Continual Prompt Tuning for Dialog State Tracking

Qi Zhu[1], **Bing Li**[1], **Fei Mi**[2], **Xiaoyan Zhu**[1], **Minlie Huang**[1*]
[1]CoAI Group, DCST, IAI, BNRIST, Tsinghua University
[2]Huawei Noah's Ark Lab
zhu-q18@mails.tsinghua.edu.cn, aihuang@tsinghua.edu.cn

Abstract

A desirable dialog system should be able to continually learn new skills without forgetting old ones, and thereby adapt to new domains or tasks in its life cycle. However, continually training a model often leads to a well-known catastrophic forgetting issue. In this paper, we present Continual Prompt Tuning, a parameter-efficient framework that not only avoids forgetting but also enables knowledge transfer between tasks. To avoid forgetting, we only learn and store a few prompt tokens' embeddings for each task while freezing the backbone pre-trained model. To achieve bi-directional knowledge transfer among tasks, we propose several techniques (continual prompt initialization, query fusion, and memory replay) to transfer knowledge from preceding tasks and a memory-guided technique to transfer knowledge from subsequent tasks. Extensive experiments demonstrate the effectiveness and efficiency of our proposed method on continual learning for dialog state tracking, compared with state-of-the-art baselines.

1 Introduction

Recently, most studies have focused on developing dialog systems for specific domains in an offline manner, assuming the data distribution stays the same. However, this is far from realistic because a deployed dialog system is often required to support new domains and provide more services constantly over time. Therefore, it is crucial for a dialog system to continually learn new tasks without forgetting old ones with high efficiency.

Previous studies on continual learning (Kirkpatrick et al., 2017; Li and Hoiem, 2018) mainly focused on solving the *catastrophic forgetting (CF)* problem (McCloskey and Cohen, 1989): when a neural model is trained on a sequence of tasks, new tasks may interfere catastrophically with old tasks. Simply storing a model version for each task to

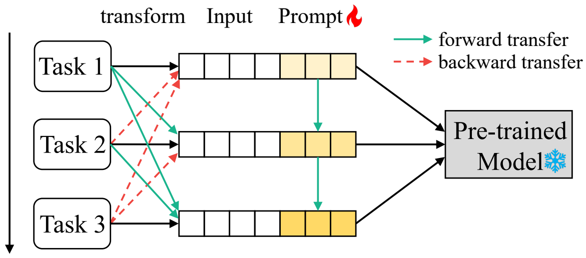

Figure 1: An illustration of *Continual Prompt Tuning*. We train a soft prompt for each task and freeze the pre-trained model. Several techniques are proposed to transfer knowledge from preceding tasks (green solid arrows) and subsequent tasks (red dashed arrows).

mitigate forgetting is prohibitive as the number of tasks grows, especially when the model size is large. To mitigate catastrophic forgetting with low computation and storage overhead, recent methods freeze the backbone model and propose to train a weight/feature mask (Mallya et al., 2018; Geng et al., 2021) or an adapter (Madotto et al., 2021) for each task independently. However, the techniques above are still not efficient enough, and they largely ignore knowledge transfer among tasks.

In this paper, we develop prompt tuning (Lester et al., 2021) for continual learning. We freeze the backbone pre-trained model and train a few prompt tokens' embeddings for each task, which is highly parameter-efficient to avoid forgetting. As illustrated by yellow components in Figure 1, we concatenate the input with a few *tunable* task-specific prompt tokens before feeding it to a *frozen* pre-trained model. Since these prompt tokens have only a small number of parameters (0.1% of the pre-trained model's parameters in our experiments), we can efficiently train and store the prompt for each task. During inference, the same pre-trained model can handle different tasks by inputting different prompts, which is friendly for deployment.

Unlike the vanilla approach of training each task's prompt from scratch and fixing it afterward, we propose *Continual Prompt Tuning*, a framework

*Corresponding author.

that enables **knowledge transfer** between tasks. We consider transferring knowledge from both preceding tasks (forward) and subsequent tasks (backward). To realize forward transfer, we propose several techniques, including continual prompt initialization, query fusion, and memory replay (green solid arrows in Figure 1). To achieve positive backward transfer, we propose a memory-guided technique that uses subsequent tasks' data to update the previous tasks' prompts selectively (red dashed arrows in Figure 1).

We conduct experiments on Dialog State Tracking (DST), a core component of a dialog system, using the Schema-Guided Dialog dataset (Rastogi et al., 2020). The model continually learns new services that have multiple slots to fill. We concatenate all slots' descriptions with the input and insert a sentinel token after each description, formulating DST as a masked spans recovering task, which is similar to the pre-training objective of T5 (Raffel et al., 2020). We empirically show that our proposed framework effectively outperforms state-of-the-art baselines on continual learning for DST, and is extremely efficient in terms of computation and storage.[1]

To summarize, our main contributions are:

1. For the first time, we develop prompt tuning for continual learning, which avoids forgetting efficiently and is friendly for deployment.

2. We investigate several techniques for forward and backward knowledge transfer based on prompt tuning, further boosting the continual learning performance.

3. Our experiments on continual DST demonstrate the superior performance and efficiency of our proposed method.

2 Related Work

2.1 Continual Learning

Continual Learning (CL) studies the problem of continually acquiring knowledge from a data stream and reusing it for future learning while avoiding forgetting. Three kinds of CL methods have been developed. *Rehearsal* methods store and replay some training samples from previous tasks (Rebuffi et al., 2017; Lopez-Paz and Ranzato, 2017). *Regularization* methods apply additional loss to aid knowledge consolidation (Kirkpatrick et al., 2017; Li and Hoiem, 2018). *Architectural* methods introduce task-specific parameters for new tasks and fix parameters for old tasks to prevent forgetting, to which our method belongs. Previous architectural methods include dynamic expanding network structure (Rusu et al., 2016), iterative network pruning and re-training (Mallya and Lazebnik, 2018), learning a parameter mask for each task individually (Mallya et al., 2018), etc.

For continual learning in dialog system, variants of general CL methods have been applied (Lee, 2017; Shen et al., 2019; Wu et al., 2019; Mi et al., 2020; Geng et al., 2021). AdapterCL (Madotto et al., 2021) is the most related to our work, which freezes the pre-trained model and learns an adapter (Houlsby et al., 2019) for each task independently. Compared with AdapterCL, our method is more parameter-efficient, and we explore the effect of both forward and backward transfer.

2.2 Prompt-based Tuning

Recent studies have found that using a textual prompt to convert downstream tasks to the language modeling task is a more effective way to use pre-trained language models than typical fine-tuning (Brown et al., 2020; Schick and Schütze, 2021). Prompts can be manual designed (Petroni et al., 2019) or generated automatically (Shin et al., 2020; Jiang et al., 2020; Gao et al., 2021). Since searching prompts in discrete spaces is sub-optimal, some works (Qin and Eisner, 2021; Liu et al., 2021; Han et al., 2021) combine hard text prompts and soft prompts whose embeddings are learned through back-propagation. Lester et al. (2021) show that freezing the pre-trained model and only tuning soft prompts, known as prompt tuning, is parameter-efficient and becomes more competitive with fine-tuning as the model size grows.

Prompt tuning differs from embedding adapter (Zhu et al., 2021) that aims to address the multilingual embedding deficiency. An embedding adapter transforms all tokens embeddings but do not affect transformer layers' computation, while prompt tuning does not change tokens embeddings but adds new tunable prompt tokens to the input, serving as context and affecting all following transformer layers. Gu et al. (2021) and Vu et al. (2021) further explore the transferability of soft prompts across tasks. While they investigate one-step adaptation, we are interested in prompt transfer in the continual learning setting.

[1]Code and data are publicly available at https://github.com/thu-coai/CPT4DST

2.3 Dialog State Tracking

Dialog State Tracking (DST) aims to capture user goals in the form of (slot, value) pairs. Traditional ontology-based classification methods (Mrkšić et al., 2017; Lee et al., 2019) require access to all candidate values. To alleviate the reliance on the ontology and improve generalization to unseen values, some work extract values from a dialog context (Xu and Hu, 2018; Gao et al., 2019) while others generate values directly to handle situations where values are missing from the context (Wu et al., 2019; Hosseini-Asl et al., 2020).

Generation-based models either generate all (slot, value) pairs in one pass (Hosseini-Asl et al., 2020; Madotto et al., 2021) or generate value for each given slot separately (Wu et al., 2019). The former are more efficient but can only predict in-domain slots and lack transferability while the latter can incorporate more information about a slot as a query, such as a brief natural language description (Rastogi et al., 2020), slot type information (Lin et al., 2021), possible values (Lee et al., 2021), and the task definition and constraint (Mi et al., 2022). Our proposed method integrates multiple slot descriptions into a single query and generates all values in one pass, which improves performance without losing efficiency.

3 Method

3.1 Overview

The goal of continual learning is to sequentially learn a model $f : \mathcal{X} \times \mathcal{T} \rightarrow \mathcal{Y}$ from a stream of tasks $\mathcal{T}_1 ... \mathcal{T}_T$ that can predict the target y given the input x and task $\mathcal{T}_k \in \mathcal{T}$. We denote the data for each task $\mathcal{T}_k$ as D_k. Our method is based on pre-trained language models. Instead of fine-tuning a pre-trained model in a traditional manner (Figure 2(a)), we freeze the model but "reprogram" it to solve task $\mathcal{T}_k$ by adding m new soft prompt tokens $P_k = P_k^1 P_k^2 ... P_k^m$ to the textual input and tuning the embeddings of P_k only. Since the prompt's parameters are much less than the model's, we save P_k for each task to avoid forgetting.

We treat each service/API as a task in continual DST (service and task are used interchangeably). To incorporate informative slot descriptions and ease the decoding process, we convert the descriptions into a query with masked spans and formulate DST as a masked spans recovering task (Sec. 3.2). To enhance knowledge transfer between tasks, we propose continual prompt initialization, query fusion, and memory replay for forward transfer (Sec. 3.3) and explore a memory-guided technique for backward transfer (Sec. 3.4).

3.2 DST as Masked Spans Recovering

In DST, each service $\mathcal{T}_k$ has a set of pre-defined slots $\mathcal{S}_k = \{s_1, ..., s_{n_k}\}$ to be tracked. The input x is a dialog and the output y consists of slot-value pairs: $\{(s_1, v_1), (s_2, v_2), ..., (s_{n_k}, v_{n_k})\}$. Similar to many NLP tasks, DST can be formulated as a text-to-text generation task. Formally, we define a function $g_k : \mathcal{X} \times \mathcal{Y} \rightarrow \mathcal{V}^* \times \mathcal{V}^*$ for each service $\mathcal{T}_k$ to transform the original data (x, y) to:

$$\tilde{x}, \tilde{y} = g_k(x, y) \tag{1}$$

where $\mathcal{V}$ is the vocabulary and $\tilde{x}, \tilde{y}$ are texts that serve as the model input and output, respectively. For example, $\tilde{x}$ can be the concatenation of x and service name, while $\tilde{y}$ is a sequence of slot-value pairs (Madotto et al., 2021) (Figure 2(a)).

Previous research has shown that incorporating a natural language description d_i for each slot s_i is beneficial (Lin et al., 2021; Lee et al., 2021). They concatenate the dialog x with each slot description d_i and decode the value v_i independently. However, separately decoding is inefficient, especially when there are many slots. To solve this, we concatenate all slot descriptions and insert a sentinel token after each description to form a query added to the input, formulating DST as a masked spans recovering task that generates all slot values in one pass:

$$\begin{aligned} \tilde{x} &= [x; Q_k; P_k] \\ Q_k &= \text{“}d_1^k : \langle \mathrm{M}_1 \rangle. \; ... \; d_{n_k}^k : \langle \mathrm{M}_{n_k} \rangle.\text{”} \\ \tilde{y} &= \text{“}\langle \mathrm{M}_1 \rangle \; v_1^k \; ... \langle \mathrm{M}_{n_k} \rangle \; v_{n_k}^k \text{”} \end{aligned} \tag{2}$$

where $[\cdot;\cdot]$ is the concatenation operation and $\langle \mathrm{M}_* \rangle$ are distinct sentinel tokens representing masked spans. The **query** Q_k contains all n_k slot descriptions for task $\mathcal{T}_k$ with n_k masked spans and $\tilde{y}$ contains corresponding slot values leaded by the sentinel tokens. If the value of a slot can not be inferred from the input, we set it to "None". We freeze the pre-trained model's parameters θ and only optimize the prompt's parameters θ_{P_k} for each service $\mathcal{T}_k$. The loss function is:

$$\mathcal{L}_{\theta_{P_k}}(D_k) = - \sum_{j=1}^{|D_k|} \log p_\theta(\tilde{y}_j^k | [x_j^k; Q_k; P_k]) \tag{3}$$

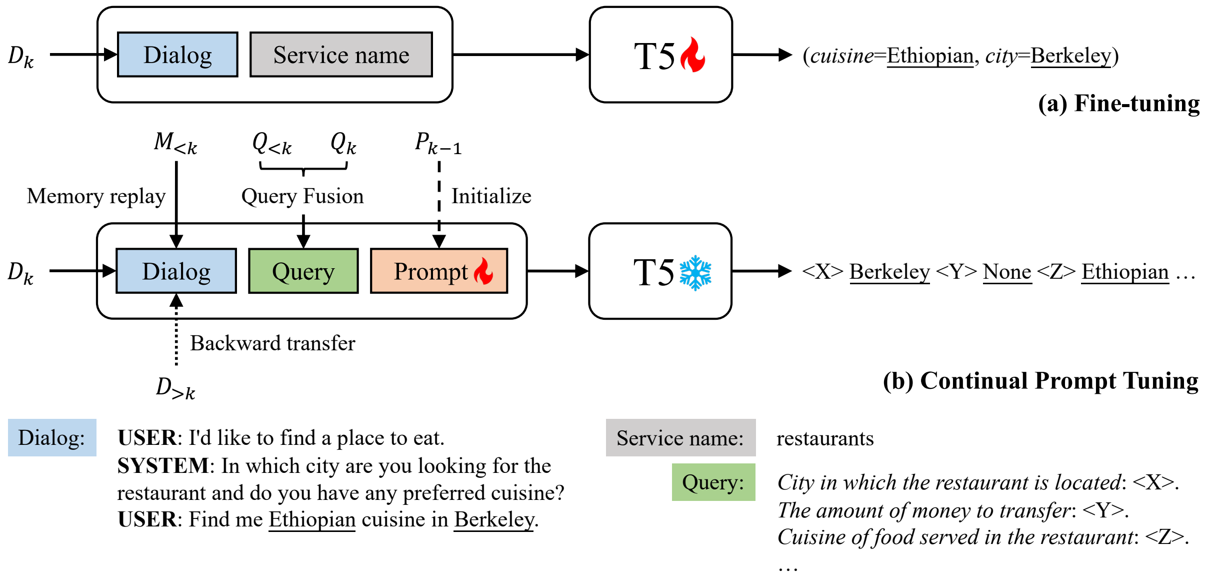

Figure 2: An illustration of *Fine-tuning* and *Continual Prompt Tuning* for continual DST. **(a)** ***Fine-tuning*** takes the dialog and current service's name as input and tunes T5 to generate slot-value pairs. **(b)** ***Continual Prompt Tuning*** feeds the dialog, query consisting of slot descriptions and sentinel tokens, and prompt tokens to frozen T5 and tunes the prompt's embeddings to generate values for all slots in the query. Continual prompt initialization, query fusion, and memory replay are proposed to enhance forward transfer while subsequent services' data will be used for backward transfer. We show an example dialog, service name, fused query, and expected outputs. Slot *names* and *descriptions* are in italic and values are underlined. Note that the second slot description in the query belongs to another service ("banks") and is inserted by query fusion.

3.3 Forward Transfer

Reusing the knowledge acquired from preceding tasks often improves and accelerates the learning on future tasks. Therefore, we propose three types of techniques for forward transfer that can be employed in combination.

3.3.1 Continual Prompt Initialization

An intuitive way to transfer knowledge is parameter initialization. We explore two continual prompt initialization strategies. **CLInit** uses last task's prompt P_{k-1} to initialize current task's prompt P_k. **SelectInit** evaluates all $\{P_j\}_{j<k}$ on the validation set of $\mathcal{T}_k$ without training and selects the one with the lowest loss to initialize P_k. The initial prompt of CLInit has been continually trained on all previous tasks, while SelectInit only considers the most relevant task without interference from its subsequent tasks. We empirically compare these two strategies in Sec. 5.3.

3.3.2 Query Fusion

We hope the model can learn to generate values according to any slot descriptions, which is a general skill that may improve performance on future tasks. However, when training on the current task, there is only one query that consists of the slot descriptions of that task in a fixed order, which may hinder the model from learning the general skill. Therefore, we propose to augment the query by mixing slot descriptions from the current and previous tasks to help the prompt better understand the correspondence between slot descriptions and values. We fuse the query Q_k with previous tasks' queries $\{Q_j\}_{j<k}$ for each sample, including three steps: 1) sample n_1 slots from $\mathcal{S}_k$ randomly, where n_1 is sampled from $[1, |\mathcal{S}_k|]$ uniformly. 2) sample n_2 slots from previous tasks' slots $\bigcup_{i<k} \mathcal{S}_i$ randomly, where n_2 is sampled from $[1, n_1]$ uniformly. 3) combine the above n_1 and n_2 slots' descriptions in a random order as new $Q_k^{'}$, and modify $\tilde{y}$ accordingly. Note that some original slots are dropped, and values for added slots are set to "None".

3.3.3 Memory Replay

Previous studies (Rebuffi et al., 2017; Lopez-Paz and Ranzato, 2017) store a few samples for each task and replay them when training on new tasks to mitigate forgetting. Since our prompt tuning framework has already resolved forgetting, we focus on how these samples benefit the current task. We assume we can store $|M|$ samples for each task ($|M|$ should be small) and denote M_i as the memory for task $\mathcal{T}_i$. When a new task $\mathcal{T}_k$ comes, we optimize P_k on D_k and $M_{<k} = \bigcup_{i<k} M_i$ jointly, changing the loss function to $\mathcal{L}_{\theta_{P_k}}(D_k + M_{<k})$.

When combined with query fusion, query Q_i for samples in the memory M_i are also fused with queries $\{Q_j\}_{j \le k, j \ne i}$ from other seen tasks, including the current task. Note that in this way, samples from other tasks can be viewed as "positive" samples to those added slots in $Q_i^{'}$ since these samples may have not "None" values for those added slots.

3.4 Memory-Guided Backward Transfer

Although fixing P_k immediately after training on task $\mathcal{T}_k$ can avoid forgetting, it also blocks the backward knowledge transfer from future tasks. Motivated by Chaudhry et al. (2019), we explore whether it is possible to improve the performance on previous tasks with the help of memory when a new task comes. Specifically, for each previous task $\mathcal{T}_i, i < k$, we initialize a new prompt $P_i^{(k)}$ to P_i and trained it on current task's data D_k with memory M_i as regularization. During training, we sample a batch from D_k and a batch from M_i synchronously and denote the gradient from each batch as g_{ori} and g_{ref}, respectively. We decide the gradient for update according to the angle between g_{ori} and g_{ref}:

$$g = \begin{cases} g_{ori}, & \text{if } g_{ori}^T g_{ref} > 0 \\ 0, & \text{otherwise} \end{cases} \tag{4}$$

which means we abort the update that will increase the loss on memory batch. We empirically find that this simple abortion is better than projecting g_{ori} onto the normal plane of g_{ref} (Chaudhry et al., 2019). After training, we update P_i to $P_i^{(k)}$ if $P_i^{(k)}$ obtains lower loss and better (or equal) performance on M_i than P_i.

4 Experimental Setup

Recently, Madotto et al. (2021) proposed a continual learning benchmark for task-oriented dialog systems and compared several classic CL methods. We adapt their data processing steps and baselines in our experiments.

4.1 Dataset

We conduct experiments on Schema-Guided Dialog dataset (SGD) (Rastogi et al., 2020) that has 44 services over 19 domains. It also provides a one-sentence description for each slot. We treat each service as a task and only consider dialogs involving a single service. We randomly split a service's dialogs into train/val/test sets at the ratio of 7:1:2. The number of training samples of each service ranges from 112 to 4.7K, and there are 2 to 10 slots for one service. More details about data statistics can be found in the Appendix (Table 8).

4.2 Evaluation Protocol

We evaluate DST performance using the widely adopted Joint Goal Accuracy (JGA) (Wu et al., 2019), which requires all slots' values are correctly predicted. We assign the target service during testing to avoid ambiguity since the same dialog can be parsed differently under different services. We denote $a_{j,i}$ as the JGA on the test set of task $\mathcal{T}_i$ right after training on task $\mathcal{T}_j$. We evaluate the CL performance as the average JGA on all tasks after training on the final task $\mathcal{T}_T$:

$$\textbf{Avg. JGA} = \frac{1}{T}\sum_{i=1}^{T} a_{T,i} \tag{5}$$

Following Lopez-Paz and Ranzato (2017), we define two metrics to measure the effect of forward transfer and backward transfer, respectively:

$$\begin{aligned} \textbf{FWT} &= \frac{1}{T-1}\sum_{i=2}^{T} a_{i-1,i} \\ \textbf{BWT} &= \frac{1}{T-1}\sum_{i=1}^{T-1} a_{T,i} - a_{i,i} \end{aligned} \tag{6}$$

FWT is the averaged zero-shot performance on new tasks, evaluating a model's generalization ability. BWT assesses the impact that learning on subsequent tasks has on a previous task. Negative BWT indicates that the model has forgotten some previously acquired knowledge.

4.3 Baselines and Training Details

We adopt the following models from Madotto et al. (2021) as baselines:

- ***Fine-tuning***: Fine-tune the model on new task data continually.
- ***Replay***: Save $|M|$ samples randomly sampled from the training set of each task $\mathcal{T}_i$ to memory M_i and jointly train the model on new task data D_k and memory $M_{<k}$.
- ***EWC***: Maintain the memory in the same way as *Replay* but use it to compute the Fisher information matrix for regularization (Kirkpatrick et al., 2017).
- ***AdapterCL***: Freeze the pre-trained model and train a residual Adapter (Houlsby et al., 2019) for each task independently (Madotto et al., 2021).

Above methods use the same input and output format as in Figure 2(a).

Prompt tuning based methods including our proposed *Continual Prompt Tuning* are list below:

- ***Prompt Tuning***: Formulate DST as a masked spans recovering task (Sec. 3.2) and only tune the prompt for each task independently.
- ***Multi-task Prompt Tuning***: *Prompt Tuning* in a multi-task manner instead of CL. Train a single prompt using all tasks' data concurrently.
- ***Continual Prompt Tuning***: *Prompt Tuning* with CLInit (Sec. 3.3.1) and query fusion (Sec. 3.3.2).
  - ***w/ memory*** with memory replay (Sec. 3.3.3).
  - ***w/ memory & backward*** with memory replay and memory-guided backward transfer (Sec. 3.4).

We use the following setting in the experiments unless otherwise specified.

Training task sequences Since a sequence of all (44) tasks is too long for the evaluation purpose, we conduct most of the experiments on 15 tasks chosen at random to save computing resources. We run *AdapterCL*, *Prompt Tuning*, and *Multi-task Prompt Tuning* 5 times with different random seeds because they are agnostic to task order. The FWT and BWT metrics for these models are left blank. We run other methods in the same 5 task orders created by random permutation. The selected tasks and ordering are listed in the Appendix (Table 9).

Hyper-parameters We use T5-small as the backbone model and reuse its sentinel tokens (Raffel et al., 2020). For each task, *Continual Prompt Tuning* first trains 10 epochs with fused query (and using memory if available) for forward transfer. Afterward, it concentrates on the current task and continues training 10 epochs on the original data of the current task. When using backward transfer, we train 5 epochs for each previous task. Other methods train 20 epochs for each task. We use AdamW and set the learning rate to 3e-5 for *Fine-tuning*, *Replay*, and *EWC*, 3e-3 for *AdapterCL*, and 0.5 for all prompt tuning based methods. We set the batch size to 16 for prompt tuning based methods and 8 for other methods. To avoid overfitting, we perform early stopping if validation performance does not improve for 5 consecutive epochs. The weight for *EWC* regularization loss is 0.01. We set the memory size $|M|$ to 50 for each task and save the same samples for all methods that require memory. We initialize prompt tokens with the tokens randomly drawn from the vocabulary. For prompt tuning based methods, we tune 100 soft prompt tokens with the embedding size 512 for each task, resulting in 51.2K parameters. To compare parameter efficiency, we adjust *AdapterCL*'s parameters for each task to be nearly 1x or 20x as ours.

5 Experiments and Analysis

The experiments are organized as follows. We compare our method with baselines in Sec. 5.1, and present a comprehensive ablation study in Sec. 5.2. We investigate the effect of prompt initialization in Sec. 5.3, and the effect of model size and prompt length in Sec. 5.4.

5.1 Main Experiment

Computation Resource Analysis. In CL, there is a trade-off between performance and computation resources. Ideally, we hope to utilize the least amount of computation resources to achieve the best performance. We take three vital resources into our consideration. **Memory** saves previous tasks' samples, which may involve privacy issue and requires extra storage. **Additional parameters** are the extra parameters we add to our model to cope with different tasks along the CL process, which should be kept to a minimum in order to scale to long task sequences. **Tunable parameters** are the trainable parameters when we learn a task, which is important for GPU memory and computation. We show the usage of these resources in Table 1 (right). *Replay* stores $|M|$ samples for each task and does not need extra parameters. *EWC* saves the Fisher information matrix and original parameters, requiring two times additional parameters. *AdapterCL*, *Prompt Tuning*, and *Continual Prompt Tuning* require no memory and only add a small number (2% or 0.1%) of additional parameters for each task, largely reducing the computational and storage overhead. Apart from the vanilla form, *Continual Prompt Tuning* can also utilize the memory if available.

CL Performance Analysis. Overall CL results of different methods are summarized in Table 1 (left). We have the following findings:

- Consistent with Madotto et al. (2021), both *Fine-tuning* and *EWC* suffer from catastrophic forgetting while replaying memory can alleviate the problem to a large extend. *Fine-tuning* and *EWC* have a low Avg. JGA because of the large negative BWT, while *Replay* improves BWT a lot thus has a high Avg. JGA.

Method	Avg. JGA	FWT	BWT	Memory	+Params	Tune Params
Fine-tuning	$14.3_{0.8}$	$8.3_{1.0}$	$-49.9_{4.4}$	-	0	1
EWC	$13.9_{1.1}$	$8.4_{0.9}$	$-50.8_{4.3}$	$\|M\|$*T	2	1
Replay	$58.6_{3.5}$	$10.9_{0.5}$	$-3.2_{2.3}$	$\|M\|$*T	0	1
AdapterCL (20x)	$49.8_{1.7}$	-	-	-	2%*T	2%
AdapterCL (1x)	$30.6_{1.1}$	-	-	-	0.1%*T	0.1%
Prompt Tuning	$48.1_{0.9}$	-	-	-	0.1%*T	0.1%
Continual Prompt Tuning	$\mathbf{59.5}_{1.4}$	$9.9_{0.7}$	0	-		
w/ memory	$60.7_{2.4}$	$\mathbf{13.7}_{0.8}$	0	$\|M\|$*T		
w/ memory & backward	$\mathbf{61.2}_{2.5}$	$\mathbf{13.7}_{0.8}$	$\mathbf{0.5}_{0.4}$	$\|M\|$*T		
Multi-task Prompt Tuning	$64.0_{1.9}$	-	-	-	0.1%	0.1%

Table 1: Performance and resource usage on 15 tasks CL in 5 random orders. Means and standard variances are reported. "T" is the total number of tasks. "+Param" and "Tune Params" are additional parameters in total and tunable parameters for each task, respectively, measured by the ratio to the pre-trained model's parameters. We adjust *AdapterCL*'s parameters for each task to nearly 1x or 20x parameters of prompt tuning based methods.

- Our proposed *Prompt Tuning* with masked spans recovering is more parameter efficient than *AdapterCL*. In terms of Avg. JGA, *Prompt Tuning* is much better than *AdapterCL* with the same size and comparable to *AdapterCL* with 20x parameters.
- Forward transfer through CLInit and query fusion is effective for *Prompt Tuning*. *Continual Prompt Tuning* improves over *Prompt Tuning* significantly and outperforms baselines.
- When memory is available, our method achieves the best results w.r.t. all metrics, closing the gap between CL and multi-task learning. Memory improves zero-shot performance (FWT) on new tasks as *Replay* is better than *Fine-tuning* and *Continual Prompt Tuning w/ memory* is better than without memory.
- Our memory-guided backward transfer effectively utilizes subsequent tasks to help previous tasks. Although minor, *Continual Prompt Tuning w/ memory & backward* is the only method that exhibits positive BWT.

	MSR	CLInit	QF	MR	Avg. JGA	FWT
1					$29.6_{1.2}$	-
2		✓			$41.8_{2.8}$	$6.7_{0.3}$
3	✓				$48.1_{0.9}$	-
4	✓	✓			$57.6_{2.5}$	$9.6_{1.2}$
5	✓	✓	✓		$59.5_{1.4}$	$9.9_{0.7}$
6	✓	✓		✓	$60.4_{1.1}$	$11.9_{0.6}$
7	✓	✓	✓	✓	$60.7_{2.4}$	$13.7_{0.8}$

Table 2: Ablation study for masked spans recovering formulation (MSR), prompt initialization (CLInit or random), query fusion (QF) and memory replay (MR).

5.2 Ablation Study

To understand the effect of different proposed techniques, we conduct an in-depth ablation study and show the result in Table 2. Row 1 and 2 do not formulate DST as a masked spans recovering (MSR) task: the input is the concatenate of the dialog, service name, and soft prompt, while the output is a sequence of slot-value pairs as in *Fine-tuning* (Figure 2(a)). Several interesting observations can be noted: **First**, formulating DST as MSR is beneficial. Using MSR achieves better CL performance regardless of learning each task independently (row 3 v.s. row 1) or continually using CLInit (row 4 v.s. row 2). Besides, MSR formulation improves zero-shot generalization on new tasks (row 4 v.s. row 2). **Second**, forward transfer through CLInit brings large improvement for CL. CLInit outperforms random initialization greatly for both using MSR formulation (row 4 v.s. 3) and not (row 2 v.s. 1). **Third**, both query fusion and memory replay are effective. When they are used separately, memory replay (row 6) boosts the performance more than query fusion (row 5), while applying them altogether achieves the best performance (row 7).

5.3 Continual Prompt Initialization

In this experiment (Table 3), we compare CLInit with other prompt initialization strategies for *Prompt Tuning* in CL. SelectInit (see Sec. 3.3.1)

Initialization	Avg. JGA	FWT
Random	$48.1_{0.9}$	-
SelectInit	$54.5_{2.0}$	$8.2_{1.3}$
CLInit	$57.6_{2.5}$	$9.6_{1.2}$

Table 3: Comparison of different prompt initialization strategies for *Prompt Tuning*.

Training task sequence	Testing tasks $\mathcal{T}_{40:44}$	$\mathcal{T}_{30:44}$	$\mathcal{T}_{15:44}$
$\mathcal{T}_{40:44}$	45.1	-	-
$\mathcal{T}_{30:44}$	54.2	59.7	-
$\mathcal{T}_{15:44}$	59.0	64.4	64.3
$\mathcal{T}_{1:44}$	60.7	67.8	69.3

Table 4: *Prompt Tuning* with CLInit on the last 5, 15, 30, and 44 (all) tasks of the same task order. We report the Avg. JGA on the last 5, 15, and 30 tasks, respectively.

selects the prompt that has the best zero-shot performance on the current task from all previous tasks' prompts for initialization. We could see that both SelectInit and CLInit outperform random initialization significantly, demonstrating the effectiveness of transferring knowledge from previous tasks through prompt initialization. CLInit is slightly better than SelectInit in both Avg. JGA and zero-shot generalization (FWT), which reveals the benefit of accumulating knowledge from all seen tasks. In contrast, the prompt initialized by SelectInit has seen *fewer* tasks and thus contains less knowledge, which might explain the slightly worse result.

Based on the observation above, we further study that whether seeing *more* preceding tasks further helps CLInit. To this end, we choose a task order of all 44 tasks at random (see Table 8 in the Appendix) and perform *Prompt Tuning* with CLInit on the last 5, last 15, last 30, and all 44 tasks separately. Formally, we train on four CL curriculums $\mathcal{T}_{40:44}$, $\mathcal{T}_{30:44}$, $\mathcal{T}_{15:44}$, and $\mathcal{T}_{1:44}$, which have the same ending. We calculate the Avg. JGA on the $\mathcal{T}_{40:44}$, $\mathcal{T}_{30:44}$, and $\mathcal{T}_{15:44}$ if possible. As illustrated in Table 4, performance on the same tasks (in the same column) increases monotonously as the number of preceding tasks grows. This pattern validates that the benefit of CLInit becomes more evident as the number of tasks increases. This finding suggests that our method is suitable for long task sequences.

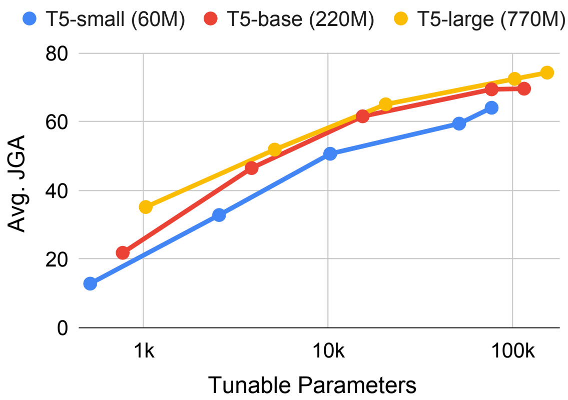

Figure 3: Avg. JGA for *Continual Prompt Tuning* with different pre-trained models and prompt lengths. The x-axis is the number of tunable parameters in log scale. The points on each curve correspond to 1, 5, 20, 100, and 150 prompt tokens from left to right.

	Prompt Length				
	1	5	20	100	150
T5-small (60M)	6.1	6.7	8.9	9.8	9.8
T5-base (220M)	5.7	9.9	12.9	18.3	15.0
T5-large (770M)	10.6	17.0	18.5	28.0	31.2

Table 5: FWT for *Continual Prompt Tuning* with different pre-trained models and prompt lengths.

5.4 Model Size and Prompt Length

In this experiment, we analyze the influence of pre-trained model size and prompt length. We vary the pre-trained model in {T5-small, T5-base, T5-large} and prompt length in {1, 5, 20, 100, 150} for *Continual Prompt Tuning* on the 15 tasks (the task order is in Table 9 in the Appendix). Figure 3 shows Avg. JGA and Table 5 shows FWT. We can observe that: **First**, when fixing the prompt length, increasing the model size improves the Avg. JGA as well as the generalization ability measured by FWT in most cases. **Second**, when the backbone model size is fixed, increasing the prompt length improves the overall performance in general. Furthermore, we found that increasing prompt token length from 20 to 100 improves Avg. JGA and FWT more than increasing it from 100 to 150, which is consistent with the finding in Lester et al. (2021). **Third**, our method becomes more parameter-efficient as the backbone model size grows. With the same number of tunable parameters (x-axis), using a larger pre-trained model achieves better Avg. JGA.

	Memory Size		
	10	50	100
Replay	$44.0_{1.0}$	$58.6_{3.5}$	$65.6_{0.8}$
CPT w/ mem.	$59.0_{3.3}$	$60.7_{2.4}$	$59.7_{3.2}$
CPT w/ mem. & back.	$58.6_{3.7}$	$61.2_{2.5}$	$60.4_{3.3}$
BWT	$-0.4_{0.5}$	$0.5_{0.4}$	$0.8_{0.4}$

Table 6: Avg. JGA for *Replay* and *Continual Prompt Tuning* (*CPT*) with memory replay (and memory-guided backward transfer) using different memory size. BWT for *CPT w/ mem. & back.* is also shown.

	Memory Size	
	fixed = 50	proportional
Replay	$58.6_{3.5}$	$55.8_{0.7}$
CPT w/ mem.	$60.7_{2.4}$	$60.3_{3.1}$
CPT w/ mem. & back.	$61.2_{2.5}$	$60.7_{3.4}$
BWT	$0.5_{0.4}$	$0.4_{0.5}$

Table 7: Avg. JGA for *Replay* and *Continual Prompt Tuning* (*CPT*) with memory replay (and memory-guided backward transfer) using the fixed/proportional memory size. The total memory sizes are the same. BWT for *CPT w/ mem. & back.* is also shown.

5.5 The Effect of Memory Size

In this section, we compare the role of memory in *Replay* and our method. We vary the memory size per task $|M|$ in {10, 50, 100} and show the performance of *Replay* and *Continual Prompt Tuning* with memory replay (and memory-guided backward transfer) in Table 6. We can find that increasing the memory size benefits *Replay* significantly. This is not surprising because *Replay* and other rehearsal methods rely on memory to solve the challenging forgetting problem. When the memory size is unlimited, *Replay* degenerates to multi-task learning, which is powerful but costly in storage and computation.

For *Continual Prompt Tuning*, however, the memory is not used for retaining the performance on previous tasks since parameters for previous tasks are saved.

- In forward transfer, the memory helps recall previous tasks' knowledge and serves as a complement to CLInit and query fusion. The influence on Avg. JGA depends on the effect of transfer learning on the current task via multi-task training ($\mathcal{L}_{\theta_{P_k}}(D_k + M_{<k})$). As shown in the row 2 in Table 6, increasing the memory size does not improve Avg. JGA significantly and may even distract the model from learning the current domain. This result suggests that our method does not need a large memory for forward transfer.
- In backward transfer, the memory gives reference gradients to guide the updates and serves as a filter to decide whether to accept the updates. Thus larger memory gives more accurate guidance. From the bottom row in Table 6, we can find that increasing memory size can improve the effect of backward transfer.

We also conduct experiments using a percentage memory budget, setting the memory size for each task proportional to task data size: $|M_i| \propto |D_i|$. This means low-resource tasks have fewer samples stored in the memory than in the original setting. We set the total memory size to 50 * T, where T is the number of tasks. As shown in Table 7, *Replay* performs much worse (58.6→55.8) in the unbalanced task memory setting while the effect on *Continual Prompt Tuning w/ mem.* is slight (60.7→60.3). Besides, our proposed backward transfer technique is still effective.

Overall, these results indicate that compared with *Replay*, our method uses the memory differently and benefits less from enlarging the memory.

6 Conclusion

In this paper, we develop prompt tuning for continual learning for the first time. We propose *Continual Prompt Tuning*, a highly parameter-efficient framework that avoids forgetting and enables forward/backward knowledge transfer among tasks. For forward transfer, we explore continual prompt initialization, query fusion, and memory replay techniques. For backward transfer, we devise a memory-guided technique. Extensive experiments on continual learning for DST demonstrate the effectiveness and efficiency of our proposed method compared with state-of-the-art baselines. Our method and findings will foster more future studies towards building more scalable, adaptable task-oriented dialog systems.

Acknowledgements

This work was supported by the National Science Foundation for Distinguished Young Scholars (with No. 62125604) and the NSFC projects (Key project with No. 61936010 and regular project with No. 61876096). This work was also supported by the Guoqiang Institute of Tsinghua University, with Grant No. 2019GQG1 and 2020GQG0005, and sponsored by Tsinghua-Toyota Joint Research Fund.

References

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D 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 Ziegler, Jeffrey Wu, Clemens Winter, Chris 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 *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc.

Arslan Chaudhry, Marc'Aurelio Ranzato, Marcus Rohrbach, and Mohamed Elhoseiny. 2019. Efficient lifelong learning with a-GEM. In *International Conference on Learning Representations*.

Shuyang Gao, Abhishek Sethi, Sanchit Agarwal, Tagyoung Chung, and Dilek Hakkani-Tur. 2019. Dialog state tracking: A neural reading comprehension approach. In *Proceedings of the 20th Annual SIGdial Meeting on Discourse and Dialogue*, pages 264–273, Stockholm, Sweden. Association for Computational Linguistics.

Tianyu Gao, Adam Fisch, and Danqi Chen. 2021. Making pre-trained language models better few-shot learners. 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 3816–3830, Online. Association for Computational Linguistics.

Binzong Geng, Fajie Yuan, Qiancheng Xu, Ying Shen, Ruifeng Xu, and Min Yang. 2021. Continual learning for task-oriented dialogue system with iterative network pruning, expanding and masking. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pages 517–523, Online. Association for Computational Linguistics.

Yuxian Gu, Xu Han, Zhiyuan Liu, and Minlie Huang. 2021. Ppt: Pre-trained prompt tuning for few-shot learning. *arXiv preprint arXiv:2109.04332*.

Xu Han, Weilin Zhao, Ning Ding, Zhiyuan Liu, and Maosong Sun. 2021. Ptr: Prompt tuning with rules for text classification. *arXiv preprint arXiv:2105.11259*.

Ehsan Hosseini-Asl, Bryan McCann, Chien-Sheng Wu, Semih Yavuz, and Richard Socher. 2020. A simple language model for task-oriented dialogue. In *Advances in Neural Information Processing Systems*, volume 33, pages 20179–20191. Curran Associates, Inc.

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*, volume 97 of *Proceedings of Machine Learning Research*, pages 2790–2799. PMLR.

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*, 8:423–438.

James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A. Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, Demis Hassabis, Claudia Clopath, Dharshan Kumaran, and Raia Hadsell. 2017. Overcoming catastrophic forgetting in neural networks. In *Proceedings of the National Academy of Sciences*, volume 114(13), pages 3521–3526. National Academy of Sciences.

Chia-Hsuan Lee, Hao Cheng, and Mari Ostendorf. 2021. Dialogue state tracking with a language model using schema-driven prompting. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 4937–4949, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Hwaran Lee, Jinsik Lee, and Tae-Yoon Kim. 2019. SUMBT: Slot-utterance matching for universal and scalable belief tracking. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 5478–5483, Florence, Italy. Association for Computational Linguistics.

Sungjin Lee. 2017. Toward continual learning for conversational agents. *arXiv preprint arXiv:1712.09943*.

Brian Lester, Rami Al-Rfou, and Noah Constant. 2021. The power of scale for parameter-efficient prompt tuning. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 3045–3059, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Zhizhong Li and Derek Hoiem. 2018. Learning without forgetting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40(12):2935–2947.

Zhaojiang Lin, Bing Liu, Seungwhan Moon, Paul Crook, Zhenpeng Zhou, Zhiguang Wang, Zhou Yu, Andrea Madotto, Eunjoon Cho, and Rajen Subba. 2021. Leveraging slot descriptions for zero-shot cross-domain dialogue StateTracking. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 5640–5648, Online. Association for Computational Linguistics.

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

David Lopez-Paz and Marc'Aurelio Ranzato. 2017. Gradient episodic memory for continual learning. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 6470–6479. Curran Associates Inc.

Andrea Madotto, Zhaojiang Lin, Zhenpeng Zhou, Seungwhan Moon, Paul Crook, Bing Liu, Zhou Yu, Eunjoon Cho, Pascale Fung, and Zhiguang Wang. 2021. Continual learning in task-oriented dialogue systems. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 7452–7467, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Arun Mallya, Dillon Davis, and Svetlana Lazebnik. 2018. Piggyback: Adapting a single network to multiple tasks by learning to mask weights. In *Proceedings of the European Conference on Computer Vision (ECCV)*.

Arun Mallya and Svetlana Lazebnik. 2018. Packnet: Adding multiple tasks to a single network by iterative pruning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

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

Fei Mi, Liangwei Chen, Mengjie Zhao, Minlie Huang, and Boi Faltings. 2020. Continual learning for natural language generation in task-oriented dialog systems. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 3461–3474, Online. Association for Computational Linguistics.

Fei Mi, Yitong Li, Yasheng Wang, Xin Jiang, and Qun Liu. 2022. Cins: Comprehensive instruction for few-shot learning in task-orienteddialog systems. In *Proceedings of the AAAI Conference on Artificial Intelligence*.

Nikola Mrkšić, Diarmuid Ó Séaghdha, Tsung-Hsien Wen, Blaise Thomson, and Steve Young. 2017. Neural belief tracker: Data-driven dialogue state tracking. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1777–1788, Vancouver, Canada. Association for Computational Linguistics.

Fabio Petroni, Tim Rocktäschel, Sebastian Riedel, Patrick Lewis, Anton Bakhtin, Yuxiang Wu, and Alexander Miller. 2019. Language models as knowledge bases? In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 2463–2473, Hong Kong, China. Association for Computational Linguistics.

Guanghui Qin and Jason Eisner. 2021. Learning how to ask: Querying LMs with mixtures of soft prompts. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 5203–5212, Online. Association for Computational Linguistics.

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. *Journal of Machine Learning Research*, 21(140):1–67.

Abhinav Rastogi, Xiaoxue Zang, Srinivas Sunkara, Raghav Gupta, and Pranav Khaitan. 2020. Towards scalable multi-domain conversational agents: The schema-guided dialogue dataset. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34(05), pages 8689–8696.

Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, Georg Sperl, and Christoph H. Lampert. 2017. icarl: Incremental classifier and representation learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Andrei A Rusu, Neil C Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. 2016. Progressive neural networks. *arXiv preprint arXiv:1606.04671*.

Timo Schick and Hinrich Schütze. 2021. It's not just size that matters: Small language models are also few-shot learners. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2339–2352, Online. Association for Computational Linguistics.

Yilin Shen, Xiangyu Zeng, and Hongxia Jin. 2019. A progressive model to enable continual learning for semantic slot filling. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 1279–1284, Hong Kong, China. Association for Computational Linguistics.

Taylor Shin, Yasaman Razeghi, Robert L. Logan IV, Eric Wallace, and Sameer Singh. 2020. AutoPrompt: Eliciting Knowledge from Language Models with Automatically Generated Prompts. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 4222–4235, Online. Association for Computational Linguistics.

Tu Vu, Brian Lester, Noah Constant, Rami Al-Rfou, and Daniel Cer. 2021. Spot: Better frozen model adaptation through soft prompt transfer. *arXiv preprint arXiv:2110.07904*.

Chien-Sheng Wu, Andrea Madotto, Ehsan Hosseini-Asl, Caiming Xiong, Richard Socher, and Pascale Fung. 2019. Transferable multi-domain state generator for task-oriented dialogue systems. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 808–819, Florence, Italy. Association for Computational Linguistics.

Puyang Xu and Qi Hu. 2018. An end-to-end approach for handling unknown slot values in dialogue state tracking. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1448–1457, Melbourne, Australia. Association for Computational Linguistics.

Yaoming Zhu, Jiangtao Feng, Chengqi Zhao, Mingxuan Wang, and Lei Li. 2021. Counter-interference adapter for multilingual machine translation. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pages 2812–2823, Punta Cana, Dominican Republic. Association for Computational Linguistics.

Task ID	Service	# Slots	# Dialogs			# Samples			Avg. tokens	
			Train	*Dev*	*Test*	*Train*	*Dev*	*Test*	*Context*	*Query*
1	events_3	5	53	7	16	312	40	105	121	47
2	banks_2	4	29	4	9	220	31	72	111	49
3	banks_1	4	144	21	42	1138	169	335	114	57
4	calendar_1	4	118	17	34	773	110	234	112	33
5	movies_3	3	33	5	10	112	18	37	72	26
6	music_2	5	231	33	67	1593	221	469	117	54
7	services_2	5	129	19	37	917	148	253	131	52
8	payment_1	4	25	3	8	233	33	89	171	52
9	media_1	4	196	28	57	1207	182	360	99	48
10	weather_1	2	58	8	17	259	39	66	77	16
11	events_1	6	202	29	58	1424	195	400	132	64
12	flights_4	7	60	9	18	290	41	87	90	77
13	travel_1	4	48	7	14	231	28	63	87	59
14	buses_2	6	111	16	32	857	120	234	137	54
15	events_2	6	400	57	115	3537	521	1067	159	59
16	alarm_1	2	58	9	17	367	49	107	101	22
17	buses_3	7	61	9	18	405	66	114	123	69
18	services_1	5	185	27	53	1241	180	352	129	58
19	buses_1	5	136	20	39	1054	143	313	138	49
20	restaurants_2	9	87	13	28	807	113	240	154	97
21	hotels_2	6	212	31	61	1569	234	460	152	73
22	ridesharing_2	3	64	9	19	380	49	108	106	34
23	rentalcars_1	6	100	14	29	840	120	242	161	59
24	movies_1	8	263	37	76	1873	250	556	122	70
25	ridesharing_1	3	74	10	22	412	57	125	103	36
26	media_3	4	56	8	16	327	42	89	95	36
27	music_3	6	17	3	5	112	19	32	114	60
28	movies_2	3	32	5	10	118	20	38	70	30
29	flights_2	7	129	19	37	822	115	251	127	75
30	services_4	5	86	13	25	680	97	208	154	49
31	flights_1	10	560	80	160	4680	667	1379	168	10
32	services_3	5	131	19	38	959	143	290	143	54
33	flights_3	8	65	10	19	420	75	116	133	79
34	trains_1	7	58	9	17	415	67	117	131	76
35	homes_2	8	62	9	18	424	56	139	140	89
36	rentalcars_2	6	77	11	23	631	91	185	157	61
37	restaurants_1	9	256	37	74	2098	297	581	153	10
38	music_1	6	68	10	20	468	73	142	118	61
39	hotels_4	7	80	12	23	559	99	141	134	72
40	media_2	5	32	4	10	215	29	71	112	59
41	hotels_3	6	90	13	26	737	100	193	157	64
42	rentalcars_3	7	44	7	13	332	55	99	148	72
43	hotels_1	7	99	14	29	868	105	250	161	71
44	homes_1	7	244	35	70	1829	282	540	159	81

Table 8: Statistics of the services we used. Average tokens of dialog context and query is calculated using T5 tokenizer. Services are arranged in the order of their appearance in our 44 task experiment (Sec. 5.3). Last 15 services are used for all our 15 task experiments.

Task order	Tasks' IDs in order														
Order1	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44
Order2	39	33	36	42	40	37	38	34	32	35	41	31	30	44	43
Order3	30	41	38	31	43	39	40	33	34	44	37	36	32	35	42
Order4	43	40	44	38	30	37	31	39	32	35	41	34	33	36	42
Order5	30	33	44	31	38	32	42	40	37	43	36	39	41	35	34

Table 9: Five task orders of all our 15 tasks experiments. We use last 15 tasks in Table 8. The task order for Section 5.4 is *Order1*.