A Human-machine Interface for Few-shot Rule Synthesis for Information Extraction

Robert Vacareanu, George C. G. Barbosa, Enrique Noriega-Atala, Gus Hahn-Powell, Rebecca Sharp, Marco A. Valenzuela-Escárcega, Mihai Surdeanu
University of Arizona
Tucson, AZ, USA
{rvacareanu, gcgbarbosa, enoriega, hahnpowell, msurdeanu}@email.arizona.edu
{bsharpataz, marcovalenzuelaescarcega}@gmail.com

Abstract

We propose a system that assists a user in constructing transparent information extraction models, consisting of patterns (or rules) written in a declarative language, through program synthesis. Users of our system can specify their requirements through the use of examples, which are collected with a search interface. The rule-synthesis system proposes rule candidates and the results of applying them on a textual corpus; the user has the option to accept the candidate, request another option, or adjust the examples provided to the system. Through an interactive evaluation, we show that our approach generates high-precision rules even in a 1-shot setting. On a second evaluation on a widely-used relation extraction dataset (TACRED), our method generates rules that outperform considerably manually written patterns. Our code, demo, and documentation is available at https://clulab.github.io/odinsynth/.

1 Introduction

Rule-based methods for information extraction address the opacity of neural architectures by producing models that are completely transparent, i.e., they are usually a collection of rules written in a declarative language. Such models are better suited for incremental improvements, as each individual rule can be explicitly interpreted. However, these benefits do not come for free: users of such systems must be familiar with the underlying declarative rule language, and, potentially, with representations of syntax such as syntactic dependencies. None of these are trivial to users outside of natural language processing (NLP), which, we argue, should be the target users of these systems.

To mitigate the above limitation, we propose a human-machine interface (HMI) that: (a) lets users synthesize rules from natural language examples, and correct them *without necessarily understanding the rule syntax* (although expert users who do have access to the actual rule produced), (b) generates rules in a *few-shot setting*, i.e., from a very small number of examples. The latter contribution is possible because our rule synthesis engine has been pretrained on a large collection of rules that were automatically generated from a large text corpus (Vacareanu et al., 2022). In other words, our rule synthesis approach is akin to prompting for language models (Liu et al., 2021). That is, we first train an open-domain rule synthesis model, and then we guide its predictions to the task of interest using a small number of examples of the desired extractions (the "prompt").

We include two types of evaluations that prove the value of the proposed approach. The first evaluation focuses on interactive sessions where users generate rules that extract mentions of named entity classes, e.g., `CITY` or `ACADEMIC INSTITUTION` from a *single example extraction*. Using six different users, we show that, despite the minimal supervision, the tool produces named entities in the corresponding classes with high precision, e.g., precision at 20 (P@20) at over 75% for the `CITY` class. This suggests that the proposed HMI is useful to domain scientists that need to perform information extraction quickly, without understanding the underlying NLP technology. For completeness, we also include a traditional, non-interactive evaluation, on the TACRED dataset (Zhang et al., 2017), through which we show that the rules synthesized by our approach outperform the manually-written rules by over 4 F1 points, even though our synthesis component is *not* re-trained on the TACRED data.

Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies: System Demonstrations, pages 64 - 70

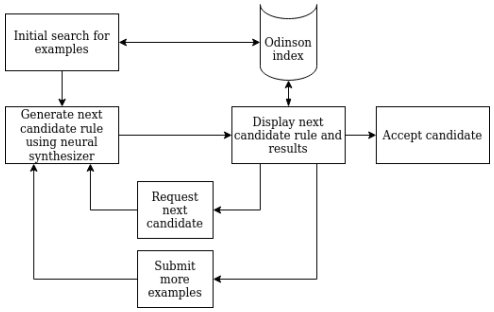

Figure 1: The architecture of our proposed system.

2 Architecture

At a high level, the proposed system consists of two main modules: a rule synthesis component, and an UI module to facilitate rapid rule prototyping with minimal programming or linguistic knowledge. The rule synthesis component consists of: (a) a searcher that explores the possible rule space, and (b) a neural scorer that prioritizes next steps during rule generation. Figure 1 summarizes the overall architecture. The user selects a handful of examples after an initial query. Then, our system proposes a rule, together with its potential extractions. The user then decides if this rule is satisfactory. If not, the user can either ask for a new rule or add new examples (positive or negative). This process repeats until the generated rule is accepted. We describe these components in detail in the next two sections.

3 Rule Synthesis

Our proposed approach for rule generation follows closely the method proposed in (Vacareanu et al., 2022). For completeness, we summarize it here as well. Our rule synthesis method uses enumerative search that is guided by a transformer-based scoring mechanism, and is optimized using search-space pruning heuristics. Our transformer model scores each potential next state (where a **state** contains the incompletely generated rule up to this point), given the current state, such that the number of states to be explored is minimized. Specifically, our system consists of two main components:

A **searcher**, with Branch and Bound (Land and Doig, 1960) as the underlying algorithm. The searcher uses the scores assigned by the scorer (below) to determine the order of exploration, choosing the state with the highest score, *regardless of its position in the search tree*. As such, it is important for the scorer to assign high scores to states that are in the sub-tree that leads to the desired final rule, and lower scores to all other states;

A **scorer**, with a transformer backbone that is initialized with a pretrained model, but fine-tuned through self-supervision, i.e., over *automatically generated rules* (see Section 3.2). The scorer uses the current state and the **specification**, i.e., the natural language examples to be matched by the generated rule, to score each potential next state.

The searcher is responsible for exploring the states in priority order (as determined by the scorer), and deciding if a given state is successful (i.e., it is a valid query and correctly extracts the requested highlighted words and nothing more). The search space can be interpreted as a tree, where the root is the initial candidate solution and the children of a node n are the candidate solutions that the node n could expand into. Given this, the searcher can be seen as iteratively applying a sequence of three operations: (a) **Expand** the current state according to the domain-specific language grammar, (b) **Score** each expanded candidate next state and insert them into the priority queue, and (c) **Select** from the queue the state with the highest score to be the next state. We repeat this process until we find a solution or we reach our step limit.

The scorer assigns a numerical value to states to establish the order of exploration. We explore two variants: a **static** variant, which assigns static weights to states based on their components, and a **contextual** neural variant based on a self-supervised method that assigns contextual state scores based on the current context.

As a simple example, consider a user that wants to learn named entities belonging to the class `CITY`. She may start with a specification based on the sentence "Regina Romero is the mayor of Tucson, Arizona, having been elected after...", in which she highlights "mayor of" as the relevant *context* representative of this class, and "Tucson" as the desired entity to be extracted. From this specification, our method would generate the rule: `[lemma=mayor] [tag=IN] (?<arg>`

`[tag=NNP]+)`,[1] which picks up "mayor" followed by a preposition (part-of-speech tag `IN`) as the context, and a sequence of 1 or more proper nouns as the entity to be extracted.

3.1 Multiple sentences

Our system can handle specifications that contain multiple sentences and their highlights. We require the enumerative searcher to find a rule that would satisfy *all* the constraints for all sentences in the specification. When scoring, we score a (current state, next potential state, single-sentence specification) triple, and then average over all sentences in the specification to obtain a final score for the (current state, next potential state) transition.

3.2 Training the neural scorer

Unlike the static scorer, the neural guiding function of the contextual scorer needs to be trained, which we do with self-supervision. Because there is no large corpus of Odinson rules, we artificially generate one with random spans of text that we randomly manipulate into rules. Our random-length text spans are chosen from the UMBC corpus (Han et al., 2013). Each token in this span is then randomly manipulated into an Odinson token constraint based on either word, lemma, or part-of-speech. For example, a span such as *the dog barked* might be converted to `[tag=DT] [word=dog] [lemma=bark]`. Then, to expose the model to additional rule components (e.g., alternation, quantifiers), we add further manipulations, again with randomization. To add alternations, we build a temporary query by replacing one of the token constraints with a wildcard that can match *any* token and query the corpus for an additional sentence that has different content in that position. This new content is added as an alternation. For example, with the temporary version of the above query `[tag=DT] [word=dog] []`,[2] we might find *A dog runs*, resulting in the following alternation: `[tag=DT] [word=dog] ([lemma=bark]|[lemma=run])`. To add a quantifier (i.e., `*`, `+`, or `?`), we select a token to modify and a quantifier to add, and check the corpus to ensure that the addition of the quantifier yields additional results.

After generating each random rule, we build a corresponding specification by querying the UMBC corpus: the retrieved sentences and their matched spans constitute specifications. However, having a specification and the corresponding rule is not enough to train our model. We also need a correct sequence of transitions from the initial placeholder to the final rule. For this, we use an Oracle to generate the shortest sequence of transitions, which we consider to be the correct sequence for our purposes. This sequence of transitions, together with the specification, forms the training data for our model. Note that we train *only* on this data, i.e., after this self-supervised training process the transformer's weights are fixed. We train using the cross-entropy loss and with a cyclical learning rate, as suggested by (Smith, 2017). Further, we employ a curriculum learning approach (Bengio et al., 2009; Platanios et al., 2019), splitting the training data by sentence length and by pattern length. We did not tune our hyperparameters.

4 User Interface

We accompany the above rule synthesis component with a user interface (UI) to facilitate rapid prototyping with minimal programming or linguistic knowledge. We showcase the UI in Figure 2, split into 5 blocks (a–e). Initially, the user has to do an initial search for sentences of interest (a). Then, she selects any relevant sentence, highlighting the parts for which she wishes to obtain a rule, e.g., highlighting *the capital city of* as the context and *Amman* as the entity of interest (b). The system then returns a potential rule which satisfies the current constraints, together with what the rule extracts (c). Note that we display the rule for the benefit of expert users, but most users are not required to understand the format of the rule. That is, a user can understand the rule's impact by analyzing what such a rule extracts. She may add negative examples (a negative example is a sentence on which the output rule should not match anything), or ask for a new rule (d). This process is repeated until the user is satisfied with the given rule (e).

[1] Our rules are generated in the Odinson rule language (Valenzuela-Escárcega et al., 2020).

[2] The Odinson wildcard, `[]`, matches any token.

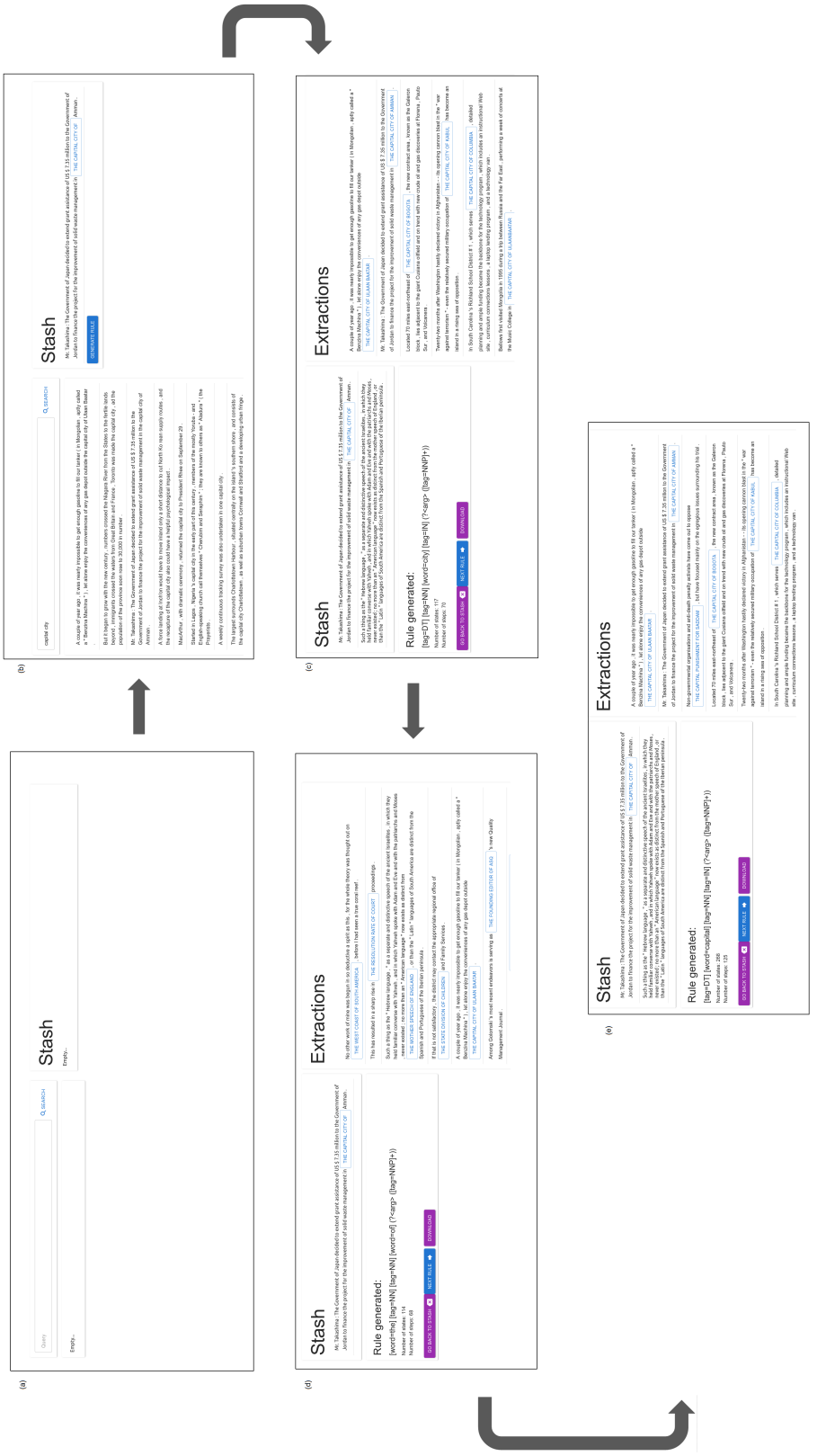

Figure 2: Walkthrough example of the user interface.

5 Evaluation

5.1 Interactive Evaluation

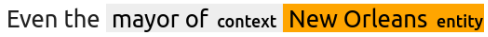

Figure 3: Example of a specification annotated by a user in the interactive evaluation. The entity is highlighted in orange and its context in gray.

We evaluated the performance of the system in an interactive scenario with a human in the loop. The purpose of the interactive evaluation is to quantify the performance of rules generated with the interface to extract specific entity types using a limited amount of examples. The user is tasked with using the interface to synthesize a rule to extract a specific named entity type, and then manually verify that the extractions indeed belong to the intended entity type. Given an entity type, the user queries an index of the UMBC corpus to pinpoint and select a pattern in one of the retrieved sentences. This pattern works as the user's specification, composed of the *context* and an *entity* of the relevant type (See section 3). Figure 3 depicts an example of a pattern selection.

Once the specification is selected, it is used to synthesize a candidate rule and retrieve a sample sentences with matches. We restrict the evaluation to contain *a single* specification, to emulate a one-shot learning scenario, where the model generates rules using a limited amount of information. The user inspects the rule and its matches to determine whether the candidate rule faithfully represents the original intent. If it does not, the next candidate rule is generated and the process is repeated. We encouraged users to repeat this process up to three times, but allowed them to repeat it a fourth time at their discretion.

The interactive evaluation is carried out for the following entity pairs: `CITY`/`CAPITAL`, `PERSON`/`ACTOR`, and `ORGANIZATION`/`ACADEMIC INSTITUTION`. Each pair represents two different levels of granularity of the same concept.

One to two users were assigned to each pair and each user was instructed to use the interface to synthesize three rules per entity type. For every rule, they retrieved and manually verified precision at 10 (P@10) and precision at 20 (P@20) on the most frequent entities in the matches from the UMBC corpus.

Table 1 contains precision at 10 and precision at 20 for each of the entity types. We can observe that using minimal supervision, i.e. using a single specification to synthesize a rule for a named entity type, the system generates rules that have high precision (P@20 $\geq$ .77) for coarse grained named entities and similar, albeit slightly lower precision (P@20 $\geq$ .63), for finer grained named entities. This is achieved with no more than three or four iterations, highlighting how domain experts can readily benefit from our proposed HMI.

Entity Type	P@10	P@20
`CITY`	.85	.85
`CAPITAL`	.83	.75
`PERSON`	.78	.77
`ACTOR`	.71	.72
`ORGANIZATION`	.93	.85
`ACADEMIC INSTITUTION`	.70	.63

Table 1: P@10 and P@20 of our rule synthesis on 6 different named entity types. `CITY`, `PERSON` and `ORGANIZATION` are coarse types. `CAPIITAL`, `ACTOR` and `ACADEMIC INSTITUTION` are fine grained types.

5.2 Non-interactive Evaluation

To facilitate a comparison with other approaches, we also include an evaluation on the TACRED dataset, a widely-used relation classification dataset (Zhang et al., 2017). In this setting, we cluster the training sentences to generate specifications. In particular, for each sentence, we compute an embedding by averaging the embeddings of the words in between the two given entities.[3] Then, we compare the similarity of two sentences by cosine similarity, and cluster similar sentences together; each cluster becomes one specification. Then, for each cluster we run our system to generate a rule, considering the words in-between the two entities as the highlighted part.

We compare our approach against several state-of-the-art approaches, as well as three baselines. Our first baseline is a traditional sequence-to-sequence approach (Sutskever et al., 2014) with transformers (Vaswani et al., 2017). We train it to decode the rule using the specification as input, akin

[3]We used GloVe (Pennington et al., 2014).

to a traditional machine translation task. Our second baseline (*Patterns*) is a rule-based system that uses the *hand-made* rules compiled for TACRED (Zhang et al., 2017). Our third baseline *No Learning* consists of directly returning a rule for trivial cases (e.g. no words or only on word in between the two entities). When there is no word, the final rule is empty. When there is one word, the final rule consists of a word, lemma, or tag constraint, depending on which will result in a shorter rule. We present our results in Table 2.

We note that both variants of our scorer (static and dynamic) perform better than the seq2seq and no-scorer baselines (34 F1 vs 28 F1). Of particular relevance for our comparison is the baseline that relies on the handcrafted patterns for TACRED. Our contextualized weights model obtains a higher F1 score (41.4 F1 vs 36.6 F1), although at the cost of precision, but with much higher recall. Our results add evidence that it might be possible to replace the human expert with a neural expert. When comparing our contextualized-scoring approach to the supervised baselines, we note that while we do not match their performance, there are two important factors to consider. First, our proposed approach is trained on *domain agnostic* data that we automatically generated, and then applied on TACRED as is, *without fine-tuning*. On the other hand, the supervised approaches that we compare with train/fine-tune on the TACRED splits. Second, the output of our approach is a set of *human-interpretable rules*, while the output of the other approaches is a statistical model that produces only the final label. In other words, previous work is much more opaque and thus more difficult to interpret, debug, adjust, maintain, and protect from hidden biases present in the training data (e.g., Kurita et al., 2019; Sheng et al., 2019).

Model	P	R	F1
Baselines			
Seq2Seq with Transformers	53.0	19.0	28.0
Patterns	86.9	23.2	36.6
No Learning	53.0	19.0	28.0
Supervised Approaches			
Joshi et al. (2020)	70.8	70.9	70.8
Zhou and Chen (2021)	–	–	74.6
Cohen et al. (2020)	74.6	75.2	74.8
Our approach			
Static weights	54.9	24.6	34.0
Contextual weights (BERT-Tiny)	57.6	29.6	39.1
Contextual weights (BERT-Mini)	57.2	32.5	41.4
Contextual weights (BERT-Small)	57.3	32.2	41.2
Contextual weights (BERT-Medium)	55.0	31.8	40.3
Contextual weights (BERT-Base)	55.6	32.4	41.0

Table 2: Results of our rule synthesis on the testing partition of TACRED (given as precision (P), recall (R), and F1 scores), compared with 3 baselines and previous supervised approaches. We include variants of our contextualized method using different transformer backbones.

6 Conclusion and Future Work

We introduced a human-machine interface that lets users synthesize rules from natural language examples, and correct them without necessarily understanding the rule syntax. In an interactive evaluation, we showed that our method is capable to rapidly generate high-precision rules for the extraction of named entities from a single example in natural language. We also demonstrated that in a traditional, non-interactive evaluation on the TACRED dataset, our method produces rules that outperform manually-written rules, despite the fact that our rule synthesis engine is not retrained on the TACRED data.

While these initial results are exciting, there is plenty of work left to do. First, the rules generated by the system are "surface" rules, i.e., they act over sequences of tokens. Adding support for the generation of rules over syntax would allow for capturing more complicated relations and over greater distances. Second, the system is configured to generate a *single* rule that captures the whole specification, which may include multiple (positive or negative) examples. This may force the system to produce complicated rules. A better alternative would be to produce several simpler rules that together capture the whole specification. Lastly, the system generates a sequence of rule candidates that are presented to the user one-by-one, which may bias the user's perspective and impact usability. For example, the current system tends to initially propose rules that are overly general, which yield low-precision results. To better understand users' preferences (i.e., do users prefer high-precision or high-recall rules initially?) user studies must be carried out.

References

Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. 2009. Curriculum learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, page 41–48, New York, NY, USA. Association for Computing Machinery.

Amir DN Cohen, Shachar Rosenman, and Yoav Goldberg. 2020. Relation classification as two-way span-prediction. *arXiv preprint arXiv:2010.04829*.

Lushan Han, Abhay L. Kashyap, Tim Finin, James Mayfield, and Jonathan Weese. 2013. UMBC_EBIQUITY-CORE: Semantic textual similarity systems. In *Second Joint Conference on Lexical and Computational Semantics (*SEM), Volume 1: Proceedings of the Main Conference and the Shared Task: Semantic Textual Similarity*, pages 44–52, Atlanta, Georgia, USA. Association for Computational Linguistics.

Mandar Joshi, Danqi Chen, Yinhan Liu, Daniel S. Weld, Luke Zettlemoyer, and Omer Levy. 2020. SpanBERT: Improving pre-training by representing and predicting spans. *Transactions of the Association for Computational Linguistics*, 8:64–77.

Keita Kurita, Nidhi Vyas, Ayush Pareek, Alan W Black, and Yulia Tsvetkov. 2019. Measuring bias in contextualized word representations. *arXiv preprint arXiv:1906.07337*.

Ailsa H. Land and Alison G. Doig. 1960. An automatic method of solving discrete programming problems. *Econometrica*, 28(3):497–520.

Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. 2021. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. *arXiv preprint arXiv:2107.13586*.

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

Emmanouil Antonios Platanios, Otilia Stretcu, Graham Neubig, Barnabás Póczos, and Tom M. Mitchell. 2019. Competence-based curriculum learning for neural machine translation. *CoRR*, abs/1903.09848.

Emily Sheng, Kai-Wei Chang, Prem Natarajan, and Nanyun Peng. 2019. The woman worked as a babysitter: On biases in language generation. 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 3398–3403.

Leslie N. Smith. 2017. Cyclical learning rates for training neural networks. In *2017 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 464–472.

Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. 2014. Sequence to sequence learning with neural networks. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'14, page 3104–3112, Cambridge, MA, USA. MIT Press.

Robert Vacareanu, Marco A Valenzuela-Escarcega, George CG Barbosa, Rebecca Sharp, and Mihai Surdeanu. 2022. From examples to rules: Neural guided rule synthesis for information extraction. *arXiv preprint arXiv:2202.00475*.

Marco A Valenzuela-Escárcega, Gus Hahn-Powell, and Dane Bell. 2020. Odinson: A fast rule-based information extraction framework. In *Proceedings of the 12th Language Resources and Evaluation Conference*, pages 2183–2191.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, undefinedukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 6000–6010, Red Hook, NY, USA. Curran Associates Inc.

Yuhao Zhang, Victor Zhong, Danqi Chen, Gabor Angeli, and Christopher D. Manning. 2017. Position-aware attention and supervised data improve slot filling. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing (EMNLP 2017)*, pages 35–45.

Wenxuan Zhou and Muhao Chen. 2021. An improved baseline for sentence-level relation extraction. *arXiv preprint arXiv:2102.01373*.