Easy, Reproducible and Quality-Controlled Data Collection with CROWDAQ

Qiang Ning[♠] **Hao Wu**[♣] **Pradeep Dasigi**[♠] **Dheeru Dua**[♢] **Matt Gardner**[♠]
Robert L. Logan IV[♢] **Ana Marasović**[♠] **Zhen Nie**[♣]
[♠]Allen Institute for AI [♢]University of California, Irvine [♣]Hooray Data Co., Ltd
{qiangn,pradeepd,mattg,anam}@allenai.org
{haowu,zhennie}@hooray.ai
{ddua,rlogan}@uci.edu

Abstract

High-quality and large-scale data are key to success for AI systems. However, large-scale data annotation efforts are often confronted with a set of common challenges: (1) designing a user-friendly annotation interface; (2) training enough annotators efficiently; and (3) reproducibility. To address these problems, we introduce CROWDAQ,[1] an open-source platform that standardizes the data collection pipeline with customizable user-interface components, automated annotator qualification, and saved pipelines in a re-usable format. We show that CROWDAQ simplifies data annotation significantly on a diverse set of data collection use cases and we hope it will be a convenient tool for the community.

1 Introduction

Data is the foundation of training and evaluating AI systems. Efficient data collection is thus important for advancing research and building time-sensitive applications.[2] Data collection projects typically require many annotators working independently to achieve sufficient scale, either in dataset size or collection time. To work with multiple annotators, data requesters (i.e., AI researchers and engineers) usually need to design a user-friendly annotation interface and a quality control mechanism. However, this involves a lot of overhead: we often spend most of the time resolving frontend bugs and manually checking or communicating with individual annotators to filter out those who are unqualified, instead of focusing on core research questions.

Another issue that has recently gained more attention is reproducibility. Dodge et al. (2019) and Pineau (2020) provide suggestions for *system* reproducibility, and Bender and Friedman (2018) and Gebru et al. (2018) propose "data statements" and "datasheets for datasets" for *data collection* reproducibility. However, due to irreproducible human interventions in training and selecting annotators and the potential difficulty in replicating the annotation interfaces, it is often difficult to reuse or extend an existing data collection project.

We introduce CROWDAQ, an open-source data annotation platform for NLP research designed to minimize overhead and improve reproducibility. It has the following contributions. First, CROWDAQ standardizes the design of data collection pipelines, and separates that from software implementation. This standardization allows requesters to design data collection pipelines *declaratively* without being worried about many engineering details, which is key to solving the aforementioned problems (Sec. 2).

Second, CROWDAQ automates qualification control via multiple-choice exams. We also provide detailed reports on these exams so that requesters know how well annotators are doing and can adjust bad exam questions if needed (Sec. 2).

Third, CROWDAQ carefully defines a suite of pre-built UI components that one can use to compose complex annotation user-interfaces (UIs) for a wide variety of NLP tasks without expertise in HTML/CSS/JavaScript (Sec. 3). For non-experts on frontend design, CROWDAQ can greatly improve efficiency in developing these projects.

Fourth, a dataset collected via CROWDAQ can be more easily reproduced or extended by *future data requesters*, because they can simply copy the pipeline and pay for additional annotations, or treat existing pipeline as a starting point for new projects.

In addition, CROWDAQ has also integrated many useful features: requesters can conveniently monitor the progress of annotation jobs, whether they are paying annotators fairly, and the agreement

[1] Crowdsourcing with Automated Qualifcation; https://www.crowdaq.com/

[2] This holds not only for collecting static data annotations, but also for collecting human judgments of system outputs.

Proceedings of the 2020 EMNLP (Systems Demonstrations), pages 127–134

level of different annotators on CROWDAQ. Finally, Sec. 4 shows how to use CROWDAQ and Amazon Mechanical Turk (MTurk)[3] to collect data for an example project. More use cases can be found in our documentation.

2 Standardized Data Collection Pipeline

A data collection project with multiple annotators generally includes some or all of the following: (1) Task definition, which describes what should be annotated. (2) Examples, which enhances annotators' understanding of the task. (3) Qualification, which tests annotators' understanding of the task and only those qualified can continue; this step is very important for reducing unqualified annotators. (4) Main annotation process, where qualified annotators work on the task. CROWDAQ provides easy-to-use functionality for each of these components of the data collection pipeline, which we expand next.

INSTRUCTION A Markdown document that defines a task and instructs annotators how to complete the task. It supports various formatting options, including images and videos.

TUTORIAL Additional training material provided in the form of multiple-choice questions with provided answers that workers can use to gauge their understanding of the INSTRUCTION. CROWDAQ received many messages from real annotators saying that TUTORIALS are quite helpful for learning tasks.

EXAM A collection of multiple-choice questions similar to TUTORIAL, but for which answers are not provided to participants. EXAM is used to test whether an annotator understands the instructions sufficiently to provide useful annotations. Participants will only have a finite number of opportunities specified by the requesters to work on an EXAM, and each time they will see a random subset of all the exam questions. After finishing an EXAM, participants are informed of how many mistakes they have made and whether they have passed, but they do not receive feedback on individual questions. Therefore, data requesters should try to design better INSTRUCTIONS and TUTORIALS instead of using EXAM to teach annotators.

We restrict TUTORIALS and EXAMS to always be in a multiple-choice format, irrespective of the original task format, because it is natural for humans to learn and to be tested in a discriminative setting.[4] An important benefit of using multiple-choice questions is that their evaluation can be automated easily, minimizing the effort a requester spends on manual inspections. Another convenient feature of CROWDAQ is that it displays useful statistics to requesters, such as the distribution of scores in each exam and which questions annotators often make mistakes on, which can highlight areas of improvement in the INSTRUCTION and TUTORIAL. Below is the JSON syntax to specify TUTORIALS/EXAMS (see Fig. 3 and Fig. 4 in the appendix).

```
"question_set": [
   {
      "type": "multiple-choice",
      "question_id": ...,
      "context": [{
         "type": "text",
         "text": "As of Tuesday, 144 of the state's
              then-294 deaths involved nursing
              homes or longterm care facilities."
      }],
      "question": {
         "question_text": "In \"294 deaths\", what
              should you label as the quantity?",
         "options": {"A": "294", "B": "294 deaths"}
      },
      "answer": "A",
      "explanation": {
         "A": "Correct",
         "B": "In our definition, the quantity
              should be \"294\"."
      }
   },
   ...
]
```

TASK For example, if we are doing sentence-level sentiment analysis, then a TASK is to display a specific sentence and require the annotator to provide a label for its sentiment. A collection of TASKS are bundled into a TASK SET that we can launch as a group. Unlike TUTORIALS and EXAMS where we only need to handle multiple-choice questions in CROWDAQ's implementation, a major challenge for TASK is how to meet different requirements for annotation UI from different datasets in a single framework, which we discuss next.

3 Customizable Annotation Interface

It is time-consuming for non-experts on the frontend to design annotation UIs for various datasets. At present, requesters can only reuse the UIs of very similar tasks and still, they often need to make modifications with additional tests and debugging. CROWDAQ comes with a variety of built-in

[3] https://www.mturk.com/

[4] E.g., we can always test one's understanding of a concept by multiple-choice questions like *Do you think something is correct?* or *Choose the correct option(s) from below.*

resources for easily creating UIs, which we will explain using an example dataset collection project centered around confirmed COVID-19 cases and deaths mentioned in news snippets.

3.1 Concepts

The design of CROWDAQ's annotation UI is built on some key concepts. First, every TASK is associated with `contexts`—a list of objects of any type: `text`, `html`, `image`, `audio`, or `video`. It will be visible to the annotators during the entire annotation process before moving to the next TASK, so a requester can use `contexts` to show any useful information to the annotators. Below is an example of showing notes and a target news snippet (see Fig. 5 in the appendix for visualization). CROWDAQ is integrated with online editors that can auto-complete, give error messages, and quickly preview any changes.

```
"contexts": [
   {
      "label": "Note",
      "type": "html",
      "html": "<p>Remember to ...</p>",
      "id": "note"
   },
   {
      "type": "text",
      "label": "The snippet was from an article
           published on 2020-05-20 10:30:00",
      "text": "As of Tuesday, 144 of the state's
           then-294 deaths involved nursing homes
           or longterm care facilities.",
      "id": "snippet"
   }
],
```

Second, each TASK may have multiple `annotations`. Although the number of dataset formats can be arbitrary, we observe that the most basic formats fall into the following categories: multiple-choice, span selection, and free text generation. For instance, to emulate the data collection process used for the CoNLL-2003 shared task on named entity recognition (Tjong Kim Sang and De Meulder, 2003), one could use a combination of a span selection (for selecting a named entity) and a multiple-choice question (selecting whether it is a person, location, etc.); for the process used for natural language inference in SNLI (Bowman et al., 2015), one could use an input box (for writing a hypothesis) and a multiple-choice question (for selecting whether the hypothesis entails or contradicts the premise); for reading comprehension tasks in the question-answering (QA) format, one could use an input box (for writing a question) and a multiple-choice question (for yes/no answers; Clark et al. (2019)), a span selection (for span-based answers; Rajpurkar et al. (2016)), or another input box (for free text answers; Kočiskỳ et al. (2018)).

These `annotation` types are built in CROWDAQ,[5] which requesters can easily use to compose complex UIs. For our example project, we would like the annotator to select a quantity from the "snippet" object in the `contexts`, and then tell us whether it is relevant to COVID-19 (see below for how to build it and Fig. 6 in the appendix for visualization).

```
"annotations": [
   {
      "type": "span-from-text",
      "from_context": "snippet",
      "prompt": "Select one quantity from below.",
      "id": "quantity",
   },
   {
      "type": "multiple-choice",
      "prompt": "Is this quantity related to
           COVID-19?",
      "options":{
         "A": "Relevant",
         "B": "Not relevant"
      }
      "id": "relevance"
   }
]
```

Third, a collection of `annotations` can form an `annotation group` and a TASK can have multiple of them. For complex TASKS, this kind of semantic hierarchy can provide a big picture for both the requesters and annotators. We are also able to provide very useful features for `annotation groups`. For example, we can put the `annotations` object above into an `annotation group`, and require 1-3 responses in this group. Below is its syntax, and Fig. 7 in the appendix shows the result.

```
"annotation_groups": [
   {
      "annotations": [
         {"id": "quantity", ...},
         {"id": "relevance", ...}
      ],
      "id": "quantity_extraction_typing",
      "title": "COVID-19 Quantities",
      "repeated": true, "min": 1, "max": 3
   }
],
```

3.2 Conditions

Requesters often need to collect some `annotations` only when certain `conditions` are satisfied. For instance, only if a quantity is related to COVID-19 will we continue to ask the type of it. These `conditions` are important because by construction, annotators will not make mistakes such as answering a question that should not be enabled at all.

As a natural choice, CROWDAQ has implemented `conditions` that take as input val-

[5] For a complete list, please refer to our documentation.

ues of multiple-choice `annotations`. The field `conditions` can be applied to any `annotation`, which will be enabled only when the conditions are satisfied. Below we add a multiple-choice question asking for the type of a quantity *only if* the annotator has chosen option "A: Relevant" in the question whose ID is "relevance" (see Fig. 8 in the appendix).

```
"annotations": [
   {"id": "quantity", ...},
   {"id": "relevance", ...},
   {
      "id": "typing",
      "type": "multiple-choice",
      "prompt": "What type is it?",
      "options":{
         "A": "Number of Deaths",
         "B": "Number of confirmed cases",
         "C": "Number of hospitalized",
         ...
      },
      "conditions":[
         {
            "id": "relevance",
            "op": "eq",
            "value": "A"
         }
      ]
   }
],
```

CROWDAQ actually supports any boolean logic composed by "AND," "OR," and "NOT." Below is an example of $\neg(Q1 = A \vee Q2 = B)$.

```
"conditions": [
   {
     "op": "not", "arg": {
       "op": "or", "args":[
         {"id": "Q1","op": "eq","value": "A"},
         {"id": "Q2","op": "eq","value": "B"}
       ]
     }
   }
]
```

3.3 Constraints

An important quality control mechanism is to implement constraints for an annotator's work such that only if the constraints are satisfied will the annotator be able to submit the instance (and get paid). An implicit constraint in CROWDAQ is that all `annotations` should be finished except for those explicitly specified as "optional."

For things that are repeated, CROWDAQ allows the requester to specify the min/max number of repetitions. This corresponds to scenarios where, for instance, we know there is at least 1 quantity (min=1) in a news snippet or we want to have exactly two named entities selected for relation extraction (min=max=2). We have already shown usages of this when introducing `annotation group`, but the same also applies to text span selectors.

CROWDAQ also allows requesters to specify a regular expression constraint. For instance, in our COVID-19 example, when the annotator selects a text span as a quantity, we want to make sure that the span selection does not violate some obvious rules. To achieve this, we define `constraints` as a list of requirements and all of them must be satisfied; if any one of them is violated, the annotator will receive an error message specified by the `description` field and also not able to submit the work.

In addition, users can specify their own constraint functions via an API. Please refer to our documentation for more details.

```
"annotations":[
   {
      "id": "quantity",
      ...,
      "constraints": [
         {
            "description": "The quantity should
                 only start with digits or
                 letters.",
            "regex": "^[\\w\\d].*$",
            "type": "regex"
         },
         {
            "description": "The quantity should
                 only end with digits, letters, or
                 %.",
            "regex": "^.*[\\w\\d%]$",
            "type": "regex"
         },
         {
            "description": "The length of your
                 selection should be within 1 and
                 30.",
            "regex": "^.{1,30}$",
            "type": "regex"
         }
      ]
   },
   ...
]
```

3.4 Extensibility

As we cannot anticipate every possible UI requirement, we have designed CROWDAQ to be extensible. In addition to a suite of built-in `annotation` types, `conditions`, and `constraints`, users can write their own components and contribute to CROWDAQ easily. All these components are separate Vue.js[6] components and one only needs to follow some input/output specifications to extend CROWDAQ.

4 Usage

We have already deployed CROWDAQ at https://www.crowdaq.com with load balancing, backend cluster, relational database, failure recovery, and user authentication. Data requesters can simply register and enjoy the convenience it provides. For users who need to deploy CROWDAQ, we provide a Docker compose configuration so that they can bring up a cluster with all the features with one

[6] https://vuejs.org/

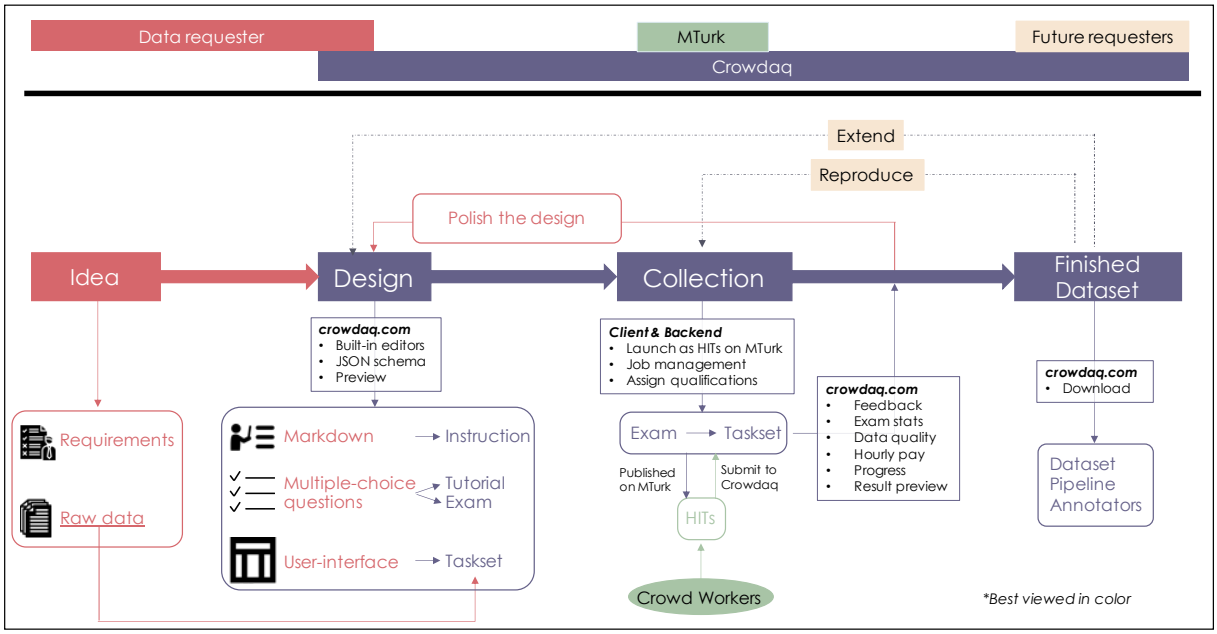

Figure 1: Data collection using CROWDAQ and MTurk. Note that this is a general workflow and one can use only part of it, or use it to build even more advanced workflows.

single command. Users will need to have their own domain name and HTTPS certificate in that case in order to use CROWDAQ with MTurk.

Figure 1 shows how a requester collects data using CROWDAQ and MTurk. The steps are: (1) identify the requirements of an application and find the raw data that one wants to annotate; (2) design the data collection pipeline using the built-in editors on CROWDAQ's website, including the Markdown INSTRUCTION, TUTORIAL, EXAM, and INTERFACE; (3) launch the EXAM and TASK SET onto MTurk and get crowd annotators to work on them; (4) if the quality and size of the data have reached one's requirement, publish the dataset. We have color-coded those components in Fig. 1 to show the responsibilities of the data requester, CROWDAQ, MTurk, and future requesters who want to reproduce or extend this dataset. We can see that CROWDAQ significantly reduces the effort a data requester needs to put in implementing all those features.

We have described how to write INSTRUCTIONS, TUTORIALS, and EXAMS (Sec. 2) and how to design the annotation UI (Sec. 3). Suppose we have provided 20 EXAM questions for the COVID-19 project. Before launching the EXAM, we need to configure the sample size of the EXAM, the passing score, and total number of chances (e.g., every time a participant will see a random subset of 10 questions, and to pass it, one must get a score higher than 80% within 3 chances). This can be done using the web interface of CROWDAQ (see Fig. 10 in the appendix).

It is also very easy to launch the EXAM to MTurk. CROWDAQ comes with a client package that one can run from a local computer (Fig. 11 in the appendix). The backend of CROWDAQ will do the job management, assign qualifications, and provide some handy analysis of how well participants are doing on the exam, including the score distribution of participants and analysis on each individual questions (Fig. 12).

The semantic difference between EXAMS and TASK SETS is handled by the backend of CROWDAQ. From MTurk's perspective, EXAMS and TASK SETS are both EXTERNALQUESTIONS.[7] Therefore, the same client package shown in Fig. 11 can also be used to launch a TASK SET to MTurk. CROWDAQ's backend will receive the annotations submitted by crowd workers; the website will show the annotation progress and average time spent on each TASK, and also provide quick preview of each individual annotations. If the data requester finds that the quality of annotations is not acceptable, the requester can go back and polish the design.

When data collection is finished, the requester can download the annotation pipeline and list of annotators from CROWDAQ, and get information about the process such as the average time spent

[7] EXTERNALQUESTION is a type of HITs on MTurk.

by workers on the task (and thus their average pay rate). Future requesters can then use the pipeline as the starting point for their projects, if desired, e.g., using the same EXAM, to get similarly-qualified workers on their follow-up project.

Although Fig. 1 shows a complete pipeline of using CROWDAQ and MTurk, CROWDAQ is implemented in such a way that data requesters have the flexibility to use only part of it. For instance, one can only use INSTRUCTION to host and render Markdown files, only use EXAM to test annotators, or only use TASK SET to quickly build annotation UIs. One can also create even more advanced workflows, e.g., using multiple EXAMS and filtering annotators sequentially (e.g., Gated Instruction; Liu et al., 2016), creating a second TASK SET to validate previous annotations, or splitting a single target dataset into multiple components, each of which has its own EXAM and TASK SET. In addition, data collection with in-house annotators can be done on CROWDAQ directly, instead of via MTurk. For instance, data requesters can conveniently create a contrast set (Gardner et al., 2020) on CROWDAQ by themselves.

We have put more use cases into the appendix, including DROP (Dua et al., 2019), MATRES (Ning et al., 2018), TORQUE (Ning et al., 2020), VQA-E (Li et al., 2018), and two ongoing projects.

5 Related Work

Crowdsourcing Platforms The most commonly used platform at present is MTurk, and the features CROWDAQ provides are overall complementary to it. CROWDAQ provides integration with MTurk, but it also allows for in-house annotators and any platform that provides crowdsourcing service. Other crowdsourcing platforms, e.g., CrowdFlower/FigureEight,[8] Hive,[9] and Labelbox,[10] also have automated qualification control as CROWDAQ, but they do not separate the format of an exam from the format of a main task; therefore it is impossible to use its built-in qualification control for non-multiple-choice tasks like question-answering. In addition, CROWDAQ provides huge flexibility in annotation UIs as compared to these platforms. Last but not least, CROWDAQ is open-source and can be used, contributed to, extended, and deployed freely.

Customizable UI To the best of our knowledge, existing works on customizable annotation UI, e.g., MMAX2[11] (Müller and Strube, 2006), PALinkA (Orăsan, 2003), and BRAT[12] (Stenetorp et al., 2012), were mainly designed for in-house annotators on classic NLP tasks, and their adaptability and extensibility are limited.

AMTI is a command line interface for interacting with MTurk,[13] while CROWDAQ is a website providing one-stop solution including instructions, qualification tests, customizable interfaces, and job management on MTurk. AMTI also addresses the reproducibility issue by allowing HIT definitions to be tracked in version control, while CROWDAQ addresses by standardizing the workflow and automated qualification control.

Sprout by Bragg and Weld (2018) is a *meta-framework* similar to the proposed workflow. They focus on *teaching* crowd workers, while CROWDAQ spends most of engineering effort to allow requesters specify the workflow *declaratively* without being a frontend or backend expert.

6 Conclusion

Efficient data collection at scale is important for advancing research and building applications in NLP. Existing workflows typically require multiple annotators, which introduces overhead in building annotation UIs and training and filtering annotators. CROWDAQ is an open-source online platform aimed to reduce this overhead and improve reproducibility via customizable UI components, automated qualification control, and easy-to-reproduce pipelines. The rapid modeling improvements seen in the last few years need a commensurate improvement in our data collection processes, and we believe that CROWDAQ is well-situated to aid in easy, reproducible data collection research.

References

Emily M. Bender and Batya Friedman. 2018. Data statements for natural language processing: Toward mitigating system bias and enabling better science. *Transactions of the Association for Computational Linguistics*, 6.

Samuel Bowman, Gabor Angeli, Christopher Potts, and Christopher D Manning. 2015. A large an-

[8] https://www.figure-eight.com/
[9] https://thehive.ai/
[10] https://labelbox.com/
[11] http://mmax2.net/
[12] https://brat.nlplab.org/about.html
[13] https://github.com/allenai/amti

notated corpus for learning natural language inference. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 632–642.

Jonathan Bragg and Daniel S Weld. 2018. Sprout: Crowd-powered task design for crowdsourcing. In *Proceedings of the 31st Annual ACM Symposium on User Interface Software and Technology*, pages 165–176.

Noam Chomsky and David W Lightfoot. 2002. *Syntactic structures*. Walter de Gruyter.

Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. 2019. BoolQ: Exploring the surprising difficulty of natural yes/no questions. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*, pages 2924–2936.

Jesse Dodge, Suchin Gururangan, Dallas Card, Roy Schwartz, and Noah A. Smith. 2019. Show your work: Improved reporting of experimental results. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 2185–2194.

Dheeru Dua, Yizhong Wang, Pradeep Dasigi, Gabriel Stanovsky, Sameer Singh, and Matt Gardner. 2019. DROP: A reading comprehension benchmark requiring discrete reasoning over paragraphs. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*.

Matt Gardner, Yoav Artzi, Victoria Basmova, Jonathan Berant, Ben Bogin, Sihao Chen, Pradeep Dasigi, Dheeru Dua, Yanai Elazar, Ananth Gottumukkala, Nitish Gupta, Hanna Hajishirzi, Gabriel Ilharco, Daniel Khashabi, Kevin Lin, Jiangming Liu, Nelson F. Liu, Phoebe Mulcaire, Qiang Ning, Sameer Singh, Noah A. Smith, Sanjay Subramanian, Reut Tsarfaty, Eric Wallace, A. Zhang, and Ben Zhou. 2020. Evaluating NLP models via contrast sets. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Timnit Gebru, Jamie Morgenstern, Briana Vecchione, Jennifer Wortman Vaughan, Hanna M. Wallach, Hal Daumé, and Kate Crawford. 2018. Datasheets for datasets. In *Proceedings of the 5th Workshop on Fairness, Accountability, and Transparency in Machine Learning*.

Tomáš Kočiskỳ, Jonathan Schwarz, Phil Blunsom, Chris Dyer, Karl Moritz Hermann, Gábor Melis, and Edward Grefenstette. 2018. The narrativeqa reading comprehension challenge. *Transactions of the Association for Computational Linguistics (TACL)*, 6:317–328.

Qing Li, Qingyi Tao, Shafiq R. Joty, Jianfei Cai, and Jiebo Luo. 2018. VQA-E: Explaining, Elaborating, and Enhancing Your Answers for Visual Questions. In *ECCV*.

Angli Liu, Stephen Soderland, Jonathan Bragg, Christopher H Lin, Xiao Ling, and Daniel S Weld. 2016. Effective crowd annotation for relation extraction. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*, pages 897–906.

Christoph Müller and Michael Strube. 2006. Multi-level annotation of linguistic data with mmax2. *Corpus technology and language pedagogy: New resources, new tools, new methods*, 3:197–214.

Qiang Ning, Hao Wu, Rujun Han, Nanyun Peng, Matt Gardner, and Dan Roth. 2020. TORQUE: A reading comprehension dataset of temporal ordering questions. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Qiang Ning, Hao Wu, and Dan Roth. 2018. A multi-axis annotation scheme for event temporal relations. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 1318–1328. Association for Computational Linguistics.

Constantin Orăsan. 2003. PALinkA: A highly customisable tool for discourse annotation. In *Proceedings of the Fourth SIGdial Workshop of Discourse and Dialogue*, pages 39–43.

Joelle Pineau. 2020. The Machine Learning Reproducibility Checklist. https://www.cs.mcgill.ca/~jpineau/ReproducibilityChecklist.pdf.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. SQuAD: 100,000+ questions for machine comprehension of text. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 2383–2392.

Pontus Stenetorp, Sampo Pyysalo, Goran Topić, Tomoko Ohta, Sophia Ananiadou, and Jun'ichi Tsujii. 2012. Brat: A web-based tool for nlp-assisted text annotation. In *Proceedings of the Conference of the European Chapter of the Association for Computational Linguistics (EACL)*.

Carson T Schütze. 2016. *The empirical base of linguistics: Grammaticality judgments and linguistic methodology*. Language Science Press.

Erik F Tjong Kim Sang and Fien De Meulder. 2003. Introduction to the conll-2003 shared task: Language-independent named entity recognition. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*.

Naushad UzZaman, Hector Llorens, James Allen, Leon Derczynski, Marc Verhagen, and James Pustejovsky.

2013. SemEval-2013 Task 1: TEMPEVAL-3: Evaluating time expressions, events, and temporal relations. *Proceedings of the Joint Conference on Lexical and Computational Semantics (*SEM)*, 2:1–9.