovec, and Dan Jurafsky. 2012. Learning attitudes and attributes from multi-aspect reviews. In *ICDM*.

Tomas Mikolov, Kai Chen, G. Corrado, and J. Dean. 2013. Efficient estimation of word representations in vector space. In *ICLR*.

Jianmo Ni, Jiacheng Li, and Julian McAuley. 2019. Justifying recommendations using distantly-labeled reviews and fine-grained aspects. In *EMNLP-IJCNLP*, pages 188–197.

Ethan Perez, Florian Strub, Harm de Vries, Vincent Dumoulin, and Aaron Courville. 2018. Film: Visual reasoning with a general conditioning layer. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence and Thirtieth Innovative Applications of Artificial Intelligence Conference and Eighth AAAI Symposium on Educational Advances in Artificial Intelligence*, AAAI'18/IAAI'18/EAAI'18. AAAI Press.

Jonas Pfeiffer, Aishwarya Kamath, Andreas Rücklé, Kyunghyun Cho, and Iryna Gurevych. 2021. AdapterFusion: Non-destructive task composition for transfer learning. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 487–503, Online. Association for Computational Linguistics.

Xipeng Qiu, Tianxiang Sun, Yige Xu, Yunfan Shao, Ning Dai, and Xuanjing Huang. 2020. Pre-trained models for natural language processing: A survey. *Science China Technological Sciences*, pages 1–26.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *JMLR*, 21:1–67.

D. Ramage, David Hall, Ramesh Nallapati, and Christopher D. Manning. 2009. Labeled lda: A supervised topic model for credit attribution in multi-labeled corpora. In *EMNLP*.

M. Rosen-Zvi, T. Griffiths, M. Steyvers, and Padhraic Smyth. 2004. The author-topic model for authors and documents. In *UAI*.

Evan Sandhaus. 2008. The new york times annotated corpus. *Linguistic Data Consortium, Philadelphia*, 6(12):e26752.

Timo Schick and Hinrich Schütze. 2021. It's not just size that matters: Small language models are also few-shot learners. In *NAACL*.

Rico Sennrich, B. Haddow, and Alexandra Birch. 2016. Controlling politeness in neural machine translation via side constraints. In *NAACL*.

Duyu Tang, Bing Qin, and Ting Liu. 2015. Learning semantic representations of users and products for document level sentiment classification. In *ACL-IJCNLP*, pages 1014–1023.

Yi Tay, A. Zhang, Anh Tuan Luu, J. Rao, Shuai Zhang, Shuohang Wang, Jie Fu, and S. C. Hui. 2019. Lightweight and efficient neural natural language processing with quaternion networks. In *ACL*.

Ashish Vaswani, Noam M. Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *NIPS*.

Mengting Wan and Julian McAuley. 2018. Item recommendation on monotonic behavior chains. In *RecSys*.

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. In *ICLR*.

Yunli Wang, Yu Wu, Lili Mou, Zhoujun Li, and Wen-Han Chao. 2019. Harnessing pre-trained neural networks with rules for formality style transfer. In *EMNLP*.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, R'emi Louf, Morgan Funtowicz, and Jamie Brew. 2020. Transformers: State-of-the-art natural language processing. In *EMNLP*.

Zhen Wu, Xin-Yu Dai, Cunyan Yin, Shujian Huang, and Jiajun Chen. 2018. Improving review representations with user attention and product attention for sentiment classification. In *AAAI*.

Keyulu Xu, Weihua Hu, J. Leskovec, and S. Jegelka. 2019. How powerful are graph neural networks? In *ICLR*.

Min Yang, Jincheng Mei, Heng Ji, Wei Zhao, Zhou Zhao, and Xiaojun Chen. 2017. Identifying and tracking sentiments and topics from social media texts during natural disasters. In *EMNLP*.

Pengcheng Yang, Xu Sun, Wei Li, and Shuming Ma. 2018. Automatic academic paper rating based on modularized hierarchical convolutional neural network. In *ACL*.

A. Zhang, Yi Tay, Shuai Zhang, Alvin Chan, A. Luu, S. C. Hui, and Jie Fu. 2021a. Beyond fully-connected layers with quaternions: Parameterization of hypercomplex multiplications with 1/n parameters. In *ICLR*.

Youjia Zhang, Jin Wang, Liang-Chih Yu, and Xuejie Zhang. 2021b. Ma-bert: Learning representation by incorporating multi-attribute knowledge in transformers. In *FINDINGS*.

Rui Zhao and K. Mao. 2017. Topic-aware deep compositional models for sentence classification. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 25:248–260.

Tony Zhao, Eric Wallace, Shi Feng, D. Klein, and Sameer Singh. 2021. Calibrate before use: Improving few-shot performance of language models. In *ICML*.

Yukun Zhu, Ryan Kiros, Rich Zemel, Ruslan Salakhutdinov, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. 2015. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. In *ICCV*, pages 19–27.

A Appendix

A.1 Descriptions of Newly Introduced Datasets

This section describes how we procured the three datasets we introduce in this paper:

1. FOOD: We used the dataset gathered in Majumder et al. (2019), which was used as a personalized recipe generation dataset. We repurposed the dataset for a new classification task and used the recipes as input text and the duration (in minutes) as output class. We removed instances with outliers: (1) recipes that took less than 5 minutes and more than 150 minutes; (2) recipes with more than 500 tokens or less than 10 tokens; and (3) tags with more than 50 labels. We also removed from the attribute vocabulary tags that explicitly indicate the recipe duration (e.g., `60-minutes-or-less`) and those that are used on almost all instances (e.g., `time-to-make`). We shuffled the data and used 10% each for the development and test sets, and the remaining 80% for the training set.

2. GOOD: We used the review corpus gathered in Wan and McAuley (2018), which was also used for spoiler detection. Since the split is unfortunately not publicly shared, we created our own split. We first removed very short (less than 32 tokens) and very long (more than 256) reviews as they were outliers. We then divided the data into three splits, with two 10K splits as the development and test sets, and the remaining split as the training set.

3. BEER: We used the review corpus gathered in McAuley et al. (2012). We removed outliers and split the dataset into three using the same method we did with Goodreads.

A.2 AttrIB Attribute Analysis

We conducted an attribute analysis to check whether the attributes in these datasets are indeed useful for the tasks. Specifically, using the training set, we create attribute-specific distribution over classes $q_z(y)$ for each attribute z to represent attribute bias, i.e.:

$$q_z(y) = \sum_{x \in \mathcal{D}_z} \mathbb{1}(x, y)/|\mathcal{D}_z| \qquad (12)$$

where $\mathcal{D}_z$ is the subset of the training set where attribute z exists and $\mathbb{1}(x, y)$ is the indicator function that returns 1 if class y is the class of training example x. We then create the same attribute-specific class distribution $p_z(y)$ for the dev set and compare the similarity of both distributions using KL-divergence. Finally, we compare with the KL-divergence of random (i.e., uniform distribution) and majority (i.e., overall class distribution) baselines to see whether attributes provide better biases than when selecting at random or the majority class.

Table 6 shows the KL-divergence of random, majority, and attributes, both when z is a single attribute and all attributes combined. We can observe two things in the table. Firstly, using a single attribute usually does not perform better than the majority baseline. This is in contrast to experiments in some of the previous work (Chen et al., 2016), where text classification models based on neural networks improved when incorporated with just one out of all attributes. This means that these models can potentially learn beyond providing attribute-specific biases, such as jointly modeling attributes and texts. Secondly, combining all attributes provides the best bias on most datasets, with the exception of POLMED. In this dataset, the audience attribute is worse than the random baseline, but it might be useful in some cases. In fact, the best configuration is actually combining both the media source and the audience attributes, which has a KL-divergence of 1.726. Therefore, attributes may either be helpful or detrimental depending on multiple factors, e.g., other attributes and (possibly) textual input. Models thus need to effectively determine if the attribute is useful for each of the examples.

Y2013		Y2014	
random	1.608	random	1.608
majority	1.358	majority	1.379
user	1.406	user	1.425
product	1.419	product	1.415
all	**1.265**	all	**1.279**
IMDB		AAPR	
random	2.300	random	0.693
majority	2.087	majority	0.693
user	2.132	author	1.357
product	2.314	research area	0.665
all	**1.873**	all	**0.664**
POLMED		FOOD	
random	2.195	random	2.769
majority	1.851	majority	2.352
politician	1.790	user	2.306
media source	**1.728**	ingredient	3.708
audience	3.640	tag	2.277
political bias	1.844		
all	1.770	all	**2.275**
GOOD		BEER	
random	0.693	random	2.195
majority	0.216	majority	1.656
user	0.246	user	1.754
book	0.326	beer	1.769
rating	0.215	rating	1.363
all	**0.215**	all	**1.362**

Table 6: KL-divergence of baseline and attribute-specific distributions over classes to the dev set distributions. Best values are in **bold**.

A.3 Cold-Start Dataset Statistics

Table 7 reports the statistics of the cold-start datasets in ATTRIB.

A.4 Parameter Analysis of Weight-based Injection

Recall that we define $\mathbf{W}_{z_j} \in \mathbb{R}^{d_h \times d_a}$ as follows:

$$\mathbf{W}_{z_j} = g_{weight}(\mathbf{z}_j) + \mathbf{C}_{weight} \tag{13}$$

In a naive setting, we can trivially use a projection function as our g_{weight}, which would linearly transform $\mathbf{z}_j \in \mathbb{R}^{d_z}$ into the shape $d_h \times d_a$. This would need a weight tensor of size $d_z \times d_h \times d_a$, which can be prohibitively large. This parameter dominates all the other parameters in the module, thus the overall parameter of the naive method is $\mathcal{O}(d_z * d_h * d_a)$.

Our parameter-saving methods remove this large tensor, but instead use three smaller parameters in hypercomplex space: the transform function $\sigma_o(\cdot)$ that is basically a linear transformation with a projection matrix of size $d_z \times d_a$ (Eq. 6), the vector $\mathbf{s}_o$ of size d_h/O^2, and the matrix $\mathbf{A}_o$ of size $O \times O$. Since we have O dimensions in our hypercomplex space, we have a total of $O*(d_z*d_a+d_h/O^2+O^2)$, which we can reduce as follows:

$$\begin{aligned} & O * (d_z * d_a + d_h/O^2 + O^2) \\ & = O * d_z * d_a + d_h/O + O^3 \\ & \approx O * d_z * d_a \\ & \approx d_z * d_a \end{aligned} \tag{14}$$

given that $O^3 \ll O*d_z*d_a$ and that we can treat O as a constant ($O = 4$ in our experiment). Thus the overall parameter when using our parameter-saving method is $\mathcal{O}(d_z * d_a)$. We emphasize that this is a huge improvement since the PLM hidden size d_h is usually the largest dimension.

The output weight $\mathbf{W}_{z_j}$ has a rank r of at most O^2+1, i.e., (1) the low-rank method (Eq. 7) outputs

	Y2013-COLD		
Dataset	20%	50%	80%
#Train	38.7K	16.1K	2.4K
#Attr. Vocab	2.6K	1.6K	0.7K
%Cold-start	20.5%	49.4%	79.6%

	IMDB-COLD		
Dataset	20%	50%	80%
#Train	44.3K	18.0K	2.5K
#Attr. Vocab	2.4K	1.5K	0.6K
%Cold-start	19.3%	48.7%	80.7%

Table 7: Cold-start dataset statistics. %Cold-start is the percentage of attributes with zero training examples. These datasets use the dev and test sets of the original datasets.

a matrix of rank $r = 1$; (2) the Kronecker product (Eq. 8) returns a matrix of rank $r = O$; and finally, (3) the sum of multiple matrices (Eq. 9) has a rank $r \leq O^2$.