Building a Natural Language Processing Model to Extract Order Information from Customer Orders for Interpretative Order Management

Mingyan Simon Lin, Clara Ga Yi Tang, Xing Jing Kom, Jia Yi Eyu, Chi Xu
Singapore Institute of Manufacturing Technology, Singapore
{simon_lin,clara_tang,kom_xing_jing,eyu_jia_yi,cxu}@simtech.a-star.edu.sg

2022 IEEE International Conference on Industrial Engineering and Engineering Management (IEEM) | 978-1-6654-8687-3/22/$31.00 ©2022 IEEE | DOI: 10.1109/IEEM55944.2022.9989801

Abstract – **Due to the increased complexity of supply chains and the various challenges that these supply chains are facing, it is important for supply chains to automate and optimize their supply chain management processes to respond to these challenges and maintain their competitive advantages. Order management plays an integral role in supply chain management, and one of the ways where the order management process can be streamlined is to adopt a no-touch approach. In this paper, we describe a natural language processing (NLP)-based engine prototype to extract and interpret order information from customer natural language orders, which will facilitate no-touch order processing. This engine prototype can then be integrated into an overall no-touch order management engine that can be used to demonstrate a reliable Advanced Available-to-Promise (AATP) process at the critical sites in a supply chain testbed.**

Keywords – **No-touch order processing, interpretative order management, natural language processing, slot filling**

I. INTRODUCTION

The current supply chains face a number of challenges, which include greater uncertainty in global trade flows, black swan events, as well as greater demand from customer for highly personalized and customized products. In order for supply chains to be more resilient to these challenges and to maintain their competitive advantages, it is important for supply chains to automate and optimize their supply chain management processes. Order management and planning systems play an integral role in supply chain management, and no-touch order processing is one of the ways where the order management process can be automated and streamlined.

To facilitate the adoption of no-touch order processing, one needs to develop a natural language processing (NLP) model to translate natural language queries into machine-formatted orders that can be interpreted easily by the order management systems. Such a task can be modelled as a slot filling task, or more generally, a named entity recognition (NER) task, where one extracts key order information from the given natural language query, either from raw text orders, or text transcripts of voice orders. A no-touch order processing system with NLP-supported interpretive order capturing and processing is useful in situations where touch screen or keyboard typing is inaccessible and there is a cost to delaying order execution. Some of these situations include customer facing or self-service scenarios for products with high volume turnover, such as customer self-service ordering in the food and beverage (F&B) in or production line scenario where operators and technicians are not hands-free, and purchase or order decisions remain relatively straightforward, such as spare parts ordering from the shopfloor within a vertical supply chain.

While there are commercial NLP solutions for NER [1-5], they work mostly on general entities, and thus require heavy customization in order to adapt their solutions to a slot filling or NER task in a supply chain context. Moreover, there may be limitations in integrating these commercial solutions into the supply chain. On the other hand, there are several works in literature that focuses on the slot filling/NER task or order information extraction in an E-commerce context, but they may not focus on extracting information that may be important, such as quantity and product attributes [6,7], and they may not be directly amenable in an order management context, due to the different nature of the queries considered in these works, where they are primarily considered in online shopping [8] assistants or in search engines [9].

In this paper, we describe an NLP model to extract order information from customer text orders, by modelling the problem as a slot filling task, and build an engine prototype based on this NLP model to interpret customers' text or voice orders. This engine prototype is part of a proof-of-concept (POC) for an overall no-touch order management engine that can be used to demonstrate a reliable AATP process at the critical sites in a supply chain testbed, which in turn can then be applied in a number of potential use-cases, such as self-service ordering in F&B, and spare parts ordering from the shopfloor within a vertical supply chain.

The rest of the paper is then organized as follows: in Section II, we will describe the overall flow of our order interpretation engine. In Section III, we will describe our NLP model for extracting order information in detail, where we describe the problem formulation and the architecture behind our NLP model. In Section IV, we describe the experimental set-up used to test our NLP model, using both open-source datasets and internal datasets, and present and discuss the results from the experiments. Finally, we will conclude the paper with a few future directions for our work.

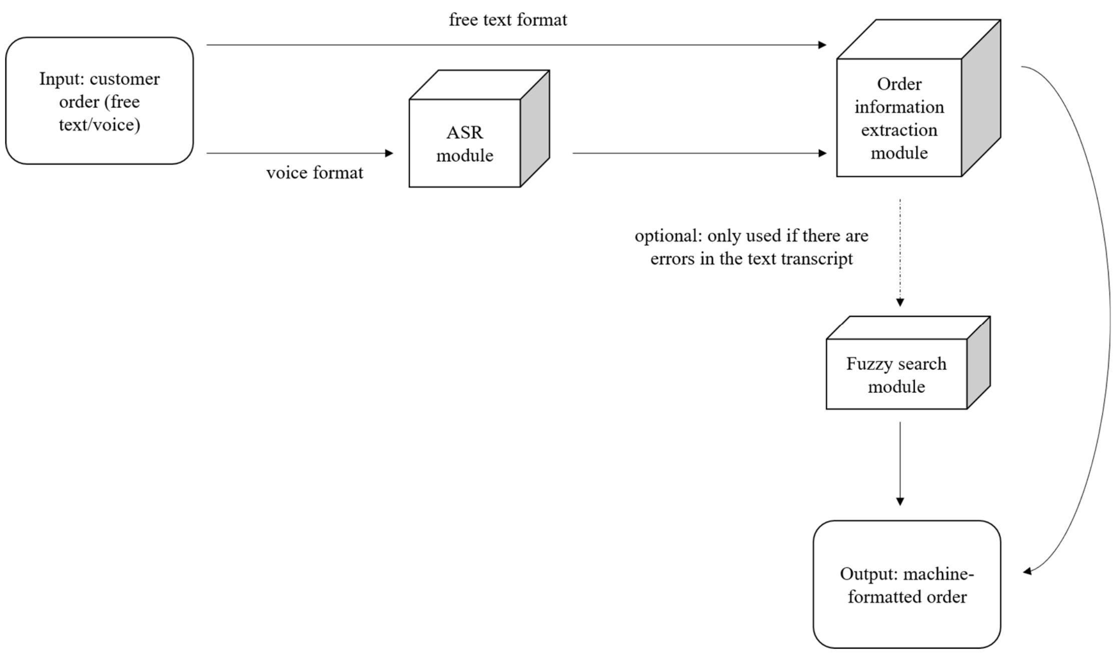
free text format
Input: customer order (free text/voice)
ASR module
voice format
Order information extraction module
optional: only used if there are errors in the text transcript
Fuzzy search module
Output: machine-formatted order

Fig. 1: Flow diagram for the order interpretation engine

II. OVERVIEW OF THE ORDER INTERPRETATION ENGINE

The overall flow diagram for the order interpretation engine is shown in Fig.1 above. Let $\{C_1, \ldots, C_k\}$ be the set of order information attributes that are required to be captured in a given customer order. The engine takes customer order in a free text/voice format as an input and yields a machine-formatted order containing the needed information for each of the attributes $C_1, \ldots, C_k$. An example is given below, where the customer seeks to order a USB cartridge with a holder, where the customer can choose 2 scents from 5 possible choices, for the cartridge, namely, lavender, citronella, eucalyptus, tea tree, and peppermint, and a color from 3 possible choices for the holder color, namely, green, white, and blue. In this case, the attributes are “scents” and “color”, and the order interpretation engine is to extract information about the attributes “scents” and “color” from the input as follows:

Input: “i want to order a cartridge scents should be citronella and tea tree holder color should be blue”

Output: {“scents”: [“citronella”, “tea tree”], “color”: “blue”}

The specifications for the modules are given as follows:

- *Automatic speech recognition (ASR) module (for voice orders)*

Given a voice order, the ASR module will automate the process of converting the voice order into free text using an ASR model, which is then used as the input for the order information extraction module.

- *Order information extraction module*

Given a free text customer order, the order information extract module will order the relevant information from the order (such as the cartridge scents and the holder color in the above example).

- *Fuzzy search module (optional)*

After the order information has been obtained, the fuzzy search module will seek to correct any errors present in the extracted information, by comparing the extracted information for each attribute C_i against a database containing all possible values for C_i and choosing the value(s) in the database that is/are most similar to the extracted information for attribute C_i based on a phonetic-based similarity metric. This module is primarily used in the case where there is a voice customer order, as there may be transcription errors arising from the ASR module, although it can also be used when there are typographical errors in a free text customer order.

III. THE ORDER INFORMATION EXTRACTION MODULE

In this section, we will describe our approach behind the order information extraction module. Formally, the order information extraction task can be modelled as a slot filling task, where given a word sequence $\mathbf{w} = (w_1, w_2, \ldots, w_n)$ that represents the natural language query, we assign a sequence of slot labels $\mathbf{s} = (s_1, s_2, \ldots, s_n)$ to $\mathbf{w}$, where each

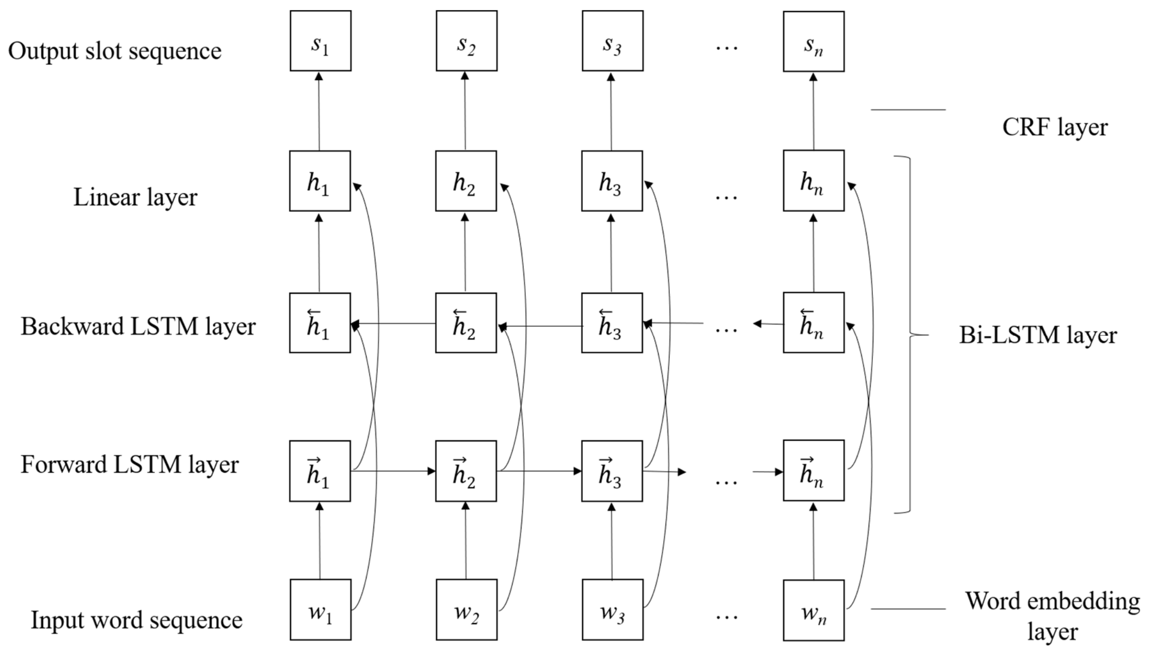

Fig. 2: Model architecture for the order information extraction module

slot label corresponds to each word token and is given a semantic label from {'O', l_1, …, l_p}. Here, the slot label is equal to some l_i if it carries a semantic meaning, and a label "O" otherwise. In an order information extraction context, we may take the subset {l_1, …, l_p} of slot labels to be equal to the set {C_1, …, C_k} of order information attributes of interest defined in Section II. In the below example from the MultiDoGo fast-food training dataset [10], *red wine* refers to a food item, while the number *1* refers to the quantity of the food item. Here, the slot labels in this example are presented in the in/out/beginning annotation format instead of the plain format used in the original corpus.

word	i	need	1	red	wine
slot label	O	O	quantity	B-food_item	I-food_item

The goal of the slot filling task is then to find the sequence **s** of slot labels for which the conditional probability score P(**s**|**w**) is maximal. The task not only involves learning the distribution of all possible slot sequences **s** based on a given word sequence **w**, but also the possible co-occurrences between the slot labels that can occur on a given word sequence.

The slot filling task is typically treated as a sequence labelling problem, and contemporary approaches to solve the sequence labelling problem involve recurrent neural networks [11-13]. Here, we describe one such approach, which we will use to build our order information extraction module. The model architecture, as shown in Fig. 2 above, is based on a recurrent neural network architecture. It consists of a word embedding layer, an encoding layer, and a decoding layer. The individual layers of the model architecture are further described as follows:

- *Word embedding layer*

The rationale of the word embedding layer is to generate a sequence of word embeddings **x** = (x_1, x_2, …, x_n) for the encoding layer of the model from a given input word sequence **w** = (w_1, w_2, …, w_n), which will serve as input features for the encoder layer, which we will describe shortly below the fold. While the word embeddings can be randomly initialized and fine-tuning during training, the performance of the slot filling task increases when word embeddings from pre-trained language models are used as features for the word embedding layer.

- *Encoder layer*

For the encoder layer, we use a bi-directional long-short term memory (Bi-LSTM) layer, along with a linear layer, with the word embeddings **x** = (x_1, x_2, …, x_n) serving as our input features. The rationale behind using bi-directional layers is to capture both left (past) and right (future) contexts, while the LSTM architecture is able to capture long-range dependencies in a given sequence of word embeddings **x** = (x_1, x_2, …, x_n).

A single LSTM memory unit consists of an input gate **i** = (i_t), output gate **o** = (o_t), forget gate **f** = (f_t) and a memory cell **c** = (c_t), and produces a hidden vector **h** = (h_t) depending on the word sequence **w** based on the following update mechanism:

$$f_t = \sigma\left(W_{fw}x_t + W_{fh}h_{t-1} + \boldsymbol{b}_f\right), \qquad (1)$$

$$i_t = \sigma(W_{iw}x_t + W_{ih}h_{t-1} + \boldsymbol{b}_i), \quad (2)$$
$$c_t = c_{t-1} \odot f_t + i_t \odot tanh(W_{cw}x_t + W_{ch}h_{t-1} + \boldsymbol{b}_c), \quad (3)$$
$$o_t = \sigma(W_{ow}x_t + W_{oh}h_{t-1} + \boldsymbol{b}_o), \quad (4)$$
$$h_t = o_t \odot tanh(c_t). \quad (5)$$

Here, σ refers to the sigmoid function, W_{nm} is an $m\times n$ weight matrix, $\mathbf{b}_m$ is an $m\times 1$ bias vector, and $\odot$ is the Hadamard product operator.

The Bi-LSTM layer consists of two LSTM layers, one which reads the input word sequence **w** from left to right (forward), and the other right to left (backward). The output of the Bi-LSTM layer is obtained by concatenating the forward hidden states $\vec{h} = (\vec{h}_t)$ and backward hidden states $\overleftarrow{\mathbf{h}} = (\overleftarrow{h}_t)$ to form a single vector $\mathbf{h} = (h_t)$ of hidden states. These hidden states are then used to form a hidden linear layer of size *k*, where *k* is equal to the number of attributes, is placed on top of the Bi-LSTM layer, and the vector **h** of hidden states is projected onto this hidden linear layer. This projection yields an $n\times k$ score matrix *P*, which is used for the subsequent decoder layer.

- *Decoder layer*

For the decoder layer, we use a conditional random field (CRF) layer. The CRF is a type of discriminative undirected probabilistic graphical model, where sentences or texts are modelled as sequences of tokens, which in turn can then be modelled as a graph. They are commonly used in slot filling tasks, as they are able to account for dependencies that can occur between output slot labels.

Next, we describe the formulation of the CRF layer. Given a word sequence $\mathbf{w} = (w_1, w_2, \ldots, w_n)$, a sequence of slot labels $\mathbf{s} = (s_1, s_2, \ldots, s_n)$, the $n\times k$ score matrix *P* derived from the encoding layer, and the transition matrix *T* of scores with entries $t_{i,j}$ the transition score from s_i to s_j, the conditional probability score P(**s**|**w**) is defined by

$$S(\mathbf{w}, \mathbf{s}) = \sum_{i=1}^{n} \left(T_{i-1,i} + P_{i,s_i}\right), \quad (6)$$
$$P(\mathbf{s}|\mathbf{w}) = \frac{exp\, S\,(\mathbf{w}, \mathbf{s})}{\sum_{\mathbf{s}'} exp\, S\,(\mathbf{w}, \mathbf{s}')}. \quad (7)$$

The loss function is then given by the negative log likelihood of the conditional probability score P(**s**|**w**), and the sequence **s** for which P(**s**|**w**) is maximal is precisely the sequence **s** for which *exp* S(**w**,**s**) is maximal. Such a sequence **s** can be found efficiently using dynamic programming [14].

IV. EXPERIMENTAL SET-UP AND RESULTS

In this section, we evaluate the effectiveness of our order information extraction module, both on open-source datasets and internal datasets.

Our first set of experiments was conducted on the fast-food train MultiDoGo dataset [14]. We took the samples that were annotated at the sentence level, and from the samples from each of the training, validation, and test datasets, we only took the samples that contain any of the slot labels “food_item”, “drink_item”, “quantity” and “size”. This yielded 4336 training samples, 576 validation samples and 1211 test samples. Next, we combined the slot labels “food_item” and “drink_item” into a single slot label “order-name” and re-labelled these slot labels in the in/out/beginning annotation format.

For our model, we tested three different word embeddings using the *en_core_web_sm* spaCy language model [15], the *all-MiniLM-L6-v2* Sentence-BERT language model [16], and the *bert-base-uncased* BERT language model [17]. The BERT word embeddings were generated by averaging the hidden outputs from the last four layers of the *bert-base-uncased* model. We set the size of both the forward and backward hidden states to 5. We used a stochastic gradient descent optimizer with learning rate 0.001 and weight decay 0.01.

Our evaluation metrics are based on [18], where additional types of errors (assignment of wrong slot labels, incorrect boundaries) were considered, as the usual evaluation metrics fail to consider the different types of error that can occur. While there are different evaluation schemas defined based on the type of errors and scenarios in which the errors occur, from which the precision, recall, and f1 scores will be computed differently, we will only consider the strict evaluation schema from [18] for brevity’s sake, where both the type of slot labels and the boundaries of the slot labels must match. Based on this evaluation schema, we set the patience value of 5 for early stopping if the validation f1 score fails to increase. Table 1 summarizes the f1 scores obtained on the MultiDoGo fast-food dataset.

TABLE 1: EXPERIMENT F1 SCORES (IN %) ON THE MULTIDOGO FAST-FOOD DATASET

Word embedding\slot label	order-name	quantity	size
spaCy	67.72	88.53	85.11
Sentence-BERT	67.01	87.56	84.40
BERT	74.61	88.08	78.47

In general, the differences between the performances of the three models on the “quantity” slot label are minimal, while the BERT based order information extraction model outperformed the other models on the “order-name” slot label. One possible reason could be the differences between the type and amount of text corpora and the amount on which these language models were trained, which allowed the BERT based model to recognize a wide range of food and drink items. On the other hand, the BERT based model underperformed the other models on the “size” slot label.

To evaluate the effectiveness of our order information extraction model on text transcripts of voice orders and to ensure that the model can be integrated into an overall order interpretation engine, our second set of experiments was conducted on an internal dataset, where the samples involve voice customer utterances involving the customer ordering a USB cartridge with a holder as described in Section II. Owing to the shortage of audio training and validation customer order samples, we generated 800 training and 200 validation synthetic text orders to train and validate our models. We evaluated our models on the text transcripts of 39 voice customer orders, generated using the NVIDIA NeMo [19] and Vosk open-source ASR models [20], and used a fuzzy search module to correct the errors in the order extraction against a pre-fixed list of scents and colors at the end. The training settings are exactly the same as that of the first set of experiments.

In lieu of the transcription errors, we evaluate the effectiveness of the order interpretation engine by counting the percentage of exact and partial matches between the scents and colors mentioned in the original voice customer order and that obtained by the engine. Here, only exact matches are reported for the slot label "color", as the customer is only allowed to choose a single color for the holder color. For brevity's sake, we only present results obtained by the BERT based model.

TABLE 2: EXPERIMENT SCORES (IN %) ON THE INTERNAL DATASET

ASR model	Scent (exact/partial)	Color (exact only)
NVIDIA NeMo	28.21/53.85	71.39
Vosk	53.85/76.92	61.54

V. CONCLUSION

In this paper, we have described an NLP approach to extract order information from customer orders and evaluated the effectiveness of our approach on both open-source (text) and internal (voice) datasets. However, we have only considered scenarios where the customer makes an order, and we have yet to consider other scenarios where the customers would like to modify or cancel their orders. To cater for such cases, a joint intent classification and slot filling model is needed to handle these scenarios. Such a joint model will also improve the performance of the individual tasks, as the model will be able to learn from the joint information given in the samples. We plan to extend our model in a future work.

ACKNOWLEDGMENT

This work is supported by A*STAR under the Industry Alignment Fund – Pre-Positioning (IAF-PP) programme (M21J6a0080), with the project title "Supply Chain (SC) 4.0 – Digital Supply Chain Development via Platform Technologies".

REFERENCES

[1] Amazon, https://aws.amazon.com/comprehend/

[2] Google Cloud, https://cloud.google.com/natural-language/

[3] Microsoft Azure, https://docs.microsoft.com/bs-latn-ba/azure/cognitive-services/text-analytics/overview

[4] IBM, https://www.ibm.com/sg-en/cloud/watson-natural-language-understanding/details

[5] SAP Community, https://blogs.sap.com/2014/09/08/text-analysis-natural-language-processing-as-a-core-feature-of-sap-hana/

[6] Abinaya K., and S. Roy, "Are you calling for the vaporizer you ordered?" Combining Search and Prediction to Identify Orders in Contact Centers," in *Proceedings of the 4th Workshop on e-Commerce and NLP (ECNLP 4)*, pp. 58–69, Aug. 2021.

[7] P. Gubbala, and X. Zhang, "A Sequence to Sequence Model for Extracting Multiple Product Name Entities from Dialog," *arXiv preprint* arXiv:2110.14843.

[8] Y. Gong *et al.*, "Deep cascade multi-task learning for slot filling in online shopping assistant." in *Proceedings of the AAAI conference on artificial intelligence*, vol. 33, no. 01, pp. 6465–6472. 2019.

[9] S. Manchanda, M. Sharma, and G. Karypis, "Distant-Supervised Slot-Filling for E-Commerce Queries," *in 2021 IEEE International Conference on Big Data (Big Data)*, pp. 677–686. IEEE, 2021.

[10] D. Peskov *et al.*, "Multi-domain goal-oriented dialogues (MultiDoGo): Strategies toward curating and annotating large scale dialogue data" 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)*, pp. 4526–4536. 2019.

[11] Z. Huang, W. Xu, and K. Yu, "Bidirectional LSTM-CRF models for sequence tagging," *arXiv preprint* arXiv:1508.01991, 2015.

[12] X. Ma, and E. Hovy, "End-to-end Sequence Labeling via Bi-directional LSTM-CNNs-CRF," in *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1064–1074. 2016.

[13] Z. Dai, X. Wang, P. Ni, Y. Li, G. Li, and X. Bai, "Named Entity Recognition Using BERT BiLSTM CRF for Chinese Electronic Health Records," in *2019 12th International Congress on Image and Signal Processing, BioMedical Engineering and Informatics (CISP-BMEI)*, pp. 1–5, 2019.

[14] J. D. Lafferty, A. McCallum, and F. C. N. Pereira, "Conditional random fields: Probabilistic models for segmenting and labeling sequence data," in *Proceedings of the 18th International Conference on Machine Learning*, pages 282–289, 2001.

[15] M. Honnibal, and I. Montani, "spaCy 2: Natural Language Understanding with Bloom Embeddings, Convolutional Neural Networks and Incremental Parsing," *GitHub*, 2017.

[16] N. Reimers, and I. Gurevych, "Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks," 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)*, pp. 3982–3992, 2019.

[17] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "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), pp. 4171–4186, 2019.

[18] I. Segura-Bedmar, P. Martínez, and M. Herrero-Zazo, “SemEval-2013 Task 9: Extraction of Drug-Drug Interactions from Biomedical Texts (DDIExtraction 2013),” in *Second Joint Conference on Lexical and Computational Semantics (* SEM), Volume 2: Proceedings of the Seventh International Workshop on Semantic Evaluation (SemEval 2013)*, pp. 341–350. 2013.

[19] O. Kuchaiev *et al.*, “Nemo: a toolkit for building ai applications using neural modules,” *arXiv preprint* arXiv:1909.09577, 2019.

[20] A. Cephei. “Vosk offline speech recognition API,” https://alphacephei.com/vosk/