Rule Based Event Extraction for Artificial Social Intelligence

Remo Nitschke*, **Yuwei Wang***, **Chen Chen***, **Adarsh Pyarelal***, **Rebecca Sharp**†
*University of Arizona, Tucson, Arizona, USA
†Lex Machina, Tucson, Arizona, USA
{nitschke, wangyw, chencc33, adarsh}@arizona.edu

Abstract

Natural language (as opposed to structured communication modes such as Morse code) is by far the most common mode of communication between humans, and can thus provide significant insight into both individual mental states and interpersonal dynamics.

As part of DARPA's Artificial Social Intelligence for Successful Teams (ASIST) program, we are developing an AI agent team member that constructs and maintains models of their human teammates and provides appropriate task-relevant advice to improve team processes and mission performance. One of the key components of this agent is a module that uses a rule-based approach to extract task-relevant events from natural language utterances in real time, and publish them for consumption by downstream components.

In this case study, we evaluate the performance of our rule-based event extraction system on a recently conducted ASIST experiment consisting of a simulated urban search and rescue mission in Minecraft. We compare the performance of our approach with that of a zero-shot neural classifier, and find that our approach outperforms the classifier for all event types, even when the classifier is used in an oracle setting where it knows how many events should be extracted from each utterance.

1 Introduction

Humans communicate with each other using both explicit (e.g., written and spoken natural language) and implicit (e.g., tone of voice, body language) modalities. While we posit that an artificially intelligent (AI) agent needs to handle both of these modalities to serve as an effective teammate on a hybrid human-machine team, in this paper, we focus on the former.

To that end, here we present a case study describing our approach for extracting events relevant to team coordination (e.g., instructions, requests, knowledge-sharing statements about the locations of people and objects, etc.) in real-time from natural language dialog. This was carried out in the context of DARPA's Artificial Social Intelligence for Successful Teams (ASIST) program,[1] a 4.5 year program aimed at developing technologies for imbuing artificial agents with *social* intelligence, i.e., the ability to construct and maintain models of their human teammates in order to provide more effective assistance. The program is structured around five large-scale experiments. One of the primary goals of these experiments is to evaluate the AI agents developed in the program on their ability to successfully predict human behavior and improve team processes. In order to do this, the agents need to understand (and perhaps contribute to) the dialog that takes place between their teammates.

This case study focuses on the third of these five experiments (ASIST Study 3), in which teams consisting of three humans and an AI advisor must work together to rescue as many victims as possible within a limited time. The mission takes place in a collapsed office building simulated in Minecraft. The human players participate remotely, communicating with each other using voice chat.[2] The goal of our event extraction component is to detect specific team coordination-related events, and relay them to AI agents, who then update their understanding of the state of the team members and the mission. The event extraction component is embedded into a larger architecture, which is described in our preregistration document (Pyarelal, 2022, 5).

Critically, the actual *design* of these missions is subject to change with little notice. There is no training data for supervised approaches, and the specificity of the domain means that many open-domain approaches are unsuitable. For these rea-

[1] https://www.darpa.mil/program/artificial-social-intelligence-for-successful-teams

[2] For further details on the experimental design, see Huang et al. (2022a).

Proceedings of the 29th International Conference on Computational Linguistic, pages 71–84
October 12–17, 2022.

sons (described further in § 2) we employ a rule-based approach for our event extraction component. This allows us to rapidly pivot and adapt to changes in requirements, as well as encode details of the specific domain.

In this setting, this work provides these key contributions:

1. Case study comparison of a rule-based and a zero-shot approach to team coordination event extraction in a (semi) real-world scenario. We describe each approach and discuss the advantages and drawbacks of each. We also show that, at least in this setting with these constraints, the rule-based system outperforms the zero-shot system and is more flexible, with richer representations.

2. Evaluation dataset annotated for twenty communication events. This data can be used to evaluate other approaches to the same task, as well as serving as an example of what to do (and not do) when designing an annotation task for event extraction.[3]

2 Motivation for Rule-Based Approach

While most current academic research focuses on machine learning (ML) based approaches to information extraction (Ahmad et al., 2021; Du and Cardie, 2020a; Nguyen et al., 2016a; Tozzo et al., 2018; Chiticariu et al., 2013), here we use a rule-based approach that was developed in response to the specific constraints of the task and the ASIST program.

1. **Rapid adaptation.** The experimental setup of ASIST is subject to change at each experiment. Thus, this is a dynamic domain where entities, events, and relations are likely to change dramatically. Hence we need a system that can quickly and reliably adapt to such changes. Further, it would be near-impossible to annotate data and train or fine-tune a neural agent on the new vocabulary of a study *prior* to its actual execution. By using rules, we can simply add, modify, or remove rules as needed. In this way, adaption is straightforward and endlessly repeatable.

2. **Structured events.** In ASIST, communicative events that shed light on individual cognitive states and team processes are of particular interest for downstream components. These events generally have a complex structure with one or more arguments. Our rule-based system allows an unlimited number of arguments for events and allows events to become arguments of other events. This leads to highly nested, but still interpretable structures. While complex structures are certainly possible with ML approaches, they are not what is supported by zero-shot approaches and we simply do not have annotated datasets of the necessary size to train such a model.

3. **Transparency.** A rule-based approach allows our system's decisions to be immediately inspectable, which helps with maintainability. The transparency of rules also makes it easier for us to inject domain knowledge into the system.[4] ML based approaches can provide attention weights, but these are not necessarily an interpretation of why the system made the prediction it did (Jain and Wallace, 2019), and regardless, they are not straightforwardly actionable.

3 Related Work

Rule-based approaches to event extraction (EE) have a long tradition in previous work (e.g., Appelt and Onyshkevych, 1998; Cunningham et al., 2002; Levy and Andrew, 2006; Hunter et al., 2008; Valenzuela-Escárcega et al., 2018; Sharp et al., 2019). These rules were written over a variety of language forms, from surface to syntactic or semantic structures. Here we use rules that combine surface and syntactic forms (Valenzuela-Escárcega et al., 2015), to allow for richer representations while also mitigating the effect of parsing errors resulting from imperfect transcriptions from the automated speech recognition system.

That said, much of the recent literature is on machine learning approaches to EE (Nguyen et al., 2016b; Liu et al., 2018; Sha et al., 2018; Wadden et al., 2019; Du and Cardie, 2020b; Nguyen and Nguyen, 2019; Xiang and Wang, 2019, *inter alia*).

[3] Due to some issues we found with our annotations described in § 5.1, we will release our annotated data set in its original form and in its corrected version.

[4] For example, due to the nature of the mission, we know that when players interact with unnamed entities, those entities can only be victims. So a statement such as "*I will go get the guy in A3*", must be about saving a victim. We can easily bake this knowledge into our rules.

Much of this work, however, relies on supervision, a luxury we do not have in this setting.

With the rise of large pretrained neural models, there are now several approaches to common natural language processing tasks that do not train or fine-tune a model, but instead use *prompting* to glean desired information from the knowledge already contained in the model (e.g., Wei et al., 2021; Liu et al., 2021; Min et al., 2021). This category of approach is desirable in scenarios such as ours, where annotated data is unavailable and would be difficult to produce at scale. For this reason, we compare our rule-based approach to a zero-shot approach based on prompting. In § 7, we show that for our scenario, the rule-based approach performed better and was able to produce richer representations.

4 Approach

Our rule-based EE system uses the Odin (Valenzuela-Escárcega et al., 2016) event extraction framework, which consists of an expressive, declarative rule language and a runtime system for applying the rules. More specifically, we have an Odin rule grammar that extracts two broad types of events from natural language: (i) *simple events* that do not have arguments,[5] and (ii) *complex events*, that can take other events as arguments.

Each event is associated with a unique span of text and is assigned a label by the rule that extracts it. The event labels we are using are organized into a hierarchical ontology. If a rule assigns a label to an event from this ontology, the parents of that label in the ontology are also assigned as labels for that event.[6]

Our system currently contains 420 active rules. These map to a total of 238 event labels, including both parent labels as well as the labels for the events we intentionally target.[7] For example, if our system detects a event label for a specific room on the map (example: "A4"), it will output that specific event label and all its parent labels like so: "Concept > EventLike > Location > Infrastructure > Room > A4".[8] This allows us to look at outputs at any level of granularity and define event arguments at any level of granularity. That is, we can define an event that only requires an "A4" label as a possible argument, or else we can define an event that takes a "Location" event label as a possible argument.

Odin's support for nested patterns allows the user to implement recursive passes on the data, as well as specifying at which pass the system should match against which specific rules. For our purposes, we use this to first look for simple events only, and then on later passes we look for complex events. The passes are *recursive* because prior extractions are part of the input for later passes. For example, a "Room" event that was extracted in pass 3 can be an argument of a "Search" event in pass 6, which can in turn be an argument of an "Instruction" event in pass 12. Figure 1 shows a visual representation of a sample utterance and the events extracted from it.

The code, rules, and documentation for our approach are publicly available on our github repository.[9]

4.1 The Label Ontology and Nested Events

Our event labels we are using are organized in a hierarchical ontology. We present a sample of this ontology in the appendix, page 13. This allows us to access labels at different levels of granularity. For example, in this domain there are different subtypes of victim entities – thus, we have different rules and labels for each sub-type. All sub-types of the victim label are hierarchically organized within a general *Victim* label. When we write event rules that need to take a *Victim*-type label as an argument (for example, the *Save* event), we can simply specify the higher ranking general *Victim* label as a possible argument, which will automatically target all subordinated labels as well. If we need to specify an event that only targets a very specific kind of victim, then that is equally possible. The nested ontology allows us to generalize over events while still keeping as much granularity as we want.

Our system does not distinguish between entities and events; it treats every label as an event. Events take other events as arguments (if so specified), leading to nested event structures. This allows us to generate complex and informative label

[5] Simple events are mostly entities, but they can also be actions or events without arguments. For this reason we prefer to call them events, as opposed to entities.

[6] See § 4.1 for details.

[7] Due to the hierarchical nature of the ontology, some event labels are never exported by a rule directly, they exist only as parents for grouping. These parent event labels can still serve as arguments of other events.

[8] Note that we do not use the term *Event* in a strict way, locations are not events in a classic or true sense, but as in our implementation they can possibly have "arguments" (i.e., relative positions, etc.), we consider them *EventLike*.

[9] Please note that v4.1.5 is the version that used for this paper. This version can be found here: https://github.com/clulab/tomcat-text/tree/v4.1.5

structures. In Figure 1, there is a *DeliberatePlan* label that takes a *MoveTo* label as a topic argument. The *MoveTo* label itself takes a *Deictic* label as a target argument. In the exported JSON, the *DeliberatePlan* label will contain this entire hierarchical structure and all superordinate labels. We present a full JSON output for the above example in the appendix, page 12.

Neural event extraction often uses a named entity recognition (NER) model to find all entities that can serve as arguments for events. While we consider all labels as "events" for practical purposes, we do have classes of labels that fall under "entities" in the ontology. However, for the actual implementation, they operate the same way as "events" do in our system. Since our events can take other events as arguments (and are not limited to entities), we can generate more informative relationships between events, and are not limited to entities as possible arguments.

4.2 Rule Writing for Dynamic Domains

The experimental setup of ASIST changes with each new experiment. These changes can be small, such as new types of player interactions, or they can entail changing the entire base task that players perform itself. While we know ahead of time that changes will happen, we can never completely predict how players will communicate under the changed environment. This has lead us to adopt a 2-stage style of rule writing:

1. **Predictive Rule Writing**: In this stage we know what the domain will be, but we do not have any actual data yet. We write rules that aim to predict *how* players will talk about certain events.

2. **Subject Data Informed Rule Writing**: Once we receive pilot data, we evaluate our predictive rules on that data and make changes to them accordingly to prepare a improved, frozen version of our system that is deployed for the actual data collection.

This approach has helped us to manage rapid adaption, while retaining a high standard of results. For an example of one of our rules please refer to the Example Rule with Commentary in the appendix.

5 Data

In order to quantitatively evaluate our approach and highlight shortcomings, we annotated an evaluation dataset[10] of in-mission dialog transcriptions for several key team coordination-related events. The dynamic nature of the experiments in ASIST means that annotations for ASIST Study 3 cannot be used as training data for ASIST Study 4 and ASIST Study 5. For this reason we (a) only annotated enough for evaluating our approaches (rather than training), and (b) only annotated for key events, which were more likely to remain central, even as the experimental paradigm shifts.

Our annotated data is drawn from live mission dialog and constitutes a subset of the ASIST study 3 dataset (Huang et al., 2022b). The dialog is transcribed in real-time using the Google Cloud Speech automatic speech recognition (ASR) system. We do not manually correct the transcription errors, as we are interested in exploring what can be achieved by our EE system in a realistic, live scenario. Due to the nature of its origin, the utterances are often messy and grammatically incorrect. Further, they contain filler phrases, repetitions, and interruptions. For example:

> *okay okay yeah so many patients I need picked up was marker with the SOS*

As mentioned above, we chose a set of 20 event labels to evaluate (see appendix for detailed explanations), consisting of labels we considered most important or interesting to us in the context of the ASIST program, and thus more likely to stay relevant. Specifically, we selected:

- 'Superset' events that account for multiple types of events. For example, a "Plan" event subsumes different kinds of utterances that are indicative of planning activity (see appendix for details).

- Events highly specific to our use case, such as "RescueInteractions".[11]

- Events where we were unsure of the specific label's performance (recall that the primary purpose of the data is to evaluate the validity of our approach).

[10] The dataset is publicly available at https://osf.io/6hr8t/.

[11] This event type subsumes different actions that players can do with/to victims.

Figure 1: Visualization of extractions produced by our rule-based system. Note that some events are simple events without arguments, while others are complex, such as the "Agree" and " Sight" events respecitvely. Additionally, events can nest, as with the "DeliberatePlan" event, which takes the "MoveTo" event as an argument. Finally, note that the utterances that serve as input are often not grammatically well-formed.

Of our 20 labels, nine are simple events (entities) and eleven represent complex events.

We tasked annotators with annotating 3686 utterances of game dialog for the labels in our set. Specifically, we asked that for each event, they (a) mark the span of all arguments, and (b) label the event as a whole. We did not ask them to indicate the labels of arguments themselves.

Concurrently, we also annotated a subset of the same data internally for precision alone[12]. This precision-based evaluation was done to gain more fine-grained insight into the arguments selected by our event labels.

5.1 Annotation Issues and Lessons Learned

We provided a manual for our two annotators with descriptions of each event type and some typical examples of how they show up in the natural language utterances. In an initial 90-minute training session, we discussed each event and walked annotators through some examples. Subsequently, we let them annotate a test set which we used to calculate inter-annotator agreement (Cohen's kappa = 0.7451).

After annotations were complete, it was evident that our system performed significantly worse when evaluated against the annotated set than it had in internal evaluations conducted for ASIST Study 1 and ASIST Study 2. We quickly realized that the annotators had slightly different conceptualizations of our events than we thought or intended. As a result, they were essentially annotating for a different task than the one we originally planned.

This was caused by a few mistakes on our part. We underestimated the degree to which the annotators would need to be intimately familiar with the domain. Their relative unfamiliarity with the domain caused them to miss instances of events that they would otherwise have identified. Another issue we observed was that annotators only annotated certain action-related events if it could be inferred that the player actually performed that action. For example, compare the following utterances:

1. *I am going to A3*
2. *We should go to A3 next*

Annotators tended to give the first utterance a "Movement" label, but not the second. However, in the context of ASIST, we want to apply a "Movement" label to both, as we would like our AI agent to be able to better predict future actions, not simply identify current and past ones.

In order to address this issue, we manually corrected all instances of disagreement between our system and the annotators. After removing same-utterance duplicates, we had 3920 annotated labels. Of those, there were 2817 disagreements with our system.[13] After manual corrections, we found 1303 disagreements remaining.[14] We would like to remark that we did not make any changes of our codebase during this process.

This method of correction is not ideal, as it risks instilling bias. If only disagreements are examined, then we could incorrectly bake in false negatives that our system and the annotators both missed. The same is potentially true for false positives from both (though due to the general nature of the issues, this was less of a concern). To check on this clear risk for bias, we subsampled 20 instances of agreement between our annotators and our system for each of our 20 event labels[15] and checked those samples for any inaccuracies. We found that in our set of 20*20 decisions, only two were faulty (one "Precedence" and one "Instruction"). Finally, we also checked 50 utterances where neither annotators nor our system had assigned any event labels. We found that in 50 utterances, there were

[12]This involved running our system on the data and manually annotating the output and its argument structure as either correct or incorrect.

[13]These were either cases where our system assigned a label and the annotators did not, or the inverse case.

[14]This means that annotators were correct in their disagreement a little under 50% of the time.

[15]Two event labels had fewer than 20 instances of agreement: "Search" and "RescueInteractions".

again only two instances where both annotators and our system had missed a "CriticalVictim" label. No other issues were found. Given this, we use the corrected annotations to evaluate both the zero-shot baseline as well as the rule-based system. We found that the corrected annotations improved the performance of *both our system and the zero-shot baseline*.

As a takeaway, we recommend creating internal test annotations to compare against annotator work at the beginning of the annotation process. By doing a mix of qualitative comparison and calculating inter-annotator agreement between the internal annotations and annotator work, this kind of situation can be avoided.

6 Evaluation

We evaluate our rule-based approach at two different levels of granularity: coarse and fine-grained. For the former, we compare the performance of our approach to that of a zero-shot classifier (see § 6.1).

For the coarse-grained evaluation, we evaluated whether the correct labels were assigned to utterances, without checking for argument structure or spans. A given utterance may contain multiple events with the same label.[16] For simplicity in this evaluation, and to streamline comparison with the zero-shot approach, here we evaluate extractions on a presence-only basis. That is, if the annotator assigned one "Victim" label to an utterance, and our EE system assigned two "Victim" labels to the same utterance, we consider this correct for the purposes of evaluation.

For the fine-grained evaluation, we annotated a subset of our data for the precision of the argument structure of our outputs. We only consider outputs that we had already annotated as correct. If all their arguments match for the correct event, we consider the extraction as a true positive.[17]

During our evaluation process, we considered the code-base "frozen". We did not make any adjustments to our system based on insights from the annotation or evaluation process until the evaluation was complete and all data was gathered. In contrast, we continually improved the zero-shot baseline approach during the evaluation process.

6.1 Zero-Shot Classification Baseline

For our zero-shot baseline we leverage the bart-large-mnli[18] checkpoint provided by Meta on Huggingface (Wolf et al., 2020). This zero-shot text classifier can take a label set and text as input and will return probability scores for the labels passed.

The model is based on a textual entailment framework which can work without annotated data of seen labels (Yin et al., 2019), an approach that has been adopted by previous work (such as Ye et al., 2020; Sainz and Rigau, 2021; Sun et al., 2021).

To recast our EE task as text classification, we provided the utterance as the text to be classified and the event labels as the classes. The labels provided were slightly adjusted to make them more amenable to the natural language expections of the approach (e.g., we changed "Move" to "movement" and "KnowledgeSharing" to "inform"). While we feel that the labels given to the zero-shot classifier could be further improved, we consider this to be beyond the scope of this work.

Since we require the classifier to be able to predict more than one event at a time, we cannot simply take an *argmax*. In an effort to more fairly compare the zero-shot model with our rule-based approach, we sampled different approaches for cut-off points of the label probability scores to serve as thresholds for extracting the events. Unfortunately, all methods sampled yielded overall micro F1 scores of < 0.1 (< 0.2 for macro F1) for the classifier.

While we believe it is likely that with further optimization we could have improved the performance of the zero-shot baseline, we hypothesize that the improvement would be limited. The data we are processing is real spoken language produced during times of stress and high focus for the participants. It is not comparable to the type of text-based data most modern large transformers are trained on.

However, we also implemented an oracle approach for the classifier that yielded much stronger performance. In our oracle approach, for each utterance we select the top n outputs of the zero-shot classifier as the given output, where n is the num-

[16] For example, a player might use the term "victim" multiple times in a single utterance.

[17] An implementation detail is that our system assigns a "GenericAction" label to events that it does not recognize. These event labels will only be exported by the system if they become an argument of a later event. When creating the dataset, if the arguments contained "GenericAction" events, we did not consider the event for annotation.

[18] https://huggingface.co/facebook/bart-large-mnli

		Precision		Recall		F1		
	Event Label	Rule-based	Zero-shot	Rule-based	Zero-shot	Rule-based	Zero-shot	Support
Simple	CriticalVictim	**0.959**	0.751	**0.729**	0.423	**0.828**	0.541	350
	Victim	**0.870**	0.691	**0.804**	0.561	**0.836**	0.619	342
	Room	**0.997**	0.572	**0.939**	0.286	**0.968**	0.381	809
	Engineer	**1.000**	0.957	**0.997**	0.609	**0.998**	0.744	294
	Transporter	**1.000**	0.303	**0.986**	0.888	**0.993**	0.451	277
	Medic	**1.000**	0.696	**1.000**	0.567	**1.000**	0.625	210
	Rubble	**1.000**	0.950	**0.986**	0.551	**0.993**	0.697	69
	MarkerBlock	**1.000**	0.272	**0.833**	0.708	**0.909**	0.393	48
	Meeting	**0.978**	0.279	**1.000**	0.856	**0.989**	0.421	90
	All simple (weighted av.)	**0.975**	0.634	**0.910**	0.508	**0.940**	0.518	2489
Complex	Move	**0.912**	0.254	**0.804**	0.595	**0.855**	0.356	296
	Precedence	**0.745**	0.112	**0.976**	0.159	**0.845**	0.132	126
	RescueInteractions	**0.792**	0.091	**0.528**	0.222	**0.633**	0.129	36
	KnowledgeSharing	**0.948**	0.246	**0.704**	0.111	**0.808**	0.154	314
	ReportLocation	**0.849**	0.127	**0.745**	0.236	**0.794**	0.165	106
	Search**	**0.818**	0.064	–	0.156	–	0.091	45
	HelpRequest	**0.805**	0.098	**0.713**	0.057	**0.756**	0.072	87
	Question	**0.705**	0.308	**0.298**	0.038	**0.419**	0.068	104
	YesNoQuestion	**0.827**	0.248	**0.684**	0.120	**0.749**	0.161	209
	Instruction	**0.700**	0.133	**0.531**	0.018	**0.604**	0.031	224
	Plan	**0.814**	0.645	**0.855**	0.045	**0.834**	0.084	447
	All complex (weighted av.)	**0.829**	0.299	**0.733**	0.165	**0.770**	0.144	1994
	All events (weighted av.)	**0.844**	0.528	**0.829**	0.472	**0.840**	0.450	4483

Table 1: Comparison of our rule-based EE system and the zero-shot baseline system for our coarse-grained evaluation. The zero-shot scores are all for the oracle setting, where we assume knowledge of the gold number of unique events in the utterance. The rule-based system does not use this oracle knowledge.

Event Label	Precision	Support
Move	0.942	120
Precedence	0.978	46
RescueInteractions	1.0	21
KnowledgeSharing	0.981	159
ReportLocation	1.0	57
Search	1.0	14
HelpRequest	0.961	26
Question	0.923	52
YesNoQuestion	1.0	21
Instruction	0.917	12
Plan	0.974	155
Weighted average	0.969	683

Table 2: Weighted average of argument structure precision scores for our rule-based approach (for events with arguments).

ber of unique gold labels for the utterance. While this method is not realistic, as it requires a-priori knowledge of the number of gold labels, it allowed us to generate a stronger baseline for comparison. Note we did *not* provide this knowledge to our rule-based approach.

7 Results

Table 1 shows the results of our evaluation. Our rule-based system achieves micro F1 scores of 0.940 and 0.770 for simple and complex events respectively.

The oracle zero-shot classifier baseline shows a micro F1 score of 0.518 for simple events and 0.144 for complex events. Our system outperforms the baseline on every event label, with the gap being particularly pronounced for domain-specific complex events such as "RescueInteractions".

In our system, both simple and complex events display consistently higher precision than recall, with "Precedence"[19] being the only exception. This outcome is expected; we designed the rules to favor precision over recall because the outputs of the EE system form inputs for further downstream tasks.

Table 2 shows the results for the precision-only fine-grained argument structure evaluation. Those events that can take arguments score a weighted average of 0.969 for precision of their argument structure.

[19]"Search" also displays higher recall than precision, but this is due to annotators not assigning any Search labels at all. We removed the recall score for the Search label from the table to reflect this fact.

8 Conclusion

Here we presented a case study on extracting team coordination events from natural language dialog. This dialog consists of live communications, which suffer from mistranscriptions, gap fillers, and interrupted or truncated utterances. While supervised neural approaches make up much of the recent literature for event extraction, these approaches become infeasible for this task due to the lack of training data. Further, the rapidly changing requirements for what should be extracted, as well as the need to embed domain-specific knowledge, led us to implement a rule-based approach.

Recently, large-scale pretrained language models have made zero shot and/or prompting-based approaches a convenient go-to for strong baselines, as these approaches make knowledge gained through large-scale pretraining accessible to various tasks. For this reason, we compare our rule-based method to a zero-shot classifier approach. We showed that in our case, our rule-based method out-performed the classifier for all event types we considered. Furthermore, the events output by our rules contain rich structure that can be used by downstream components for inference.

References

Wasi Uddin Ahmad, Nanyun Peng, and Kai-Wei Chang. 2021. GATE: Graph Attention Transformer Encoder for cross-lingual relation and event extraction. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021*, pages 12462–12470. AAAI Press.

Douglas E Appelt and Boyan Onyshkevych. 1998. The common pattern specification language. In *Proc. of the TIPSTER Workshop*, pages 23–30.

Laura Chiticariu, Yunyao Li, and Frederick R. Reiss. 2013. Rule-based information extraction is dead! Long live rule-based information extraction systems! In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 827–832, Seattle, Washington, USA. Association for Computational Linguistics.

Hamish Cunningham, Diana Maynard, Kalina Bontcheva, and Valentin Tablan. 2002. Gate: an architecture for development of robust hlt applications. In *Proceedings of the 40th annual meeting on association for computational linguistics*, pages 168–175. Association for Computational Linguistics.

Xinya Du and Claire Cardie. 2020a. Event extraction by answering (almost) natural questions. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing, EMNLP 2020, Online, November 16-20, 2020*, pages 671–683. Association for Computational Linguistics.

Xinya Du and Claire Cardie. 2020b. Event extraction by answering (almost) natural questions. *arXiv preprint arXiv:2004.13625*.

Lixiao Huang, Jared Freeman, Nancy Cooke, John Colonna-Romano, Matthew D Wood, Verica Buchanan, and Stephen J Caufman. 2022a. Exercises for artificial social intelligence in Minecraft search and rescue for teams.

Lixiao Huang, Jared Freeman, Nancy Cooke, John "JCR" Colonna-Romano, Matt Wood, Verica Buchanan, and Stephen Caufman. 2022b. Artificial Social Intelligence for Successful Teams (ASIST) Study 3.

Lawrence Hunter, Zhiyong Lu, James Firby, William A Baumgartner, Helen L Johnson, Philip V Ogren, and K Bretonnel Cohen. 2008. Opendmap: an open source, ontology-driven concept analysis engine, with applications to capturing knowledge regarding protein transport, protein interactions and cell-type-specific gene expression. *BMC bioinformatics*, 9(1):78.

Sarthak Jain and Byron C. Wallace. 2019. Attention is not Explanation. 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)*, pages 3543–3556, Minneapolis, Minnesota. Association for Computational Linguistics.

Roger Levy and Galen Andrew. 2006. Tregex and Tsurgeon: tools for querying and manipulating tree data structures. In *Proc. of LREC*.

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*.

Xiao Liu, Zhunchen Luo, and Heyan Huang. 2018. Jointly multiple events extraction via attention-based graph information aggregation. *arXiv preprint arXiv:1809.09078*.

Bonan Min, Hayley Ross, Elior Sulem, Amir Pouran Ben Veyseh, Thien Huu Nguyen, Oscar Sainz, Eneko Agirre, Ilana Heinz, and Dan Roth. 2021. Recent advances in natural language processing via large pre-trained language models: A survey. *arXiv preprint arXiv:2111.01243*.

Thien Huu Nguyen, Kyunghyun Cho, and Ralph Grishman. 2016a. Joint event extraction via recurrent neural networks. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 300–309, San Diego, California. Association for Computational Linguistics.

Thien Huu Nguyen, Kyunghyun Cho, and Ralph Grishman. 2016b. Joint event extraction via recurrent neural networks. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 300–309.

Trung Minh Nguyen and Thien Huu Nguyen. 2019. One for all: Neural joint modeling of entities and events. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pages 6851–6858.

Adarsh Pyarelal. 2022. Tomcat (uaz + tamu) study 3 preregistration.

Oscar Sainz and German Rigau. 2021. Ask2transformers: Zero-shot domain labelling with pre-trained language models. *arXiv preprint arXiv:2101.02661*.

Lei Sha, Feng Qian, Baobao Chang, and Zhifang Sui. 2018. Jointly extracting event triggers and arguments by dependency-bridge rnn and tensor-based argument interaction. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32.

Rebecca Sharp, Adarsh Pyarelal, Benjamin Gyori, Keith Alcock, Egoitz Laparra, Marco A. Valenzuela-Escárcega, Ajay Nagesh, Vikas Yadav, John Bachman, Zheng Tang, Heather Lent, Fan Luo, Mithun Paul, Steven Bethard, Kobus Barnard, Clayton Morrison, and Mihai Surdeanu. 2019. Eidos, INDRA, & Delphi: From free text to executable causal models. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics (Demonstrations)*, pages 42–47, Minneapolis, Minnesota. Association for Computational Linguistics.

Yi Sun, Yu Zheng, Chao Hao, and Hangping Qiu. 2021. Nsp-bert: A prompt-based zero-shot learner through an original pre-training task–next sentence prediction. *arXiv preprint arXiv:2109.03564*.

Alex Tozzo, Dejan Jovanović, and Mohamed Amer. 2018. Neural event extraction from movies description. In *Proceedings of the First Workshop on Storytelling*, pages 60–66, New Orleans, Louisiana. Association for Computational Linguistics.

Marco A Valenzuela-Escárcega, Özgün Babur, Gus Hahn-Powell, Dane Bell, Thomas Hicks, Enrique Noriega-Atala, Xia Wang, Mihai Surdeanu, Emek Demir, and Clayton T Morrison. 2018. Large-scale automated machine reading discovers new cancer-driving mechanisms. *Database*, 2018.

Marco A. Valenzuela-Escárcega, Gus Hahn-Powell, and Mihai Surdeanu. 2016. Odin's runes: A rule language for information extraction. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16)*, pages 322–329,

Portorož, Slovenia. European Language Resources Association (ELRA).

Marco A. Valenzuela-Escárcega, Gus Hahn-Powell, Mihai Surdeanu, and Thomas Hicks. 2015. A domain-independent rule-based framework for event extraction. In *Proceedings of ACL-IJCNLP 2015 System Demonstrations*, pages 127–132, Beijing, China. Association for Computational Linguistics and The Asian Federation of Natural Language Processing.

David Wadden, Ulme Wennberg, Yi Luan, and Hannaneh Hajishirzi. 2019. Entity, relation, and event extraction with contextualized span representations. *arXiv preprint arXiv:1909.03546*.

Jason Wei, Maarten Bosma, Vincent Y Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M Dai, and Quoc V Le. 2021. Finetuned language models are zero-shot learners. *arXiv preprint arXiv:2109.01652*.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander Rush. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online. Association for Computational Linguistics.

Wei Xiang and Bang Wang. 2019. A survey of event extraction from text. *IEEE Access*, 7:173111–173137.

Zhiquan Ye, Yuxia Geng, Jiaoyan Chen, Jingmin Chen, Xiaoxiao Xu, Suhang Zheng, Feng Wang, Jun Zhang, and Huajun Chen. 2020. Zero-shot text classification via reinforced self-training. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 3014–3024.

Wenpeng Yin, Jamaal Hay, and Dan Roth. 2019. Benchmarking zero-shot text classification: Datasets, evaluation and entailment approach. *arXiv preprint arXiv:1909.00161*.

Appendix

List of event labels and their meanings

What follows is a list of event labels we used for the evaluation. We provide explanations for each event and examples, some from the data used for this paper.

Simple Events

1. **CriticalVictim**: ASIST mission participants can encounter regular victims and critical victims. Critical victims require the entire team to save them. Examples: “type C”, “critical guy”, “c type”

2. **Victim**: Regular victims can be saved by the team member with the medic role. Examples: “guy in C3”, “type a person”, “b type victim”

3. **Room**: This is a collection of different room type events. We have specific event labels for every room on the map of any given experiment. Examples: “A2”, “room”, “office”

4. **Engineer**: One of the possible roles a player can assume. The engineer can clear rubble that is blocking the players’ path. Examples: “rubble guy”, “engineer”, “shovel guy”

5. **Transporter**: Assuming the transporter role allows players to move faster and to transport regular victims. Examples: “transporter”, “transport specialist”, “scout”

6. **Medic**: The medic can triage victims. Triaging victims yields points for the team. Examples: “medic”, “medical specialist”, “healer”

7. **Rubble**: Players will encounter rubble blocking their path on their mission. Examples: “gravel”, “rubble”, “rebel” (common mistranscription), “blockage”

8. **MarkerBlock**: Participants can drop a series of different markers on the floor. These can have different meanings and are used to mark important rooms etc. This event label is a collection of all specific marker block event labels. Examples: “c victim marker”, “gravel marker”, “victim block”, “threat sign”

9. **Meeting**: One of the participants will have a list of meetings that were going on when the building collapsed and their location. This is a collection of event labels capturing the terms used for those meetings. Examples: “management meeting”, “lunch”

Complex Events

10. **Move**: A collection of different event labels capturing participants discussing movement of themselves and others. Examples: “I’m on my way”, “Can you come to A2?”, “entering c1”

11. **Precedence**: A collection of different event labels capturing participants discussing temporal precedence. The event arguments aim to sort the two actions as an initial and subsequent action. Examples: “same thing M1 M1 M3 and then we can go i4a and then I2”, “after I4 let’s go to j4”

12. **RescueInteractions**: Participants can triage, wake up, and stabilize victims. This is a collection of labels capturing all these cases. Examples: “heading back to get all the victims”, “yeah I can get this one let’s wake up the there he’s awake”

13. **KnowledgeSharing**: This event captures participants relaying information about entities that exist around them. Examples: “there’s loads of victims in here”, “there’s a critical condition in the back”

14. **ReportLocation**: This event captures participants reporting their own location or the location of other entities. It also captures participants relaying certain information about the location.[20] Examples: “yeah I’m right I’m right by C6”, “M3 is a trap room”

15. **Search**: This event captures participants talking about searching a room for victims. Examples: “we have searched F4 F4 is lunch this is medic”, “have you checked a4a”

16. **HelpRequest**: Participants requesting help. This is a collection of different event labels which can be summarized under this umbrella. Examples: “if someone could help me I am trapped in j4”, “and transporter if you can also assist in j4”

17. **Question**: Participants asking content questions. Examples: “this is the engineer how do you guys know whether they goes north or south”

18. **YesNoQuestion**: Participants asking binary questions. Examples: “do we clear this out”

19. **Instruction**: Participants giving instructions. Examples: “go to the middle section there’s about 45 criticals so we know there’s points there”

20. **Plan**: Participants engaging in planning. Examples: “okay if we want to start on i4a just to get some people moving”, “okay I’ll take care of the B type”

[20] This is handled by different event labels which are summarized under this event label for the purpose of the evaluation.

Listing 1: Example JSON output for "I'll head there first oh jeez what."


```
{
  "participant_id": "----",
  "asr_msg_id": "----",
  "text": "I'll head there first oh
      jeez what.",
  "utterance_source": {
    "source_type": "message_bus",
    "source_name": "agent/asr/final"
  },
  "extractions": [
  {
    "labels": [
    "DeliberatePlan",
    "Commitment",
    "Plan",
    "Communicate",
    "SimpleAction",
    "Action",
    "EventLike",
    "Concept"
    ],
    "span": "will head there",
    "arguments": {
      "topic": [
      {
        "labels": [
        "MoveTo",
        "Move",
        "SimpleAction",
        "Action",
        "EventLike",
        "Concept"
        ],
        "span": "head there",
        "arguments": {
          "target": [
          {
            "labels": [
            "Deictic",
            "Inferred",
            "Location",
            "EventLike",
            "Concept"
            ],
            "span": "there",
            "arguments": {},
            "attachments": [],
            "start_offset": 10,
            "end_offset": 15,
            "rule": "deictic_detection"
          }
          ]
        },
        "attachments": [
        {
          "text": "I",
          "agentType": "Self",
          "labels": [
          "Self",
          "Entity",
          "Concept"
          ],
          "span": [
          0
          ]
        },
        {
          "value": "future"
        }
        ],
        "start_offset": 5,
        "end_offset": 15,
        "rule": "move_deixis_action"
      }
      ]
    },
    "attachments": [
    {
      "text": "I",
      "agentType": "Self",
      "labels": [
      "Self",
      "Entity",
      "Concept"
      ],
      "span": [
      0
      ]
    },
    {
      "value": "future"
    }
    ],
    "start_offset": 1,
    "end_offset": 15,
    "rule":
        "commit_to_something_plan-type"
  },
  {
    "labels": [
    "MoveTo",
    "Move",
    "SimpleAction",
    "Action",
    "EventLike",
    "Concept"
    ],
    "span": "head there",
    "arguments": {
      "target": [
      {
        "labels": [
        "Deictic",
        "Inferred",
        "Location",
        "EventLike",
        "Concept"
        ],
        "span": "there",
        "arguments": {},
        "attachments": [],
        "start_offset": 10,
        "end_offset": 15,
        "rule": "deictic_detection"
      }
      ]
    },
    "attachments": [
    {
      "text": "I",
      "agentType": "Self",
      "labels": [
      "Self",
      "Entity",
      "Concept"
      ],
      "span": [
```

```
        0
        ]
      },
      {
        "value": "future"
      }
      ],
      "start_offset": 5,
      "end_offset": 15,
      "rule": "move_deixis_action"
    }
    ]
}
```

Listing 2: Sample of a subsection of the Label Ontology

```
- Communicate:
  - Instruction:
    - HelpCommand # for players instructing others to help, f.e.: "Help the engineer!"
  - ReportStatus:
    - Stuck # for players stuck in a room (for whichever reason), has some overlap with AmTrapped
    - ReportLocation # players reporting on their, or other players locations: "I'm in B2"
    - RoleDeclare # players declaring their role to the team
    - RoomStatus:
      - RoomClear # "A1 is clear"
      - ReportThreatRoom # players declaring rooms as threatrooms: "A1 is a threat room"
  - KnowledgeSharing # players reporting that something exists: "There is a critical victim here" or "I have some rubble in B2"
  - Need: # labels for players discussing the needs of the team or their own needs
    - NeedRole
    - NeedItem
    - NeedAction
```

Example Rule with Commentary

Listing 3: Example Rule

```
- name: "Help_command"
  label: HelpCommand
  example: "You should help him."
  priority: ${ rulepriority }
  pattern: |
  trigger =
      [lemma=/assist|help|aid|support/]
  agent: Entity = >nsubj
      [!mention=Self] |<xcomp >nsubj
      [!mention=Self]
  helpee: Entity = >ccomp >nsubj
      [!mention=You] | >ccomp
      [!mention=You] | >dep
      [!mention=You] | >dobj
      [!mention=You]
  location: Location? = >/${preps}/|
    >/advmod/
```

- `name`: Every Odin rule requires a unique name.
- `label`: This field defines the actual label that this rule will export. The label's place in the ontology is defined elsewhere

- `example`: We try and provide an example with each rule, for easier development.
- `priority`: This field defines at which iteration the rule will be applied. In this case this is defined via a vaiable.
- `pattern`: The pattern field defines the trigger and its arguments.
- `trigger`: Rules with an argument need a trigger field. If the trigger is found, the rule starts searching for the arguments. In this case, the trigger is defined as series of possible lemmas.
- `agent: Entity` – This field states that this rule takes an "agent" argument which as to carry the label "Entity" (or any of the subordinate labels of entity)
- `>nsubj [!mention=Self] |<xcomp >nsubj [!mention=Self]:` These statements define the *dependency relationship* that the agent argument needs to have with respect to the trigger. In plain English, this statement requires: An outgoing "nsubj" dependency, at the end of which there may **not** be an event label *You*, or an outgoing "ccomp" dependency, at the end of which there may **not** be an event label *You*, and so on.