NLUBroker: A Flexible and Responsive Broker for Cloud-based Natural Language Understanding Services

Lanyu Xu
Wayne State University
lanyu.xu@wayne.edu

Arun Iyengar
IBM T.J. Watson Research Center
aruni@us.ibm.com

Weisong Shi
Wayne State University
weisong@wayne.edu

Abstract

Cloud-based Natural Language Understanding (NLU) services are getting more and more popular with the development of artificial intelligence. More applications are integrated with cloud-based NLU services to enhance the way people communicate with machines. However, with NLU services provided by different companies powered by unrevealed AI technology, how to choose the best one is a problem for users. To our knowledge, there is currently no platform that can provide guidance to users and make recommendations based on their needs. To fill this gap, in this paper, we propose NLUBroker, a platform to comprehensively measure the performance indicators of candidate NLU services, and further provide a broker to select the most suitable service according to the different needs of users. Our evaluation shows that different NLU services leading in different aspects, and NLUBroker is able to improve the quality of experience by automatically choosing the best service. In addition, reinforcement learning is used to support NLUBroker by an intelligent agent in a dynamic environment, and the results are promising.

1 Introduction

The widespread development of computerized natural language understanding (NLU) has resulted in several cloud-based NLU products and services [1, 6, 9, 17] and has greatly affected the way people interact with machines. Dialog systems, especially goal-oriented dialog systems, are now commonly used to support applications requiring NLU capabilities [10]. With time-sensitive and compute-intensive requirements, NLU services use cloud computing to train language models with machine learning algorithms for high accuracy and scalable service. In developed applications, NLU services are often integrated with other services such as storage and analytic services on the cloud as well as third-party applications to provide enhanced functionality.

With the burgeoning of NLU services, two challenging but practical questions arise: (a) *how good is one NLU service compared to others?* (b) *how to choose the proper service for an application with a specific purpose?* Answering these two questions will greatly benefit application developers using NLU services.

Utterance	play	a	symphony	by	Beethoven
Slot	O	O	B-music_item	O	B-artist
Intent	PlayMusic				

Table 1: An example utterance with annotations of semantic slots in IOB format and intent, which indicates the slot of the music item, and the artist with the intent PlayMusic.

We are able to answer the first question by analyzing service performance qualitatively through some general metrics. NLU performance is represented in two parts: intent detection and slot filling. Intent detection is the utterance level classification, to classify the end user intents given diversely expressed utterances in natural language. Popular detection methods include support vector machines (SVM), convolutional neural networks (CNN) [14] and recurrent neural networks (RNN) [16]. Slot filling is the word level sequential labeling task to understand utterances with finer granularity and usually performs well on conditional random field (CRF) [20], RNN [23] and attention model [8]. Take a play music utterance as an example, "*play a symphony by Beethoven*" (Table 1)[1]. Different slot types are labeled for each word, and a specific intent is labeled for the whole utterance. Thus, to compare the performance of different NLU services, the performance of intent and slot filling can be used, as well as other metrics such as cost and responsiveness which are generally important considerations for a cloud-based service. The second question is an extension of the first question and can be resolved once the performance of each service provider is compared and analyzed. Based on the comparison result, users will find the most appropriate one for the required scenario easily.

In this paper, we propose NLUBroker to address the two

[1] IOB format: a common token tagging format. B- prefix indicates the beginning of a tag, I- prefix indicates the tag is inside a chunk, and O means a token belongs to no chunk.

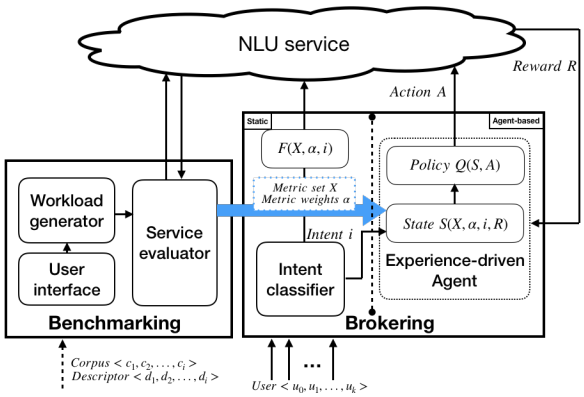


Figure 1: The design of NLUBroker.

aforementioned questions. NLUBroker has a general benchmarking module to make a comprehensive evaluation of NLU services. This module contributes to the brokering module by initializing broker parameters with the evaluation results. The brokering module takes over the data flow distribution for the application when several cloud providers are available. By leveraging a reinforcement learning (RL) algorithm, the brokering module can achieve better scalability. The remainder of this paper is organized as follows. Section 2 describes the NLUBroker design, including the benchmarking and the brokering, as well as the underlying mechanisms. NLUBroker is evaluated in Section 3. Related work is introduced in Section 4. Finally, Section 5 concludes the paper.

2 Design

In this section, we introduce the design of NLUBroker. Our system is based on the assumption that users have access to train and use models on multiple service providers. Figure 1 illustrates the system architecture of NLUBroker which is composed of two components: the benchmarking module and the brokering module. The two components are detachable from each other since each module itself can have its own input parameters, while the two modules work together to provide the comprehensive functionality as a fair NLUBroker.

2.1 Benchmarking

As the fundamental part for the brokering module, the benchmarking module is proposed for two purposes: (1) **guide** users by providing efficient evaluation and comparison in terms of functionality, performance, and availability. Users can easily choose the most appropriate service for their specific purposes based on the evaluated information; (2) **initialize** the brokering module by giving the score of the service providers according to the evaluation metrics and user preferences.

User interface allows users to have self-defined descriptors and corpora. Consider a user interested in finding the most appropriate service for a chatbot; several arguments can be specified using the descriptor: workload size, training and testing ratio, target services, target workloads, target metrics, evaluate criterion, and so on. A default training corpus is also provided by the system for users who don't have a specified dataset.

Workload generator is responsible for validating, cleaning, and transforming the input data according to the descriptor. It removes redundant data which only differs by space, punctuation, or letter-case, and converts data to the required format. Then it generates workload files based on the training/testing data ratio, workload size, intent type, slot type, *etc.*.

Service evaluator collects three key parameters for NLU services: intent, slot type, and utterance to generate qualified files to train NLU models on different service providers. It helps the user to train models based on the provided corpus and descriptor on multiple service providers under the given quota. Once models are trained well on the cloud and ready to use, service evaluator sends queries to different models using APIs provided by services. To evaluate NLU services, the service evaluator uses target metrics defined by users in the descriptor. It also provides several metrics by default, ranging from general system-related metrics such as availability to NLU-specific metrics like prediction accuracy.

With the three aforementioned components, the benchmarking module is extensible to test a variety of cloud NLU services with different corpora. By allowing users to have self-defined input, the benchmarking module helps users not only on making research comparisons, but also having a better understanding of the difference among NLU services.

2.2 Brokering

Let's assume a scenario where an application developer designs a personal assistant application that uses the cloud-based NLU service to reduce development costs. The service type includes queries for a variety of purposes, such as finding nearby restaurants, inquiring about the local weather, advice on travel routes, and more. In order to read the end user's input more accurately, the brokering module can take advantage of the benchmarking results and always push the query to the top-ranked service provider. Figure 1 depicts this process. A lightweight multi-classification method is deployed in the brokering module to obtain the intent of each incoming query, since the service performance is affected by the intent type as validated in Section 3. The ranking formula of the available service provider is then generated based on the results contributed by the benchmarking module (blue arrow in Figure 1), as shown in Equation (1). We use quality of experience (QoE) to indicate the service ranking. The higher the QoE, the better the service. In the static approach (left side in Figure 1 **Brokering**), QoE is the only indicator for brokering. Each element in the vector QoE ranges from 0 to 1 to represent performance of available providers on different intents. Set X' stands for normalized metrics X. The normalization of n

Algorithm 1 Algorithm in experience-driven agent

Input: State space S, action space A, discount rate θ, learning rate μ

1: Initialize $Q(s,a)$, T^{t_0};
2: **while** q **do**
3: **for** query q_i with intent i coming at time t **do**
4: interact with the environment to get S_i;
5: according to policy $\pi^t(s,a)$, take action A_i^t: choose provider p_j;
6: broker q_i to p_j, get reward $R(s,a)$: qoe^t_{ji}, new state S^t_j and waiting time T^t_{ji};
7: $Q(s,a) \leftarrow (1-\mu) * Q(s,a) + \mu * (R(s,a) + \theta * Q(s,a) * \log(T^t_{ji}/T^{last}_j)$;
8: $T^{last}_j \leftarrow T^t_{ji}$
9: **end for**
10: **end while**

different metrics varies depending on their impact on overall performance. Users are free to assign values for set α with a sum of one to decide the weights of corresponding metrics.

$$QoE = \sum_{k=1}^{n} \alpha_k * X'_k, \quad where \quad \sum_{k=1}^{n} \alpha_k = 1 \tag{1}$$

$$QoE = \alpha * \log \frac{10 * T_{min}}{T} + \beta * \frac{F}{F_{max}} + \gamma * \frac{M_{min}}{M}, \quad where \quad \alpha + \beta + \gamma = 1 \tag{2}$$

Specifically, in this paper, we consider the response time T, prediction performance F, as well as the monetary cost M to rank the service by Equation (2). The logarithm of time takes the notion that the marginal improvement of perceived quality is reduced at a faster response rate for the end user. The accuracy and cost are simply chosen to be normalized by the extreme value. The constant 10 is multiplied to avoid a negative value. The weights α, β, and γ depend on users' assessment of the priority of the corresponding metric. Note that the brokering module also has the ability to optimize by accuracy, responsiveness, and cost, by simply setting the corresponding parameter to 1 and the others to 0. We consider QoE in the rest of the paper since it is a comprehensive indicator that combines all of these factors.

2.3 Experience-driven agent

Equation (1) embodies the basic principle of the brokering module, which is to always choose the best service provider for queries given the predicted intent in a static way. While in reality, a static distribution is not always applicable due to quotas imposed by the service. For example, for a web application embedded with an NLU service, when the number of end user requests under one intent bursts, the static way will reach the limitation of the best service. Considering the joint QoE of clients, we resort to reinforcement learning (RL) [25] to improve the brokering module by balancing load among the services. RL refers to an *agent* learning continuously from the interaction with the *environment* to achieve a specific goal: the maximum expected QoE. As shown in Algorithm 1, for each incoming query q_i at a specific state s (intent) on time t, the agent follows a policy $\pi^t(s)$ to take action. A policy defines the probability distribution over actions. $\pi^t(s,a)$ means the probability that action a is taken in state s at time t. When there is peak traffic access, the agent will be aware of the possible service congestion and distribute to the vacant service providers promptly. Therefore, we have considered the latency factor in Q (Algorithm 1 line 7) for the policy. qoe^t_{ji} is the reward for query q_i at time t and is calculated after the action A^t_i is taken using Equation 1, demonstrating the end user experience. θ is introduced as the discount rate to avoid a possible infinity problem in the reward, since the agent is updating dynamically according to the current network situation, server state, and user preferences.

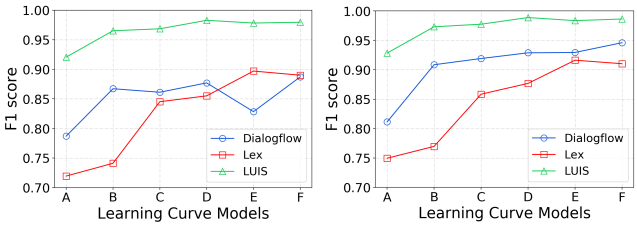
Compared with accessing a single cloud NLU service provider, the brokering module makes it possible to provide a real-time choice and combine the advantages of different cloud services to improve the end user experience, which is not possible using a single service provider. Moreover, users can pre-define the weights for performance metrics based on their preferences so that the brokering module can provide reasonable optimization for them. In addition, by interacting with the environment promptly, the agent can effectively distribute the high-concurrency query data to multiple service providers to ensure the timeliness of the query results.

3 Implementation

To build a fair NLUBroker, we trained the models from scratch to achieve an impartial comparison because built-in slot types for each service are different in content, size, and purpose. In addition, we carefully read the materials about instruction, access and usage limitation, *etc.* to ensure the evaluation and implementation are conducted legitimately. Considering the cost and time limitation, we only choose three of the most representative and widely used NLU service providers to evaluate: *Google Dialogflow*, *Amazon Lex*, and *Microsoft LUIS*. NLUBroker is extensible for other NLU services.

NLUBroker is deployed on an Intel Fog Reference node, which has one Xeon E3-1275 v5 processor equipped with 32GB DDR4 memory and 256GB PCIe SSD. Network speed is ensured to be consistent and stable during the experiments.

For benchmarking purposes we use a labeled public NLU dataset **SNIPS** collected by the company *snips.ai* and labeled with intent and slot type [5]. It includes nearly 14K utterances with seven common used intents in human communications: *AddToPlaylist, BookRestaurant, GetWeather, PlayMusic, RateBook, SearchCreativeWork, SearchScreeningEvent*. For each intent, there are a group of slot types related to it.



(a) All intents (b) Without *SearchCreativeWork*

Figure 2: Learning curve results.

3.1 Provider Performance

This part introduces results from the benchmarking module evaluating three state-of-the-art NLU services from both algorithmic and system aspects.

Learning Curve. The learning curve is one important indicator for evaluating the quality of the service, especially for users who hope to get an accurate model with limited training data. We randomly select from a corpus to compose the training data set with the sizes of 10, 50, 100, 200, 500, 800, and 1000 for each intent and use *LearningCurve A* to *G* to label the trained model. Lex is limited to *F* because of restrictions. From the benchmark, we randomly select 80% as the training set for each intent, with the remaining 20% in the testing set. To avoid possible bias and to exercise the general capability of models, we test models five times, by randomly choosing testing sets from testing utterances each time. Figure 2 illustrates how the average F1 score of trained models varies with training data size. LUIS maintains stable and accurate performance all the time. Dialogflow's performance on *E* is lower than expected. Lex starts at the lowest score and experiences a steep increase when training data grows from *B* to *C*, where the training data size grows from two to three digits for each intent. The gap between Lex and Dialogflow narrows with more training data, and Lex exceeds Dialogflow on *E*. The unusually low F1 score of *E* for Dialogflow arises because *SearchCreativeWork* has an extremely low F1 score, since Dialogflow is confused by *SearchCreativeWork* and *SearchScreenEvent*. After removing this intent, both Dialogflow and Lex obtain stable improvement as shown in Figure 2(b). In general, good performance requires more than 100 utterances to train for each intent. When training sets are small, LUIS performs best.

Intent detection. Figure 3 shows the average F1 score for each intent. Dialogflow predicts most of the intents accurately, leaving *SearchCreativeWork* and *SearchScreeningEvent* intents, whose F1 scores are less than 0.8. Lex shows an obvious deficiency in predicting *BookRestaurant*. LUIS has stable performance and high accuracy (more than 0.9) among all intents. On the whole, *AddToPlaylist, GetWeather, PlayMusic, RateBook* are so-called "good intent" achieving high F1 scores across different services, while the remaining three intents get various results. For users, it is essential to select the appropriate service depending on the target intent.

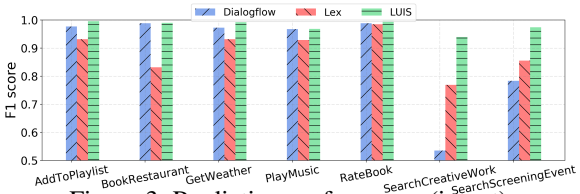


Figure 3: Prediction performance (intent).

Slot filling. Similarly to intent detection, slot filling performance varies among different slots. We define "bad" slot types as those whose F1 score is less than 0.6, and we find it is hard for all three services to correctly detect phrases in *PlayMusic* with label *album*, *track*, and *genre*. One reason for this is the limited amount of training data of these slot types, which is only one-fourth to one-third of the training data for other slot types. Thus, the model is not strong enough to understand the context structure for these slot types. The other "bad" slot types are related to names, such as a street name (*poi*), a food name (*cuisine*), or a restaurant name (*restaurant_name*). Interestingly, each service varies in failed slot types and numbers, reflecting the different drawbacks in back-end models.

Responsiveness. Response time is evaluated with the test set in *F* and measured by the operation time for each query. The results reported in Table 2 show the average response time and the time spent completing 99% of queries. Though on average it takes less than 0.7 seconds for all three services to process a query, there is a big difference in end-to-end latency among the three services. On average, operation time consumed by Lex is three times that of Dialogflow, and 1.7 times that of LUIS.

Service		Dialogflow	Lex	LUIS
Intent (F1)		88.73	89.01	**97.95**
Slot (F1)		83.56	**87.78**	85.38
Response (s)	Average	**0.206**	0.604	0.357
	99%	**0.388**	0.835	0.512
Availability (%)		99.970	99.991	**99.998**
Cost/1,000 requests ($)		2.00	**0.75**	3.00
Limit requests/minute		600	N/A	**3,000**

Table 2: Evaluation summary.

Availability. In order to measure availability, we queried NLU services over 30 consecutive days and measured the failure rate. In each minute of the availability test, the framework sends a query to the cloud services and records any error message received. All three services show good performance in the availability tests, and failures rarely occur (Table 2). A higher number of failures occurred in Dialogflow because the gateway was out of service for several consecutive minutes.

[Summary and insights] Table 2 summaries the evaluation results of three state-of-the-art NLU service providers. Cost and request limitations are provided by documentation for each service provider, and we compare only the enterprise version. The results from the evaluation module reveal that, in general, the three services all performed well: average F1 score around 0.9, average latency less than 1s, and average availability higher than 99.9%. Users thus need to measure

the indicators to find the service that best meets their requirements. The three state-of-the-art NLU services have trade-offs between accuracy and other performance metrics. There is no absolute winner achieving the shortest response time and highest accuracy at the same time. And the prediction performance is affected by the training data size and intent type. Thus, if the user obtains a dataset with larger data, or in a different distribution, it is worthwhile to rerun the benchmarking to generate a new model.

3.2 NLUBroker Performance

The lightweight multiclassification algorithm for NLUBroker is logistic regression provided by Scikit-learn, with the semantic hash as the embedding method [22]. This combination is chosen given the small overhead (0.1ms/request) with good accuracy (91.0%). Both the word embedding model and trained logistic regression model are saved and launched once with NLUBroker.

QoE optimization. We evaluate the NLUBroker performance using QoE as shown in Table 3 for both the static and agent-based method. The static method uses Equation 2 to choose the best service provider. The agent-based method follows the Algorithm 1 proposed in Section 3. We set both the learning rate μ and discount factor θ to 0.5 in the experiment, which means that the agent treats historical performance and current performance equally and learns from both at the same speed. As an example to show the capability of NLUBroker to optimize QoE, we choose the weights as 0.8, 0.1, 0.1 for accuracy, response time, and cost, considering the trade off [13]. The NLUBroker performance is shown in Table 3. Though it has the trade-off among parameters, compared with accessing a single cloud NLU service provider, NLUBroker makes it possible to combine the advantages of different cloud services to obtain a more balanced performance with higher QoE in both static and agent-based approaches. Considering the simplicity of deployment, the static approach is the best choice when queries coming in slow speed and small volume.

Throughput. To test the scalability of NLUBroker, we increase the number of threads to get the throughput of the system and compare with systems without NLUBroker. When getting "too many requests" messages from the server, NLUBroker is able to distribute requests to the suboptimal service. Since Lex is not accepting multiple requests at the same time, we compare NLUBroker with the other two. The throughput results (Figure 4) show that the performance of Dialogflow and LUIS is consistent with the request limitation claimed in the document, while NLUBroker has the capability to balance workload by distributing blocked requests to a second choice. The static approach will retry three times before brokering the blocked request to the second choice service. The agent-based approach has the highest throughput when the thread number increases, since it constantly monitors the response time of previous responses in the same action and quickly switches

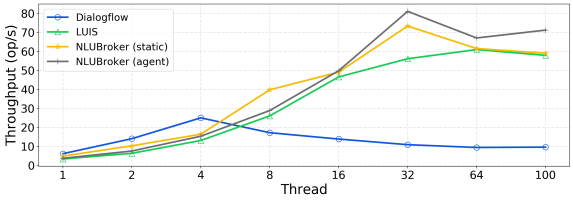


Figure 4: Throughput performance.

	Dialogflow	Lex	LUIS	NLUBroker (static)	NLUBroker (agent)
F1 (%)	88.73	89.01	**97.95**	97.23	97.38
Response (s)	**0.206**	0.604	0.357	0.25	0.32
Cost	2.00	**0.75**	3.00	2.29	2.68
QoE (%)	84.73	87.92	88.31	**90.17**	**90.28**

Table 3: QoE optimization for NLUBroker.

to other available services without retrying the operation.

4 Related Work

Benchmarking. Different types of benchmarks are available for different computing applications and services [4, 7, 15, 27, 28]. However, the required workload and evaluation indicators in NLU services differ from the existing benchmarks, making it still at an early stage. Resnik *et al.* evaluated several NLP services with different case studies [21]. [3] examined the prediction performance of several NLU services on two different corpora. Dialogue systems are evaluated with similar principles [18, 26]. These works focus at the algorithmic level, while our benchmarking module evaluates NLU services from both an algorithmic and system perspective.

Brokering. Located between cloud customers and providers, brokers are helping customers to select the most suitable cloud service [2, 11, 12, 19, 24]. For example, [11] proposes a split algorithm for request splitting and provisioning across multiple cloud platforms. *ARank* ranks the candidate services based on service quality. However, there is no suitable broker designed specifically for NLU services.

5 Conclusion

In this work, NLUBroker, an extensible NLU service broker, is proposed to help users choose a suitable service provider based on preferences and enable the cross-cloud resource acquisition for QoE of applications. NLUBroker evaluates the performance of several state-of-the-art NLU service providers and shows that there is no absolute winner in different query scenarios in terms of accuracy, responsiveness, and cost. With these insights, NLUBroker automatically distributes NLU requests based on user preferences to make full use of the advantage of each service provider to achieve the overall optimality. A reinforcement learning-based agent is leveraged to deal with high concurrency requests in a dynamic environment, while ensuring accuracy and responsiveness.

6 Discussion

In this paper, we propose NLUBroker, a platform to comprehensively measure the performance indicators of candidate NLU services, and further provide a broker to select the most suitable service according to the different needs of users. And we list the discussion as follows:

[Feedback] One of the core parts is about how to design the brokering module, so we are looking for suggestions on the methodology of NLUBroker, especially for the more efficient policy for the reinforcement learning-based agent.

[Controversial points] The privacy issue is always a problem between the cloud customer and service provider, and this issue also exists in NLUBroker. Before the data is uploaded to the cloud, it will go through the broker, which may increase the potential risk. In our design, NLUBroker only helps requests to select the optimal service, while never saving any data in any circumstances.

[Discussion types] This paper is likely to generate discussions on the different performance of cloud-based NLU services, and the reinforcement learning-based agent. Specifically, it should lead to discussions on how the brokering module contributes to the quality of experience for users.

[Open issue] This paper is focused on the functionality of NLUBroker. We do not address the types of natural language understanding algorithms used by the cloud service providers.

[Under what circumstances might the whole idea fall apart] The ideas in the paper will continue to be valid as long as Web services for natural language understanding services continue to be used.,

References

[1] AWS. Amazon Lex – Build Conversation Bots, 2019. https://aws.amazon.com/lex.

[2] Ramsha Baig, Waqas A Khan, Irfan Ul Haq, and Irfan Muhammad Khan. Agent-based sla negotiation protocol for cloud computing. In *2017 International Conference on Cloud Computing Research and Innovation (ICCCRI)*, pages 33–37. IEEE, 2017.

[3] Daniel Braun, Adrian Hernandez-Mendez, Florian Matthes, and Manfred Langen. Evaluating natural language understanding services for conversational question answering systems. In *Proceedings of the 18th Annual SIGdial Meeting on Discourse and Dialogue*, pages 174–185, 2017.

[4] Brian F Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan, and Russell Sears. Benchmarking cloud serving systems with ycsb. In *Proceedings of the 1st ACM symposium on Cloud computing*, pages 143–154. ACM, 2010.

[5] Alice Coucke, Alaa Saade, Adrien Ball, Théodore Bluche, Alexandre Caulier, David Leroy, Clément Doumouro, Thibault Gisselbrecht, Francesco Caltagirone, Thibaut Lavril, et al. Snips voice platform: an embedded spoken language understanding system for private-by-design voice interfaces. *arXiv preprint arXiv:1805.10190*, pages 12–16, 2018.

[6] Facebook. Wit.ai, 2019. https://wit.ai/.

[7] Yu Gan, Yanqi Zhang, Dailun Cheng, Ankitha Shetty, Priyal Rathi, Nayan Katarki, Ariana Bruno, Justin Hu, Brian Ritchken, Brendon Jackson, et al. An open-source benchmark suite for microservices and their hardware-software implications for cloud & edge systems. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 3–18. ACM, 2019.

[8] Chih-Wen Goo, Guang Gao, Yun-Kai Hsu, Chih-Li Huo, Tsung-Chieh Chen, Keng-Wei Hsu, and Yun-Nung Chen. 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)*, volume 2, pages 753–757, 2018.

[9] Google. Dialogflow, 2018. https://dialogflow.com/.

[10] Jan-Gerrit Harms, Pavel Kucherbaev, Alessandro Bozzon, and Geert-Jan Houben. Approaches for dialog

management in conversational agents. *IEEE Internet Computing*, 2018.

[11] Ines Houidi, Marouen Mechtri, Wajdi Louati, and Djamal Zeghlache. Cloud service delivery across multiple cloud platforms. In *2011 IEEE International Conference on Services Computing*, pages 741–742. IEEE, 2011.

[12] Arezoo Jahani, Farnaz Derakhshan, and Leyli Mohammad Khanli. Arank: A multi-agent based approach for ranking of cloud computing services. *Scalable Computing: Practice and Experience*, 18(2):105–116, 2017.

[13] Jiarong Jiang, Adam Teichert, Jason Eisner, and Hal Daume. Learned prioritization for trading off accuracy and speed. In *Advances in Neural Information Processing Systems*, pages 1331–1339, 2012.

[14] Yoon Kim. Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*, 2014.

[15] Ang Li, Xiaowei Yang, Srikanth Kandula, and Ming Zhang. Cloudcmp: comparing public cloud providers. In *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*, pages 1–14. ACM, 2010.

[16] Grégoire Mesnil, Yann Dauphin, Kaisheng Yao, Yoshua Bengio, Li Deng, Dilek Hakkani-Tur, Xiaodong He, Larry Heck, Gokhan Tur, Dong Yu, et al. Using recurrent neural networks for slot filling in spoken language understanding. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 23(3):530–539, 2015.

[17] Microsoft. Language Understanding (LUIS), 2018. https://www.luis.ai/home.

[18] Fabrizio Morbini, Kartik Audhkhasi, Kenji Sagae, Ron Artstein, Dogan Can, Panayiotis Georgiou, Shri Narayanan, Anton Leuski, and David Traum. Which asr should i choose for my dialogue system? In *Proceedings of the SIGDIAL 2013 Conference*, pages 394–403, 2013.

[19] Przemyslaw Pawluk, Bradley Simmons, Michael Smit, Marin Litoiu, and Serge Mankovski. Introducing stratos: A cloud broker service. In *2012 IEEE fifth international conference on cloud computing*, pages 891–898. IEEE, 2012.

[20] Christian Raymond and Giuseppe Riccardi. Generative and discriminative algorithms for spoken language understanding. In *Eighth Annual Conference of the International Speech Communication Association*, 2007.

[21] Philip Resnik and Jimmy Lin. Evaluation of nlp systems. *The handbook of computational linguistics and natural language processing*, 57:271–295, 2010.

[22] Kumar Shridhar, Amit Sahu, Ayushman Dash, Pedro Alonso, Gustav Pihlgren, Vinay Pondeknath, Fotini Simistira, and Marcus Liwicki. Subword semantic hashing for intent classification on small datasets. *arXiv preprint arXiv:1810.07150*, 2018.

[23] Aditya Siddhant, Anuj Goyal, and Angeliki Metallinou. Unsupervised transfer learning for spoken language understanding in intelligent agents. *arXiv preprint arXiv:1811.05370*, 2018.

[24] Smitha Sundareswaran, Anna Squicciarini, and Dan Lin. A brokerage-based approach for cloud service selection. In *2012 IEEE Fifth International Conference on Cloud Computing*, pages 558–565. IEEE, 2012.

[25] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[26] Johannes Twiefel, Timo Baumann, Stefan Heinrich, and Stefan Wermter. Improving domain-independent cloud-based speech recognition with domain-dependent phonetic post-processing. In *AAAI*, pages 1529–1536, 2014.

[27] Lei Wang, Jianfeng Zhan, Chunjie Luo, Yuqing Zhu, Qiang Yang, Yongqiang He, Wanling Gao, Zhen Jia, Yingjie Shi, Shujie Zhang, et al. Bigdatabench: A big data benchmark suite from internet services. In *High Performance Computer Architecture (HPCA), 2014 IEEE 20th International Symposium on*, pages 488–499. IEEE, 2014.

[28] Yifan Wang, Shaoshan Liu, Xiaopei Wu, and Weisong Shi. CAVBench: A benchmark suite for connected and autonomous vehicles. In *Proceedings of the Third IEEE/ACM Symposium on Edge Computing*, pages 30–42. IEEE, 2018.