# MOSS: End-to-End Dialog System Framework with Modular Supervision

**Weixin Liang,**[*1,3] **Youzhi Tian,**[*1,2] **Chengcai Chen,**[4] **Zhou Yu**[3]
[1]Zhejiang University, [2]University of California, Davis,
[3]Stanford University, [4]Xiaoi Robot Technology Co., Ltd
{victor_liang, 3150101340}@zju.edu.cn, arlenecc@xiaoi.com, joyu@ucdavis.edu

## Abstract

A major bottleneck in training end-to-end task-oriented dialog system is the lack of data. To utilize limited training data more efficiently, we propose Modular Supervision Network (MOSS), an encoder-decoder training framework that could incorporate supervision from various intermediate dialog system modules including natural language understanding, dialog state tracking, dialog policy learning and natural language generation. With only 60% of the training data, MOSS-all (i.e., MOSS with supervision from all four dialog modules) outperforms state-of-the-art models on CamRest676. Moreover, introducing modular supervision has even bigger benefits when the dialog task has a more complex dialog state and action space. With only 40% of the training data, MOSS-all outperforms the state-of-the-art model on a complex laptop network trouble shooting dataset, LaptopNetwork, that we introduced. LaptopNetwork consists of conversations between real customers and customer service agents in Chinese. Moreover, MOSS framework can accommodate dialogs that have supervision from different dialog modules at both framework level and model level. Therefore, MOSS is extremely flexible to update in real-world deployment.

## Introduction

Most current end-to-end generative dialog models require thousands of annotated dialogs to train a simple information request task (Lei et al. 2018). It is difficult and time consuming to collect human-human dialogs (Serban et al. 2015). Due to the task constraints, it is even impossible to collect a large number of dialogs. In contrast, traditional modular framework (Williams and Young 2007) requires less training data (Lowe et al. 2017). Traditional modular framework is a pipeline of the following four functional modules developed independently: a natural language understanding module that maps the user utterance to a distributed semantic representation; a dialog state tracking module that accumulates the semantic representation across different turns to form the dialog state; a dialog policy learning module that decides system dialog act based on the dialog state, and a natural language generation module that maps the obtained dialog act to natural language. However, each module in such traditional modular system is independently optimized. Therefore, it is difficult to update each module whenever new training data come. For example, when the natural language understanding module is retrained with new data, all the other modules that depend on it become sub-optimal due to the fact that they were trained on the output distributions of the older version of the module.

To combine the benefits from both modular and end-to-end systems, we propose to follow the idea of modular systems by injecting rich supervision from each dialog module in an end-to-end trainable framework. Under MOSS framework, dialog modules such as natural language understanding, dialog state tracking, dialog policy learning and natural language generation share an encoder but have their own decoders. Decoders of different modules are connected through hidden states rather than symbolic outputs. Then all the modules can be optimized jointly to avoid error propagation and model mismatch. In addition, since MOSS produces output from individual modules during testing, we can easily locate the error by checking the modular output.

MOSS is also a flexible framework that can be used in a plug-and-play fashion by removing supervision from some modules. The plug-and-play feature offers options at two levels to enable full utilization of all available annotations. At framework level, for example, if the data do not have natural language understanding supervision, we can create a new instance (model) of MOSS framework by removing the natural language understanding module in MOSS. As a general rule of thumb, the more supervision the model has, the better the performance is, and potentially the less number of dialogs are required to reach good performance. Our results show that, MOSS-all (MOSS with supervision from all four dialog modules) on only 60% of the training data outperforms state-of-the-art models on CamRest676 including TSCP (Lei et al. 2018). At model level, we could patch the performance of an individual module of a specific model by adding incompletely annotated training dialogs. For example, we observe a large performance improvement of natural language generation on MOSS-all with 60% of the training data when we add the additional 40% training dialogs in raw format (i.e., without any annotations).

[*]Equal contribution.

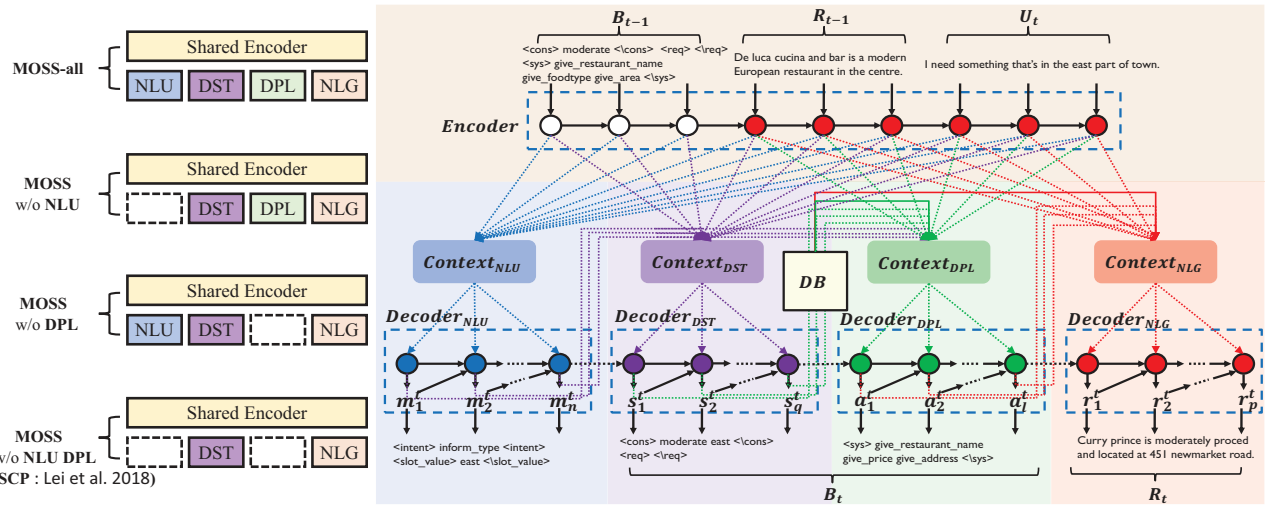


Figure 1: Modular Supervision Framework (MOSS): The left part shows several instances of MOSS framework in plug-and-play fashion: MOSS-all (MOSS with supervision from all four dialog modules), MOSS w/o NLU, MOSS w/o DPL, MOSS w/o NLU DPL (which is actually TSCP (Lei et al. 2018)). The right part shows the detailed architecture of MOSS-all with one decoder and four decoders (Natural Language Understanding, Dialog State Tracking, Dialog Policy Learning and Natural Language Generation). The black dash lines connecting different modules represent shared hidden states. The colored dash lines represent modular attention, by which MOSS-all feeds input to each module.

Theoretically, introducing modular supervision has even bigger benefits when the dialog task has more complex dialog states and action spaces. To prove MOSS's ability on complex tasks, we collect and annotate LaptopNetwork, a dataset on the laptop network malfunction trouble-shooting task from real-world dialogs. Compared to existing datasets, LaptopNetwork has a more complex and realistic dialog structure since the dialogs are between real users and professional computer maintenance engineers. Different from previous information request tasks, LaptopNetwork has more actions as the dialogs are driven by the goal of fixing the network. On LaptopNetwork, MOSS-all (MOSS with all supervision) outperforms state-of-the-art model with only 40% of the training data. Based on our experiments on both LaptopNetwork and CamRest676, we summarize the take-aways for how to efficiently build a dataset to solve a task. We release the code and data[1].

## Related work

Different end-to-end trainable task-oriented dialog systems inject supervision differently. Eric et al. (2017) proposed to use an attention sequence-2-sequence (Seq2Seq) (Sutskever, Vinyals, and Le 2014) encoder-decoder model without intermediate dialog module's supervision except for the natural language generation part. Such systems require thousands of dialogs to learn one simple task. It is not clear if such systems can work well on complex tasks (Lowe et al. 2017; He et al. 2018). Lee 2014 suggested that there is a positive correlation between end-to-end dialog performance and dialog state tracking performance. So we believe incorporating dialog state tracking supervision will improve overall system performance. NDM and LIDM (Wen et al. 2017b; 2017a) incorporated dialog state tracking supervision via a separately-trained belief tracker. TSCP (Lei et al. 2018) introduced a two-decoder pipeline that combines two dialog modules together. Specifically, it jointly trained belief span decoding (dialog state tracking) and response generation. Shu et al. (2018) extended TSCP (Lei et al. 2018) by separately decoding information slot and predicting requested slot for dialog state tracking. All these approaches outperform Eric et al. (2017). None of them incorporated supervision from dialog policy learning. Though both NDM and LIDM (Wen et al. 2017b; 2017a) have policy network components, they are single layer MLPs functioning as the glue that binds the system modules together. Their policy network component does not incorporate supervision from dialog policy learning. However, the dialog policy learning is important because it decides the system's next action. The system dialog act can guides the language generation. There is also work that incorporates supervision from dialog policy but not natural language understanding (Liu et al. 2018). However, incorporating natural language understanding supervision improve performance for tasks in which user utterances have a large number of intents and slots. Although Li et al. 2017 incorporated supervision from all four modules, it feeds the symbolic output from NLU to downstream modules and could not avoid error propagation. Therefore, we propose MOSS, an encoder-decoder based end-to-end trainable framework that can incorporate supervision from all intermediate dialog modules, including natural language understanding (NLU), dialog state tracking (DST), dialog policy learning (DPL) and natural language generation (NLG).

[1]https://github.com/YouzhiTian/MOSS-End-to-End-Dialog-System-Framework-with-Modular-Supervision

Most existing task-oriented dialog datasets, such as Wen et al. (2017b) and Budzianowski et al. (2018), are collected in the Wizard-of-Oz (WOZ) role-play paradigm. In such a paradigm, the users are asked to conduct the task with detailed instruction. It improved the efficiency in collecting domain-specific data and ensures coherence and consistency between the two conversation partners. However, the user action space is relatively small compared to the real-world dialog because of the predefined constraints. In addition, the users are role-playing instead of having a real need to talk to the system, so the dialogs are different from practical usage. Towards tackling tasks with more dialog acts, Lewis et al. (2017); He et al. (2018); Wang et al. (2019) collected negotiation and persuasion dialogs by asking the two Turkers negotiate or persuade each other to reach an agreement. However, these tasks are still not collected from real users. The only real human-human real-world dialog system is a domain-specific IT helpdesk dataset (Vinyals and Le 2015). But unfortunately, this dataset is not public. Therefore, to test MOSS's ability to handle complex tasks, we publish an annotated real-world dataset, LaptopNetwork. It contains dialogs between real users and computer maintenance engineers on solving laptop network issues.

## MOSS: Modular Supervision Network

MOSS is an encoder-decoder based end-to-end trainable framework that could incorporate supervision from various intermediate dialog system modules. Figure 1 (right) shows the detailed architecture of MOSS-all (i.e., MOSS with supervision from all four dialog modules). Inspired by traditional modular architecture, MOSS-all has a unified encoder and four separate decoders. Each decoder aligns with a dialog module so the supervision can be introduced from each decoder. Between different modules, we transfer knowledge via cross-modular attention and shared hidden states without relying on symbolic outputs. We jointly optimize the four decoders to avoid error propagation. Moreover, as Figure 1 (left) shows, with different instantiations, MOSS framework can accommodate dialogs that have supervision from different dialog modules in a plug-and-play fashion.

### Methodology

We first present the architecture of MOSS-all and then describe how the plug-and-play feature deals with incomplete annotations. For each dialog turn $t$, the system inputs are: the state summary of the previous turn $B_{t-1} = [S_{t-1}; A_{t-1}]$ (the concatenation of the dialog state $S_{t-1}$ and the system act $A_{t-1}$ of previous turn), the system response utterance $R_{t-1}$ of previous turn and the user utterance $U_t$. We formulate each module into a sequence-to-sequence (Seq2Seq) framework with $[B_{t-1}, R_{t-1}, U_t]$ as the input sequence.

**Natural Language Understanding (NLU) Module** The NLU module generates a distributed semantic representation $M_t = (m_0^t, m_1^t, \ldots, m_n^t)$ of the user utterance $U_t$. $M_t$ is the concatenation of user intent and the extracted values for slot filling. The NLU module could be formulated as:

$$M_t = Seq2Seq_{NLU}(B_{t-1}, R_{t-1}, U_t)$$

**Diaog State Tracking (DST) Module** The DST module maintains the dialog state $S_t = (s_0^t, s_1^t, \ldots, s_n^t)$, which is the concatenation of user expressed constraints and requests. DST achieves this by accumulating user semantic representation $M_t$ across different turns $0, 1, \ldots, t$. So the DST module could be formulated as:

$$S_t = Seq2Seq_{DST}(B_{t-1}, R_{t-1}, U_t | M_t)$$

**Dialog Policy Learning (DPL) Module** The DPL module generates system act $A_t = (a_1^t, a_2^t, \ldots, a_l^t)$ based on the current dialog state $S_t$. It could be formulated as:

$$A_t = Seq2Seq_{DPL}(B_{t-1}, R_{t-1}, U_t | M_t, S_t)$$

**Natural Language Generation (NLG) Module** The natural language generation (NLG) module then maps the dialog act to its surface form $R_t = (r_1^t, r_2^t, \ldots, r_p^t)$. So the NLG module could be formulated as:

$$R_t = Seq2Seq_{NLG}(B_{t-1}, R_{t-1}, U_t | M_t, S_t, A_t)$$

**Plug-and-Play: Dealing with Incomplete Annotations** The plug-and-play feature offers options at both framework level and model level to deal with incomplete annotations. At framework level, to accommodate dialogs that lack supervision from different dialog modules, we could create different instances (models) of MOSS framework by removing the corresponding decoder in MOSS as shown in Figure 1 (left). We further adopt the down-stream module(s) by removing the condition dependencies on the module(s) to be removed. For example, if we remove the dialog policy learning module, then we get MOSS without supervision from dialog policy learning (MOSS w/o DPL) and re-formulate the NLG module as:

$$R_t = Seq2Seq_{NLG}(B_{t-1}, R_{t-1}, U_t | M_t, S_t)$$

where $A_t$ is removed in the condition.

At model level, for a specific instance (model), we could patch the performance of an individual module by adding incompletely-annotated training dialogs. For example, if the performance of natural language generation is not satisfactory, we could add raw training dialogs without any annotations. For these training dialogs, we calculate the loss solely based on the natural language generation module and back-propagate the gradient to the entire model. The flexibility offered by these two levels of plug-and-play encourages the maximum utilization of all available annotations and the practical updates in deployed systems.

### Building Blocks: Encoder and Decoder

**Encoder** An encoder is shared by all modules under MOSS framework. For each dialog turn $t$, a shared bidirectional GRU encodes the following three input: the state summary of the previous turn $B_{t-1} = [S_{t-1}; A_{t-1}]$ (the concatenation of the dialog state $S_{t-1}$ and the system act $A_{t-1}$ of previous turn), the system response utterance $R_{t-1}$ of previous turn and the user utterance $U_t$.

$$\widetilde{B}_{t-1}, \widetilde{R}_{t-1}, \widetilde{U}_t, h_E^t = Encoder(B_{t-1}, R_{t-1}, U_t)$$

where $\widetilde{B}_{t-1}, \widetilde{R}_{t-1}, \widetilde{U}_t$ are the encoder states when encoding each token of $B_{t-1}$,$R_{t-1}$ and $U_t$ respectively. $h_E^t$ is the last encoder hidden state.

**Decoder** The decoders in all modules (NLU, DST, DPL, NLG) have the same structure. Each decoder is implemented as an attention (Bahdanau, Cho, and Bengio 2015) based unidirectional GRU augmented with the copy mechanism (Gu et al. 2016). The decoder input is a sequence of distributed representations $X = (x_1, x_2, \ldots, x_n)$. In addition, the initial decoder hidden state $h_0$ could be assigned as prior knowledge. The decoder output is $Y = (y_1, y_2, \ldots, y_m)$, a sequence of the probability of output tokens. We also records $\widetilde{Y} = (\widetilde{y}_1, \widetilde{y}_2, \ldots, \widetilde{y}_m) = (h_1, h_2, \ldots, h_m)$, the decoder hidden states when decoding $Y$ because it would be used by its downstream modules. The decoder could be formulated as:

$$Y, \widetilde{Y} = Decoder_{\varphi}(X, h_0)$$

where $\varphi$ is the module name which could be NLU, DST, DPL or NLG. The loss is defined as negative log likelihood.

## Natual Language Understanding (NLU) Decoder

Traditionally, a NLU module processes intent detection and slot filling separately: intent detection is treated as a semantic utterance classification problem, and slot filling is treated as a sequence labeling task. We jointly formulate intent detection and slot filling as a sequence generation problem, which solves multi-intent problem.

The NLU module maps user utterance $U_t$ to user semantic representation $M_t$ with the help of the information in previous turns $(B_{t-1}, R_{t-1})$. we formulate $Decoder_{NLU}$ as:

$$M_t, \widetilde{M_t} = Decoder_{NLU}([\widetilde{B}_{t-1}, \widetilde{R}_{t-1}, \widetilde{U}_t], h_E^t)$$

Note that $\widetilde{M_t}$ is the decoder hidden states when decoding user semantic representation $M_t$. It will be used as the input of later modules. The initial hidden state of $Decoder_{NLU}$ is initialized as the last hidden state $h_E^t$ of the encoder.

## Dialog State Tracking (DST) Decoder

MOSS formulates DST into a sequence-to-sequence framework with copy mechanism. So the DST module can solve the out-of-vocabulary words problem of traditional classification-based methods, as users may mention values for the informable slots which have never appeared in the training data. The DST module tracks dialog state $S_t$ by accumulating user semantic representation $M_t$ across different turns.

The DST decoder also takes system response utterance $\widetilde{R}_{t-1}$, user utterance $\widetilde{U}_t$ as input. Different from condensed context like state summary of previous turn $B_{t-1}$, $\widetilde{R}_{t-1}, \widetilde{U}_t$ is the immediate dialog context of this turn. The immediate dialog context might contain information that's not in the condensed context. So we formulate the DST decoder as:

$$S_t, \widetilde{S}_t = Decoder_{DST}([\widetilde{B}_{t-1}, \widetilde{R}_{t-1}, \widetilde{U}_t, \widetilde{M_t}], \tilde{m}_n^t)$$

Here $Decoder_{DST}$ is initialized with the last hidden state of the NLU decoder $\tilde{m}_n^t$ as prior.

| Model | Mat | Succ.F1 | BLEU |
|---|---|---|---|
| KVRN | N/A | N/A | 0.134 |
| NDM | 0.904 | 0.832 | 0.212 |
| LIDM | 0.912 | 0.840 | 0.246 |
| TSCP | 0.927 | 0.854 | 0.253 |
| MOSS w/o DPL | 0.932 | 0.856 | 0.251 |
| MOSS w/o NLU | 0.932 | 0.857 | 0.255 |
| MOSS-all $\times$ 60% | 0.947 | 0.857 | 0.202 |
| MOSS $\times$ (60%all + 40%raw ) | 0.947 | 0.859 | 0.221 |
| MOSS-all | **0.951** | **0.860** | **0.259** |

Table 1: Performance comparison on CamRest676 among the baselines, MOSS-all, and several variants of MOSS.

## Dialog Policy Learning (DPL) Decoder

We formulate DPL as a sequence-to-sequence problem to enable MOSS to generate multiple system acts. The DPL module predicts the system acts $A_t$ by considering both the dialog states $S_t$ and the query results from the external database DB. Following Wen et al. (2016), the DPL module forms the database query by taking the union of the maximum values of each informable slot in dialog state $S_t$ (Wen et al. 2017b). The DB returns a one-hot vector $k_t$ representing different degrees of matching in the DB (no match, 1 match, ... or more than 5 matches). As language model type condition (Wen et al. 2016), $k_t$ is concatenated with the word embedding of each $a_j^t$, $j \in [1, \ldots, l]$ as the new embedding.

$$emb'(a_j^t) = \begin{pmatrix} emb(a_j^t) \\ k_t \end{pmatrix}$$

The DPL decoder explicitly conditions on the state summary of this turn $S_t$ to generate the system act $A_t$.

$$A_t, \widetilde{A}_t = Decoder_{DPL}([\widetilde{R}_{t-1}, \widetilde{U}_t, \widetilde{S}_t], \tilde{s}_q^t)$$

The hidden state of $Decoder_{DPL}$ is initialized as the last hidden state $\tilde{s}_q^t$ of $Decoder_{DST}$.

## Natural Language Generation (NLG) Decoder

The NLG decoder converts the system dialog acts $A_t$ into system response $R_t$. The NLG also conditions on DB query result $k_t$ in the same way as the DPL. The NLG decoder initializes its hidden state with the last hidden state $\tilde{a}_l^t$ of the $Decoder_{DPL}$ as the prior knowledge of system acts $A_t$.

$$R_t, \widetilde{R}_t = Decoder_{NLG}([A_t, \widetilde{R}_{t-1}, \widetilde{U}_t], \tilde{a}_l^t)$$

Finally, we sum up the cross-entropy losses of the four decoders and optimize the four decoders jointly to avoid error propagation and model mismatch:

$$\mathcal{L} = \mathcal{L}_{NLU} + \mathcal{L}_{DST} + \mathcal{L}_{DPL} + \mathcal{L}_{NLG}$$

# Restaurant Search Task

We first use CamRest676 (Wen et al. 2017b) dataset to show MOSS's advantage on existing task-oriented dialog

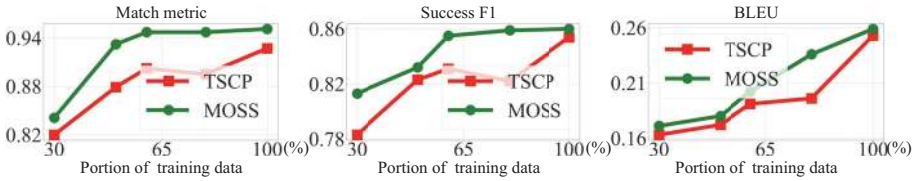


Figure 2: The detailed perforamance change of TSCP and MOSS-all on CamRest676 using a different amount of data.

| | |
|---|---|
| **User**: | I'm looking for an expensive restaurant in the south. |
| **True**: | Hi!We have Mexican,Italian,Indian and Chinese restaurants that fit the bill. Do you have any preferences? |
| **MOSS-all**: | What type of food are you interested in the south part of town? (act: ***"ask_food"***) |
| **TSCP**: | Curry Garden is a expensive Indian restaurant in the south part of town. |

Table 2: An example of MOSS-all and TSCP on CamRest676. The bold word indicates MOSS-all's correct prediction on dialog policy.

datasets. We annotate CamRest676 with five user intents (e.g., *Inform_type_change*, *Goodbye*) and 10 system dialog acts (e.g., *give_foodtype*, *ask_food*). We follow Wen et al. (2017b); Lei et al. (2018) to split the data as 3:1:1 for training, validation and testing.

## Baselines and Metrics

We compare our model against a set of state-of-the-art models: *(i)* KVRN (Eric et al. 2017) is an attention seq2seq encoder-decoder model without intermediate dialog module's supervision except for the natural language generation; *(ii)* NDM (Wen et al. 2017b) and *(iii)* LIDM (Wen et al. 2017a) incorporate dialog state tracking supervision via a separately-trained belief tracker; *(iv)* TSCP (Lei et al. 2018) could be viewed as an instance of MOSS without supervision from natural language understanding and dialog policy learning. *(v- viii)* We also evaluate some variants of MOSS shown in Figure 1 (left). Following Lei et al. (2018), we use three evaluation metrics: entity match rate (Mat) on dialog state, success F1 (Succ.F1) on requested slots and BLEU (Papineni et al. 2002) on generated system utterances.

## Results

The first key takeaway is that the more supervision the model has, the better the performance is. As shown in Table 1, in terms of overall performance, we have *(i)* KVRN $<$ *(ii)* NDM $\approx$ *(iii)* LIDM $<$ *(iv)* TSCP $<$ *(v)* MOSS w/o DPL $\approx$ *(vi)* MOSS w/o NLU $<$ *(ix)* MOSS-all. We note that this performance ranking is the same as the ranking of how much supervision each system receives: *(i)* KVRN only incorporates supervision from one dialog module (i.e., natural language generation); *(ii, iii, iv)* NDM, LIDM, TSCP incorporate supervision from two dialog modules (i.e., dialog state tracking and natural language generation); *(v, vi)* MOSS without dialog policy learning (MOSS w/o DPL) and MOSS without natural language understanding (MOSS w/o NLU) incorporate supervision from three dialog modules; *(ix)* MOSS-all incorporates supervision from all four modules and outperform all models on all three metrics.

| **Response and request** |
|---|
| **Sys:** Unfortunately there are no Thai restaurants in the **north**, do you want to change an area to look for ? **User:** How about west area? I also want the address, phone number, and the price range? |
| **MOSS-all.NLU: *ask_Inf*** : west address phone price |
| **True.NLU:** *inform_Type_Change* : west address phone price |
| **MOSS-all.DST:** constraints: Thai **north** requests: address phone price |
| **True.DST:** constraints: Thai west request: address phone price |

Table 3: An MOSS-all error analysis example. The underlined words indicate the correct outputs while the bold parts indicate the incorrect outputs.

Another takeaway is that models that have access to more detailed supervision need fewer number of dialogs to reach good performance. Row 7 in Table 1 shows that with only 60% training data, MOSS-all outperforms state-of-the-art baselines in terms of task completion (Mat and success F1). As for language generation quality (BLEU), MOSS-all with 60% training data performs worse. We suspect that it is partially because MOSS-all with 60% training data has seen fewer number of dialogs and thus has a weaker natural language generation module. We validate this hypothesis by training MOSS-all with 60% training data with all annotations plus the left 40% training data without any annotation (i.e., MOSS-all $\times$ 60% + 40%raw, Row 8 in Table 1). We observe a large improvement on the BLEU score.

MOSS-all $\times$ 60% + 40%raw also shows the plug-and-play feature at model level. An instance of MOSS framework (e.g., MOSS-all) could accommodate dialogs that have supervision from different dialog modules (e.g., all four modules v.s. only natural language generation module). The plug-and-play feature at model level allows us to patch the performance of an individual module (e.g., natural language

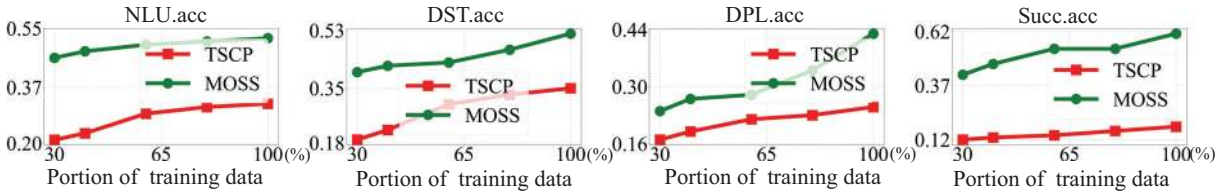


Figure 3: The detailed performance change of TSCP and MOSS-all on LaptopNetwork using a different amount of data.

generation) by adding incompletely annotated dialogs.

Compared to MOSS-all with only 60% training data, MOSS-all using all data only improves the performance slightly from 0.947 to 0.951 in Mat and 0.857 to 0.867 in Succ.F1. The improvement is not huge because restaurant search is a relatively simple task. TSCP's performance drops drastically by reducing the training data (from 0.927 to 0.902 in Mat and 0.854 to 0.831 in Succ.F1). If limited training data is available, MOSS would potentially outperform TSCP much more significantly (0.947 VS 0.902 in Mat and 0.857 VS 0.831 in Succ.F1). Figure 2 shows the detailed performance change between the two models using a different amount of data.

For traditional modular dialogue systems, we note that Braunschweiler and Papangelis construct a traditional modular dialogue system and compare it against NDM on CamRest676 dataset in a synthesized speech scenario. The results show that NDM significantly outperforms the traditional modular dialogue system. While our method outperforms NDM with a huge margin. Therefore, we expect MOSS also outperforms traditional modular dialogue systems.

**Case Study** Since TSCP is the best among all the baselines, we select TSCP to compare against MOSS in the case study. Table 2 presents an example from the testing set. We found after incorporating supervision from dialog policy MOSS performs better than TSCP. MOSS-all learns to ask the user for more information (act: "ask_food") when there are too many matched results in the database. In contrast, TSCP instead acts as there is only one restaurant satisfying the user's constraint, though TSCP tracts the dialog state correctly. We suspect this error is caused because TSCP replies with the utterance it has seen the most in a similar context without distinguishing even similar context may lead to different dialog act choice.

**Error analysis** The output from individual modules in MOSS helps to locate its error easily. Table 3 shows an error in the generated dialog state of MOSS ("north" v.s. "west"). The natural language understanding produced correct slots but in the dialog act intent prediction (*"ask_info"* v.s. *"inform_type_change"*), it produced wrong values. So the DST receives the wrong information. For such errors, given that *"inform_type_change"* occurs much less than other tags like *"ask_info"*, one solution is to collect more examples on these two confusing dialog acts for training.

## Laptop Network Troubleshooting

In this section, we first introduce a complex laptop network troubleshooting dataset-LaptopNetwork. We then evaluate MOSS on LaptopNetwork, showing that when the dialog task has a more complex dialog state and action space, introducing modular supervision has even bigger benefits.

### LaptopNetwork Dataset

We collect LaptopNetwork, a real-world laptop network troubleshooting task in Chinese. Different from dialogs generated by crowd-source workers (Wen et al. 2017b), LaptopNetwork is more realistic since it involves real customers with technical problems and professional computer maintenance engineers on an online typing after-sales service platform. In LaptopNetwork, the engineer might give different solutions based on the feedback from customers. LaptopNetwork contains 618 dialogs from 97 engineers with 14.76 turns per dialog, 51.67 Chinese characters per utterance and 985.82 seconds of time elapsed per dialog on average.

Developing an annotation scheme for LaptopNetwork is challenging because dialogs are full of computer maintenance terminologies such as "network adaptor" and "winsock" that are difficult to understand for naive annotators. So to guarantee the annotation quality, we hired two well-trained computer technicians and went through many iterations to polish the dialog annotation scheme. Finally we identified 60 different dialog acts with 12 of them as network solution related dialog acts. Compared to dataset on information search task like CamRest676 which only has two dialog acts: request and provide information, our proposed network troubleshooting task is much more complex. as the system needs to first understand users' problem and then collaborate with user to perform troubleshooting to resolve the problem. After finalizing the dialog annotation scheme, the two technicians then annotate each utterance with one or more dialog acts. The two experts reached $\kappa = 0.852$ inter-annotator agreement on 300 turns.

### Baselines and Metrics

Table 1 shows TSCP (Lei et al. 2018) perform the best among the baselines on CamRest676. So we compare our model against TSCP and some variants of MOSS on LaptopNetwork. We augment the belief span $B_t$ originally introduced in Lei et al. (2018) by concatenating user act, old $B_t$ and system act in TSCP. This augmentation makes sure that TSCP has access to the same annotations as MOSS, otherwise TSCP could hardly generate reasonable response.

| TN | User | System utterance (Ground truth) | System utterance (MOSS) | System utterance (TSCP) |
|---|---|---|---|---|
| 1 | The wireless network is not working. | What is the operating system of your computer? | What is your computer model? | Computer model? |
| 2 | My computer operating system is win7. | Can you show me your computer's Mobility Center? | Are you able to find the wireless signal? | Computer model? |
| 3 | I have sent you the result. | Launch your computer console window, enter 'netsh winsock reset'. | Launch your computer console window, enter 'netsh winsock reset'. | Show me your network adaptor. |
| 4 | Okay, let me try. | Don't forget to restart your computer. | Don't forget to restart the computer. | Show me your network adaptor. |
| 5 | My problem is solved! | Next time, just enter 'netsh winsock reset' into the console. | Next time you can try typing 'netsh winsock reset'. | Can I help you with anything else? |
| 6 | Thank you! | You're welcome. Can I help you with anything else? | You're welcome. Can I help you with anything else? | Can I help you with anything else? |
| 7 | No! Thank you for your help! | Goodbye! Have a nice day. | Goodbye! Have a nice day. | Can I help you with anything else? |

Figure 4: An example dialog generated by MOSS-all and TSCP. TN denotes the turn number.

| Model | NLU.acc | DST.acc | DPL.acc | Succ.acc | BLEU |
|---|---|---|---|---|---|
| TSCP | 0.32 | 0.35 | 0.25 | 0.18 | 0.050 |
| MOSS w/o DPL | 0.51 | 0.34 | – | – | 0.109 |
| MOSS w/o NLU | – | 0.45 | 0.40 | 0.50 | 0.115 |
| MOSS $\times$ 40% | 0.48 | 0.42 | 0.27 | 0.47 | 0.063 |
| MOSS-all | **0.52** | **0.52** | **0.43** | **0.61** | **0.122** |

Table 4: Performance comparison on LaptopNetwork among MOSS-all, TSCP, and several variants of MOSS.

Since LaptopNetwork is more complex than CamRest676, we add more metrics to capture different perspectives for model performance evaluation. To evaluate the performance of all four modules respectively, we calculate: natural language understanding accuracy **NLU.acc**, the accuracy of user dialog act and slots; dialog state tracking accuracy **DST.acc**, the accuracy of user expressed constraints and requests; dialog policy learning accuracy **DPL.acc**, the accuracy of system dialog act and slots. In LaptopNetwork, whether the system can give an accurate solution to solve the problem is important. So we design **Succ.acc** to capture the system's task completion rate. Because the task is very complex, as long as the system provides the correct solution, the task is considered successful.

## Results

As expected, introducing modular supervision has even bigger benefits when the dialog task has a more complex dialog state and action spaces. As shown in Table 4, with only 40% training data, MOSS-all can outperform the TSCP on all the metrics. Figure 3 shows a consistent large performance gap between TSCP and MOSS-all on LaptopNetwork using a different amount of data.

With 100% training data, MOSS-all significantly outperforms TSCP on all the metrics mentioned above. For task completion rate (Succ.acc), MOSS-all outperforms the state-of-the-art model by **42%**. We suspect that the big performance boost comes from the additional modular supervision MOSS-all has. For the complex task, user dialog act and system dialog act are very effective supervision to facilitate dialog system learning. Without such supervision, we observe that TSCP tends to repeat trivial system responses that are frequently seen in the training data (more details in Case Study). Therefore, TSCP achieves moderate, but not high scores for all the metrics. MOSS-all also outperforms the state-of-the-art model by **7%** in language generation quality. It is not surprising that with the supervision from DPL, the generated dialog act can guide the NLG module to generate a response with the correct intent.

We now examine the performance change in each perspective when removing dialog policy learning module (MOSS w/o DPL) or natural language understanding module (MOSS w/o NLU). Without dialog policy learning module, MOSS w/o DPL achieves comparable natural language understanding accuracy (NLU.acc) but degraded dialog state tracking accuracy (DST.acc) and natural language generation quality (BLEU). Without dialog policy learning module, MOSS w/o DPL exhibits difficulty in directly learning the correlation between dialog state tracking and natural language generation. Without natural language understanding module, MOSS w/o NLU lacks the semantic information from user utterance and performs worse in downstream tasks (i.e., dialog state tracking, dialog policy learning, natural language generation).

**Case Study** Figure 4 presents an example in LaptopNetwork. Without supervision from NLU and DPL, it is difficult to generate correct system acts and responses in complex tasks. So TSCP tends to repeat trivial system responses

(turn 1&2 ; turn 3&4; turn 5&6&7) that are frequently seen in the training data. In contrast, with supervision from NLU and DPL, MOSS understands the dialog context better and reacts with proper system acts and responses: MOSS is able to make inquiries (turn 1&2), give solutions (turn 3), remind users important steps in the solution (turn 4) and close the dialog politely (turn 5&6&7).

## Discussion

Our experiments provide some guidance for managing the budget of constructing a new dialog dataset. For dialog tasks that have more complex dialog states and action space like LaptopNetwork, supervision from all four modules leads to much higher performance and requires significantly fewer number of dialogs (e.g., 40% in LaptopNetwork). Therefore, annotating natural language understanding and dialog policy learning should be prioritized during the construction of such datasets. For simple dialog tasks like information search tasks (e.g., CamRest676), the benefits of adding more supervision is still huge. Moreover, it is possible to automatically annotate the natural language understanding and dialog policy learning in these simple tasks. In CamRest676 for example, we obtain annotations for natural language understanding by calculating the difference of the current and previous dialog states. We also obtain annotations for dialog policy learning by reusing the regular expressions designed for delexicalization of system response in (Wen et al. 2017b). Although collecting more dialogs is important, if it is possible to get detailed annotations for free, we suggest to incorporate these supervision first.

## Conclusion

We propose Modular Supervision Network (MOSS), an end-to-end trainable framework that incorporates supervision from various intermediate dialog system modules. Our experiments show that the more supervision the model has, the better the performance. If more supervision is included, the model needs less number of training dialogs to reach state-of-the-art performance. In addition, such benefit is observed even larger when the dialog task has a more complex dialog state and action space for example, LaptopNetwork. We introduce LaptopNetwork, which is a complex real-world laptop network malfunction trouble-shooting task. Moreover, MOSS framework accommodates dialogs that have supervision from different dialog modules at both framework level and model level. At framework level we create different models with different modules removed; at model level we support feeding dialogs with annotations for different modules into the same model. Such property is extremely useful in real-world industry setting.

## References


Bahdanau, D.; Cho, K.; and Bengio, Y. 2015. Neural machine translation by jointly learning to align and translate. In *ICLR*.

Braunschweiler, N., and Papangelis, A. 2018. Comparison of an end-to-end trainable dialogue system with a modular statistical dialogue system. In *INTERSPEECH*, 576–580. ISCA.

Budzianowski, P.; Wen, T.; Tseng, B.; Casanueva, I.; Ultes, S.; Ramadan, O.; and Gasic, M. 2018. Multiwoz - A large-scale multi-domain wizard-of-oz dataset for task-oriented dialogue modelling. In *EMNLP*, 5016–5026. Association for Computational Linguistics.

Eric, M.; Krishnan, L.; Charette, F.; and Manning, C. D. 2017. Key-value retrieval networks for task-oriented dialogue. In *SIGDIAL Conference*, 37–49. Association for Computational Linguistics.

Gu, J.; Lu, Z.; Li, H.; and Li, V. O. K. 2016. Incorporating copying mechanism in sequence-to-sequence learning. In *ACL (1)*. The Association for Computer Linguistics.

He, H.; Chen, D.; Balakrishnan, A.; and Liang, P. 2018. Decoupling strategy and generation in negotiation dialogues. In *EMNLP*, 2333–2343. Association for Computational Linguistics.

Lee, S. 2014. Extrinsic evaluation of dialog state tracking and predictive metrics for dialog policy optimization. In *SIGDIAL Conference*, 310–317. The Association for Computer Linguistics.

Lei, W.; Jin, X.; Kan, M.; Ren, Z.; He, X.; and Yin, D. 2018. Sequicity: Simplifying task-oriented dialogue systems with single sequence-to-sequence architectures. In *ACL (1)*, 1437–1447. Association for Computational Linguistics.

Lewis, M.; Yarats, D.; Dauphin, Y. N.; Parikh, D.; and Batra, D. 2017. Deal or no deal? end-to-end learning for negotiation dialogues. *CoRR* abs/1706.05125.

Li, X.; Chen, Y.; Li, L.; Gao, J.; and Çelikyilmaz, A. 2017. End-to-end task-completion neural dialogue systems. In *IJCNLP(1)*, 733–743. Asian Federation of Natural Language Processing.

Liu, B.; Tür, G.; Hakkani-Tür, D.; Shah, P.; and Heck, L. P. 2018. Dialogue learning with human teaching and feedback in end-to-end trainable task-oriented dialogue systems. In *NAACL-HLT*, 2060–2069. Association for Computational Linguistics.

Lowe, R. T.; Pow, N.; Serban, I. V.; Charlin, L.; Liu, C.; and Pineau, J. 2017. Training end-to-end dialogue systems with the ubuntu dialogue corpus. *D&D* 8(1):31–65.

Papineni, K.; Roukos, S.; Ward, T.; and Zhu, W. 2002. Bleu: a method for automatic evaluation of machine translation. In *ACL*, 311–318. ACL.

Serban, I. V.; Lowe, R.; Henderson, P.; Charlin, L.; and Pineau, J. 2015. A survey of available corpora for building data-driven dialogue systems. *arXiv preprint arXiv:1512.05742*.

Shu, L.; Molino, P.; Namazifar, M.; Liu, B.; Xu, H.; Zheng, H.; and Tur, G. 2018. Incorporating the structure of the belief state in end-to-end task-oriented dialogue systems. In *NeurIPS 2018 Conversational AI Workshop*.

Sutskever, I.; Vinyals, O.; and Le, Q. V. 2014. Sequence to sequence learning with neural networks. *CoRR* abs/1409.3215.

Vinyals, O., and Le, Q. V. 2015. A neural conversational model. *CoRR* abs/1506.05869.

Wang, X.; Shi, W.; Kim, R.; Oh, Y.; Yang, S.; Zhang, J.; and Yu, Z. 2019. Persuasion for good: Towards a personalized persuasive dialogue system for social good. *CoRR* abs/1906.06725.

Wen, T.; Gasic, M.; Mrksic, N.; Rojas-Barahona, L. M.; Su, P.; Ultes, S.; Vandyke, D.; and Young, S. J. 2016. Conditional generation and snapshot learning in neural dialogue systems. In *EMNLP*, 2153–2162. The Association for Computational Linguistics.

Wen, T.; Miao, Y.; Blunsom, P.; and Young, S. J. 2017a. Latent intention dialogue models. In *ICML*, volume 70 of *Proceedings of Machine Learning Research*, 3732–3741. PMLR.

Wen, T.-H.; Vandyke, D.; Mrkšić, N.; Gasic, M.; Rojas Barahona, L. M.; Su, P.-H.; Ultes, S.; and Young, S. 2017b. A network-based end-to-end trainable task-oriented dialogue system. In *EACL*, 438–449. Valencia, Spain: Association for Computational Linguistics.

Williams, J. D., and Young, S. J. 2007. Partially observable markov decision processes for spoken dialog systems. *Computer Speech & Language* 21(2):393–422.