Unfreeze with Care: Space-Efficient Fine-Tuning of Semantic Parsing Models

Weiqi Sun
Amazon Alexa AI
New York, USA
weiqisun@amazon.com

Haidar Khan
Amazon Alexa AI
New York, USA
khhaida@amazon.com

Nicolas Guenon des Mesnards
Amazon Alexa AI
New York, USA
mesnarn@amazon.com

Melanie Rubino
Amazon Alexa AI
New York, USA
rubinome@amazon.com

Konstantine Arkoudas
Amazon Alexa AI
New York, USA
arkoudk@amazon.com

ABSTRACT

Semantic parsing is a key NLP task that maps natural language to structured meaning representations. As in many other NLP tasks, SOTA performance in semantic parsing is now attained by fine-tuning a large pretrained language model (PLM). While effective, this approach is inefficient in the presence of multiple downstream tasks, as a new set of values for all parameters of the PLM needs to be stored for each task separately. Recent work has explored methods for adapting PLMs to downstream tasks while keeping most (or all) of their parameters frozen. We examine two such promising techniques, prefix tuning and bias-term tuning, specifically on semantic parsing. We compare them against each other on two different semantic parsing datasets, and we also compare them against full and partial fine-tuning, both in few-shot and conventional data settings. While prefix tuning is shown to do poorly for semantic parsing tasks off the shelf, we modify it by adding special token embeddings, which results in very strong performance without compromising parameter savings.

CCS CONCEPTS

• **Computing methodologies** → **Natural language processing**; **Neural networks**.

KEYWORDS

semantic parsing, pretrained language model, prefix tuning, BitFit

ACM Reference Format:
Weiqi Sun, Haidar Khan, Nicolas Guenon des Mesnards, Melanie Rubino, and Konstantine Arkoudas. 2022. Unfreeze with Care: Space-Efficient Fine-Tuning of Semantic Parsing Models. In *Proceedings of the ACM Web Conference 2022 (WWW '22), April 25–29, 2022, Virtual Event, Lyon, France.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3485447.3511942

WWW '22, April 25–29, 2022, Virtual Event, Lyon, France

ACM ISBN 978-1-4503-9096-5/22/04…$15.00
https://doi.org/10.1145/3485447.3511942

1 INTRODUCTION

Large transformer-based language models that have been pretrained without supervision on huge amounts of text have proven to be remarkably effective in powering downstream NLP tasks ranging from classification and regression to generation tasks such as question answering, summarization, and semantic parsing. Two common ways of transferring knowledge from a large pretrained language model (PLM) M to a downstream task T are: (a) using M as a *feature extractor*, e.g., solely for the purpose of encoding a given input and passing the output to a subsequent model trained on T; and (b) *fine-tuning* M on T's training data, after replacing M's output layer by a task-specific head. Fine-tuning has become the de facto method of adapting a PLM to a downstream task, as most SOTA results in NLP have been obtained by this approach.

While fine-tuning is very effective, it is susceptible to catastrophic forgetting unless the learning rate is carefully calibrated,[1] and can also be prone to overfitting in low-data settings. More importantly for present purposes, it requires updating and storing values for *all* of M's parameters, which can easily number in the billions. In the presence of multiple downstream tasks (a frequent scenario in larger organizations), this approach becomes too inefficient. Accordingly, a good deal of effort has recently focused on adapting a PLM M to downstream tasks in ways that do not require updating all of M's parameters.

We distinguish between *mutative* methods of adapting M, which modify some or all of M's parameters, and *non-mutative* methods, which *freeze* all of M and instead learn additional parameters on a task-specific basis. Mutative methods do not need to modify all of M's parameters. Indeed, it is common to modify only a few top layers of M. Non-mutative methods have the considerable advantage that multiple downstream tasks can share the same PLM, as they access it in a read-only way that does not change its weights. We introduce an additional distinction that partitions non-mutative methods into those that are *opaque*, i.e., methods that do not need to know anything about the internal structure of M; and *transparent* methods that are coupled with the specific architecture of M.

In this paper we study the effectiveness of two adaptation techniques specifically in connection with semantic parsing. We focus

[1] If the learning rate is too large then M will forget what it learned during pre-training. On the other hand, if the learning rate is too small then M will not adapt to T well enough.

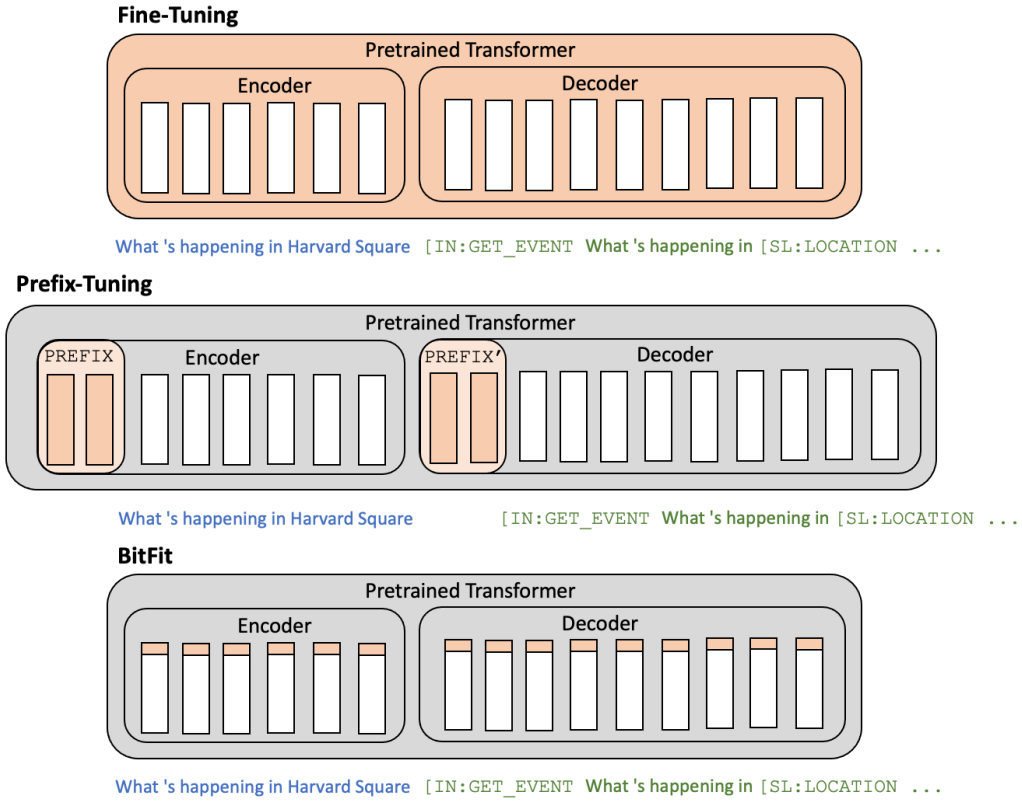

Figure 1: Fine-tuning updates all parameters. Prefix-tuning freezes Transformer parameters and only optimizes the prefixes. BitFit freezes all the weight matrices and only updates the bias vector in the Transformer layers. White blocks are the hidden vectors, orange blocks represent the tunable model components, and grey blocks are frozen.

on task-oriented parsing rather than the generation of general-purpose meaning representations (such as AMR [3]), although the techniques we discuss should be generally applicable. Semantic parsing (especially task-oriented parsing) is a key task for digital voice assistants such as Google Assistant and Alexa. It also has a vital role to play in unlocking the full potential of the web, particularly in enabling sophisticated NLP-powered searches of web content. Yet to the best of our knowledge there has been no prior investigation of adaptation techniques in the semantic parsing field. The two techniques that we investigate are prefix tuning, a transparent non-mutative method recently introduced by Li and Liang [13]; and BitFit [22], a mutative technique that only fine-tunes bias terms; see Figure 1. We compare both approaches against full fine-tuning, as well as *partial fine-tuning*, which only modifies the weights of a few top layers of the PLM. We study both regular- and low-resource data settings.

We view semantic parsing as a sequence-to-sequence (seq2seq) task, where the input sequence x is a natural-language utterance and the output sequence y is a linearized semantic tree in some appropriate representation. An example from the TOP dataset [7] is shown in Figure 2. As described in Section 4, in this paper we work with the latest version of the TOP dataset [6], as well as PIZZA [2], a new semantic-parsing dataset in the domain of pizza ordering.

Prefix tuning was previously applied to two generation tasks [13], summarization and table-to-text. In both of these, the decoded outputs are pure natural-language text sequences. By contrast, in semantic parsing the output sequences contain *special tokens* representing the non-terminals of the semantic grammar of the domain

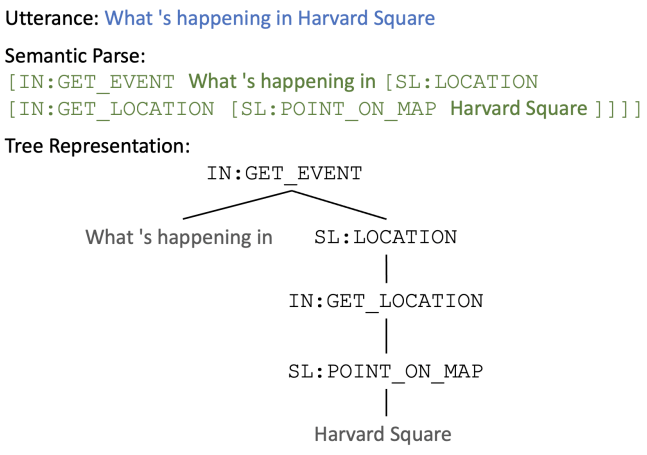

Figure 2: A sample parse tree from the TOP dataset.

at hand, often containing bracketing symbols such as parentheses, square brackets, colons, and so on, e.g., `[IN:GET_LOCATION` in Figure 2. This does not present a problem for full fine-tuning, because full fine-tuning updates the entire model during training, including the embedding layer, and hence the subwords corresponding to these special tokens can be modified as needed. However, this is not possible during prefix tuning, which freezes the entire PLM during training. Consequently, as shown by our results, a straightforward application of prefix tuning to semantic parsing will not work.

We solve this problem by adding the domain's special tokens to the vocabulary, expanding the embedding matrix accordingly, and then unfreezing only those matrix entries that correspond to special tokens. More specifically, we unfreeze the entire matrix but set the gradients to zero for those cells that do not involve special tokens. We demonstrate that this change has a dramatic impact on the performance of prefix tuning for semantic parsing, while adding only a trivial number of additional parameters to the overall budget. As a result, prefix tuning performance on semantic parsing tasks is competitive with regular fine-tuning in full data regimes and outperforms in low data settings.

The rest of the paper is structured as follows: in Sections 2 and 3 we introduce the two lightweight tuning techniques we consider: prefix tuning and BitFit. We present the datasets and experimental setup in Section 4, and our full-data and few-shot results in Section 5. Section 6 analyzes prefix tuning and its variants. Section 7 reviews related work, and Section 8 presents our conclusions.

2 PREFIX TUNING

Following recent work by Li and Liang [13], we use prefix tuning as an alternative to fine-tuning a PLM for conditional generation, applying it to semantic parsing. Prefix tuning is inspired from in-context learning (aka *prompting*) with very large PLMs such as GPT-3 [5], whereby desired outputs are generated without any task-specific fine-tuning. Instead, prompting freezes the PLM's parameters and attempts to guide the model by prefacing the input sequence with a natural language text that essentially gives instructions on what output to generate, along with a few (input, output) examples. Choosing the right text for prompting is important for performance but can be challenging [10, 11].

Prefix tuning builds on this idea by formulating prompting as an optimization task, one that optimizes a set of task-specific continuous column vectors prepended to the input, or *prefixes*, instead of input-specific discrete natural language tokens. This allows for more expressive prompt forms while limiting the number of trainable parameters to as few as 0.1% of the PLM parameters.

We start with a pretrained transformer-based encoder-decoder language model capturing a distribution $p_\phi(y|x)$ [12, 20], where ϕ are the model's parameters, the input x is encoded by a bidirectional encoder, and the target sequence y is generated autoregressively by the decoder, conditioned on the encoded input x and previously generated outputs. We define $z = [x; y]$ as the concatenation of x and y.

For the semantic parsing task, we prepend prefixes to both the encoder and decoder to obtain $z = [PREFIX; x; PREFIX'; y]$; see Figure 1. The prefixes are prepended to the key and value of each attention layer in the transformer architecture, including the encoder self-attention, decoder self-attention, and encoder-decoder cross attention. The prefix length *PrefixLength* is a hyperparameter and all hidden-layer representations of the prefixes are optimized. The prefix parameters are stored in a trainable matrix P_θ with the following number of parameters:

$$\begin{aligned} Size(P_\theta) &= PrefixLength \times Size(layers) \\ &\quad \times HidDim \times 2 \times 3, \end{aligned} \tag{1}$$

where *HidDim* is the dimension of the hidden state in a single attention layer, the 2 multiplier is due to the attention key and value parameters, and the 3 multiplier is due to the three aforementioned types of attention layers. The model is then trained similarly to the original language model, but now only modifying the prefix parameters θ, as the ϕ parameters remain frozen. Furthermore, these prefix parameters are trained on task-specific training data. Each semantic parsing domain is considered a different task and results in a different set of task-specific prefix-tuning parameters.

2.1 Special Token Embeddings

Prefix tuning was shown to give strong results in the summarization and table-to-text tasks [13], but its effectiveness on the semantic parsing task is not a priori clear. In both summarization and table-to-text, the output sequences are pure natural language. In a semantic parsing task, by contrast, the output semantic representations often[2] contain special labels representing non-terminal nodes, which are usually formulated by concatenating multiple tokens together with underscores, colons, brackets, and so on, such as `[IN:GET_REMINDER_DATA_TIME`. In prefix tuning, these special labels are tokenized by the PLM's subword tokenizer. The resulting tokenized sequences will deviate from the data given to the PLM during pretraining, making it much more difficult for the decoder to generate the correct output sequences; we show this in Section 5.

To address this problem, we introduce *special token embeddings* to the prefix tuning method: We add the non-terminal labels as special tokens to the tokenizer vocabulary, expand the pretrained embedding layer to the size of the new vocabulary, and tune the embedding vectors of the special tokens, E_η, during training along with the prefixes, while keeping the embeddings of all other tokens fixed. With special token embeddings, the number of additional parameters that need to be trained and stored is

$$Size(E_\eta) = Size(labels) \times EmbDim, \tag{2}$$

where *EmbDim* is the dimension of the embedding vectors and the number of labels is domain specific.

During training, instead of initializing the special token embeddings randomly, we use the averaged embedding vectors of their subword tokens with the original vocabulary as the initial values. Furthermore, we remove the open bracket from the label tokens and rely on the subword tokenization to split the open bracket from labels.

In Section 5 we demonstrate that these special token embeddings significantly improve the performance of prefix tuning on semantic parsing, especially in the full-data setting.

[2]But not always; sometimes semantic parsing models are trained to output natural language directly, see Section 7.

3 BITFIT

BitFit [22], or Bias-terms Fine-tuning, has recently been proposed as a method of fine-tuning an encoder-based PLM for downstream tasks, such as GLUE [21] sentence classification tasks, by adapting only the bias parameters of the model. Specifically, for a pretrained transformer encoder containing linear components $\mathbf{W} \cdot \mathbf{x} + \mathbf{b}$, BitFit fine-tunes only the vector of bias terms $\mathbf{b}$. This is shown to achieve competitive performance with full fine-tuning and other lightweight fine-tuning methods on sentence and token-level classification tasks, especially in low resource settings. In addition, it is shown that the bias parameters are special in this sense, and that a random subset of parameters selected from the model do not achieve similar fine-tuning performance.

Here we apply BitFit to a new task, semantic parsing, which involves autoregressive generation and an encoder-decoder architecture, and compare its performance against other adaptation techniques. We extend the set of fine-tuned parameters to include the biases in the decoder. Thus, the set of fine-tuned parameters includes the biases of the query, key, and value projections in the encoder self-attention, the decoder self-attention, and the decoder cross-attention. In addition, we follow Zaken et al. [22] and include the biases in the layer norm and feed-forward components from the encoder and decoder.

4 EXPERIMENTAL SETUP

4.1 Datasets

We use two public datasets for task-oriented semantic parsing in our experiments: the `TOPv2` dataset [6], and a recently open-sourced food-ordering dataset called `PIZZA` [2].

The TOPv2 dataset contains 180K examples in 8 distinct domains: alarm, event, messaging, music, navigation, reminder, timer, and weather. Each example in this dataset contains three segments: a domain label, a natural language utterance, and a target semantic tree consisting of intent nodes, slot nodes, and leaves that contain fragments of the utterance text. The original semantic representations provided in the dataset are in the TOP format. In this paper we convert the TOP format to a TOP-Decoupled format [1] by removing terminals that are not associated with any semantic nodes in the parsing tree. For example, the source utterance in Figure 2, *"What's happening in Harvard Square"*, is mapped to the target semantics:

```
[IN:GET_EVENT
  [SL:LOCATION
    [IN:GET_LOCATION
      [SL:POINT_ON_MAP Harvard Square ] ] ] ]
```

This representation differs from the TOP format in that tokens that do not carry any semantics are removed: {*What's happening in*} do not appear in the above target. Only children representing resolvable entities of interest are kept.

We report results on three TOPv2 domains, selected so as to have diverse profiles in structural complexity (see table below):

- `Navigation` is the domain with the largest average tree depth and number of slots.
- `Reminder` has the second deepest semantics, and has the largest intent catalog of all 8 domains.
- `Weather` is on the other end of the spectrum: it is the domain with the smallest average depth and smallest catalog sizes.

The following table lists the number of examples, intents, and slots, as well as the average semantic-tree depth for these three domains:

	Weather	Navigation	Reminder
#Train	23,054	20,998	17,840
#Dev	2,667	2971	2,526
#Test	5,682	6,075	5,767
#Intent	7	17	19
#Slot	11	33	32
Depth	1.93	2.68	2.45

We also experiment with a low-data regime and use the **10SPIS** (samples per intent and slot) setting used by Chen et al. [6] for the *Reminder* and *Weather* domains; relevant statistics can be found below. As no low-resource splits were provided for the *Navigation* domain, we sampled its 10SPIS fold ourselves.

	Weather	Navigation	Reminder
#Train	87	156	219
#Dev	59	132	161
#Test	5,682	6,075	5,767

The test set is the same as for the full-data setting.

The PIZZA dataset contains about 2 million synthetically generated pizza and/or drink orders mapped to their target semantics, used for training, and about 1.7K human-generated and annotated orders used for validation and testing. The dataset also comes in the TOP format. We follow the same rules used in the TOPv2 dataset to form the TOP-Decoupled semantics.

For the PIZZA dataset, we simulate a few-shot scenario as follows: Using the human-annotated validation set of 348 utterances, we randomly sample 200 utterances from it, and then incrementally build 30-, 50-, and 100-shot datasets. For each of these three datasets, we also use validation sets of sizes 30, 50, and 100, respectively. We report results on the provided test set, which contains 1,357 utterances.

	30 shots	50 shots	100 shots
#Train	30	50	100
#Dev	30	50	100
#Test	1,357	1,357	1,357

As a point of comparison, the average semantic tree depth of the human-generated portion of the `PIZZA` dataset is around 3.6.

4.2 Metric

Even though the training data is in TOP-Decoupled format, the trained model might still produce terminals that are not associated with slot nodes, and which therefore do not contribute to the output's semantics; we remove such terminals for evaluation purposes. Moreover, since sibling order in a parse tree does not alter its semantics (in both datasets), we ignore ordering differences when computing the semantics-only exact-match metric. We refer to this as the Unordered Semantics-Only Exact-Match metric (EM).

4.3 Architectures and Hyperparameters

The PLM we use in all of our experiments is BART-Large, a 24-layer transformer-based encoder-decoder model pretrained on a masked span prediction task on 160GB of unlabeled text [12]. The pretraining task involves masking contiguous spans of an input sequence, encoding the masked sequence with the 12-layer encoder, and generating the original text using the 12-layer decoder. Therefore, the number of *layers* in Equation 1 is 12. The dimensions of the hidden states, *HidDim*, and the embedding vectors, *EmbDim*, are both 1024.

To study the effectiveness of different tuning strategies, we use full fine-tuning (FT) as the baseline model. We then compare the accuracy of the two adaptation techniques: prefix tuning with and without special token embeddings (†Prefix and Prefix respectively) and BitFit (BitFit). We report main results with fixed prefix lengths of 30 and 5 for the full data setting, as well as 20 and 5 for the low data setting in the next section, and study the effect of prefix length on model accuracy more systematically in Section 6. We also include partial fine-tuning (FT-Top2) in the comparison, which only fine-tunes the top 2 *decoder* layers of the pretrained BART-Large model. The intuition behind this approach is that the upper PLM layers contain more specialized information.

To establish a fair comparison across various tuning strategies, we allocate an equal amount of trials (16) to each method. Instead of grid search, we select the tunable hyperparameters and ranges (as described in the Appendix A) for each method and then conduct a random search [4]. All other hyperparameters are fixed during the random search and their values are listed in Table 1. Each trial is repeated over three random seeds and the hyperparameters with the best average performance on the validation set are selected for final evaluation.

hyperparameter	value
beam size	6
max epoch	50
early stopping patience	4
max target length	200
gradient accumulation step	3
evaluation frequency	1 epoch*

Table 1: Values of the fixed hyperparameters in all experiments. *In low data settings, we duplicate the training data multiple times to have similar number of training examples as the full data settings (20,000 examples), so that the number of updates between evaluations are similar across all experiments.

5 RESULTS

5.1 Full Data Setting

Table 2 presents results for the full-data setting. We see that both prefix tuning ($\texttt{Prefix}_{\texttt{len}}$) and BitFit significantly underperform both full and partial fine-tuning. The biggest performance drop is as large as 20.72 absolute points. In the case of prefix tuning, this is likely because the target sequences are not natural language. Because the PLM is not fine-tuned, the decoder cannot learn anything about the rather unique non-terminal labels that occur in the outputs (and are unlikely to have appeared in the datasets used to pretrain the PLM). In the case of BitFit, we find that while tuning bias terms works well for sentence classification tasks, its performance degrades significantly on token-level tasks (we also tried BitFit for named entity recognition, in addition to semantic parsing) and tasks involving generation.

We solve the prefix tuning problem by adding the special labels to the tokenizer vocabulary and tuning the embedding vectors of these special tokens together along with the prefixes. Table 2 shows that with special token embeddings added in this way ($^{\dagger}\texttt{Prefix}_{\texttt{len}}$), the exact match accuracy of prefix tuning can be boosted by 9.61 to 17.49 absolute points, with only a negligible number of additional task-specific parameters added (0.01% of the total parameter size). Note that the number of task-specific parameters added by the special token embeddings is proportional to the number of non-terminal labels, including both intents and slots, in the dataset. The numbers reported in Table 2 are obtained from the Reminder domain, which has the largest number of non-terminal labels in the TOPv2 dataset. Therefore, it represents an upper bound on the number of task-specific parameters for that entire dataset.

The results with prefix lengths of 30 and 5 suggest that in harder domains like Navigation, longer prefixes are preferable, as they help to learn complex semantic structures. By contrast, in relatively easy domains like Weather, shorter prefixes work better, as they avoid overfitting.

Compared to full fine-tuning, prefix tuning with special token embeddings can achieve competitive performance with only 0.1% of the PLM's parameters. It also decidedly outperforms the BitFit method, with a similar number of parameters.

5.2 Low Data Setting

The few-shot results on both PIZZA and TOPv2, presented in Table 3, show that off-the-shelf prefix tuning outperforms all other tuning strategies, including full fine-tuning, partial fine-tuning, and prefix tuning with special token embeddings. On average, it outperforms full fine-tuning by 2.76 absolute points, while requiring a significantly smaller number of parameters.

Because models are more prone to overfitting in low-data regimes, we hypothesize that, with more parameters available to the model, full fine-tuning and prefix tuning with special token embeddings overfit the training data and produce worse results on test data. Furthermore, according to He et al. [8], the lower layers of a PLM capture more generic features while upper layers capture more task-specific features. Therefore, as the lowest layer in a PLM, if the special token embeddings in the embedding layer come to overfit the training data, the model will not be able to generalize well to unseen data.

Overall, prefix tuning with special token embeddings in the full-data regime and without special token embeddings in the low-data regime significantly outperforms the other lightweight tuning strategies. It is competitive with the full fine-tuning baseline when there is plenty of training data and performs even better in few-shot scenarios.

	TOPv2				Task Specific Parameters	
	Navigation	Reminder	Weather	Avg.	Number	% of PLM Parameters
FT	83.08	82.66	89.81	85.18	406,291,456	100%
FT-Top2	77.38	75.29	88.66	80.44	33,629,273	8.28%
Prefix_{30}	63.05	65.61	72.75	67.14	2,211,840	0.54%
Prefix_{5}	62.36	69.29	76.75	69.46	368,640	0.09%
$^{\dagger}\text{Prefix}_{30}$	**80.41**	**79.12**	87.49	82.34	2,264,064	0.56%
$^{\dagger}\text{Prefix}_{5}$	79.85	78.90	**88.88**	**82.54**	420,864	0.10%
BitFit	67.55	62.87	70.66	67.03	320,399	0.08%

Table 2: EM accuracy of models trained with the full TOPv2 datasets averaged over 3 seeds (left), and the number and percentage of task specific parameters (right). Symbol † indicates the special token embeddings.

	TOPv2 10SPIS				PIZZA			
	Navigation	Reminder	Weather	Avg.	30 Shots	50 Shots	100 Shots	Avg.
FT	44.89	51.46	62.75	53.03	65.70	72.00	83.50	73.73
FT-Top2	10.25	17.87	26.95	18.36	24.48	42.82	60.22	42.51
Prefix_{20}	46.75	56.41	68.15	57.10	**67.99**	**71.93**	**85.55**	**75.16**
Prefix_{5}	45.24	**56.98**	**69.19**	**57.13**	66.72	71.53	83.37	73.88
$^{\dagger}\text{Prefix}_{20}$	46.02	52.71	56.21	51.65	62.58	69.39	82.74	71.57
$^{\dagger}\text{Prefix}_{5}$	**47.50**	55.32	63.86	55.56	64.39	71.24	82.91	72.84
BitFit	2.88	13.25	43.66	19.93	48.35	57.49	70.22	58.69

Table 3: EM accuracy of few-shot learning on TOPv2 dataset with 10SPIS (left) and PIZZA dataset (right). Symbol † indicates the special token embeddings.

6 ANALYSIS

6.1 Prefix Length

To study the effect of prefix length on model accuracy, we trained prefix tuning models (with and without special token embeddings) in the Navigation domain with prefix lengths ranging from 0 to 300. The results are plotted in Figure 3. We see that as long as the prefix length is not zero, the models deliver decent accuracy even with prefix lengths as short as 1. Model performance increases as the prefix length increases, until it reaches an optimal length (80 with special token embeddings and 10 without). Then there is a performance drop, before the model finally attains a steady state.

The variance of model accuracy with special token embeddings is much smaller than that of the model without such embeddings, suggesting that the special token embeddings help to stabilize the model and make prefix tuning more robust with respect to prefix length.

Figure 3 also demonstrates that special token embeddings dramatically improve the prefix tuning performance across all the prefix lengths. The average performance boost is 19.98 absolute points.

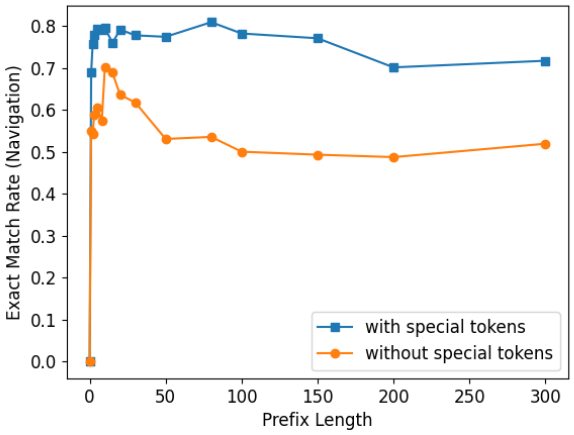

Figure 3: Effect of prefix length on exact match rate in Navigation domain.

6.2 Prefix Variants

In the prefix tuning model used to obtain the main results, we *prepend* prefixes to *all attention layers* in both the encoder and the decoder. Here we explore two variants of this technique: 1) inserting these vectors in different locations; and 2) prepending prefixes only in the top 2 decoder layers.

Different vector locations: Because BART's decoder is autoregressive and proceeds from left to right, prefixes are the only reasonable choice for where to add task-specific hidden vectors. However, encoder layers are bidirectional and symmetric, and therefore these vectors don't need to be prepended to the left of the attention layers. Accordingly, we investigate the effect of different location choices by:

	TOPv2			
	Navigation	Reminder	Weather	Avg.
Prefix_{30}	63.05	65.61	72.75	67.14
Prefix_{5}	62.36	69.29	76.75	69.46
$^{\dagger}\text{Prefix}_{30}$	80.41	79.12	87.49	82.34
$^{\dagger}\text{Prefix}_{5}$	79.85	78.90	88.88	82.54
$^{\dagger}\text{Suffix}_{30}$	77.77	78.63	**89.90**	82.10
$^{\dagger}\text{Suffix}_{5}$	76.96	80.01	89.42	82.13
$^{\dagger}\text{Prefix\&Suffix}_{30}$	80.13	**80.18**	89.12	**83.14**
$^{\dagger}\text{Prefix\&Suffix}_{5}$	**80.35**	78.65	88.96	82.65
$\text{Prefix}^{\text{top-2-layers}}$	33.30	35.28	55.56	41.38

Table 4: EM accuracy of different prefix tuning variants. Symbol † indicates the special token embeddings.

- *appending* these vectors to the right of the encoder layers as suffixes (†`Suffix`); and by
- splitting the prefix vectors into a prefix sequence and a suffix sequence (†`Prefix&Suffix`).

In the †`Prefix&Suffix` model, the prefix vectors are split evenly with rounding up on the prefix side. For example, a $^{\dagger}\text{Prefix\&Suffix}_{5}$ model includes a prefix of length 3 and a suffix of length 2.

Table 4 shows that †`Suffix` models produce almost identical performance as the †`Prefix` models on average, while the †`Prefix&Suffix` model can slightly improve model performance by an average of 0.80 and 0.11 absolute points for prefix lengths of 30 and 5, respectively. This demonstrates that prefix tuning is not sensitive to prefix location.

Prefix tuning only the top 2 decoder layers: Inspired by the fine-tuning of the top 2 decoder layers, we study $\text{Prefix}^{\text{top-2-layers}}$, a prefix-tuning variant that only prepends prefixes to the top two decoder layers. An additional motivation behind this experiment is that even though prefix tuning can dramatically reduce the number of task-specific parameters, it cannot reduce training latency, since the gradients still need to back-propagate to the very first layer in the encoder. On the other hand, with prefixes in only the top 2 decoder layers, the backward propagation can be terminated immediately after the lowest prefix has been processed, which should make the training process significantly faster.

The last row in Table 4 shows the resulting model accuracy when we only add prefixes to the top 2 decoder layers. We report the results we obtain from the best prefix length in each domain instead of using a fixed prefix length. As can be seen, performance drops radically for partial prefix tuning. EM accuracy regresses by 26.92 absolute points on average compared to full prefix tuning without special tokens, and by more than 40 absolute points compared to full prefix tuning with special tokens.

6.3 Effect of Special Token Embeddings

The results in Section 5 demonstrate that special token embeddings can dramatically improve the performance of prefix tuning in semantic parsing tasks. By adding embedding vectors for non-terminal labels, a frozen PLM with tunable prefixes *and* tunable special token embeddings can learn the distributed representations of the latter, which help to guide the decoder in the right directions.

Domain	Max.	Min.	Mean	Median
Navigation	160	10	40.41	37
†Navigation	84	5	18.34	17
Reminder	179	11	46.52	44
†Reminder	77	5	22.00	21
Weather	72	10	26.13	23
†Weather	34	5	12.10	12

Table 5: Statistics of target length in Navigation, Reminder, and Weather domain. Symbol † indicates the special token embeddings.

Besides this contribution, we hypothesize that special token embeddings can also reduce the target sequence length and further improve model performance. In a transformer-based PLM such as BART, both source and target sequences are tokenized by the PLM's *subword* tokenizer. Since the non-terminal labels in semantic representations are created artificially and tend to contain multiple words and special characters, they are usually tokenized to long subword sequences. For example, a typical intent label, `IN:GET_REMINDER_DATA_TIME`, in the Reminder domain, is tokenized to 12 subwords: `["IN", ":", "GET", "_", "REM", "IND", "ER", "_", "D", "ATE", "_", "TIME"]`. Therefore, these special non-terminal labels cause the target sequences to be unnecessarily long, thereby complicating the decoding process.

The issue can be mitigated with special token embeddings. By adding the non-terminal labels to the tokenizer subword vocabulary, these labels will stay as whole units after tokenization, which will reduce the overall target sequence length. Table 5 and Figure 4 show the statistics and percentiles, respectively, of the target sequence lengths in the Navigation, Reminder, and Weather domains. We can clearly see that special token embeddings significantly reduce target length. The mean target length is reduced by 53.67% on average across three domains. With shorter target sequences, the decoder should be able to generate correct outputs easier.

To evaluate the effect of target sequence length on overall performance, we compare the full fine-tuning models with and without special token embeddings. Table 6 shows that special token embeddings consistently boost model performance in all three domains. The average performance improvement is 1.09 absolute points. In full fine-tuning, the entire embedding layer, including the embedding vectors of the subwords corresponding to the non-terminal labels, is updated. Thus, the distributed representations of the non-terminal labels can be learned regardless of special token embeddings, which indicates that in full fine-tuning the performance improvement introduced by special token embeddings comes primarily from the reduced length of the target sequences.

Therefore, adding special token embeddings to prefix tuning method helps not only to learn the distributed representations of non-terminal labels in a given semantic parsing task, but also to facilitate the decoding process by reducing overall target sequence length.

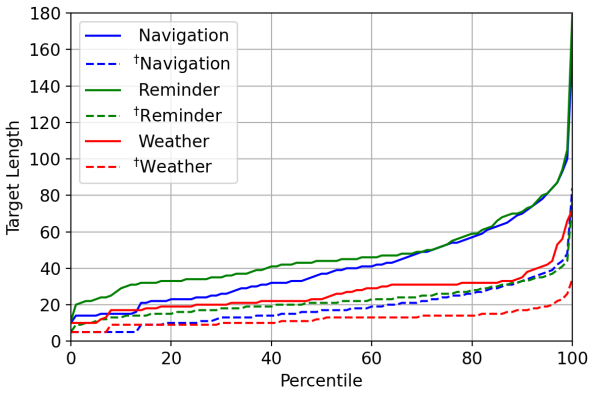

Figure 4: Percentile of target length in Navigation (blue), Reminder (green), and Weather (red) domains with (dash lines) and without (solid lines) special tokens. Symbol † indicates the special token embeddings.

	TOPv2			
	Navigation	Reminder	Weather	Avg.
FT	83.08	82.66	89.81	85.18
†FT	**83.17**	**84.33**	**91.33**	**86.27**

Table 6: EM accuracy of fine-tuned models with (bottom) and without (top) special token embeddings. Symbol † indicates the special token embeddings.

7 RELATED WORK

Semantic parsing has been a staple of NLP for decades, particularly in connection with question answering [23, 24]. More recently, semantic parsing has received increased attention due to its relevance to digital voice assistants. That is especially the case for variants such as task-oriented parsing (which is the category of the TOPv2 dataset).

As with other NLP tasks, SOTA results in semantic parsing, including task-oriented parsing, have been obtained by fine-tuning large pretrained language models [6, 14, 17], particularly pretrained encoder-decoder architectures such as BART [12]. Shin et al. [18] leverages the PLM's ability to generate natural language by reframing semantic parsing as a paraphrasing task. They use the PLM to map utterances into a canonical (but highly restricted) natural-language representation that can then be automatically mapped to the final output target.

PLMs also help with bootstrapping a task-oriented parser from a small amount of training data. Chen et al. [6] approach this by fine-tuning the PLM with other domain data and also by utilizing meta-learning techniques. Tran and Tan [19] tackle the problem by using a PLM to generate synthetic training data.

The issue with fine-tuning large pretrained models is that doing so requires a lot of space, and this becomes infeasible when there are many downstream tasks. As we saw, transparent non-mutative methods such as prefix tuning [13] adapt a large PLM in parameter-efficient ways. Other approaches add task-specific layers, or *adapters*, between layers of the PLM [9, 15, 16]. These adapter approaches are not as space-efficient, as they generally require up to 30 times more tunable parameters in order to achieve performance comparable to prefix tuning [13]. There are also opaque non-mutative techniques, such as side-tuning [25], which further facilitate sharing across multiple tasks.

8 CONCLUSIONS

We studied a transparent non-mutative fine-tuning strategy, prefix tuning, and a lightweight mutative fine-tuning strategy, BitFit, specifically for semantic parsing tasks, and compared their performance against each other and also against full and partial fine-tuning. We observed that both BitFit and prefix tuning perform poorly in regular data settings, and BitFit also performs poorly in few-shot settings. To solve the problem with prefix tuning, we proposed the addition of special-token embeddings and demonstrated that this can dramatically improve the technique's performance while adding only a trivial number of task-specific parameters. With the help of special token embeddings, prefix tuning becomes competitive with full fine-tuning in the full data setting and outperforms full fine-tuning in the low data regime, despite modifying only 0.1% of the total number of PLM parameters.

ACKNOWLEDGMENTS

The authors thank Xiang Lisa Li for helpful discussions on prefix tuning.

REFERENCES

[1] Armen Aghajanyan, Jean Maillard, Akshat Shrivastava, Keith Diedrick, Michael Haeger, Haoran Li, Yashar Mehdad, Veselin Stoyanov, Anuj Kumar, Mike Lewis, and Sonal Gupta. 2020. Conversational Semantic Parsing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics, Online, 5026–5035.

[2] Konstantine Arkoudas, Nicolas Guenon des Mesnards, Melanie Rubino, Sandesh Swamy, Saarthak Khanna, and Weiqi Sun. 2021. Pizza: a task-oriented semantic parsing dataset. https://github.com/amazon-research/pizza-semantic-parsing-dataset

[3] Laura Banarescu, Claire Bonial, Shu Cai, Madalina Georgescu, Kira Griffitt, Ulf Hermjakob, Kevin Knight, Philipp Koehn, Martha Palmer, and Nathan Schneider. 2013. Abstract Meaning Representation for Sembanking. In *Proceedings of the 7th Linguistic Annotation Workshop and Interoperability with Discourse*. Association for Computational Linguistics, Sofia, Bulgaria, 178–186. https://aclanthology.org/W13-2322

[4] James Bergstra and Yoshua Bengio. 2012. Random Search for Hyper-Parameter Optimization. *Journal of Machine Learning Research* 13, 10 (2012), 281–305. http://jmlr.org/papers/v13/bergstra12a.html

[5] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey 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. arXiv:2005.14165 [cs.CL]

[6] Xilun Chen, Asish Ghoshal, Yashar Mehdad, Luke Zettlemoyer, and Sonal Gupta. 2020. Low-Resource Domain Adaptation for Compositional Task-Oriented Semantic Parsing. *CoRR* abs/2010.03546 (2020). arXiv:2010.03546 https://arxiv.org/abs/2010.03546

[7] Sonal Gupta, Rushin Shah, Mrinal Mohit, Anuj Kumar, and Mike Lewis. 2018. Semantic Parsing for Task Oriented Dialog using Hierarchical Representations. arXiv:1810.07942 [cs.CL]

[8] Ruidan He, Linlin Liu, Hai Ye, Qingyu Tan, Bosheng Ding, Liying Cheng, Jia-Wei Low, Lidong Bing, and Luo Si. 2021. On the Effectiveness of Adapter-based Tuning for Pretrained Language Model Adaptation. arXiv:arXiv:2106.03164

[9] 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 *Proceedings of the 36th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 97)*, Kamalika Chaudhuri and Ruslan Salakhutdinov (Eds.). PMLR, 2790–2799. http://proceedings.mlr.press/v97/houlsby19a.html
[10] Zhengbao Jiang, Frank F. Xu, Jun Araki, and Graham Neubig. 2020. How Can We Know What Language Models Know? *Transactions of the Association for Computational Linguistics* 8 (07 2020), 423–438. https://doi.org/10.1162/tacl_a_00324 arXiv:https://direct.mit.edu/tacl/article-pdf/doi/10.1162/tacl_a_00324/1923867/tacl_a_00324.pdf
[11] Brian Lester, Rami Al-Rfou, and Noah Constant. 2021. The Power of Scale for Parameter-Efficient Prompt Tuning. arXiv:2104.08691 [cs.CL]
[12] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Ves Stoyanov, and Luke Zettlemoyer. 2019. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. *arXiv preprint arXiv:1910.13461* (2019).
[13] Xiang Lisa Li and Percy Liang. 2021. Prefix-Tuning: Optimizing Continuous Prompts for Generation. arXiv:2101.00190 [cs.CL]
[14] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. RoBERTa: A Robustly Optimized BERT Pretraining Approach. arXiv:1907.11692 [cs.CL]
[15] 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*. Association for Computational Linguistics, Online, 487–503. https://aclanthology.org/2021.eacl-main.39
[16] S-A Rebuffi, H. Bilen, and A. Vedaldi. 2017. Learning multiple visual domains with residual adapters. In *Advances in Neural Information Processing Systems*.
[17] Subendhu Rongali, Luca Soldaini, Emilio Monti, and Wael Hamza. 2020. Don't Parse, Generate! A Sequence to Sequence Architecture for Task-Oriented Semantic Parsing. In *Proceedings of The Web Conference 2020*. 2962–2968.
[18] Richard Shin, Christopher H. Lin, Sam Thomson, Charles Chen, Subhro Roy, Emmanouil Antonios Platanios, Adam Pauls, Dan Klein, Jason Eisner, and Benjamin Van Durme. 2021. Constrained Language Models Yield Few-Shot Semantic Parsers. *CoRR* abs/2104.08768 (2021). arXiv:2104.08768 https://arxiv.org/abs/2104.08768
[19] Ke Tran and Ming Tan. 2020. Generating Synthetic Data for Task-Oriented Semantic Parsing with Hierarchical Representations. In *Proceedings of the Fourth Workshop on Structured Prediction for NLP*. Association for Computational Linguistics, Online, 17–21. https://doi.org/10.18653/v1/2020.spnlp-1.3
[20] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is All You Need. https://arxiv.org/pdf/1706.03762.pdf
[21] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. 2019. GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding. arXiv:1804.07461 [cs.CL]
[22] Elad Ben Zaken, Shauli Ravfogel, and Yoav Goldberg. 2021. BitFit: Simple Parameter-efficient Fine-tuning for Transformer-based Masked Language-models. *arXiv preprint arXiv:2106.10199* (2021).
[23] John M. Zelle and Raymond J. Mooney. 1996. Learning to Parse Database Queries Using Inductive Logic Programming.. In *AAAI/IAAI, Vol. 2*, William J. Clancey and Daniel S. Weld (Eds.). MIT Press, 1050–1055.
[24] Luke S. Zettlemoyer and Michael Collins. 2005. Learning to Map Sentences to Logical Form: Structured Classification with Probabilistic Categorial Grammars. In *UAI '05, Proceedings of the 21st Conference in Uncertainty in Artificial Intelligence, Edinburgh, Scotland, July 26-29, 2005*. AUAI Press, 658–666.
[25] Jeffrey O Zhang, Alexander Sax, Amir Zamir, Leonidas Guibas, and Jitendra Malik. 2020. Side-Tuning: A Baseline for Network Adaptation via Additive Side Networks. arXiv:1912.13503 [cs.LG]

A HYPERPARAMETER TUNING

In prefix tuning without a fixed prefix length, we tune the learning rate (`lr`), batch size (`bsz`), prefix length (`prefix_length`), and the intermediate dimension of prefixes (`mid_dim`). The detailed values of these hyperparameters in each random search trial is listed in Table 7. In the prefix tuning experiments with fixed prefix length, we overwrite the `prefix_length` value in the trial by the fixed length.

In other tuning strategies, including full fine-tuning, partial fine-tuning, and BitFit, we only tune the learning rate (`lr`) and batch size (`bsz`), as specified in Table 8.

trial	lr	bsz	prefix_length	mid_dim
0	3.19E-05	16	50	800
1	1.13E-04	4	50	300
2	3.00E-04	8	100	800
3	4.62E-05	16	30	600
4	4.27E-04	4	50	300
5	1.79E-05	16	30	300
6	5.89E-05	4	50	800
7	3.88E-05	16	20	600
8	5.93E-06	16	30	300
9	6.23E-05	16	100	300
10	5.81E-06	8	100	600
11	1.25E-04	4	50	800
12	2.64E-05	16	20	600
13	3.98E-05	4	20	300
14	1.25E-05	4	100	800
15	3.70E-05	16	20	300

Table 7: Hyperparameter values in each random search trial for prefix tuning.

trial	lr	bsz
0	1.61E-04	4
1	7.34E-05	16
2	2.52E-04	4
3	1.22E-04	4
4	1.49E-05	8
5	3.69E-04	8
6	5.82E-05	4
7	1.50E-05	8
8	4.07E-05	16
9	1.40E-05	16
10	5.40E-06	16
11	1.50E-05	8
12	9.82E-05	4
13	7.07E-05	4
14	1.42E-04	8
15	4.38E-05	4

Table 8: Hyperparameter values in each random search trial for full fine-tuning, partial fine-tuning, and BitFit.