# Task-wrapped Continual Learning in Task-Oriented Dialogue Systems

Min Zeng[1], Haiqin Yang[2*], Xi Chen[1], Yike Guo[1*]

[1]Hong Kong University of Science and Technology
[2]Independent Researcher
min.zeng.u@gmail.com, haiqin.yang@gmail.com

## Abstract

Continual learning is vital for task-oriented dialogue systems (ToDs), and AdapterCL, equipped with residual adapters, has proven effectiveness in this domain. However, its performance is limited by training separate adapters for each task, preventing global knowledge sharing. To address this, we propose **Task-wrapped Continual Learning (TCL)**, a novel framework that employs **Task-Wrapped Adapters (TWAs)**, to simultaneously learn both global and task-specific information through parameter sharing. TCL leverages task-conditioned hypernetworks to transfer global knowledge across tasks, enabling TWAs to start from more informed initialization, efficiently learning task-specific details while reducing model parameters. Additionally, the simple, linear structure of both hypernetworks and TWAs ensure stable training, with task-free inference supported through effective loss utilization. Across 37 ToD domains, TCL consistently outperforms AdapterCL, significantly reducing forgetting. Remarkably, by setting the task embedding dimension to 1, TCL achieves a 4.76% improvement over AdapterCL while using only 46% of the parameters. These findings position TWA as a lightweight, powerful alternative to traditional adapters, offering a promising solution for continual learning in ToDs. The code is available at https://github.com/cloversjtu/TCL.

## 1 Introduction

Task-oriented dialogue systems (ToDs) play a crucial role in natural language processing (NLP), enabling smart assistants to understand user intents and facilitate task completion through natural language interactions (Louvan and Magnini, 2020; Balaraman et al., 2021; Algherairy and Ahmed, 2025). These systems typically comprise multiple components, including natural language understanding (NLU), dialogue state tracking (DST), dialogue policy (DP) and natural language generation (NLG), or are implemented end-to-end. ToDs must be adaptable to new tasks and capable of continuous learning (Biesialska et al., 2020; Zeng et al., 2024). To meet this requirement, continual learning (CL) methodologies have emerged, allowing models to seamlessly integrate new knowledge over time (Mendez and Eaton, 2023). However, CL faces the challenge of *catastrophic forgetting* (CF), where performance on previously learned tasks deteriorates as new tasks are introduced, risking the erasure of acquired knowledge.

To mitigate CF, researchers have explored various approaches, including regularization-based, rehearsal-based, and architectural-based methods (Biesialska et al., 2020; Mendez and Eaton, 2023; Wang et al., 2024). Among these, architectural-based methods like AdapterCL (Madotto et al., 2021) have shown promise in ToDs. However, while effective, AdapterCL tends to prioritize task-specific information, potentially overlooking shared knowledge across tasks, and its linear resource expansion poses challenges for scalability.

To address these limitations, we propose **Task-wrapped Continual Learning (TCL)**, which introduces a novel, lightweight module called the **Task-Wrapped Adapter (TWA)** to jointly learn global and local information. TCL places a TWA atop each transformer layer of a pre-trained base model, allowing it to capture task-specific (local) insights while preserving core representations. Each TWA consists of three key layers: down-projection, up-projection, and normalization. Unlike conventional residual adapters (Houlsby et al., 2019), which initialize parameters randomly, TCL leverages two task-conditioned hypernetworks (Von Oswald et al., 2019) to initialize the weights and biases of the projection and normalization layers. Different from HyperFormer (Mahabadi et al., 2021), originally designed for multi-task learning and facilitating

*The corresponding author.



*Findings of the Association for Computational Linguistics:*
*NAACL 2025*, pages 3173–3183

multiple embeddings (task, adapter position, and layer ID), TCL employs only task embeddings to drive the initialization process. The parameters of both hypernetworks are trained across all tasks, capturing global knowledge, while the parameters of TWAs are initialized by the hypernetworks, enabling TWAs to start from more informed initialization to continually learn task-specific, fine-grained information.

TCL is highly parameter-efficient, achieving a 4.76% improvement over the state-of-the-art (SOTA) AdapterCL (Madotto et al., 2021) using only 46% of the parameters when the task embedding hidden size is set to 1 for the Intent Recognition (INTENT) task. Moreover, TCL demonstrates robust performance across multiple ToDs tasks over 37 domains, including INTENT (Casanueva et al., 2020; Coucke et al., 2018), DST (Mrkšić et al., 2016; Xu and Hu, 2018), NLG (Press et al., 2017), and building end-to-end (E2E) dialogue systems (Serban et al., 2016). Notably, TCL meets or even exceeds the multi-task upper bound in INTENT in high-capacity settings, benefiting from positive transfer across tasks.

In summary, our key contributions include:

- We propose **Task-wrapped Continual Learning (TCL)**, a novel framework employing TWAs to jointly learn both global and local information across tasks, addressing the limitations of existing CL methods.
- TCL leverages TWAs, whose parameters are initialized by two task-conditioned hypernetworks, enabling effective knowledge transfer across tasks, learning task-specific information from more well-informed starting points, and maintaining parameter efficiency through a lightweight design.
- Experimental results across 37 domains in ToDs show that TCL consistently outperforms the state-of-the-art AdapterCL. In particular, TCL attains a 4.76% improvement in INTENT utilizing only 46% of the parameters and demonstrates robust performance across tasks, including INTENT, DST, NLG, and E2E dialogue systems.

## 2 Related Work

Continual learning (CL) aims to learn from new tasks while retaining knowledge of previously learned ones (Wang et al., 2024; Kirkpatrick et al., 2017). Approaches to address CF in CL generally fall into three categories: regularization-based, rehearsal-based, and architectural-based methods.

*Regularization*-based methods constrain model updates to protect important parameters from previous tasks. For example, Elastic Weight Consolidation (EWC) (Kirkpatrick et al., 2017) prevents significant updates to crucial parameters to maintain earlier performance. Adaptively Regularized Prioritized Exemplar Replay (ARPER) (Mi et al., 2020) combines prioritized replay with EWC-based adaptive regularization to mitigate forgetting.

*Rehearsal*-based methods store examples from past tasks in memory and replay them during the new task training. For example, ICaRL (Rebuffi et al., 2017) applies clustering techniques to manage memory constraints and select key examples. Gradient episodic memory (GEM) (Lopez-Paz and Ranzato, 2017) calculates losses from stored examples and prevents the loss increase when updating. A-GEM (Chaudhry et al., 2018) refines GEM by constraining the average loss across previous tasks rather than addressing them individually. LAMOL (Sun et al., 2019) employs a language model to generate pseudo-samples, avoiding the need for memory storage. In DST, Cho et al. (2023) reformulates the task into example-guided question-answering and deploys dialogue-level replay to enhance continual learning performance. DCL (Zeng et al., 2024) leverages a Dirichlet distribution-based CVAE (Zeng et al., 2019), which offers greater flexibility in modeling utterance-level characteristics and generates more realistic pseudo samples compared to the conventional Gaussian-based CVAE.

*Architectural*-based methods modify the network by adding task-specific parameters to capture unique local information and reduce CF. For example, AdapterCL (Madotto et al., 2021) introduces residual adapters (Houlsby et al., 2019) atop transformer layers to approximate tasks. CPT4DST (Zhu et al., 2022) employs prompt tuning for DST to reduce forgetting. ACM (Zhang et al., 2022) reduces adapter numbers by incorporating LAMOL, but increasing training time in the two-stage procedure. O-LoRA (Wang et al., 2023) learns tasks in orthogonal low-rank subspaces to minimize interference without requiring data storage. MetaLTDS (Xu et al., 2023) combines architectural and rehearsal techniques, using task-specific masks and uncertainty-based sampling for memory selection. SAPT (Zhao et al., 2024) integrates adapters and shared attention to enhance knowledge transfer.

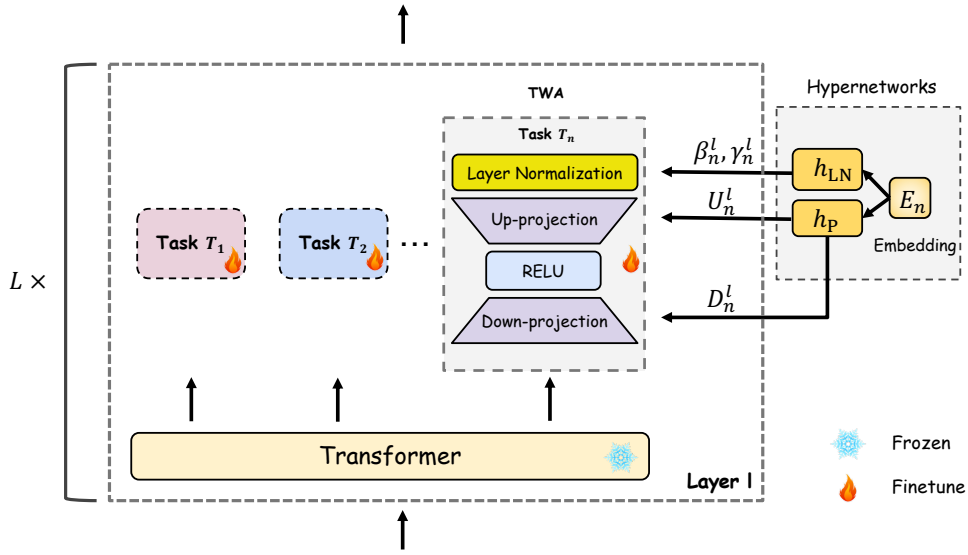


Figure 1: Illustration of TCL at layer $l$ for Task $T_n$: TCL places a TWA atop each transformer layer, which has fixed pre-trained parameters and only fine-tunes the parameters in TWAs and two conditioned hyperparameters. Each TWA is initialized by the hypernetworks, whose parameters are driven by the task embeddings, and learned incrementally to capture both global and task-specific information. See the main text for more details.

Given the effectiveness of AdapterCL in mitigating forgetting in ToDs, there is strong motivation to further optimize its efficiency and task generalization capabilities.

## 3 Methodology

### 3.1 Task Definition

Given a sequence of $N$ tasks, $T_1, \ldots, T_N$, each task, $T_n$ consists of $N_n$ samples in $\mathcal{D}_n = \{(x_i, y_i)\}_{i=1}^{N_n}$, where $(x_i, y_i)$ represent a general input-output pair. While learning on task $T_n (n > 2)$, we have no access to examples from previous tasks. The final goal is to train a model $f_\theta$, parameterized by $\theta$, to predict the output response $y$ such that $y = f_\theta(x)$ and optimize the model's average performance on all tasks after training on the entire sequence of tasks. In this paper, we focus on tasks in task-oriented dialogue systems; more detailed task definitions are provided in Appendix A.

### 3.2 Overview

As illustrated in Fig. 1, TCL employs a pre-trained language model as its backbone and integrates a Task-Wrapped Adapter (TWA), a novel and lightweight module, atop each transformer layer (with $L = 12$ for GPT-2) to absorb both global and local information. TWA consists of three core layers: down-projection, up-projection, and normalization. Traditional residual adapters train each adapter independently (Madotto et al., 2021; Houlsby et al., 2019), missing the opportunity to leverage global knowledge shared across tasks. In contrast, TCL utilizes TWA, which relies on two linear task-conditioned hypernetworks–one for the projection layers and another for the normalization layer–to initialize their parameters, thereby enhancing the model's ability while reducing the number of parameters.

In summary, TCL offers several key advantages: (1) **A unified framework for capturing global and local information:** The hypernetworks' parameters are solely driven by task embeddings and are learned incrementally, enabling TCL to efficiently absorb the global information across tasks. Subsequently, the task-conditioned hypernetworks globally initialize the parameters of TWAs at each layer, transferring the global information into TWAs for each task. This strategy allows TWAs to continually fine-tune from more informed starting points and adapt to absorb fine-grained and task-specific (local) information, ensuring that each task benefits from distinct, tailored TWA representations. (2) **Parameter efficiency and training stability:** By reusing hypernetworks, TCL significantly reduces the number of parameters required. Additionally, the linear structure of both the hypernetworks and TWAs ensures stable training while effectively capturing available information. Training stability is crucial in continual learning, where models are updated incrementally.

In the following, we outline the structure of TWAs, the training process, and the task-free inference procedure in TCL.

### 3.3 TWA and Parameter Initialization

**Architecture** Building on the residual adapter concept from AdapterCL (Madotto et al., 2021), we place TWA atop each transformer to jointly learn both global and local task-specific information. TWA consists of three core layers: down-projection, up-projection, and normalization.

Given an input hidden state $\hat{x}_i \in \mathbb{R}^h$ (where $h$ is the input dimension), the output of TWA at layer $l$ for task $T_n$, denoted as $A_n^l(\cdot)$, is computed by:

$$A_n^l(\hat{x}_i) = \text{LN}_n^l(U_n^l(\text{GELU}(D_n^l(\hat{x}_i)))) + \hat{x}_i, \quad (1)$$

where $D_n^l \in \mathbb{R}^{h\times b}$ and $U_n^l \in \mathbb{R}^{b\times h}$ represent the corresponding parameters for the down-projection and up-projection layers, respectively, and $\text{LN}_n^l(\cdot)$ refers to layer normalization. Here, $b$ denotes the bottleneck dimension, and GELU is the activation function. The initialization of $D_n^l$ and $U_n^l$ and those in $\text{LN}_n^l$ are detailed below.

In summary, TWA differs from AdapterCL in two key aspects: (1) **Parameter initialization:** In AdapterCL, the weights and biases of residual adapters for each task and layer are initialized randomly. In contrast, the parameters of TWAs are initialized using two task-conditioned, globally learned hypernetworks (Von Oswald et al., 2019), allowing TWAs to begin continual learning from more informed starting points, with global knowledge already embedded. (2) **Normalization procedure:** TWA employs post-normalization (Post-N) after the up-projection layer, whereas AdapterCL applies pre-normalization (Pre-N) before the down-projection layer. Post-normalization provides more controllable outputs for the subsequent layer, potentially improving performance (He et al., 2020).

**Layer Normalization in TWA** Following conventional layer normalization (Ba et al., 2016), the layer normalization in TWA is defined as:

$$\text{LN}_n^l(\hat{x}_i) = \gamma_n^l \odot \frac{\hat{x}_i - \mu_n}{\sigma_n} + \beta_n^l, \quad (2)$$

where $\odot$ is the element-wise multiplication between two vectors, and $\mu_n$ and $\sigma_n$ denote the mean and standard deviation of the data in task $T_n$, respectively. The learnable parameters $\gamma_n^l$ and $\beta_n^l$ at layer $l$ have the same dimension as $\hat{x}_i$, representing the scale and shift, respectively. The mechanism for their initialization is explained below.

**Hypernetworks for Initialization** To initialize the parameters in the three core layers of a TWA, we employ two task-conditioned hypernetworks: one for the projection layers and another for the normalization layer. A hypernetwork is an auxiliary network that generates weights for other networks, offering rapid adaptation and robust performance across diverse tasks (Ha et al., 2016). While this setup is inspired by HyperFormer (Mahabadi et al., 2021), it differs in two key aspects: (1) HyperFormer is designed for multi-task learning, whereas our TCL is tailored for continual learning. (2) TCL simplifies the hypernetwork configuration, driven solely by task embeddings with a linear structure, enabling incremental learning across tasks. This design allows TCL to efficiently capture global information within the hypernetwork parameters while reducing the number of learnable parameters.

Let $E_n \in \mathbb{R}^d$ ($d$ is the task embedding dimension) be the task embedding for task $T_n$. We initialize it from a standard normal distribution.

The parameters for the projection layers are initialized by a hypernetwork, $h_{\text{P}}(\cdot)$, defined as:

$$(U_n^l, D_n^l) = h_{\text{P}}(E_n) = (W^U, W^D)E_n, \quad (3)$$

where $W^U \in \mathbb{R}^{(b\times h)\times d}$ and $W^D \in \mathbb{R}^{(h\times b)\times d}$ are the hypernetwork's parameters, globally learned across tasks, to initialize the up-projection and down-projection layers in TWA, respectively. These parameters serve to ensure that TWAs benefit from global task knowledge while maintaining the flexibility to adapt to task-specific nuances.

The hypernetwork for the normalization layer, $h_{\text{LN}}(\cdot)$, initializes the scale and shift parameters as:

$$(\gamma_n^l, \beta_n^l) = h_{\text{LN}}(E_n) = (W^\gamma, W^\beta)E_n, \quad (4)$$

where $W^\gamma \in \mathbb{R}^{h\times d}$ and $W^\beta \in \mathbb{R}^{h\times d}$ represent the hypernetwork parameters to initialize the scaling and shifting parameters at each normalization layer.

The two task-conditioned hypernetworks offer the following advantages: (1) Both hypernetworks are linear networks, driven solely by task embeddings, which ensures simplicity and stability during training. (2) The parameters of these hypernetworks are shared and learned across all tasks, enabling them to capture global information efficiently. The global initialization allows TWAs can begin from well-informed starting points and progressively absorb task-specific information.

### 3.4 Model Training

Let $\theta_n$ be the learnable parameters of TCL for task $T_n$:

$$\theta_n = \{E_n, W^U, W^D, W^\gamma, W^\beta, \{U_n^l\}_{l=1}^L, \{D_n^l\}_{l=1}^L, \{\gamma_n^l\}_{l=1}^L, \{\beta_n^l\}_{l=1}^L\}, \quad (5)$$

where $E_n$ is the task embedding, $W^U$, $W^D$, $W^\gamma$, and $W^\beta$ are the hypernetwork parameters, and $U_n^l$, $D_n^l$, $\gamma_n^l$, and $\beta_n^l$ ($l = 1, \ldots, L$) are the task-specific parameters for each layer $l$ in TWA.

Given an input utterance $x$, we simplify its expansion as $x = x_1 \ldots x_M$ and define the corresponding loss for task $T_n$ as:

$$L_n(x) = -\sum_{j=1}^{M} \log p_{\theta_n}(x_j | x_{<j}), \quad (6)$$

where $M$ is the number of tokens. We then expand this loss to the $(x, y)$-pair as $L_n(x \odot y)$, where $\odot$ denotes the concatenation of $x$ and $y$.

Given the training data $\mathcal{D}_n$ from task $T_n$, TCL is trained to learn the parameters $\theta_n$ by minimizing the following loss:

$$L_n(\mathcal{D}_n) = -\sum_{i=1}^{N_n} L_n(x_i \odot y_i), \quad (7)$$

where the sequence of $x \odot y$ is padded to ensure uniform length if necessary.

### 3.5 Task-free Inference

Subsequent methods (Zhu et al., 2022; Zhang et al., 2022) developed after AdapterCL (Madotto et al., 2021) typically assume the task-ID is provided during inference to reduce task complexity. However, in real-world scenarios, the task-ID is usually unknown, and labeling can be expensive. To better mimic real-world situations, we predict the task-ID before generating the output. Specifically, given an input utterance $x$, we determine the task-ID by selecting the one with the least loss as

$$\hat{n} = \arg\min_n L_n(x), \quad (8)$$

where $L_n$ is defined in Eq. (6).

For example, given an input utterance $x$ (e.g., "`I'm looking for a restaurant in San Francisco`"), TCL enumerates all $L_n$ on $x$, calculating the losses by Eq. (6), and yields results: ("`TM19_coffee`", 1.8), ("`TM19_restaurant`", 0.12), ..., ("`MWOZ_train`", 2.1). TCL then identifies the task as "`TM19_restaurant`" for $x$ because it has the least loss.

## 4 Experiments

### 4.1 Datasets

Following the datasets used in AdapterCL, as described in Appendix B, we conduct experiments on four datasets spanning 37 domains: Task-Master 2019 (TM19) (Byrne et al., 2019), Task-Master 2020 (TM20) (Byrne et al., 2019), Schema Guided Dialogue (SGD) (Rastogi et al., 2020), and Multi-domain WoZ (MultiWOZ) (Budzianowski et al., 2018). These datasets are continually learned in the E2E setting. To ensure a fair comparison, we follow the experimental setup used in AdapterCL (Madotto et al., 2021) and create 5 learning curriculum by randomly permuting the 37 domains. The datasets' statistics and descriptions are provided in Appendix C.

### 4.2 Baselines

To demonstrate the efficiency of our method, we conduct a comparative analysis with the following strong baselines: (1) **Finetune** (Yogatama et al., 2019): Directly fine-tunes on sequential tasks without addressing catastrophic forgetting; (2) **L2** (Yoon et al., 2017): Regularizes the network with a fixed quadratic constraint for each weight; (3) **EWC** (Kirkpatrick et al., 2017): Imposes constraints on the loss to minimize updates of crucial parameters in previous tasks; (4) **A-GEM** (Chaudhry et al., 2018): Applies gradient-based constraints, effectively preserving knowledge of previous tasks; (5) **LAMOL** (Sun et al., 2019): Uses the language model as both a learner and generator, replaying pseudo-samples from previous tasks; (6) **AdapterCL** (Madotto et al., 2021): An architectural method that inserts residual adapters (Houlsby et al., 2019) atop each transformer layer, training task-specific adapters; (7) **O-LoRA** (Wang et al., 2023): A parameter-efficient architectural method that learns tasks in different low-rank vector subspaces ensuring orthogonality to reduce catastrophic forgetting; (8) **SAPT** (Zhao et al., 2024): Incorporates an adapter for each new task and leverages a shared attention framework to enhance knowledge transfer across tasks; (9) **MULTI** (Caruana, 1997): Multi-task learning, serving as the upper bound.

### 4.3 Experimental Settings

Following the setup in AdapterCL (Madotto et al., 2021), we apply the same hyperparameters to guarantee a fair comparison. The batch size is set to

| Models | INTENT *Accuracy* ↑ | DST *JGA* ↑ | NLG *EER* ↓ | NLG *BLEU* ↑ |
|---|---|---|---|---|
| Finetune (Yogatama et al., 2019) | 4.08 ± 1.4 | 4.91 ± 4.46 | 48.73± 3.81 | 6.38 ± 0.6 |
| L2 (Yoon et al., 2017) | 3.74 ± 1.4 | 3.81 ± 3.44 | 55.68 ± 7.09 | 5.4 ± 0.9 |
| EWC (Kirkpatrick et al., 2017) | 3.95 ± 1.3 | 5.22 ± 4.46 | 58.2 ± 3.66 | 5.06 ± 0.5 |
| A-GEM (Chaudhry et al., 2018) | 34.04 ± 6.36 | 6.37 ± 4.0 | 62.09 ± 6.88 | 4.54 ± 0.6 |
| LAMOL (Sun et al., 2019) | 7.49 ± 6.35 | 4.55 ± 3.48 | 66.11 ± 6.97 | 3.0 ± 0.9 |
| AdapterCL (Madotto et al., 2021) | 90.46 ± 0.6 | **35.06**± 0.52 | 31.78 ± 1.28 | 16.76 ± 0.34 |
| O-LoRA (Wang et al., 2023) | 32.26 ± 8.35 | 11.61 ± 1.29 | 47.75 ± 0.69 | 14.6 ± 0.72 |
| SAPT (Zhao et al., 2024) | 53.98 ± 3.86 | 22.17 ± 0.79 | 48.63 ± 0.68 | 14.47 ± 0.28 |
| TCL | **95.22** ± 0.6 | 33.45 ± 0.59 | **28.64** ± 1.29 | **17.85** ± 0.41 |
| MULTI (Upper Bound) (Caruana, 1997) | 95.45 ± 0.1 | 48.9 ± 0.2 | 12.56 ± 0.2 | 23.61 ± 0.1 |

Table 1: Comparison results of TCL and baselines. The best results are highlighted in bold.

10, and the learning rate is 6.25e-3. The gradient accumulation step is 8 and AdamW (Loshchilov and Hutter, 2017) is employed as the optimizer. The hidden state dimension is 768, the task embedding dimension is 1, and the bottleneck size is 24. All experiments are conducted on an NVIDIA A100-80G GPU. We train TCL for 10 epochs, with a training time of approximately 24 hours.

### 4.4 Evaluation Metrics

We employ four well-defined metrics to evaluate model performance (Madotto et al., 2021):

- **Accuracy** measures the exact match between generated intents and the gold labels, evaluating the model's performance on the INTENT recognition task.
- **Average Joint Generalization Accuracy (JGA)** (Wu et al., 2019) is the primary evaluation metric in DST, assessing the correctness of slot-value pairs across all tasks. It is calculated as $\text{JGA} = \frac{1}{N}\sum_{i=1}^{N} R_{N,i}$, where $R_{i,j}$ is the evaluation metric achieved on task $T_j$ after training on task $T_i$.
- Slot **Error Rate (EER)** (Wen et al., 2015) measures the ratio of missing slots in the response to the total number of slots in NLG.
- **BLEU** (Papineni et al., 2002) evaluates the similarity between generated texts and the reference texts based on n-gram overlap in NLG. Higher BLEU scores indicate better performance.

## 5 Results and Analysis

### 5.1 Overall Evaluation Results

To assess the overall performance of our proposed TCL across all tasks, we conduct experiments in the E2E setting and evaluate performance on three specific tasks: INTENT, DST, NLG. The average score across all tasks is adopted as an overarching metric for reliability consideration. Results in Table 1 show that

| Models | Size | Task Dim. | INTENT *Accuracy* ↑ |
|---|---|---|---|
| AdapterCL | 346M (100%) | - | 90.46± 0.6 |
| TCL | 162M (46%) | 1 | 95.22± 0.6 |
| TCL | 218M (63%) | 4 | 95.49± 0.52 |
| TCL | 293M (85%) | 8 | **95.55**± 0.31 |

Table 2: Comparison results of TCL and AdapterCL in INTENT with varying task embedding dimensions (in short Task Dim.).

- TCL significantly surpasses the state-of-the-art (SOTA) model, AdapterCL: (1) In INTENT, TCL improves upon AdapterCL by 4.76%; (2) In DST, both TCL and AdapterCL show significant improvement over the baselines. However, TCL's JGA slightly trails behind AdapterCL due to TCL's tendency to generate more diverse and comprehensive responses. For example, the predicted response: "`FindTrains (date_of_journey=12th of this month, journey_start_time=6:20am)`" contains the correct intent. However, it is marked incorrect for JGA since the golden label, "`FindTrains (date_of_journey=12th of this month)`", does not include the second slot-value pair, "(journey_start_time=6:20am), which is also correct from the context. JGA requires exact matches of slot-value pairs, treating minor discrepancies as errors. Thus, although TCL may produce more informative responses, it may fail to achieve perfect alignment with the golden truth. (3) For NLG, TCL outperforms AdapterCL

| Model | Bottleneck | Size | INTENT *Accuracy* ↑ | DST *JGA* ↑ | NLG *EER* ↓ | NLG *BLEU* ↑ |
|---|---|---|---|---|---|---|
| AdapterCL | 24 | 162M (46%) | 89.91± 0.23 | 29.47± 0.58 | 32.51± 0.97 | 15.51± 0.45 |
| TCL | 24 | 162M (46%) | 95.22± 0.6 | 33.45± 0.59 | 28.64± 1.29 | 17.85± 0.41 |
| TCL | 32 | 173M (50%) | 95.45± 0.18 | 33.5± 0.36 | 28.15± 0.43 | 17.63± 0.19 |
| TCL | 48 | 197M (57%) | **95.75**± 0.42 | **35.24**± 0.51 | **26.1**± 0.38 | **17.88**± 0.23 |

Table 3: Comparison of AdapterCL (bottleneck size 24) and TCL with varying bottleneck dimensions ($b$ = 24, 32, and 48)

with improvements of 3.14% in EER and 1.09 in BLEU. Lower EER indicates more comprehensive and complete generated responses, with fewer missing slots and values. A higher BLEU score reflects better alignment between generated and reference text, with the 1.09 increase highlighting TCL's strength in generating linguistically accurate and content-rich responses.

- TCL achieves an accuracy of 95.22%, closely approaching the upper bound set by MULTI at 95.45%. This remarkable performance is achieved without seeing all available data simultaneously, highlighting TCL's potential for continual learning.

### 5.2 Ablation Study

**Dimension of Task Embedding** We conduct experiments in INTENT with varying task embedding dimensions in the hypernetworks to evaluate parameter efficiency. The results, reported in Table 2, show that: (1) TCL consistently outperforms AdapterCL across all settings, with task embedding dimensions of 1, 4, and 8. As the dimension increases, TCL's performance gradually improves, attributed to the increased information carried by larger task embeddings. (2) TCL yields even better performance than MULTI (95.49% and 95.55% vs. 95.45%) when the dimension is set to 4 and 8, respectively. (3) When the dimension is set to 1, TCL contains 162M parameters, only 46% of AdapterCL's 346M. This demonstrates TCL's higher computational efficiency, driven by its ability to share information across tasks.

**Dimension of Bottleneck** Table 3 reports the results assessing the effect of the bottleneck size in {24, 32, 48}. The findings show that as the bottleneck size increases, TCL's performance improves accordingly with a slight increase in model size. Notably, when the bottleneck size is 48, TCL attains the best performance in all cases, compared to a bottleneck size of 24: In INTENT, a 0.53% improvement in accuracy; In DST, a 1.79% improvement in JGA; and in NLG, a 2.54% decrease in EER and 0.03 improvement BLEU score. This suggests that larger bottleneck sizes allow TCL to capture more information, enhancing performance. Nevertheless, a trade-off exists between performance and computational cost. Overall, our TCL attains favorite performance when the size is 24, a small scale.

| Models | task-ID | INTENT *Accuracy* ↑ |
|---|---|---|
| AdapterCL | without | 90.46± 0.60 |
| TCL | without | 95.22± 0.60 |
| TCL | with | **95.56**± 0.25 |

Table 4: Comparison results of TCL in INTENT with or without given task-ID during inference.

**Task-free Inference** Table 4 shows the impact of with or without task-ID during inference. The results show that when the task-ID is not provided, TCL consistently outperforms AdapterCL in INTENT, demonstrating TCL's superior model capability. Notably, performance further improves when the task-ID is given. When TCL is equipped with a task-ID during inference, the accuracy in INTENT reaches 95.56%, reflecting a 0.34% improvement compared to scenarios without providing the task-ID. This underscores TCL's versatility in handling both scenarios and highlights its potential for real-world applications.

| Models | INTENT *Accuracy* ↑ | DST *JGA* ↑ | NLG *EER* ↓ | NLG *BLEU* ↑ |
|---|---|---|---|---|
| TCL (Pre-N) | 94.96± 0.22 | **33.7**± 0.43 | **28.28**± 1.36 | 17.74± 0.57 |
| TCL (Post-N) | **95.22**± 0.60 | 33.45± 0.59 | 28.64± 1.29 | **17.85**± 0.41 |

Table 5: Comparison of TCL with Pre-N vs. Post-N.

**Pre-N vs. Post-N** We conduct experiments to assess the impact of Pre-N vs. Post-N configurations in TCL. Table 5 shows that TCL with Post-N performs better in INTENT and NLG (BLEU) while

| **Ex.1:** | **USER: I'd like to see a romantic comedy.** |
|---|---|
| Reference | movie_search (genre="romantic comedy.") |
| AdapterCL | movie_ticket (name_movie="a romantic comedy") |
| TCL | movie_search (genre="romantic comedy") |
| **Ex.2:** | **Perfect and do you know where the shop is located? USER: Yes, I do. API-OUT:** |
| Reference | Thanks for confirming. Please note There is an $89 fee for inspection that could go towards repair if you choose to use their services. |
| AdapterCL | great. I've booked your appointment. do you know where the shop is located? |
| TCL | ok, I'll book that appointment. please note that there is a $89 service fee but that can be applied to the cost of repairs. |

Table 6: Comparison of generated outputs for TCL and AdapterCL with the Ground Truth (Reference).

showing a slight lag in DST and NLG (EER) compared to Pre-N. It is important to note that although Post-N shows lower JGA compared to Pre-N in DST, this does not necessarily indicate poor performance. Instead, it might be attributed to TCL's preference for generating more comprehensive responses when using Post-N. Similar issues occur for evaluating EER in NLG.

### 5.3 Case Study

Table 6 illustrates two cases to compare the generated results of TCL and AdapterCL with respect to the ground truth (reference).

In Ex. 1, given the input utterance "`USER: I'd like to see a romantic comedy.`", TCL accurately predicts the correct task-ID, "`movie_search`", and generates the corresponding output, "`genre = romantic comedy`". As a comparison, AdapterCL wrongly predicts the task-ID to "`movie_ticket`" and yields a wrong slot, "`name_movie`", though the filling "`a romantic comedy`" is partially correct. This implies a failure of AdapterCL to capture the true intent.

In Ex. 2, given the input utterance, "`Perfect and do you know where the shop is located? USER: Yes, I do. API-OUT`", AdapterCL encounters challenges in accurately generating output response. The generated response "`do you know where the shop is located?`" by AdapterCL is redundant since the user already provided in the input utterance. As a comparison, TCL accurately confirms the appointment and reminds that "`there is $89 service fee`", which adds information that this fee can be applied to the cost of repairs. This example highlights TCL's capability of providing more relevant and informative responses.

## 6 Conclusion

In this paper, we propose Task-wrapped Continual Learning (TCL), a novel continual learning framework to mitigate forgetting in Task-oriented Dialogue systems (ToDs) by utilizing Task-wrapped Adapters (TWAs) to capture both global and task-specific information. TCL employs two task-conditioned hypernetworks, driven by task embeddings, to initialize the parameters of TWAs. The hypernetwork parameters are shared across tasks and learned incrementally, allowing TCL to absorb the global information while reducing the model size. Meanwhile, TWAs continually learn from the globally initialized parameters, providing more informed starting points to capture task-specific information. The simple structure of both hypernetworks and TWAs ensures stable training in TCL, while the effective loss function supports task-free inference, favoring real-world applications. Experimental results across 37 domains in ToDs show that TCL consistently outperforms the state-of-the-art AdapterCL. Notably, TCL achieves a 4.76% improvement in INTENT while utilizing only 46% of the parameters, and demonstrates robust performance across all tested ToDs tasks.

## 7 Limitations

The limitations of this work can be summarized as follows:

1. **Scope of applications:** Our experiments are confined to Task-oriented Dialogue systems, in line with the scope of existing baselines. Future work should explore the generalizability of our TCL by applying it to a broader range of applications.
2. **Architecture variation:** Within our TCL, we utilize straightforward linear network architec-

tures for both hypernetworks and TWAs. Despite achieving significant improvements with a stable training process, it would be valuable to investigate the performance of TCL with more complex network architectures.

## References


Atheer Algherairy and Moataz Ahmed. 2025. Prompting large language models for user simulation in task-oriented dialogue systems. *Comput. Speech Lang.*, 89:101697.

Lei Jimmy Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. 2016. Layer normalization. *CoRR*, abs/1607.06450.

Vevake Balaraman, Seyedmostafa Sheikhalishahi, and Bernardo Magnini. 2021. Recent neural methods on dialogue state tracking for task-oriented dialogue systems: A survey. In *Proceedings of the 22nd Annual Meeting of the Special Interest Group on Discourse and Dialogue, SIGdial 2021, Singapore and Online, July 29-31, 2021*, pages 239–251. Association for Computational Linguistics.

Magdalena Biesialska, Katarzyna Biesialska, and Marta R. Costa-jussà. 2020. Continual lifelong learning in natural language processing: A survey. In *Proceedings of the 28th International Conference on Computational Linguistics, COLING 2020, Barcelona, Spain (Online), December 8-13, 2020*, pages 6523–6541. International Committee on Computational Linguistics.

Paweł Budzianowski, Tsung-Hsien Wen, Bo-Hsiang Tseng, Inigo Casanueva, Stefan Ultes, Osman Ramadan, and Milica Gašić. 2018. Multiwoz–a large-scale multi-domain wizard-of-oz dataset for task-oriented dialogue modelling. *arXiv preprint arXiv:1810.00278*.

Bill Byrne, Karthik Krishnamoorthi, Chinnadhurai Sankar, Arvind Neelakantan, Daniel Duckworth, Semih Yavuz, Ben Goodrich, Amit Dubey, Andy Cedilnik, and Kyu-Young Kim. 2019. Taskmaster-1: Toward a realistic and diverse dialog dataset. *arXiv preprint arXiv:1909.05358*.

Rich Caruana. 1997. Multitask learning. *Machine learning*, 28:41–75.

Iñigo Casanueva, Tadas Temčinas, Daniela Gerz, Matthew Henderson, and Ivan Vulić. 2020. Efficient intent detection with dual sentence encoders. *arXiv preprint arXiv:2003.04807*.

Arslan Chaudhry, Marc'Aurelio Ranzato, Marcus Rohrbach, and Mohamed Elhoseiny. 2018. Efficient lifelong learning with a-gem. *arXiv preprint arXiv:1812.00420*.

Hyundong Cho, Andrea Madotto, Zhaojiang Lin, Khyathi Raghavi Chandu, Satwik Kottur, Jing Xu, Jonathan May, and Chinnadhurai Sankar. 2023. Continual dialogue state tracking via example-guided question answering. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, EMNLP 2023, Singapore, December 6-10, 2023*, pages 3873–3886. Association for Computational Linguistics.

Alice Coucke, Alaa Saade, Adrien Ball, Théodore Bluche, Alexandre Caulier, David Leroy, Clément Doumouro, Thibault Gisselbrecht, Francesco Caltagirone, Thibaut Lavril, et al. 2018. Snips voice platform: an embedded spoken language understanding system for private-by-design voice interfaces. *arXiv preprint arXiv:1805.10190*.

David Ha, Andrew Dai, and Quoc V Le. 2016. Hypernetworks. *arXiv preprint arXiv:1609.09106*.

Ruining He, Anirudh Ravula, Bhargav Kanagal, and Joshua Ainslie. 2020. Realformer: Transformer likes residual attention. *arXiv preprint arXiv:2012.11747*.

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 *Proc. Intl. Conf. Machine Learning (ICML)*, pages 2790–2799.

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

David Lopez-Paz and Marc'Aurelio Ranzato. 2017. Gradient episodic memory for continual learning. In *Proc. Advances in Neural Information Processing Systems (NIPS)*, volume 30.

Ilya Loshchilov and Frank Hutter. 2017. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*.

Samuel Louvan and Bernardo Magnini. 2020. Recent neural methods on slot filling and intent classification for task-oriented dialogue systems: A survey. In *Proceedings of the 28th International Conference on Computational Linguistics, COLING 2020, Barcelona, Spain (Online), December 8-13, 2020*, pages 480–496. International Committee on Computational Linguistics.

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 *Proc. Conf. on Empirical Methods in Natural Language Processing (EMNLP)*.

Rabeeh Karimi Mahabadi, Sebastian Ruder, Mostafa Dehghani, and James Henderson. 2021. Parameter-efficient multi-task fine-tuning for transformers via shared hypernetworks. *arXiv preprint arXiv:2106.04489*.

Jorge A. Mendez and Eric Eaton. 2023. How to reuse and compose knowledge for a lifetime of tasks: A survey on continual learning and functional composition. *Trans. Mach. Learn. Res.*, 2023.

Fei Mi, Liangwei Chen, Mengjie Zhao, Minlie Huang, and Boi Faltings. 2020. Continual learning for natural language generation in task-oriented dialog systems. In *Proc. Conf. on Empirical Methods in Natural Language Processing (EMNLP)*, pages 3461–3474.

Nikola Mrkšić, Diarmuid O Séaghdha, Tsung-Hsien Wen, Blaise Thomson, and Steve Young. 2016. Neural belief tracker: Data-driven dialogue state tracking. *arXiv preprint arXiv:1606.03777*.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, pages 311–318.

Ofir Press, Amir Bar, Ben Bogin, Jonathan Berant, and Lior Wolf. 2017. Language generation with recurrent generative adversarial networks without pre-training. *arXiv preprint arXiv:1706.01399*.

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*, 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*, pages 2001–2010.

Iulian Vlad Serban, Alessandro Sordoni, Yoshua Bengio, Aaron C Courville, and Joelle Pineau. 2016. Building end-to-end dialogue systems using generative hierarchical neural network models. In *Proc. AAAI Conf. on Artificial Intelligence*, pages 3776–3784.

Fan-Keng Sun, Cheng-Hao Ho, and Hung-Yi Lee. 2019. Lamol: Language modeling for lifelong language learning. *arXiv preprint arXiv:1909.03329*.

Johannes Von Oswald, Christian Henning, Benjamin F Grewe, and João Sacramento. 2019. Continual learning with hypernetworks. *arXiv preprint arXiv:1906.00695*.

Liyuan Wang, Xingxing Zhang, Hang Su, and Jun Zhu. 2024. A comprehensive survey of continual learning: Theory, method and application. *IEEE Trans. Pattern Anal. Mach. Intell.*, 46(8):5362–5383.

Xiao Wang, Tianze Chen, Qiming Ge, Han Xia, Rong Bao, Rui Zheng, Qi Zhang, Tao Gui, and Xuanjing Huang. 2023. Orthogonal subspace learning for language model continual learning. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 10658–10671, Singapore. Association for Computational Linguistics.

Tsung-Hsien Wen, Milica Gasic, Nikola Mrksic, Pei-Hao Su, David Vandyke, and Steve Young. 2015. Semantically conditioned lstm-based natural language generation for spoken dialogue systems. *arXiv preprint arXiv:1508.01745*.

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. *arXiv preprint arXiv:1905.08743*.

Puyang Xu and Qi Hu. 2018. An end-to-end approach for handling unknown slot values in dialogue state tracking. *arXiv preprint arXiv:1805.01555*.

Qiancheng Xu, Min Yang, and Ruifeng Xu. 2023. Balanced meta learning and diverse sampling for lifelong task-oriented dialogue systems. In *AAAI*, pages 13843–13852.

Dani Yogatama, Cyprien de Masson d'Autume, Jerome Connor, Tomas Kocisky, Mike Chrzanowski, Lingpeng Kong, Angeliki Lazaridou, Wang Ling, Lei Yu, Chris Dyer, et al. 2019. Learning and evaluating general linguistic intelligence. *arXiv preprint arXiv:1901.11373*.

Jaehong Yoon, Eunho Yang, Jeongtae Lee, and Sung Ju Hwang. 2017. Lifelong learning with dynamically expandable networks. *arXiv preprint arXiv:1708.01547*.

Min Zeng, Haiqin Yang, Wei Xue, Qifeng Liu, and Yike Guo. 2024. Dirichlet continual learning: Tackling catastrophic forgetting in nlp. In *UAI*.

Min Zeng, Yisen Wang, and Yuan Luo. 2019. Dirichlet latent variable hierarchical recurrent encoder-decoder in dialogue generation. In *Proc. Conf. on Empirical Methods in Natural Language Processing and Joint Conf. on Natural Language Processing (EMNLP-IJCNLP)*, pages 1267–1272.

Yanzhe Zhang, Xuezhi Wang, and Diyi Yang. 2022. Continual sequence generation with adaptive compositional modules. In *Proc. Conf. of the Association for Computational Linguistics (ACL)*.

Weixiang Zhao, Shilong Wang, Yulin Hu, Yanyan Zhao, Bing Qin, Xuanyu Zhang, Qing Yang, Dongliang Xu, and Wanxiang Che. 2024. Sapt: A shared attention framework for parameter-efficient continual learning of large language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 11641–11661.

Qi Zhu, Bing Li, Fei Mi, Xiaoyan Zhu, and Minlie Huang. 2022. Continual prompt tuning for dialog state tracking. In *Proc. Conf. of the Association for Computational Linguistics (ACL)*, pages 1124–1137.

## A Task Description

The tasks in task-oriented dialogue systems include:

- The **INTENT** task aims to recognize user queries into specific intents. For example, the query $Q$ of "`I would like to book a flight to Los Angeles on Dec. 26th.`" would be categorized as the "flight_booking" intent. By accurately classifying intents, systems can efficiently route queries and generate appropriate responses.
- The **Dialogue State Tracking (DST)** task aims to interpret and manage users' intents and contextual information in conversations. For instance, in a conversation about flight booking, a user might raise a query $Q$, "`I would like to book a flight to Los Angeles on Dec. 26th.`" Then DST identifies key entities such as the "`flight_booking(destination = Los Angeles, date = Dec. 26th)`" to comprehend the user's travel request. The extracted entities enable the system to generate relevant responses aligned with the user's preferences.
- The **Natural Language Generation (NLG)** task utilizes the extracted information "`flight_booking(destination = Los Angeles, date = Dec. 26th)`" from DST to generate a response to the user's query. The system might respond "American Airlines - Departure: 11:30 AM, Arrival: 2:30 PM, Price: $280. Would this flight meet your requirements?"
- The **End-to-End (E2E)** task refers to a unified approach where the system processes users' input and generates responses without component segmentation. For query $Q$ of "`I would like to book a flight to Los Angeles on Dec. 26th.`", an E2E system directly interprets the request and generates a tailored response, "I found a few flights to Los Angeles on Dec. 26th. One option is a flight departing at 11:30 AM, arriving at 2:30 PM, with American Airlines for $280. Would you like to book this flight, or should I look for other options?" without task-specific modules like INTENT or DST. This approach can simplify architecture and thus enhance system efficiency.

## B Dataset Description

We describe the details of the datasets as follows:

- The Task-Master **(TM19)** dataset introduces task-oriented dialogues across six domains (Byrne et al., 2019): ordering pizza, creating auto repair appointments, setting up ride service, ordering movie tickets, ordering coffee drinks, and making restaurant reservations. TM19 has a richer and more diverse language than MultiWOZ.
- The Task-Master 2020 **(TM20)** dataset consists of dialogues in seven domains (Byrne et al., 2019): restaurants, food ordering, movies, hotels, flights, music, and sports, which encompass a wider range of themes and contexts. TM19 includes written "self-dialogues" and two-person spoken dialogues, whereas TM20 solely comprises two-person spoken dialogues. Furthermore, TM19 primarily focuses on task-oriented dialogues, while TM20 features many search-oriented and recommendation-oriented dialogues, spanning various domains.
- The Schema-Guided Dialogue **(SGD)** dataset (Rastogi et al., 2020) contains over multi-domain conversations spanning 16 domains such as movies, music, banks, etc.
- The Multi-Domain Wizard-of-Oz **(MultiWOZ)** dataset (Budzianowski et al., 2018) is a multi-domain dialogue dataset including 5 domains: attraction, hotel, restaurant, taxi and train.

## C Dataset Statistics

Table 7 summarizes the dataset statistics.

| Name | Train | Valid | Test | Dom. | Intent | Turns |
|---|---|---|---|---|---|---|
| TM19 | 4,403 | 551 | 553 | 6 | 112 | 19.97 |
| TM20 | 13,839 | 1,731 | 1,734 | 7 | 128 | 16.92 |
| MultiWOZ | 7,906 | 1,000 | 1,000 | 5 | 15 | 13.93 |
| SGD | 5,278 | 761 | 1,531 | 19 | 43 | 14.71 |
| Total | 31,426 | 4,043 | 4,818 | 37 | 280 | 16.23 |

Table 7: Dataset Statistics