OpenSLU: A Unified, Modularized, and Extensible Toolkit for Spoken Language Understanding

Libo Qin[1]*, **Qiguang Chen**[2]*, **Xiao Xu**[2], **Yunlong Feng**[2], **Wanxiang Che**[2]

[1]School of Computer Science and Engineering

[1]Central South University, China

[2]Research Center for Social Computing and Information Retrieval

[2]Harbin Institute of Technology, China

lbqin@csu.edu.cn, {qgchen,xxu,ylfeng,car}@ir.hit.edu.cn

Abstract

Spoken Language Understanding (SLU) is one of the core components of a task-oriented dialogue system, which aims to extract the semantic meaning of user queries (e.g., intents and slots). In this work, we introduce OpenSLU, an open-source toolkit to provide a unified, modularized, and extensible toolkit for spoken language understanding. Specifically, OpenSLU unifies 10 SLU models for both single-intent and multi-intent scenarios, which support both non-pretrained and pretrained models simultaneously. Additionally, OpenSLU is highly modularized and extensible by decomposing the model architecture, inference, and learning process into reusable modules, which allows researchers to quickly set up SLU experiments with highly flexible configurations. OpenSLU is implemented based on PyTorch, and released at https://github.com/LightChen233/OpenSLU.

1 Introduction

Spoken Language Understanding (SLU), which is used to extract the semantic frame of user queries (e.g., intents and slots) (Tur and De Mori, 2011). Typically, SLU consists of two sub-tasks: intent detection and slot filling. Take the utterance shown in Figure 1 as an example, given "*Listen to Rock Music*", the outputs include an intent class label (i.e., `Listen-to-Music`) and a slot label sequence (i.e., `O, O, B-music-type, I-music-type`).

Since intent detection and slot filling are highly tied (Qin et al., 2021c), dominant methods in the literature explore joint models for SLU to capture shared knowledge (Goo et al., 2018; Wang et al., 2018; Qin et al., 2019). Recently, Gangadharaiah and Narayanaswamy (2019) shows that, in the Amazon internal dataset, 52% of examples contain multiple intents. Inspired by this observation, various SLU works shift their eye from single-intent SLU to multi-intent SLU scenario (Gangadharaiah and Narayanaswamy, 2019; Qin et al., 2020b; Casanueva et al., 2022; Moghe et al., 2022).

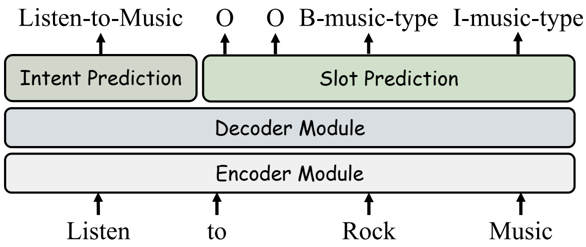


Figure 1: An example of spoken language understanding. `Listen-to-Music` stands for the intent label while `{O, O, B-music-type, I-music-type}` denotes the slot sequence labels.

Thanks to the development of neural network, especially the successful use of large pretrained models, remarkable success have been witnessed in SLU. Nevertheless, there still lacks a unified open-source framework to facilitate the SLU community. In this work, we make the first attempt to introduce OpenSLU, a unified, modularized, and extensible toolkit for SLU, which aims to help researchers to set up experiments and develop their new models quickly. The main features of OpenSLU are:

- **Unified and modularized toolkit**. OpenSLU is the first unified toolkit to support both single-intent and multi-intent SLU scenarios. Meanwhile, it is highly modularized by decoupling SLU models into a set of highly reusable modules, including data module, model module, evaluation module, as well as various common components and functions. Such modularization allows users to quickly re-implement SLU baselines or develop their new SLU models by re-using provided modules or adding new modules.

- **Extensible and flexible toolkit**. OpenSLU is configured by configuration objects, which is extensible and can be initialized from YAML files. This enables users can easily develop

*Equal Contribution

Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics Volume 3: System Demonstrations, pages 95–102
July 10-12, 2023

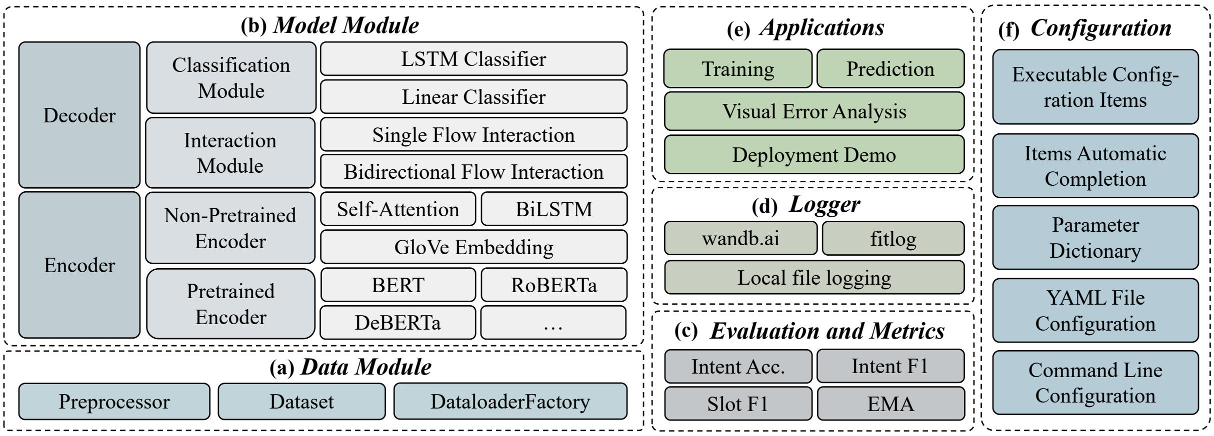


Figure 2: An overall workflow of `OpenSLU`, which consists of (a) Data Module, (b) Model Module, (c) Evaluation and Metrics, (d) Logger, (e) Applications and (f) Configuration.

their models by simply extending the configurations. Additionally, we provide various interfaces of various common functions or modules in SLU models, including Encoder and Decoder module. Besides, the interfaces of our toolkit are fully compatible with the PyTorch interface, which allows seamless integration and flexibly rewriting any sub-module in the toolkit.

- **Visualization Tool.** We provide a visualization tool to help users to view all errors of the model directly. With the help of visualization tool, we can get a clearer picture: where we are and where we should focus our efforts to improve the performance of the model, which helps to develop a more superior framework.

To our knowledge, this is the first unified, modularized, and extensible toolkit for SLU. We hope our work can help researchers to quickly initiate experiments and spur more breakthroughs in SLU[1].

2 Architecture and Design

Figure 2 illustrates the overall workflow of `OpenSLU`. In this section, we describe the (a) Data Module (§2.1); (b) Model Module; (§2.2); (c) Evaluation and Metrics (§2.3) and other common modules (Logger, Applications and Configuration module) (§2.4).

2.1 Data Module

`OpenSLU` offers an integrated data format in the data module (see Figure 2(a)) for SLU models, which can be denoted as: *raw text* → `Preprocessor` → `Dataset` → `DataLoaderFactory` → *model input*.

Given the input *raw text*, *Preprocessor* sub-module first pre-process different raw texts to an integrated $.jsonl$ format that contains slot, text and intent, which is formatted as:

```
{
  "slot": [List of Slot Value],
  "text": [List of Text],
  "intent": [Intent Value]
}.
```

The `Dataset` sub-module offers a range of data processing operations to support both pretrained and non-pretrained models. For pretrained models, these operations include lowercase conversion, BPE-tokenization, and slot alignment, while for non-pretrained models, the sub-module handles word-tokenization and vocabulary construction.

Finally, `DataLoaderFactory` sub-model is used for creating `DataLoader` to manage the data stream for models.

2.2 Model Module

As shown in Figure 2(b), the overall model module contains encoder module(§2.2.1) and decoder module (§2.2.2).

2.2.1 Encoder

For the encoder module, we implement both non-pretrained models and pretrained models. In non-pretrained models, we offer the widely used SLU encoders including self-attentive (Vaswani et al., 2017; Qin et al., 2019) and BiLSTM (Hochreiter and Schmidhuber, 1997; Goo et al., 2018; Liu et al.,

[1]Video introduction about OpenSLU is available at https://youtu.be/uOXh47m_xhU.

2020b) encoder. Additionally, we support auto-load GloVe embedding (Pennington et al., 2014).

In pretrained models, OpenSLU supports various encoders including BERT (Devlin et al., 2019), RoBERTa (Liu et al., 2020a), ELECTRA (Clark et al., 2020), DeBERTa$_{v3}$ (He et al., 2021).

2.2.2 Decoder

Since slot filling and intent detection are highly related, dominant methods in the literature employ joint models to capture the shared knowledge across the related tasks (Goo et al., 2018; Wang et al., 2018; Chen et al., 2019). To support the joint modeling paradigm, the decoder in OpenSLU contains two sub-modules: (1) interaction module for capturing interaction knowledge for slot filling and intent detection and (2) classification module for the final prediction results.

Interaction Module. As summarized in Qin et al. (2021c), the interaction module consists of two widely used interaction types, including *single flow interaction* and *bidirectional flow interaction*.

- **Single Flow Interaction** refers to the flow of information from intent to slot in one direction as illustrated in Figure 3(a). A series of studies (Goo et al., 2018; Li et al., 2018; Qin et al., 2019) have achieved remarkable improvements in performance by guiding slot filling with intent detection information.

- **Bidirectional Flow Interaction** stands for the bidirectional cross-impact between intent detection and slot filling can be considered, which is shown in Figure 3(b). Another series of works (Wang et al., 2018; E et al., 2019; Liu et al., 2019; Qin et al., 2021a) build the bidirectional connection across slot filling and intent detection to enhance each other.

Based on the two types of interaction, users can easily design the interaction module and interaction order via our provided classic interaction modules and customized configurations.

Classification Module. It aims to transform hidden states after the interaction module into final classification logits. There are two types of classification modules supported by OpenSLU:

- **MLP Classifier.** Multi-Layer Perceptron (MLP) Classifier is a fundamental classification decoding algorithm. Nevertheless, the method ignores the dependency across tokens.

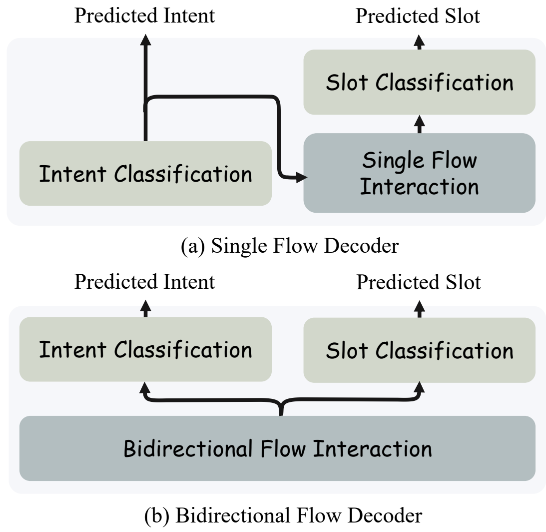


Figure 3: A brief illustration of Single Flow Decoder (a) vs. Bidirectional Flow Decoder (b).

- **LSTM Classifier.** It indicates that we adopt an LSTM classifier for the final prediction, which has the advantage of modeling the dependency of tokens (from left to right). However, it is an autoregressive classification module for SLU, which cannot be parallel to speed up the decoding prediction.

To improve the quality of SLU prediction results, we also implement several SLU tricks, like teacher-forcing and token-level intent detection (Qin et al., 2019). Users can switch between different prediction strategies by simply setting up the hyper-parameter to improve performance.

2.3 Evaluation and Metrics

Following Goo et al. (2018); Qin et al. (2021c), we support various metrics for SLU (shown in Figure 2(c)), including Slot F1 Score, Intent Accuracy, Intent F1, and Exactly Match Accuracy (EMA).

- **Slot F1 Score** (Goo et al., 2018; Qin et al., 2019) is used for assessing slot filling performance. This metric is calculated as the harmonic mean between precision and recall.

- **Intent Accuracy** (Goo et al., 2018; Qin et al., 2019) is a measure used to evaluate the accuracy of intent detection, based on the ratio of correctly predicted intents.

- **Intent F1 Score** (Gangadharaiah and Narayanaswamy, 2019; Qin et al., 2020b) is adopted to evaluate the macro F1 Score of the predicted intents in the multi-intent detection.

```
python run.py \
        --dataset atis \
        --model slot-gated
```

(a) Example for reproducing existing models.

```
accelerate config
accelerate launch \
    run.py \
    --dataset atis \
    --model slot-gated
```

(b) Example for multi-GPU finetuning.

```
python visualization.py \
   --config_path visual.yaml \
   --output_path outputs.jsonl
```

(c) Example for visualization.

```
python run.py \
    --dataset atis \
    --config_path config/dca_net.yaml
```

(d) Example for run your own model.

```
class NewInteraction(BaseInteraction):
 def __init__(self, **config):
  self.config = config
  ...

 def forward(self, hiddens: HiddenData):
  ...
  intent, slot = self.func(hiddens)
  hiddens.update_slot_hidden_state(slot)
  hiddens.update_intent_hidden_state(intent)
  return hiddens
```

(e) Example for implementing a new encoder model.

```
class NewDecoder(BaseDecoder):
 def __init__(self,
              intent_classifier,
              slot_classifier,
              interaction=None):
  ...
  self.int_cls = intent_classifier
  self.slot_cls = slot_classifier
  self.interaction = interaction

 def forward(self, hiddens: HiddenData):
  ...
  interact = self.interaction(hiddens)
  slot = self.slot_cls(interact.slot)
  intent = self.int_cls(interact.intent)
  return OutputData(intent, slot)
```

(f) Example for implementing a new decoder model.

Figure 4: Example usage of `OpenSLU`.

- **Exact Match Accuracy** (Goo et al., 2018; Qin et al., 2019, 2020b) takes intent detection as well as slot filling into account simultaneously. This metric is calculated as the ratio of sentences for which both the intent and slot are predicted correctly within a sentence.

2.4 Common Modules

Logger. We provide a generic `Logger` component to help users to track the process of model building including wandb.ai, fitlog and local file logging (see Figure 2(d)).

Applications. We provide complete scripts in the `Application` (see Figure 2(e)) for training, prediction, visual error analysis, and the final stage of model deployment.

Configuration. As shown in Figure 2(f), our toolkit employs `Configuration` module to manage the model configuration, training parameters, and training and analysis data. We will introduce more details in Section Toolkit Usage (§3).

3 Toolkit Usage

3.1 Reproducing Existing Models

For reproducing an existing model implemented by `OpenSLU` on different datasets, users are required only to specify the dataset and model by setting hyper-parameters, i.e., *model* and *dataset*. Experiments can be reproduced in a simple command line instruction, as shown in Figure 4 (a). This instruction aims to fine-tuning `Slot-Gated` (Goo et al., 2018) model on ATIS (Hemphill et al., 1990) dataset. With YAML configuration files, we can modify hyper-parameters conveniently, which allows users can reproduce various experiments quickly without modifying the source code. In addition, we designed `OpenSLU` to work on a variety of hardware platforms. If the hyper-parameter *device* is set to "*cuda*", CUDA devices will be used. Otherwise, CPU will be employed by default. As shown in Figure 4 (b), we also support distributed training on multi-GPU by setting hyper-parameters and command line parameters.

3.2 Customizable Combination Existing Components

As the model is designed as reusable modules, users can easily reuse modules via the call of interface or configuration files. More specifically, for the interface, users can call common-used encoder and decoder modules in one line of code from the pre-configured library. For configuration files, users can combine existing component libraries only through configuration files, thus creating a customized model.

It can be useful for users in cross-cutting areas, such as biology, that are unfamiliar with using Python code to create models, as it allows them to create their own models without using any Python code. Such features can potentially make it easier to build and test models more rapidly. Similarly, the customized model can be trained by specifying the relevant configuration file path and running simple command line instructions, as shown in Figure 4(d).

3.3 Implementing a New SLU Model

Since `OpenSLU` split the model into fine-grained components, users can directly reuse modules through configuration files. Specifically, when users aim to implement a new SLU model, only a few key innovative modules need to be rewritten by users, including a specific Model class and 2 functions as follows:

Model	ATIS			SNIPS		
	Slot F1.(%)	Intent Acc.(%)	EMA(%)	Slot F1.(%)	Intent Acc.(%)	EMA(%)
Non-Pretrained Models						
Slot Gated (Goo et al., 2018)	94.7	94.5	82.5	93.2	97.6	85.1
Bi-Model (Wang et al., 2018)	95.2	96.2	85.6	93.1	97.6	84.1
Stack Propagation (Qin et al., 2019)	95.4	96.9	85.9	**94.6**	97.9	87.1
DCA Net (Qin et al., 2021a)	**95.9**	**97.3**	**87.6**	94.3	**98.1**	**87.3**
Pretrained Models						
Joint BERT (Chen et al., 2019)	**95.8**	**97.9**	**88.6**	96.4	98.4	91.9
RoBERTa (Liu et al., 2020a)	95.8	97.8	88.1	95.7	98.1	90.6
ELECTRA (Clark et al., 2020)	95.8	96.9	87.1	95.7	98.3	90.1
DeBERTa$_{v3}$ (He et al., 2021)	95.8	97.8	88.4	**97.0**	**98.4**	**92.7**

Table 1: Main results of single-intent SLU. All baseline results are re-implemented by `OpenSLU`.

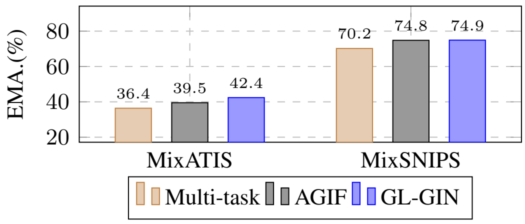


Figure 5: Main results of multi-intent SLU on EMA. All baseline results are re-implemented by `OpenSLU`.

• *__init__()* function. This function aims for parameter initialization, global variable definition, and so on. All modules can be inserted into the system by configuring the *__model_target__* hyperparameters, so as to quickly and automatically build the model.

• *forward()* function. This function mainly focuses on forward data flow and learning the parameters according to the pre-defined configuration.

In most cases, rewriting `Interaction` module is enough for building a new SLU model. As shown in Figure 4(e), this module accepts `HiddenData` data object as input and returns with `HiddenData` data object. `HiddenData` contains the *hidden_states* for intent and slot, and other helpful information. With the advancement of SLU research, patterns of decoders become increasingly complex (Xing and Tsang, 2022; Cheng et al., 2022). Therefore, to further meet the needs of complex exploration, we provide the `BaseDecoder` class, and the user can simply override the *forward()* function in class, which accepts `HiddenData` as input data format and `OutputData` as output data format, as shown in Figure 4(f).

4 Experiments

Extensive reproduction experiments are conducted to evaluate the effectiveness of `OpenSLU`.

4.1 Data Settings

In single-intent SLU, we employ two widely used benchmarks including ATIS (Hemphill et al., 1990) and SNIPS dataset (Coucke et al., 2018).

In multi-intent SLU scenario, we support 2 widely used datasets: MixATIS and MixSNPIS (Qin et al., 2020b), which are collected from the ATIS, SNIPS by simple conjunctions, e.g., “and”, to connect sentences with different intents.

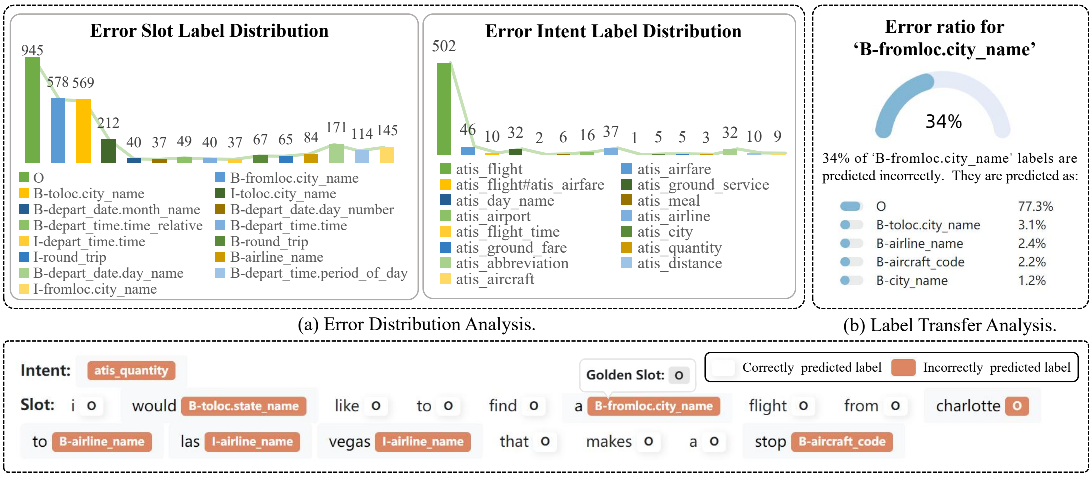
4.2 Result Reproduction

We implement various state-of-the-art SLU models. For single-intent SLU methods, we re-implement the following baselines: (1) `Slot Gated` (Goo et al., 2018); (2) `Bi-Model` (Wang et al., 2018); (3) `Stack Propagation` (Qin et al., 2019); (4) `DCA Net` (Qin et al., 2021a); (5) `Joint Bert` (Chen et al., 2019); (6) `RoBERTa` (Liu et al., 2020a); (7) `ELECTRA` (Clark et al., 2020); (8) `DeBERTa`$_{v3}$ (He et al., 2021). For multi-intent SLU methods, we adopt the following baselines: (1) `AGIF` (Qin et al., 2020b); (2) `GL-GIN` (Qin et al., 2021b).

The reproduction results are illustrated in Table 1, we observe that `OpenSLU` toolkit can reproduce the comparable results reported in previous works, which verify the effectiveness of `OpenSLU`. In addition, `OpenSLU` can outperform some reported results in previous published work, which further shows the superiority of `OpenSLU`. Meanwhile, the same trend can be observed in multi-intent SLU setting, which is shown in Figure 5.

4.3 Visualization Analysis

According to a number of studies (Vilar et al., 2006; Wu et al., 2019; Ribeiro et al., 2020; Paleyes et al., 2022), model metrics tests alone no longer adequately reflect the model’s performance. To help researchers further improve their models, we pro-



(c) Instance Analysis.

Figure 6: Visual analytics consists of three main functions including Error Distribution Analysis (a), Label Transfer Analysis (b) and Instance Analysis (c).

vide a tool for visual error analysis including three main parts: (a) error distribution analysis; (b) label transfer analysis; and (c) instance analysis (see Figure 6). And the visual analysis interface can be run with the command as shown in the Figure 4(c).

4.3.1 Error Distribution Analysis.

We provide error distribution analysis that presents the number and percentage of label errors predicted by the model. By viewing the error distributions, the model can be easily analyzed and studied qualitatively (Caubrière et al., 2020). As a result, the weaknesses of each system can be better understood and improvements can be made to the model in the future.

Take the error in Figure 6(a) as an example, a large number of '`atis_flight`' labels are incorrectly predicted compared with all other labels. Therefore, we should pay more attention on how to improve the performance of '`atis_flight`' labels.

4.3.2 Label Transfer Analysis.

Label Transfer Analysis module first offers the percentage of incorrect predictions for each label and provides the probability of being incorrectly predicted as each of the other labels to present a fine-grained statistics for a better understanding of issues such as invisible bias in the model (Wu et al., 2019; Ribeiro et al., 2020).

For example, Figure 6(b) shows the details in incorrect prediction on '`B-fromloc.city_name`'. We observe 34% of '`B-fromloc.city_name`' predict incorrectly and 77.3% of error labels are predicted as '`O`'. By having access to this information, users can be better guided to improve their data or label learning methods to prevent those error predictions.

4.3.3 Instance Analysis.

In order to provide a better case study, `OpenSLU` offers a instance-level analysis view by highlighting error results and interactively checking all golden labels (shown in Figure 6(c)). Such instance analysis allows users to examine data on a case-by-case basis in an intuitive way. This can be seen easily in Figure 6(c), where token '*a*' is predicted as '`B-fromloc.city_name`' instead of '`O`'.

Furthermore, we also deploy `OpenSLU` into the Gradio[2] platform, which allows users to connect the demo directly to the public network and access it via the computer or mobile device.

5 Conclusion

This paper introduces `OpenSLU`, a unified, modularized, and extensible toolkit for spoken language understanding. In our toolkit, we implement 10 models on both single- and multi-intent SLU settings, both covering the categories of non-pretrained and pretrained language models. Our toolkit can be easily applied to other SLU settings, which is extensible to support seamless incorporation of other external modules. To the best of our knowledge, this is the first open-resource toolkit for SLU and we hope `OpenSLU` can attract more breakthroughs in SLU. In the future, we can extend `OpenSLU` to support cross-lingual (Qin et al., 2020a; Zheng et al., 2022) and profile (Xu et al., 2022) SLU scenario.

[2]https://www.gradio.app

Acknowledgements

This work was supported by the National Key R&D Program of China via grant 2020AAA0106501 and the National Natural Science Foundation of China (NSFC) via grant 62236004 and 61976072. Libo Qin is the corresponding author.

References

Inigo Casanueva, Ivan Vulić, Georgios Spithourakis, and Paweł Budzianowski. 2022. NLU++: A multi-label, slot-rich, generalisable dataset for natural language understanding in task-oriented dialogue. In *Findings of the Association for Computational Linguistics: NAACL 2022*, pages 1998–2013, Seattle, United States. Association for Computational Linguistics.

Antoine Caubrière, Sahar Ghannay, Natalia Tomashenko, Renato De Mori, Antoine Laurent, Emmanuel Morin, and Yannick Estève. 2020. Error analysis applied to end-to-end spoken language understanding. In *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 8514–8518. IEEE.

Qian Chen, Zhu Zhuo, and Wen Wang. 2019. Bert for joint intent classification and slot filling. *arXiv preprint arXiv:1902.10909*.

Lizhi Cheng, Wenmian Yang, and Weijia Jia. 2022. A scope sensitive and result attentive model for multi-intent spoken language understanding. *arXiv preprint arXiv:2211.12220*.

Kevin Clark, Minh-Thang Luong, Quoc V. Le, and Christopher D. Manning. 2020. ELECTRA: Pre-training text encoders as discriminators rather than generators. In *International Conference on Learning Representations*.

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*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Haihong E, Peiqing Niu, Zhongfu Chen, and Meina Song. 2019. A novel bi-directional interrelated model for joint intent detection and slot filling. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 5467–5471, Florence, Italy. Association for Computational Linguistics.

Rashmi Gangadharaiah and Balakrishnan Narayanaswamy. 2019. Joint multiple intent detection and slot labeling for goal-oriented dialog. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 564–569, Minneapolis, Minnesota. Association for Computational Linguistics.

Chih-Wen Goo, Guang Gao, Yun-Kai Hsu, Chih-Li Huo, Tsung-Chieh Chen, Keng-Wei Hsu, and Yun-Nung Chen. 2018. Slot-gated modeling for joint slot filling and intent prediction. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 753–757, New Orleans, Louisiana. Association for Computational Linguistics.

Pengcheng He, Jianfeng Gao, and Weizhu Chen. 2021. Debertav3: Improving deberta using electra-style pre-training with gradient-disentangled embedding sharing. *arXiv preprint arXiv:2111.09543*.

Charles T. Hemphill, John J. Godfrey, and George R. Doddington. 1990. The ATIS spoken language systems pilot corpus. In *Speech and Natural Language: Proceedings of a Workshop Held at Hidden Valley, Pennsylvania, June 24-27,1990*.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780.

Changliang Li, Liang Li, and Ji Qi. 2018. A self-attentive model with gate mechanism for spoken language understanding. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3824–3833, Brussels, Belgium. Association for Computational Linguistics.

Yijin Liu, Fandong Meng, Jinchao Zhang, Jie Zhou, Yufeng Chen, and Jinan Xu. 2019. CM-net: A novel collaborative memory network for spoken language understanding. 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 1051–1060, Hong Kong, China. Association for Computational Linguistics.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2020a. RoBERTa: A robustly optimized BERT pretraining approach.

Zihan Liu, Genta Indra Winata, Zhaojiang Lin, Peng Xu, and Pascale Fung. 2020b. Attention-informed mixed-language training for zero-shot cross-lingual task-oriented dialogue systems. *Proceedings of the AAAI*

Conference on Artificial Intelligence*, 34(05):8433–8440.

Nikita Moghe, Evgeniia Razumovskaia, Liane Guillou, Ivan Vulić, Anna Korhonen, and Alexandra Birch. 2022. Multi3nlu++: A multilingual, multi-intent, multi-domain dataset for natural language understanding in task-oriented dialogue. *arXiv preprint arXiv:2212.10455*.

Andrei Paleyes, Raoul-Gabriel Urma, and Neil D. Lawrence. 2022. Challenges in deploying machine learning: A survey of case studies. *ACM Comput. Surv.*, 55(6).

Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. GloVe: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar. Association for Computational Linguistics.

Libo Qin, Wanxiang Che, Yangming Li, Haoyang Wen, and Ting Liu. 2019. A stack-propagation framework with token-level intent detection for spoken language understanding. 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 2078–2087, Hong Kong, China. Association for Computational Linguistics.

Libo Qin, Tailu Liu, Wanxiang Che, Bingbing Kang, Sendong Zhao, and Ting Liu. 2021a. A co-interactive transformer for joint slot filling and intent detection. In *ICASSP 2021 - 2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 8193–8197.

Libo Qin, Minheng Ni, Yue Zhang, and Wanxiang Che. 2020a. Cosda-ml: Multi-lingual code-switching data augmentation for zero-shot cross-lingual nlp. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*, pages 3853–3860. International Joint Conferences on Artificial Intelligence Organization. Main track.

Libo Qin, Fuxuan Wei, Tianbao Xie, Xiao Xu, Wanxiang Che, and Ting Liu. 2021b. GL-GIN: Fast and accurate non-autoregressive model for joint multiple intent detection and slot filling. 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 178–188, Online. Association for Computational Linguistics.

Libo Qin, Tianbao Xie, Wanxiang Che, and Ting Liu. 2021c. A survey on spoken language understanding: Recent advances and new frontiers. In *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*, pages 4577–4584. International Joint Conferences on Artificial Intelligence Organization. Survey Track.

Libo Qin, Xiao Xu, Wanxiang Che, and Ting Liu. 2020b. AGIF: An adaptive graph-interactive framework for joint multiple intent detection and slot filling. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 1807–1816, Online. Association for Computational Linguistics.

Marco Tulio Ribeiro, Tongshuang Wu, Carlos Guestrin, and Sameer Singh. 2020. Beyond accuracy: Behavioral testing of NLP models with CheckList. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4902–4912, Online. Association for Computational Linguistics.

Gokhan Tur and Renato De Mori. 2011. *Spoken language understanding: Systems for extracting semantic information from speech*. John Wiley & Sons.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

David Vilar, Jia Xu, L. F. D'Haro, and Hermann Ney. 2006. Error analysis of statistical machine translation output. In *International Conference on Language Resources and Evaluation*, pages 697–702.

Yu Wang, Yilin Shen, and Hongxia Jin. 2018. A bi-model based RNN semantic frame parsing model for intent detection and slot filling. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 309–314, New Orleans, Louisiana. Association for Computational Linguistics.

Tongshuang Wu, Marco Tulio Ribeiro, Jeffrey Heer, and Daniel Weld. 2019. Errudite: Scalable, reproducible, and testable error analysis. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 747–763, Florence, Italy. Association for Computational Linguistics.

Bowen Xing and Ivor Tsang. 2022. Co-guiding net: Achieving mutual guidances between multiple intent detection and slot filling via heterogeneous semantics-label graphs. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 159–169, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Xiao Xu, Libo Qin, Kaiji Chen, Guoxing Wu, Linlin Li, and Wanxiang Che. 2022. Text is no more enough! a benchmark for profile-based spoken language understanding. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 11575–11585.

Bo Zheng, Zhouyang Li, Fuxuan Wei, Qiguang Chen, Libo Qin, and Wanxiang Che. 2022. HIT-SCIR at MMNLU-22: Consistency regularization for multilingual spoken language understanding. In *Proceedings of the Massively Multilingual Natural Language Understanding Workshop (MMNLU-22)*.