Towards Need-Based Spoken Language Understanding Model Updates: What Have We Learned?

Quynh Do* Amazon Alexa AI doquynh@amazon.de

Judith Gaspers* Amazon Alexa AI gaspers@amazon.de

Daniil Sorokin* Amazon Alexa AI dsorokin@amazon.de

Patrick Lehnen Amazon Alexa AI plehnen@amazon.de

Abstract

In productionized machine learning systems, online model performance is known to deteriorate over time when there is a distributional drift between offline training and online application data. As a remedy, models are typically retrained at fixed time intervals, implying high computational and manual costs. This work aims at decreasing such costs in productionized, large-scale Spoken Language Understanding systems. In particular, we develop a need-based re-training strategy guided by an efficient drift detector and discuss the arising challenges including system complexity, overlapping model releases, observation limitation and the absence of annotated resources at runtime. We present empirical results on historical data and confirm the utility of our design decisions via an online A/B experiment.

1 Introduction

Traditionally, a Spoken Language Understanding (SLU) system like Google Assistant, Siri or Alexa, is a cascade of an Automatic Speech Recognition (ASR) component converting speech into text followed by a Natural Language Understanding (NLU) component that interprets the meaning of the text through domain classification (DC), intent classification (IC) and slot filling (SF). Once deployed to customers, the machine learning (ML) models implemented for DC, IC and SF may experience distributional drifts between offline training and online application data over time which leads to serious performance degrades. This is known as a *model drift* phenomenon. In most real-world cases, model drift can be avoided by retraining the ML models with regular cadence. However, this strategy often entails a significant amount of human, computational and environmental costs in each release cycle, especially for large systems. Hence, it should be optimized with an intelligent decision making mechanism that is able to automatically predict if a particular model needs to be updated or can be left in place as is for another release cycle.

Given a deployed ML model, *drift detection* is a task to identify model drifts when the model is applied on a new data set, and therefore can be used to guide decisions on when to retrain the model. However, academic work in this field often makes simplifying assumptions that render the problem more manageable but do not hold in the industrial practice, e.g., that there is exactly one model that needs to be kept or updated, and that it is possible to create non-overlapping detection and reference data windows, which are aligned with a model release cycle. This makes productionizing drift detection nontrivial.

This paper describes our effort to develop a drift detector to produce decisions whether to update the NLU models in the next release cycle for a productionized SLU system. First, we generally discuss the challenges that may arise when applying drift detection on a large-scale SLU production system: i) the complexity when the SLU system is supporting multiple domains with several ML models and architectures; ii) the possible overlapping model releases in production; iii) the limited number of observations for each individual ML model; iv) the absence of annotated SLU data at runtime. Moreover, for each challenge, we consider the necessary system design decisions and possible solutions that are needed to apply drift detection in practice. Finally, we describe our offline and online experiments on real-world SLU data to confirm the utility of our design decisions.

2 Background on drift detection

Lu et al. (2018) classifies automatic drift detection methods into three categories: i) methods which rely on labeled data to monitor error rates, ii) methods which use distance measures to estimate the similarity between distributions of previous and

*These authors contributed equally to this work

current (unlabeled) data, and iii) methods that make use of multiple hypothesis tests to detect concept change. While the first category requires manually labeled data representing the current data distribution, the last two categories require at least two data windows: a *reference window* containing the instances that belong to the same distribution that was used to train the most recent model, and a *detection window* which represents the current data distribution. The detection window can consist of unlabeled data only (Gemaque et al., 2020). Thus, the methods of the second and the third categories can be both classified as unsupervised drift detection methods (Elsahar and Gallé, 2019; Qin et al., 2021). For example, Koh (2016) compare the reference and detection data windows by Hoeffding bound. The difference in terms of sample means between both the windows is compared to a value ϵ defined by the Hoeffding bound. Then, a drift is signaled when this difference is greater than ϵ. An obvious advantage of unsupervised methods is that they do not require labeled data. However, it can be difficult to interpret the drift signal due to the lack of an indication on how much the performance drop is.

While drift detection is considered as one of the stages in modern AI workflows, corresponding work in NLP and SLU has been limited. Recently, Do et al. (2021) have proposed a regression model to detect temporal performance drop in SLU. The authors evaluated their approach via small-scale simulated release cycles for a joint IC+SF model in isolation, thus abstracting away from the complexity of a production SLU system. They built one regression model per domain, assuming the availability of a large number of previous model releases for training the regression model, which is unrealistic for many real-world scenarios.

3 NLU drift detection definition

In this work, we consider multi-domain NLU as one part of a more complex production SLU system, leaving out ASR and other components. The main NLU tasks include DC, IC and SF, and there are different ways to approach them. Usually, pipelined systems are constructed with DC being applied as the first step to determine the domain for a given utterance. Subsequently, the utterance is fed into the corresponding domain-specific IC+SF model that jointly detects the intent and extracts semantic slots from the utterance. For instance, an ASR transcribed utterance "play Hello by Adele" can be parsed into {domain: Music, intent: "play", song name (slot): "Hello", artist (slot): "Adele"}.

To simplify the problem, we focus on building a drift detector to decide on IC+SF model updates. In our experiments, we make *an assumption* that DC models also face distributional drifts when their corresponding IC+SF models do, since their data ages are usually similar.

Given a trained IC+SF model $\mathcal{M}$, a data *reference window*, $\mathcal{W}_{ref}$, containing the testing instances considered belonging to the data distribution at the time $\mathcal{M}$ was developed offline, and a *detection window*, $\mathcal{W}_{detect}$, which contains the testing utterances representing the data distribution when the model update decision needs to be made. Then, we define that $\mathcal{M}$ has suffered a drift if the error rate on the detection window exceeds the error rate on the reference window by a certain threshold:

$$\Delta_E = \mathrm{E}(\mathcal{M}, \mathcal{W}_{detect}) - \mathrm{E}(\mathcal{M}, \mathcal{W}_{ref}) > \alpha \quad (1)$$

where α is a drift threshold, and E is a pre-defined function to compute an error rate. In this work, E is a semantic error rate and defined as follows:

$$\textit{SemER} = \frac{\#(\text{slot+intent errors})}{\#\text{slots in reference} + 1} \quad (2)$$

A drift detector should be able to identify automatically whether $\mathcal{M}$ has suffered a drift or not.

4 NLU system and challenges

In this section, we describe the considered NLU system, and discuss the potential challenges arising when applying drift detection on such a system.

4.1 NLU architecture and Challenge 1 - system complexity

In this work, we consider a real-world SLU system with multiple domains and each of them has a single IC+SF model. Each IC+SF model is a combination of a pre-trained encoder and two separate decoders for the target tasks. Depending on its domain, the IC+SF model may receive gazetteers as an additional token-level input in parallel to the BERT-encoder embeddings, resulting in two variants of IC+SF model architectures. In the rest of this paper, we refer the gazetteer-based and non-gazetteer architectures as *Gaz* and *Non-Gaz*, respectively.

Traditionally, academic work on drift detection often assumes that there is exactly one model that

needs to be kept or updated to make the problem more manageable. Unfortunately, it does not hold in industrial practice. As in our case, a real-world SLU system is often multi-domain. Each domain IC+SF model can be updated individually and there is no requirement for these models to have the same architecture. Thus, we face the first challenge: A separate drift detector should be developed per domain and architecture or a single detector needs to cover all supported domains.

4.2 NLU lifecycle and Challenge 2 - overlapping model releases

Figure 1 presents a simplified SLU model production lifecycle, where each release has three main phases: Build, Deploy and Production. Let us apply the drift detection definition (Sec. 3) to the lifecycle of a single IC+SF domain model. For IC+SF model $\mathcal{M}_N^{\mathcal{D}}$ that was released for a domain $\mathcal{D}$ during a release cycle N, the task is to predict if there is a drift after $\mathcal{M}_N^{\mathcal{D}}$ was deployed (that is if $\Delta_E > \alpha$) using the data window $\mathcal{W}_{ref}$ collected during the model build and the window $\mathcal{W}_{detect}$ collected after the deployment. Consequently, the drift detection decision indicates if $\mathcal{M}_N^{\mathcal{D}}$ needs to be updated during the release cycle $N+1$. Thus, the detection window $\mathcal{W}_{detect}$ must be available before the build phase of release $N+1$, but after the deployment of N.

However, in practice, the build, deploy and production phases of subsequent releases may overlap significantly which make defining disjoint $\mathcal{W}_{ref}$ and $\mathcal{W}_{detect}$ for releases N and $N+1$ our second challenge. For a complex and large-scale production system, the required time for each phase is considerable and the human and computational cost of each phase are usually distributed between teams. Once the model development team is finished with the latest model build, it is handed over for deployment and the team can start the work on the next release. Therefore, overlaps tend to occur between the cycles of two consecutive releases. In Figure 1, the Build phase of release $N+1$ starts before the model of release N goes to production. In this case, the data from the detection window for the current release after its deployment is not yet available at the start of the build phase for the next release. Thus, $\mathcal{W}_{detect}$ is not available for application of a drift detector.

To overcome this challenge, we take that the online data collected after the development of a

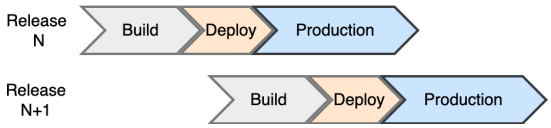

Figure 1: A simplified NLU lifecycle with three phases per release: Build, Deploy and Production. The phases overlap for subsequent releases and the development for the next release $N+1$ may start as soon as the previous release is deployed.

model for a release N during its build phase is indicative of the drift that might occur after the model go to the deployment stage.

Since the build phase of a model is sufficiently long so that new data can be collected that didn't go into the model development, we will use this data for $\mathcal{W}_{detect}$. We adjust the definitions for the reference and detection windows as follows:

- *Reference window*: De-identified online data that was manually transcribed and labeled with domain, intent and slots and that was collected before the build phase of a release. This data is used for testing during the build phase. We use the annotated domain labels to split that data between IC+SF domain models. The gold domain labels are used to decide the data flow for each domain's IC+SF model.

- *Detection window*: De-identified online data that was automatically transcribed and automatically classified into domains with the current release DC model and that was collected after the Build phase of N has started, but before the Build phase of $N+1$. We use the automatically generated domain labels to filter the data for a specific domain.

These definitions result in a time gap between the defined detection window and the real online window, during which the IC+SF model will be deployed. We evaluate this decision in an online experiment in Section 7.

4.3 Challenge 3 - limited observations

The drift detection problem is domain dependent and different domains may experience different drift patterns. For example, the Video domain should observe large drifts more often than the Calendar domain. That would call for learning a separate drift detection function for each domain in production. To learn a drift detector, we need to collect historical data of model releases and reference and detection windows. And to learn a domain

specific drift detector the same data needs to be collected per domain, leading to the third challenge:

For a real-world SLU system, only a very limited amount of available historical data points may be available for training per domain. The availability of historical data for learning a drift detector is restricted by the age of the production system and privacy guidelines for data and model storage. This implies that a per-domain drift detector needs to be learned on a handful of data points, which is often infeasible.

To avoid this issue, we build a single drift detector for all IC+SF domain models of the same type (Gaz or Non-Gaz). We evaluate a single unified drift detection function for multiple domains in the next sections in offline experiments.

4.4 Challenge 4 - the absence of annotated data at runtime

The amount of unlabeled live data flowing into a production SLU system may be on the order of a million of utterances per day. However, due to the associated costs, only a comparatively small subset gets manually transcribed and annotated, leading to the forth challenge: Manually labeled data for any given period in time may be limited, with potentially only few – or in rare cases even none – manually labeled utterances being available for low-frequency domains in certain time periods. In addition, since manually transcribing and annotating data takes time, this data may not be available at runtime to construct the detection window. As mentioned in the previous section, we use the ASR and DC components to obtain textual per-domain data. The amount of data instances per window can be huge in production, which makes data processing a challenge and may slow down the process. As drift detection should enable quick decisions, we downsample the data amounts to a reasonable size. Since we cannot rely on manually annotated data to extract the features for the detection window, we focus on methods that require only unlabeled data at runtime.

5 Supervised drift detection from unsupervised signals

5.1 Learning problem definition

Motivated by recent work, which successfully predicts a performance drop using unsupervised signals (Elsahar and Gallé, 2019; Do et al., 2021), we aim to learn a function to map from a set of measures estimating the similarity between the reference and detection data windows to a binary label indicating whether a significant performance drop occurs or not. By predicting directly if a model drift and performance drop occurred instead of estimating a magnitude of the drift, we simplify the learning problem, so that it can be approached with only a limited amount of training data points available. At the same time, the drift detector predictions remain clearly interpretable for the user.

More formally, we learn a function predicting if there is a performance drop when the test data window for model $\mathcal{M}$ trained on $\mathcal{W}_{train}$ is moved from $\mathcal{W}_{ref}$ to $\mathcal{W}_{detect}$:

$$F(f_1(x), f_2(x), \ldots) \rightarrow \{Drop, \textit{No-Drop}\} \quad (3)$$

where $f_1, f_2, \ldots$ are features, x is a data instance containing the information about $\mathcal{M}$, $\mathcal{W}_{train}$, $\mathcal{W}_{ref}$ and $\mathcal{W}_{detect}$. The *No-Drop* label indicates that $\Delta_{SemER} \leq \alpha$ while the *Drop* label indicates that $\Delta_{SemER} > \alpha$.

5.2 Features and learning algorithms

For $f_1, f_2, \ldots$ in Equation 3, we explore 17 features representing the differences between $\mathcal{W}_{ref}$ and $\mathcal{W}_{detect}$ as follows:

Discriminative classifier: Disriminative classifiers have been used for drift detection (Elsahar and Gallé, 2019; Do et al., 2021). When $\mathcal{W}_{ref}$ and $\mathcal{W}_{detect}$ are separable by a discriminative classifier, it is likely that there is a drift. In this work, we apply a Logistic Regression classifier trained on the pre-trained BERT sentence representations as the discriminative classifier, and its 5-fold cross-validation accuracy on each of the unions of ($\mathcal{W}_{ref}$, $\mathcal{W}_{detect}$), ($\mathcal{W}_{ref}$, $\mathcal{W}_{train}$), and ($\mathcal{W}_{detect}$, $\mathcal{W}_{train}$) is used as the drift signal.

Distributional distance: For each of the window pairs ($\mathcal{W}_{ref}$, $\mathcal{W}_{detect}$), ($\mathcal{W}_{ref}$, $\mathcal{W}_{train}$), and ($\mathcal{W}_{detect}$, $\mathcal{W}_{train}$), we compute Euclidean and Cosine distances between two mean pre-trained BERT sentence representations.

Prediction confidence differences: Confidence scores have been proven to be effective in detecting drifts (Do et al., 2021). We use the SF and IC prediction confidence scores separately. For each case, we compute the difference between the mean confidence scores that model $\mathcal{M}$ obtains on the two data windows. We also compute the average, minimum and maximum difference between the per tag mean IC confidence scores on the two data windows $\mathcal{W}_{ref}$ and $\mathcal{W}_{detect}$.

Kullback-Leibler (KL) divergence at token level: KL divergence measures the difference between two probability distributions and is also known as the relative entropy. It has been used as an effective measure in drift detection (Lindstrom et al., 2011). We consider each data window as a large text and compute the KL divergence between the token distributions of the two texts representing each of the data window pairs among $\mathcal{W}_{ref}$, $\mathcal{W}_{detect}$ and $\mathcal{W}_{train}$.

Targeting fast drift detection, to learn F we apply classical binary classification algorithms like *multinomial logistic regression, k-nearest neighbors, or decision tree*. In some cases, it may be desired to reduce the amount of features, e.g., when we have a limited amount of training data points. Therefore, we include feature selection methods into our experiments.

6 Offline experiment

We evaluate the proposed feature set and the selected learning algorithms for Gaz and Non-Gaz architectures using historic data.

6.1 Experimental set-up

In offline experiments, we build two drift detection models for two different IC+SF architectures: i) *Gaz*: Including domains which use gazetteers as an extra input. ii) *Non-Gaz*: Including domains which do not use gazetteers. Following previous work (Do et al., 2021), we define the baseline and metrics as follows. Given N NLU models and historic performance data, a subset $S \subseteq N$ are the models that have no drift (that is, on historic data windows $\Delta_{SemER} \leq \alpha$) and thus belong to the class *No-Drop*. Using the learned drift detector F, $P \subseteq N$ models are predicted as low-risk of being drifted, i.e. predicted as *No-Drop*. To evaluate F, we compute precision: $P = \frac{|S \cap P|}{|P|}$ and recall: $R = \frac{|S \cap P|}{|S|}$ for the *No-Drop* class.

We consider precision for the class *No-Drop* as the most important metric in a user-facing setting. This class includes the models that can be left in place for the next release, thus having the potential to negatively impact customer satisfaction. If a drifted model is left in place, then there is the risk of increased friction for the customers. Recall does not have an impact on customer satisfaction, but on training costs, which we consider less important than customer satisfaction. We aim to reduce training costs without negatively impacting customers.

α	B_p	Model	Feat.	R	P
0.0	45.0	MLR	ALL	55.6	56.8
		Knn	ALL	46.7	55.3
		DT	ALL	57.8	56.5
		MLR	SUB	55.6	**75.8**
		Knn	SUB	68.9	70.5
		DT	SUB	57.8	68.4
0.002	46.0	MLR	ALL	56.5	61.9
		Knn	ALL	43.5	54.1
		DT	ALL	60.9	56.0
		MLR	SUB	58.7	**75.0**
		Knn	SUB	56.5	59.1
		DT	SUB	58.7	61.4
0.01	59.0	MLR	ALL	72.9	**75.4**
		Knn	ALL	66.1	70.9
		DT	ALL	66.1	72.2
		MLR	SUB	84.7	73.5
		Knn	SUB	72.9	72.9
		DT	SUB	83.1	**75.4**

Table 1: Evaluation results for non-Gaz drift detection model on class No-Drop. B_p, R, P are the precision baseline, Recall and Precision, respectively.

Therefore, our goal is to build a drift detector which reaches a high precision and an acceptable recall for the drift class *No-Drop*. We compare our models against a baseline (B_p) obtained by selecting instances for the *No-Drop* class randomly.

We collected past NLU model release data points resulting in 134 instances for *Gaz* and 100 instances for *Non-Gaz* model architectures. For each release model instance, the utterances representing $\mathcal{W}_{ref}$ and $\mathcal{W}_{detect}$ windows are sampled. All data used in our experiments was de-identified.

We compare the performance of three binary classifiers to learn F: multinomial logistic regression (MLR), k-nearest neighbors (Knn) and decision tree (DT). The classifiers are built using all features (ALL) defined in Section 5.1 or a selected subset of features (SUB) obtained with correlation-based feature selection (Hall, 1999). For training classifiers and feature selection, we use scikit-learn (Pedregosa et al., 2011). We report 10-fold cross-validation performance for the *No-Drop* class with three different drift thresholds α, used to assign the gold labels based on the historic data: 0.0, 0.02, and 0.1.

6.2 Results

Table 1 and Table 2 show the offline evaluation results for the two model architecture types. In most cases, drift detection models outperform the precision baseline. MLR reaches the best precision of 75.8% to select models to be left in place for the next release on non-gazetteer data instances ($\alpha =$

α	B_p	Model	Feat.	R	P
0.0	47.01	MLR	ALL	50.8	58.2
		Knn	ALL	38.1	45.3
		DT	ALL	68.3	59.7
		MLR	SUB	49.2	**72.1**
		Knn	SUB	50.8	59.3
		DT	SUB	69.8	65.7
0.002	50.75	MLR	ALL	63.2	58.1
		Knn	ALL	42.6	49.2
		DT	ALL	50.0	65.4
		MLR	SUB	76.5	64.2
		Knn	SUB	60.3	64.1
		DT	SUB	61.8	**64.6**
0.01	61.19	MLR	ALL	78.0	61.0
		Knn	ALL	48.8	58.0
		DT	ALL	85.4	71.4
		MLR	SUB	93.9	65.8
		Knn	SUB	59.8	68.1
		DT	SUB	80.5	**72.5**

Table 2: Evaluation results for Gaz drift detection model on class No-Drop. B_p, R, P are the precision baseline, Recall and Precision, respectively.

0.0) compared to the random baseline precision of 45.0%. There seems be no benefit to use a higher threshold to create the gold Drop/No-Drop labels to train a drift detector, so we set α to 0.0 in the online experiment.

Feature selection is often useful in boosting the drift detection performance. From 17 features, depending on the dataset, 1 to 4 features were normally selected. The following features were selected at least once: The Euclidean distance between the mean pre-trained BERT sentence representations, the difference between two mean SF confidence scores, the difference between two mean IC confidence scores, and discriminative classification score. Among these feature, the difference between the two mean SF confidence scores seems to be the most important feature, as it is selected in all cases.

7 Online A/B experiment

We conducted an online A/B experiment to evaluate our drift detection approach in the context of a complex large-scale production SLU system (i.e., including several components in addition to NLU, such as ASR). We picked a point in time in which all domain NLU models were scheduled for re-training and re-deployment. The production system with all models updated served as the A model. Our B system comprises exactly the same components, except that we re-trained only a subset of the domain NLU models, according to the decision of our drift detector. We acknowledge that it would be desirable to include another baseline model with none of the domain models updated into the comparison. Yet, simply leaving all domain models in place as they are increases the risk of model drift, thus potentially increasing friction for the customers being exposed to such a system.

We generated the features for each domain IC+SF model and applied our MLR drift detection model with feature selection and drift threshold $\alpha = 0.0$ (see Section 6). A low threshold for the SemER drop also reduces the risk of negatively impacting customers. If the predicted class was *No-Drop*, the IC+SF model for the domain was left in place, otherwise it was retrained on current traffic. Roughly half of the domain models were re-trained vs left in place. In this work, we assumed that a DC model faces model drift simultaneously with its corresponding IC+SF model as their data ages were usually similar. Therefore, the DC model also followed the same retraining decision as its corresponding IC+SF unless there was a special circumstance like the appearance of a new domain in the next release. Both, the A and the B systems were deployed, and we ran our experiment over 10 days.

By applying a need-based approach to domain model re-training in the B system, our main goal is to decrease costs for model re-training compared to the A system (in which all domain models are re-trained), while keeping model performance the same. To measure the impact on performance, we have monitored online friction metrics that reflect the overall end-to-end system performance (as opposed to the NLU only component in the offline experiments). Unlike in the offline experiments, all metrics were computed automatically and concern overall system performance rather than NLU in isolation. We compared the A and B system's performances using the selected online metrics, indicating that there was no significant increase recorded in any of the error metrics for the B system compared to the A system. Thus, we conclude that even though we left around half of the domain models in place according to the decision of our drift detector, there was no negative effect on overall system performance. Due to re-training fewer models, we observed a decrease in costs for expensive GPU instances of 46.4% for training IC+SF models (no GPUs were used for detector building).

8 Conclusions

We presented an efficient drift detection approach to guide IC+SF model retraining decisions, which requires only unlabeled data during the application phase in a multi-domain large-scale SLU system. We discussed the challenges that we faced while developing the approach and the corresponding design decisions to address them. We presented experimental results using historical data and we evaluated our approach via both offline and online experiments with a large-scale SLU system, confirming the utility of our design decisions.

Acknowledgements

We would like to thank Yannick Versley, Caglar Tirkaz, Tobias Falke and Debjit Paul for valuable feedback on this work.

References

Quynh Do, Judith Gaspers, Daniil Sorokin, and Patrick Lehnen. 2021. Predicting temporal performance drop of deployed production spoken language understanding models. In *Proc. Interspeech*.

Hady Elsahar and Matthias Gallé. 2019. To annotate or not? predicting performance drop under domain shift. 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 2163–2173, Hong Kong, China. Association for Computational Linguistics.

Rosana Noronha Gemaque, Albert Franca Josua Costa, Rafael Giusti, and Eulanda Miranda dos Santos. 2020. An overview of unsupervised drift detection methods. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 10.

Mark A. Hall. 1999. *Correlation-based Feature Selection for Machine Learning*. Ph.D. thesis.

Yun Sing Koh. 2016. Cd-tds: Change detection in transactional data streams for frequent pattern mining. *2016 International Joint Conference on Neural Networks (IJCNN)*, pages 1554–1561.

Patrick Lindstrom, Brian Mac Namee, and Sarah Jane Delany. 2011. Drift detection using uncertainty distribution divergence. In *2011 IEEE 11th International Conference on Data Mining Workshops*, pages 604–608.

Jie Lu, Anjin Liu, Fan Dong, Feng Gu, Joao Gama, and Guangquan Zhang. 2018. Learning under concept drift: A review. *IEEE Transactions on Knowledge and Data Engineering*, page 1–1.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

Libo Qin, Tianbao Xie, Wanxiang Che, and Ting Liu. 2021. A survey on spoken language understanding: Recent advances and new frontiers.