Named Entity Recognition for E-Commerce Search Queries

Bhushan Ramesh Bhange[*], Xiang Cheng[†], Mitchell Bowden[*], Priyanka Goyal[†], Thomas Packer[†], Faizan Javed[†]

March 8, 2020

Abstract

Named entity recognition (NER) is a critical step in search query understanding. In the domain of e-commerce, identifying the entities of brand and product type can help a search engine retrieve relevant products and thus offer an engaging shopping experience. However, NER in search is a challenging task due to ambiguity, noise, and lack of context. Recent research shows promising results on shared benchmark NER tasks using deep learning methods, but there are still unique challenges in the industry regarding domain knowledge, training data, and model production. This paper proposes an end-to-end framework to address these challenges and deliver a deep learning model in production. This framework firstly prepares three sets of training data to meet different requirements. Then the three datasets are used to iteratively train a bidirectional GRU-CRF model. Lastly, the best model is deployed as a real-time web service for the search engine at homedepot.com. Using this approach, the best model lifts the F1 score from 69.5 to 93.3 on the holdout test data. In both the A/B test and day-to-day use, we see significant improvements in user engagement and revenue.

Keywords: named entity recognition, e-commerce, search engine, neural networks, deep learning

1 Introduction

The search engine at homedepot.com processes billions of search queries and generates tens of billions of dollars in revenue every year for The Home Depot (THD). One of the fundamental challenges in a search engine is to understand a search query and extract entities, which is critical to retrieve the most relevant products. This task can be framed as named entity recognition (NER) which is a common information retrieval task to locate, segment, and categorize a pre-defined set of entities from unstructured text. Since its introduction in the early 1990s, NER has been studied extensively and is evolving rapidly [1, 2], especially after the adoption of deep learning and related techniques in recent years [2].

However, there is a gap between academic research and industrial applications of NER. Recent research works often use the latest neural architectures [2] and language models [3, 4, 5] to improve performance on popular NER shared tasks and datasets [6]. The focused outcome is often marginal improvement of F1 scores, while the feasibility in real-world applications is not often considered.

1.1 Recent Research in NER In recent years, deep-learning-based NER together with embeddings has become increasingly popular in the research community. Collobert et al. proposed the first neural network architecture for NER [7] and later experimented pre-trained word embeddings [8] as model features. Since then various neural architectures and word representations have been studied [2]. The most popular approach is to use recurrent neural networks (RNN) over word, sub-word, and/or character embeddings [2, 9]. Long Short Term Memory (LSTM) and its variants (e.g. Gated Recurrent Unit, i.e. GRU) are the most common neural architectures for NER as well as for sequence tagging [9, 10, 11, 12]. Recently, transformer-based language models [13, 4] have been tested on benchmark NER tasks and claim the state-of-art performance [14, 15].

It is exciting to see the booming research based on deep learning methods and latest language models. However, industrial applications still seem being left behind due to unique challenges as described below.

1.2 Industry Applications and Challenges Applying deep learning to NER in e-commerce is a new and active research area in the industry. Among those most closely related to our work, three papers [16, 17, 18] explore deep learning for NER in e-commerce search, two papers use conditional random fields (CRFs) [19, 20], two papers [16, 18] perform an evaluation on queries in production, and five papers [21, 22, 20, 16, 18] leverage a large number of unlabeled queries available to commercial search engines or perform weakly supervised learning.

Despite the above progress, we believe challenges remain in industrial applications such as noisy customer-

[*]The Home Depot, Irving, TX
[†]The Home Depot, Atlanta, GA

intent signals; the need for a large amount of custom labeled training data; and real-time and memory-limited production environments. In this work, we propose a framework (see Section2) to address these challenges and deliver a model in production.

1.3 NER at The Home Depot THD is a leading home improvement retailer for consumers and professionals. This domain is rich in entity types and relationships, and customer search queries often reflect this richness. To determine the shopping target of the customer query, NER is a vital query understanding step.

The legacy NER system in production used predefined taxonomies of brands and product types, recognized using a sequential greedy search. Beginning with brands and then product types, the longest matching token sequence is labeled as the corresponding entity. Common challenges include queries containing multiple product types or multiple brands but only one shopping target, ambiguity between product types and brands, and new product types not in pre-defined taxonomy. Three examples are listed in Table 1 to demonstrate the challenges. These mislabels are often closely related to specific brands, product types, and the context in the query, so it is hard to fix at scale. The motivation to develop a deep learning NER system is to let the catalog and customer data determine the best tagging of entities in the query to solve the problems of ambiguity and intent.

The rest of this paper is organized as follows. Section 2 describes the problem, our proposed solution in terms of an end-to-end framework, and the neural architecture. Section 3 covers offline experiments and results while Section 4 covers online deployment and measured impact to business.

2 An End-to-End Solution

We propose an end-to-end framework to solve the NER challenge at THD. In this section, we first define the problem, then elaborate on the framework, and describe the model architecture in the end.

2.1 Problem Definition Given a search query as a sequence of word tokens, the task of NER is to identify the important entities, which is formulated as a sequence tagging problem using the beginning-inside-outside (BIO) tagging format. We focus on the two most important entities for e-commerce: brand (`BRD`) and product type (`PRD`). Therefore, two entities are translated into five labels as shown in Table 2. The key task is to build a machine learning model for sequence tagging. The evaluation metric of the model is the exact-match F1 score [6] where only the correct prediction of the whole entity is considered as a true positive. The baseline is the legacy NER system in production as described in Section 1.3.

2.2 End-to-End Framework Our end-to-end framework consists of three phases as shown in Fig. 1. Each step or node is developed in a modular way for the convenience of development and optimization.

Phase I: Training Data Preparation. An ideal set of training data for deep learning should have three characteristics: 1) large volume, 2) high quality labels, 3) high coverage of label values (i.e. all brands and product types here). However, it is often too time-consuming to prepare such training data. We find that it is more realistic to prepare separate datasets to meet these three requirements.

The starting point are two foundational datasets in an e-commerce business: the product catalog (node `I.1` in Fig. 1) and customer behavior data (node `I.2`). Product catalog data is the ground truth for the key entities of all products sold on homedepot.com which has more than 14K brands, 11K product types, and 3 million items. Customer behavior data stores customers' shopping journeys, including searches, impressions, clicks, and purchases.

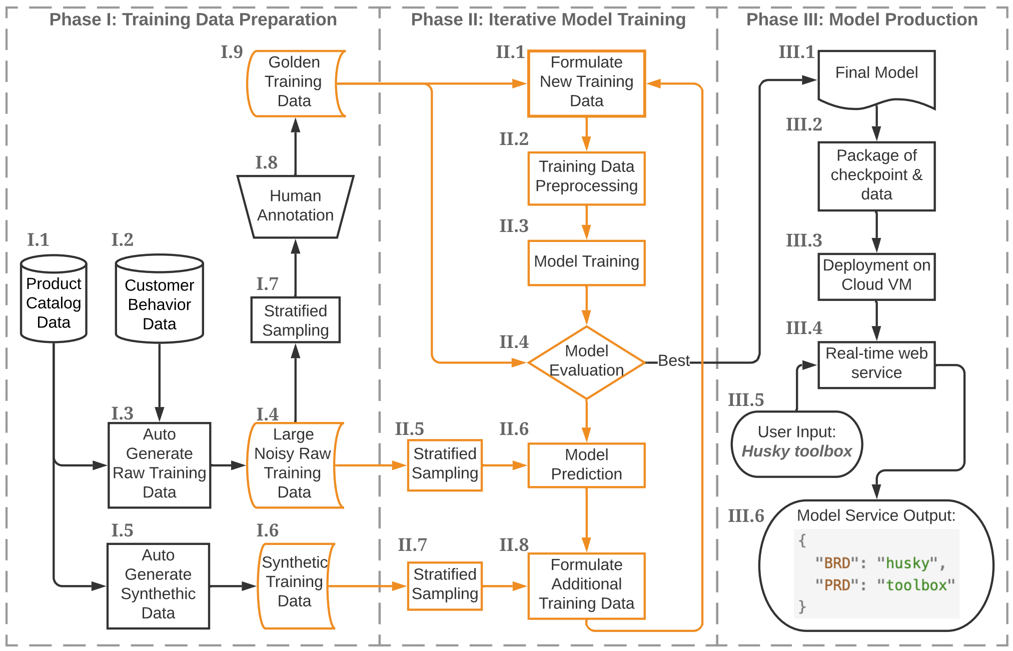
Firstly, a large amount of training data (`I.4`) is automatically generated using a rule-based algorithm (`I.3`) by matching the tokens in product catalog (`I.1`) and customer behavior data (`I.2`). This dataset can be noisy due to noisy customer behavior data and imperfect matching algorithm, but it is still important to capture the patterns in real customers' search queries.

Secondly, to prepare a set of high quality "golden" data (`I.9`), we sample a small amount of data from `I.4` for manual annotation. To avoid bias or over-fitting our model on a small number label sequence patterns, we stratified sample the data by entity pattern. Here a pattern is defined as a sequence of consequential entities (`BRD, PRD, O`), instead of individual labels. See Table 3 for the top three patterns and examples. The reason for sampling by patterns is that we find that our model is sensitive to the order of entities in the training data, which makes sense for a sequence tagging model. This set of sampled and manually annotated golden data is vital to build the first model in Phase II and to measure the real model performance in each iteration.

Thirdly, a set of synthetic training data (`I.6`) consisting of query-like strings is created by a rule-based algorithm (`I.5`) using all the brands and products in our product catalog (`I.1`). The synthetic pattern is usually simple, such as brand-only queries and product-

Table 1: The examples where legacy NER mislabeled entities. `"weed eater"` is ambiguous because it could be either a brand or a product type. The second query has two product types (`"ice maker"` and `"fridge"`), but the longer one is labeled. Lastly, `"table and chair set"` is not in the existing product taxonomy.

Search query	NER brand	NER product type	True brand	True product type
`weed eater light weight`	`weed eater`	`light`		`weed eater`
`fridge no ice maker`		`ice maker`		`fridge`
`cosco table and chair set`	`cosco`	`table`	`cosco`	`table and chair set`



```
{
  "BRD": "husky",
  "PRD": "toolbox"
}
```

Figure 1: An End-to-End Framework of the NER solution for the search engine at homedepot.com. The process highlighted in orange could be generalized to other problems using any supervised machine learning models.

Table 2: The 5 labels in this NER problem.

Label	Description
`B-BRD`	beginning of a brand
`I-BRD`	inside of a brand
`B-PRD`	beginning of a product type
`I-PRD`	inside of a product type
`O`	outside

Table 3: The examples of the top three patterns in the large volume of training data (`I.4`).

$\overbrace{\texttt{milwaukee}}^{BRD}\ \overbrace{\texttt{cheap}}^{O}\ \overbrace{\texttt{drill}}^{PRD}$
$\overbrace{\texttt{ge}}^{BRD}\ \overbrace{\texttt{7.4 cu ft}}^{O}\ \overbrace{\texttt{dryer}}^{PRD}\ \overbrace{\texttt{gas}}^{O}$
$\overbrace{\texttt{behr}}^{BRD}\ \overbrace{\texttt{paint}}^{PRD}\ \overbrace{\texttt{discount}}^{O}$

type-only queries. All the distinct brands and products are covered so that the model can potentially recognize them all.

The output of Phase I are three separate training datasets to meet the three requirements of an ideal training dataset. Their corpus statistics are shown in Table 4.

Phase II: Iterative Model Training. In Phase II, we iteratively train the model with more training data and aim to incrementally improve the model performance.

The first iteration starts with the golden data (`I.9`). 15% of the golden data is randomly held out as test data; the rest of the data is randomly split into training (90% of remaining data) and validation ("dev") data (10%). Then the training data is pre-processed to balance the

Table 4: Corpus statistics. The numbers of queries, tokens, and entities in the three datasets in Phase I.

Dataset	Large (I.4)	Synthetic (I.6)	Golden (I.9)
Query	2,737,399	23,710	16,915
Token	16,914,607	48,568	89,692
BRD	2,509,132	14,058	14,915
PRD	2,694,413	9,652	14,774

labels (`II.2`), which requires domain-specific knowledge here. For example, we identify 50 tokens which can be either a brand or a product, such as `instant pot` and `anchor`. Such queries are balanced by oversampling entities with fewer queries so that no bias is generated due to skewed training data distribution.

Then the model is trained (`II.3`) until the validation set F1 score stops improving or the maximum epoch is reached. After the training, the model is evaluated (`II.4`) on the test data to see whether this iteration has the best-performed model. If not, we stratified sample (`II.5`) from I.4 as done in `I.7`, for model inference (`II.6`). If the prediction on a query matches the noisy labels in `I.4`, we pass the query to the next step as candidates for additional training data. The motivation is to reduce the noise in training data by looking for consensus between the noisy labels produced by `I.3` and the noisy labels produced by `II.6`. Similarly, we sample (`II.7`) synthetic data (`I.6`) to increase the coverage of brands and product types so that all possible values are covered in a few iterations. In the end, we formulate new additional training data from `II.6` and `II.8` for the next iteration.

A question we had during early development was whether this iterative self-training process would accumulate errors, thereby biasing itself toward erroneous patterns of predictions in `II.6` such as a reduced set of label sequence patterns. We reduce the chance of the model drifting in this way by stratified sampling by label sequence patterns in steps `I.7`, `II.5`, and `II.7`. Beyond that, the final judge of this iterative process is the F1 score on the held-out test set. Despite the chance of the model drifting away from correct label patterns, we see that this iterative process can indeed improve the model performance iteration by iteration. More results are shown in Section 3.

Phase III: Model Production. Model production is the key to help THD search engine find more relevant products. This phase starts with the best model selected from Phase II. In node `III.2`, the neural graph, weights, and required data (vocabulary, word embeddings, etc) are packed for the next step. Then the model package is deployed on a cloud virtual machine (`III.3`). To use it in a real-time search engine, in node `III.4`, the other necessary components are also added to be able to parse a raw user search queries and to convert NER model prediction into machine readable outputs, which enable the deep learning model as a real-time web service for our search engine. The sample input and output of this service are shown in `III.5` and `III.6`.

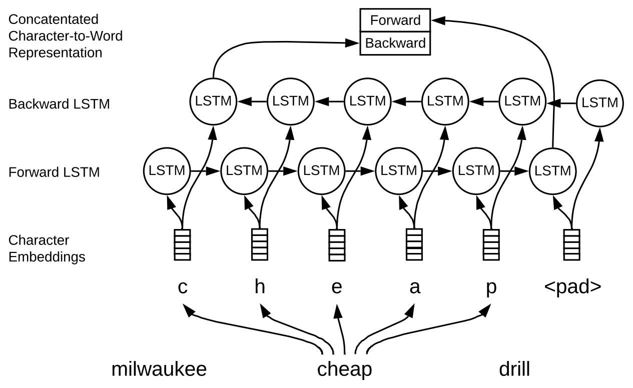


Figure 2: The character-to-word subgraph of our deep learning model. Both forward and backward word representations are learned from character embeddings using a BiLSTM layer.

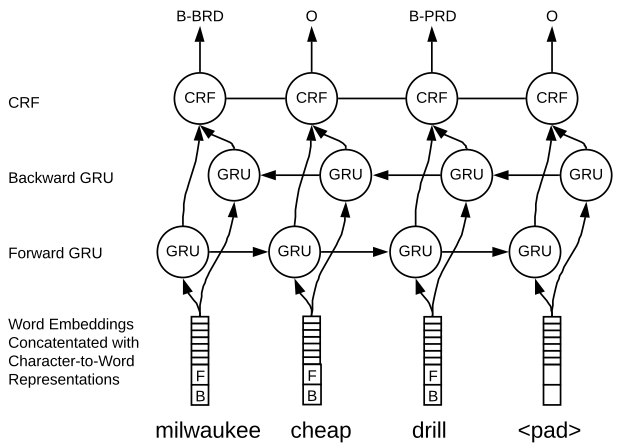

B-BRD
O
B-PRD
O
CRF
Backward GRU
Forward GRU
GRU
Word Embeddings Concatentated with Character-to-Word Representations
F
B
milwaukee
cheap
drill
<pad>

Figure 3: The word-to-label-sequence subgraph of our deep learning model. Forward ("F") and backward ("B") character-based word embeddings are concatenated to word embeddings as input to the bidirectional GRU (BiGRU) which in turn provides features for the CRF.

2.3 Model Architecture We base our model on the bidirectional RNN-CRF, a popular, recent approach to sequence tagging [2, 9]. Other recent work helps us to finalize the architecture. Huang et al. first demonstrated that BiLSTM-CRF can effectively use both left and right context as well as the statistical sequential dependency among token labels [10]. Lample et al. [12] further showed that BiLSTM-CRF with pre-trained word embeddings, character embeddings, and dropout performed the best on CoNLL-2003 [6]. The GRU, a simplified variant of the LSTM, has also shown state-of-the-art performance [9]. After numerous experiments (see details in Section 3), our final neural architecture is a BiGRU-CRF with a BiLSTM subgraph for character-level embedding, as shown in Figures 2 and 3. This neural architecture is implemented using Tensorflow.

3 Experiments & Results

We systematically run experiments on our framework, model architectures, embeddings, and hyperparameters. The evaluation metric is the model performance on the holdout test data described in Section2.2.

3.1 Iterative Model Training The process of iterative model training in Phase II (Section 2.2) shows improved model performance on the test data as shown in Fig.4 and Table 5. As we iteratively add more training data, the coverage of brands and product types increases and reaches 100% at iteration 7, which is important for this model to potentially recognize all brands and product types in real application. The F1 scores also saturate after iteration 7.

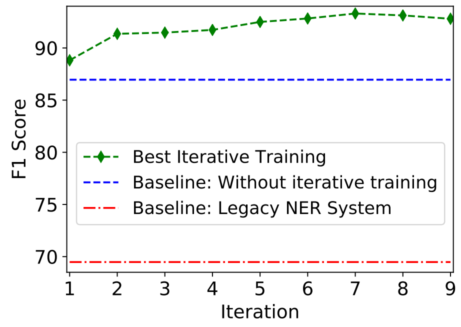


Figure 4: F1 score on the test data of our model over each iteration (green diamonds) and two baselines (dashed lines). The red line on bottom is for the legacy NER system. The blue line in the middle is for one-pass training using all the data (I.4 + I.6 + I.9 in Fig. 1).

The advantage of this iterative training process is justified in three aspects. Firstly, the F1 score is iteratively improved from 87.1 at iteration 1 to 93.3 at iteration 7. Secondly, compared to simply training the model once using all the data (Fig. 4 blue horizontal line), this iterative process performs better in every iteration. Lastly, compared to the most important baseline, the legacy NER system in production, our best model increases the F1 score from 69.5 to 93.3. This significant improvement shows the superiority of the model and justifies an A/B test in production. The model at iteration 7 is selected for production because of its best F1 score and complete coverage of all the brands and product types.

Table 5: The F1 scores and the numbers of training queries, unique brands, and unique product types by iteration.

i	training	unq. BRD	unq. PRD	dev F1	test F1
1	14,378	3400	3239	89.32	88.82
2	19,911	3936	3768	95.81	91.36
3	31,510	4402	4216	96.52	91.47
4	57,379	5374	5131	97.70	91.73
5	115,564	7544	7004	98.68	92.49
6	241,629	12,082	10,305	99.19	92.83
7	484,390	14,102	11,111	99.49	93.30
8	992,823	14,102	11,111	99.66	93.12
9	2,089,573	14,102	11,111	99.81	92.79

Table 6: Four neural architectures tested on golden data, with and without layers of character-based embedding.

models	char emb.	dev F1	test F1
BiLSTM	No	85.77	85.05
	Yes	86.99	86.23
BiLSTM-CRF	No	87.69	86.72
	Yes	88.57	88.44
BiGRU	No	85.42	85.57
	Yes	86.53	87.09
BiGRU-CRF	No	87.12	87.04
	Yes	**88.71**	**88.82**

A major disadvantage is its computational cost. On average, the whole process with nine iterations takes about 20 hours on a GPU machine (NVIDIA Tesla K80). Therefore, most of the experiments below are tested on golden data only (iteration 1), which is much faster. However, from the finished iterative model trainings, we find that the model performance at iteration 1 is a good indicator of the final model performance, which can justify the experiments using the golden data in the next three subsections.

3.2 Tested Models The final neural architecture in production is BiGRU-CRF with BiLSTM character-based word representations, but we tested four different architectures as shown in Table 6.

The character-based word embedding is produced by a BiLSTM layer [12] (Fig. 2). It essentially extracts features for each word using a character-level model. With character-based embeddings, the F1 score is consistently better as shown in Table 6. We believe the reason is that it can help bridge the gap between common query words and uncommon query words that contain common sub-word character sequences.

We also compare BiGRU and BiLSTM, with BiGRU showing comparable or better performance. We select BiGRU because it has 25% fewer parameters and is thus faster to train and execute.

The CRF layer helps to predict the most likely sequence and to forbid invalid sequence transitions, such as `B-BRD`→`I-PRD` and `O`→`I-BRD`. The results in Table 6 also show that it consistently improves the F1 score.

3.3 Word Embeddings We test pre-trained Word2vec embedding [23], pre-trained GloVe [3], custom Word2vec, and custom GloVe embedding. The custom embeddings are trained using the top 50 million search queries and product titles on homedepot.com.

The custom Word2vec embedding is selected for three reasons as shown in Table 7. Firstly, domain-specific embeddings are a better semantic representation in terms of similar words. For example, our custom embeddings correctly associate `"milwaukee"` with other brands while off-the-shelf embeddings associate it with cities. Secondly, the vocabulary coverage is much higher (99.9% vs. 16% to 40%). Lastly, the model performance is also better or comparable to the pre-trained embeddings.

3.4 Hyper-parameters We tune our model using the following five hyper-parameters (chosen values in parentheses): dropout (0.3), max. epoch (20), word and character dimensionality (300 and 100), and optimization method (`adam`). Since the training cost is expensive for deep learning models, we do not grid-search all the combinations of these parameters. Instead, we optimize one hyper-parameter at a time by fixing the rest of the hyper-parameters.

4 Productization

Productization is the key to delivering a real-world impact, and there are two correlated challenges, speed and cost. Firstly, the model execution has to be fast enough to serve thousands of queries per second and help retrieve search results within a time limit. The speed has a direct impact on user satisfaction and conversion rate. Secondly, the cost of serving the model has to be reasonably low to justify the return on investment. More details are explained below.

Table 7: Top three similar words for `"milwaukee"`, vocabulary coverage, and F1 score at iteration 1 of the four word embeddings.

embedding	similar words	vocab coverage	F1
pre-trained GloVe	chicago detroit minneapolis	39.8%	87.56
pre-trained Word2vec	springfield harvey wisconsin	15.5%	83.99
custom GloVe	m18 dewalt drill	99.9%	87.58
custom Word2vec	dewalt makita ridgid	**99.9%**	**88.82**

4.1 Deployment The first challenge is speed, and a practical solution is to leverage an existing platform with customized optimization. Specifically, the model is deployed using Tensorflow Serving, a flexible and high-performance model serving system by Google, on the Google Cloud Platform (GCP). This can provide a stable environment but still not an optimized one. Thus, we customize the optimization by reducing the servable model size which has a direct impact on the model inference time. This involves converting the variables in a Tensorflow checkpoint into constants stored directly in the model graph, stripping out unreachable parts of the graph, folding constants, folding batch normalizations, removing training and debug operations, etc. In our experience, the optimized model has a significantly smaller size, faster loading, and faster inference.

The other challenge is cost. Serving the deep learning model on a GPU machine would be fast but also much more expensive. We manage to optimize the model for CPU to meet our performance requirement and deploy it on a GCP virtual machine instances with custom CPU machine type (4 vCPUs, 3.75 GB memory). This deployment automatically scales to real-time traffic, leading to a very cost-effective solution.

The model has been live in production for an extended period of time. We see that the model service can serve 99% of the search queries within 26 milliseconds, including the time for query pre-processing, model inference, and post-processing. This is highly satisfactory for our search engine. This concludes a successful engineering deployment of the model.

4.2 A/B Test and Business Impact The deep learning model service was A/B tested against the legacy NER system with equally random traffic split at homedepot.com. With millions of search visits from real online shoppers, we saw a significant improvement in both click-through-rate and revenue per search. The estimated business impact is $60M in incremental revenue. The cost is minimal comparing to its benefit and the existing search engine operating cost, so this project is yielding a high return on investment. Detailed metrics measured from the A/B test are confidential and not disclosed here.

5 Conclusion

Our work demonstrates an end-to-end solution to apply a state-of-art deep learning model in a domain-specific industrial problem, i.e. named entity recognition for e-commerce search queries. We propose a framework to address the remaining challenges regarding training data, domain knowledge, and production. We solve the training data challenge by leveraging general-purpose data to prepare three separate sets of training data. Then these three datasets are used to iteratively train the model using customized domain-specific word embeddings. We demonstrate that this iterative process is effective at improving model performance. The best model has been deployed in production as a real-time web service by using both the existing platform and customized optimizations. The model A/B test and day-to-day use at homedepot.com show stable model performance, increasing user engagement, and increased revenue, which proves the significant value of our work. Moreover, the framework highlighted may be generalized to solve other domain-specific industrial problems in data science and machine learning.

References

[1] David Nadeau and Satoshi Sekine. A survey of named entity recognition and classification. *Lingvisticæ Investigationes*, 30(1):3–26, 2007.

[2] Vikas Yadav and Steven Bethard. A survey on recent advances in named entity recognition from deep learning models. *arXiv preprint arXiv:1910.11470*, 2019.

[3] Jeffrey Pennington, Richard Socher, and Christopher Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

[4] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[5] Matthew E Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. *arXiv preprint arXiv:1802.05365*, 2018.

[6] Erik F Sang and Fien De Meulder. Introduction to the conll-2003 shared task: Language-independent named entity recognition. *arXiv preprint cs/0306050*, 2003.

[7] Ronan Collobert and Jason Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th international conference on Machine learning*, pages 160–167, 2008.

[8] Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. Natural language processing (almost) from scratch. *Journal of machine learning research*, 12(Aug):2493–2537, 2011.

[9] Changki Lee. Lstm-crf models for named entity recognition. *IEICE Transactions on Information and Systems*, 100(4):882–887, 2017.

[10] Zhiheng Huang, Wei Xu, and Kai Yu. Bidirectional lstm-crf models for sequence tagging. *arXiv preprint arXiv:1508.01991*, 2015.

[11] Xuezhe Ma and Eduard Hovy. End-to-end sequence labeling via bi-directional lstm-cnns-crf. *arXiv preprint arXiv:1603.01354*, 2016.

[12] Guillaume Lample, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami, and Chris Dyer. Neural architectures for named entity recognition. *CoRR*, abs/1603.01360, 2016.

[13] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[14] Yijin Liu, Fandong Meng, Jinchao Zhang, Jinan Xu, Yufeng Chen, and Jie Zhou. Gcdt: A global context enhanced deep transition architecture for sequence labeling. *arXiv preprint arXiv:1906.02437*, 2019.

[15] Papers With Code. Papers with code - named entity recognition, 2020.

[16] Chao-yuan Wu, Amr Ahmed, Gowtham Ramani Kumar, and Ritendra Datta. Predicting latent structured intents from shopping queries. *WWW 2017*, 2017.

[17] Bodhisattwa Prasad Majumder, Aditya Subramanian, Abhinandan Krishnan, Shreyansh Gandhi, and Ajinkya More. Deep recurrent neural networks for product attribute extraction in ecommerce. *arXiv preprint ArXiv:1803.11284*, 2018.

[18] Musen Wen, Deepak Kumar Vasthimal, Alan Lu, Tian Wang, and Aimin Guo. Building large-scale deep learning system for entity recognition in e-commerce search. In *Proceedings of the 6th IEEE/ACM International Conference on Big Data Computing, Applications and*

Technologies, pages 149–154, 2019.

[19] Brooke Cowan, Sven Zethelius, Brittany Luk, Teodora Baras, Prachi Ukarde, and Daodao Zhang. Named entity recognition in travel-related search queries. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, AAAI'15, pages 3935–3941. AAAI Press, 2015.

[20] Ajinkya More. Attribute extraction from product titles in ecommerce. *CoRR*, abs/1608.04670, 2016.

[21] Jiafeng Guo, Gu Xu, Xueqi Cheng, and Hang Li. Named entity recognition in query. In *Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval*, pages 267–274. ACM, 2009.

[22] Duangmanee (Pew) Putthividhya and Junling Hu. Bootstrapped named entity recognition for product attribute extraction. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, EMNLP '11, pages 1557–1567, Stroudsburg, PA, USA, 2011. Association for Computational Linguistics.

[23] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.