Errudite: Scalable, Reproducible, and Testable Error Analysis

Tongshuang Wu[1], **Marco Tulio Ribeiro**[2], **Jeffrey Heer**[1], and **Daniel S. Weld**[1]

[1]Paul G. Allen School of Computer Science & Engineering, University of Washington
[2]Microsoft Research
{wtshuang,jheer,weld}@cs.washington.edu
marcotcr@microsoft.com

Abstract

Though error analysis is crucial to understanding and improving NLP models, the common practice of manual, subjective categorization of a small sample of errors can yield biased and incomplete conclusions. This paper codifies model and task agnostic principles for informative error analysis, and presents *Errudite*, an interactive tool for better supporting this process. First, error groups should be precisely defined for reproducibility; Errudite supports this with an expressive domain-specific language. Second, to avoid spurious conclusions, a large set of instances should be analyzed, including both positive and negative examples; Errudite enables systematic grouping of relevant instances with filtering queries. Third, hypotheses about the cause of errors should be explicitly tested; Errudite supports this via automated counterfactual rewriting. We validate our approach with a user study, finding that Errudite (1) enables users to perform high quality and reproducible error analyses with less effort, (2) reveals substantial ambiguities in prior published error analyses practices, and (3) enhances the error analysis experience by allowing users to test and revise prior beliefs.

1 Introduction

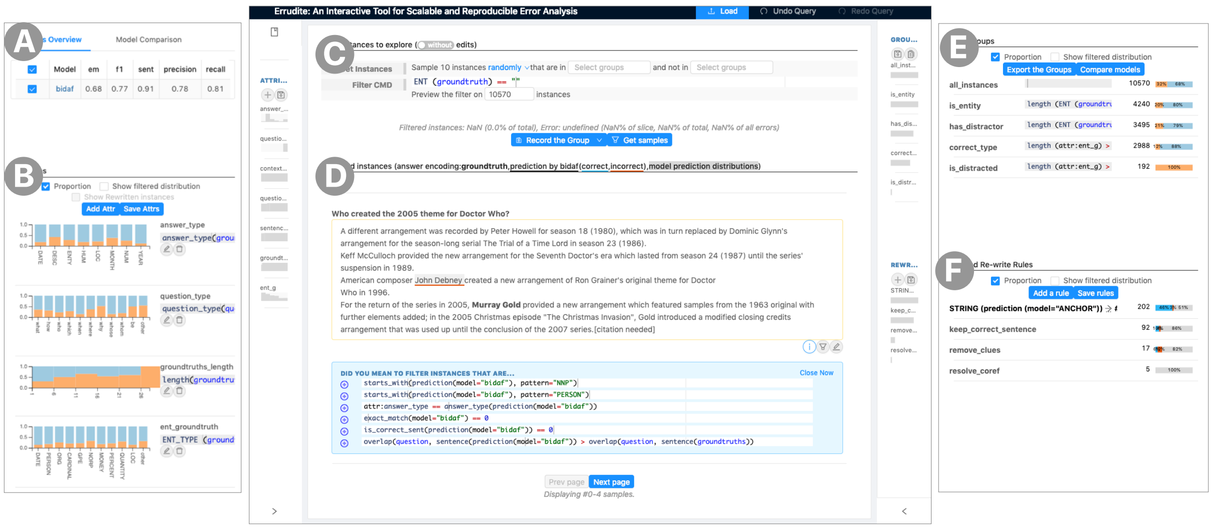
The attempt to analyze when, how, and why models fail (*error analysis*) is a crucial part of the development cycle. Understanding model shortcomings helps NLP developers revise their models, uncover bugs, make deployment decisions, and communicate model performance. Two common forms of error analysis are (1) *data grouping*, where aggregate metrics are computed for particular slices of interest (e.g., accuracy over question types in machine comprehension, per-label performance in semantic role labeling) (Liu et al., 2017; He et al., 2017), and (2) *counterfactual error analysis*, where one modifies the input data to assess if expectations are met, such as adding irrelevant data to see if new errors are introduced (Jia and Liang, 2017; Ribeiro et al., 2018).

In practice, however, groupings and counterfactual tests are very coarse or limited. The input to most NLP tasks is unstructured text, which makes systematic in-depth error analysis challenging. Even answering simple questions such as *"how accurate is my model when person names are involved?"* requires extensive coding, and the use of additional tools such as NER or POS taggers. Due to such difficulties, a common alternative is to group a subset of error samples with manual labels on potential error causes.

While useful, the high cost of manual labeling limits analyses to small samples. We surveyed 10 papers with error analyses that examine a sample of incorrect predictions, e.g., (Wadhwa et al., 2018; Min et al., 2017)[1], and found the sample sizes ranged from 50 to 200 model errors ($\mu = 85.5$, a range corroborated by our user study survey) — frequently covering less than 5% of the total errors. Such small samples are likely unrepresentative of the true error distribution, resulting in high sampling error in the analysis. Furthermore, due to subjectivity, the labels themselves are not precisely defined (Chang et al., 2017). Indeed, our user study (§5) reveals that inter-researcher agreement is very low even for simple labels, an inconsistency that greatly harms reproducibility.

Focusing exclusively on errors — while overlooking successful predictions for instances with similar attributes — may also lead researchers to make biased conclusions, and mistakenly prioritize groups that are in fact well-handled on average (Rondeau and Hazen, 2018). Finally, there may be multiple plausible explanations for an error, with the true cause not immediately apparent.

[1]The full list of papers is provided in Appendix C.



	Model	em	f1	sent	precision	recall
✓	bidaf	0.68	0.77	0.91	0.78	0.81

Proportion ✓ Show filtered distribution ☐ Show Rewritten instances ☐

answer_type	`answer_type(grou`
question_type	`question_type(qu`
groundtruths_length	`length(groundtru`
ent_groundtruth	`ENT_TYPE (ground`

ances to explore (without edits)

et Instances: Sample 10 instances randomly that are in Select groups and not in Select groups

Filter CMD:
```
ENT (groundtruth) == ""
```
Preview the filter on 10570 instances

Filtered instances: NaN (0.0% of total), Error: undefined (NaN% of slice, NaN% of total, NaN% of all errors)

d instances (answer encoding:groundtruth,prediction by bidaf(correct,incorrect),model prediction distributions)

Who created the 2005 theme for Doctor Who?

A different arrangement was recorded by Peter Howell for season 18 (1980), which was in turn replaced by Dominic Glynn's arrangement for the season-long serial The Trial of a Time Lord in season 23 (1986).
Keff McCulloch provided the new arrangement for the Seventh Doctor's era which lasted from season 24 (1987) until the series' suspension in 1989.
American composer John Debney created a new arrangement of Ron Grainer's original theme for Doctor Who in 1996.
For the return of the series in 2005, **Murray Gold** provided a new arrangement which featured samples from the 1963 original with further elements added; in the 2005 Christmas episode "The Christmas Invasion", Gold introduced a modified closing credits arrangement that was used up until the conclusion of the 2007 series.[citation needed]

DID YOU MEAN TO FILTER INSTANCES THAT ARE...
```
starts_with(prediction(model="bidaf"), pattern="NNP")
starts_with(prediction(model="bidaf"), pattern="PERSON")
attr:answer_type == answer_type(prediction(model="bidaf"))
exact_match(model="bidaf") == 0
is_correct_sent(prediction(model="bidaf")) == 0
overlap(question, sentence(prediction(model="bidaf")) > overlap(question, sentence(groundtruths))
```
Displaying #0-4 samples.

oups

Proportion ✓ Show filtered distribution ☐

all_instances		10570	32% 68%
is_entity	`length (ENT (groundtru`	4240	20% 80%
has_distractor	`length (ENT (groundtru`	3495	21% 79%
correct_type	`length (attr:ent_g) >`	2988	12% 88%
is_distracted	`length (attr:ent_g) >`	192	100%

d Re-write Rules

Proportion ✓ Show filtered distribution ☐

STRING (prediction (model="ANCHOR")) → #	202	46% 3% 51%
keep_correct_sentence	92	10% 86%
remove_clues	17	6% 12% 82%
resolve_coref	5	100%

Figure 1: The Errudite interface, with (A) model overview; (B) attribute histograms (§4, enlarged version in Figure 5); (C) filtering panel for users to specify DSL queries (§3.1), (D) instance list displaying filtered examples; (E) list of saved groups (§3.2 and Figure 3) and (F) rewrite rules (§3.3). See §4 for more details. [2]

Q: Who created the 2005 theme for Doctor Who?

> ...John Debney created a new arrangement of Ron Grainer's original theme for Doctor Who in 1996. For the return of the series in 2005, **Murray Gold** provided a new arrangement... featured sampled from the 1963 original.

Figure 2: An example MC error with the **ground truth** and the prediction both being "PERSON" entities.

Figure 2 illustrates an incorrect prediction from a machine comprehension (MC) model that could be caused by the presence of a *distractor* entity with the same type as the ground truth (PERSON), the need to perform multi-sentence reasoning, a combination of both, or something else altogether. In a manual analysis, researchers may gravitate to the first or most salient explanation, without verifying them via counterfactual analysis (e.g., by removing the *distractor*).

We present an error analysis methodology grounded in three *principles*: hypothesized error causes should be (1) formalized in a precise and reproducible manner, (2) applied to all instances rather than a small sample of errors, and (3) tested explicitly via counterfactual analysis. We instantiate these principles in the design of an interactive system called *Errudite*. At the core of Errudite is an expressive domain-specific language (DSL) for precisely querying instances based on linguistic features. The DSL concretizes unambiguous error hypotheses, allows grouping to scale to all instances, and enables rewriting for counterfactual testing. For example, it makes it easy to create a precise group containing all instances where the ground truth and the prediction share entity type (which would include the example in Figure 2), verify how often the model gets distracted, and check if the model turns to the correct entity when the distractor is removed. This sequence is precisely what we use to illustrate the design of Errudite (§3). At each step in the sequence, Errudite helps users inspect and refine their hypotheses in real time with interactive visualizations (Figure 1) and query suggestions based on programming-by-demonstration (§4). We validate our methodology and Errudite via a user study (§5), where MC experts applied it to gain valuable and reproducible insights into model behavior. The same users, when given identical descriptions of an error type from a prior published analysis and asked to reproduce it, produced groups that vary in size from 13.8% to 45.2% of all errors — which illustrates the ambiguity in subjective manual labeling.

In summary, we contribute: (1) an enumeration of key challenges for NLP error analysis: manual, subjective inspection of a small sample of errors can be ambiguous, biased, and miss the root cause of errors; (2) principles for informative error analysis: precise and reproducible, scalable, and testable; (3) the design of Errudite, an interactive graphical tool that instantiates these principles by systematically grouping and rewriting instances using a domain-specific language; and (4) a user study and case studies comparing Errudite with *status quo* error analysis practices. Errudite is available as an open source resource at https://github.com/uwdata/errudite, together with all analyses in this paper for easy replication.

[2] Video demo: https://youtu.be/Dil5i0AYyu8.

2 Task, Dataset, and Model

While our proposed error analysis principles and tool are model and task agnostic, we describe and evaluate them in the context of Machine Comprehension (MC). MC systems aim to answer questions about facts in some reference text (context), potentially requiring complex reasoning (Joshi et al., 2017). Error analysis for MC is challenging by virtue of the fact that both inputs (question and context) and output (answer) are unstructured text, which makes it ideal for our purpose. Furthermore, various prior analyses with particular semantic groups are available for comparison and replication (e.g., cases that involve paraphrasing or coreference (Chen et al., 2016; Weissenborn et al., 2017; Wadhwa et al., 2018)).

Specifically, we analyze Bi-Directional Attention Flow (BiDAF) (Seo et al., 2016) on SQuAD v1.1 (Rajpurkar et al., 2016) in the rest of the paper. SQuAD contains 100,000+ crowdsourced question-answer pairs about Wikipedia articles. Each question refers to one paragraph of an article, and the corresponding answer is guaranteed to be a span in that paragraph context. BiDAF[3] is a hierarchical multi-stage end-to-end neural network. It has been widely referenced as a strong baseline model (Wang et al., 2018; Clark and Gardner, 2017). Because both SQuAD and BiDAF are common in MC, experts can test and verify prior beliefs about model strengths and weaknesses.

3 Error Analysis Principles & Errudite

We identify three principles (abbreviated to the three subsection titles) for effective and unbiased error analysis, and describe tactics in Errudite that instantiate them.[4]

3.1 Precise and Reproducible Hypotheses

Manual labeling of errors involves forming qualitative descriptions that implicitly refer to characteristics of the input and/or model output, often in an ambiguous form. For example, *"the model is bad on long questions"* refers to questions that have more than N tokens, with N left open to interpretation. In order to make error analysis scalable (not dependent on manual labels) and reproducible (unambiguous), our first principle is therefore **P1: error hypotheses should be defined precisely with concrete descriptions**, e.g., describing questions as "longer than 20 tokens" rather than "long." Errudite enables this through a domain-specific language (DSL) with **targets**, **attribute extractors** and **operators**, in increasing order of abstraction.

Targets are primitives which allow users to access inputs and outputs at different levels of granularity, such as the question (`q`), passage context (`c`), ground truth (`g`), the prediction of a model `m` (denoted by `p(m)`), `sentence` and `token`. Targets can be composed, e.g., `sentence(g)` extracts the sentence that contains the ground truth span.

Attribute extractors act on targets to extract fundamental instance metadata (e.g., `length(q)` returns the length of a question). These include (1) basic extractors like `length`, (2) general purpose linguistic features like token `LEMMA`, `POS` tags, and entity (`ENT`) annotations, (3) standard prediction performance metrics such as `f1` or `accuracy`, (4) between-target relations such as `overlap(t1, t2)`, and (5) domain-specific attributes (e.g., for MC or VQA) such as `question_type` and `answer_type` (Wadhwa et al., 2018; Shen et al., 2017). Table 1 provides an abridged listing of extractors, with example values from Figure 2.[5]

Finally, extractors are composable through standard logical and numerical **operators**, serving as building blocks for more complex attributes. For example, to create a boolean attribute that checks if the ground truth span contains an entity, the `!=` operator is used, yielding `ENT(g)!=""`. A more complex example is counting the number of times the ground truth entity appears in the passage context: `count(token(c, pattern=ENT(g)))`. Being reusable and composable makes extractors much more expressive than predefined attributes, and helps formulate much richer hypotheses.

Errudite's data grouping and rewriting (introduced below) are both supported by these abstractions in the DSL. Precise hypotheses and queries enable reproducible analyses that can be shared between research groups, and automatically applied to new datasets and models.

3.2 Analyze All Relevant Instances

Random spot checking of errors can lead to confirmation bias and spurious conclusions (Rondeau

[3]We used the implementation from Allennlp (Gardner et al., 2017): `https://allennlp.org/models`.

[4]Additional use cases on Machine Comprehension (MC) and Visual Question Answering (VQA) are in Appendix A.

[5]A complete list is available in Appendix D.

Function Name	Definition	DSL Code and resulting Output Values

sentence, token	Extractors for desired spans from targets — sentences or sub-phrases.	`sentence(g)`→`For the return of...`
		`token(c,pattern="PERSON")`→`[John,...]`

exact_match, f1, is_correct_sent	Performance functions that measure different levels of correctness.	`f1(m) == 0`, `exact_match(m) == 0`
		`is_correct_sent(m) == False`

length	Length of the target.	`length(q) == 9`, `length(g) == 2`
POS, ENT, LEMMA	Tokens in the target that have certain patterns of POS tags, named entity, etc.	`ENT(g,get_root=True) == "PERSON"`

has_pattern, starts_with, ends_with	To check whether the target contains certain pattern. `pattern` automatically detects queries on POS tags and entity types.	`starts_with(q,pattern="who VBZ") == True`
		`has_pattern(g,pattern="PERSON") == True`

overlap(t1,t2)	The ratio of `t1` tokens that also occur in `t2`.	`overlap(q, sentence(g)) == 0.25`

Table 1: Definitions for a subset of attribute extractors, including sample values from the example in Figure 2.

and Hazen, 2018). To avoid these, we propose **P2: error prevalence should be assessed over the entire dataset.** Grouping queries created with the DSL can scale the analysis to cover not only errors that are otherwise missed by small samples, but also *correct cases* that are typically overlooked. We now provide an example that illustrates the pitfalls of not following this principle, and how including all of the relevant successes and failures can lead to different insights than looking at a small sample of mistakes.

Distractor Example. The *distractor hypothesis* states that BiDAF is good at matching questions to entity types (e.g., knowing when a PERSON is expected as an answer), but is often distracted by other spans with the same entity type (e.g., other PERSONs), leading to wrong predictions as the in Figure 2. This is a hypothesis independently raised by four out of ten user study participants (§5).[6]

Consider the group `is_distracted`, defined by the following query:

```
ENT(g) != ""                                      1
and count(token(c, pattern=ENT(g))) >             2
   count(token(g, pattern=ENT(g)))                3
and ENT(g) == ENT(p(m))                           4
and f1(m) == 0                                    5
```

The query can be broken down into the following conditions: the ground truth is an entity (line 1); there are potential distractors – i.e., there are more tokens matching the ground truth entity type (`ENT(g)`) in the whole context than in the ground truth (lines 2-3); the prediction entity type matches the ground truth one (line 4); and the prediction is incorrect (line 5). Starting from all instances, we can subset groups by applying these conditions successively in order. Errudite conveys useful statistics about the groups via visualizations, as in Figure 3.

[6]Participants tested the hypothesis for a specific entity type (numbers). We present a more general case here.

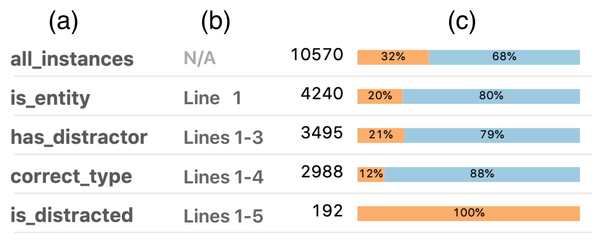


Figure 3: Saved groups with their (a) manually created and semantically meaningful names, (b) query definitions, and (3) sizes and error rates (orange indicates errors, blue indicates correct predictions.)

If we *only* consider `is_distracted`, without also considering correct predictions, we might conclude that the distractor hypothesis is correct: the 192 instances in the group are all cases where BiDAF predicts a wrong span that has the same entity type as the ground truth, and the group accounts for 5.7% of all BiDAF errors. However, looking at the groups in succession reveals a different, and more complete story: BiDAF predicts the exact correct span (exact match) 68% of the time overall, which rises to 80% when the ground truth is an entity. When other entities with the same type are present in the passage, BiDAF is still 79% accurate (i.e., it is not particularly worse when there are potential distractors), and conditioned on having matched the question to the right entity type, it is quite accurate (88% exact match). The user study participants who previously believed the distractor hypothesis either rejected or revised it after creating similar groups.

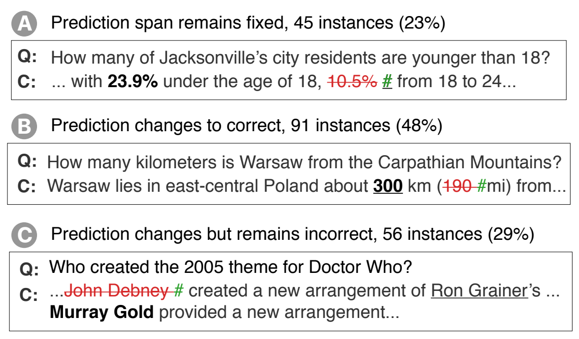
3.3 Explicitly Test Error Hypotheses

In the example from the previous section, the presence of distractors in the context of a wrong prediction does not necessarily indicate that distractors were the root cause of the mistake. To isolate the essential cause of errors, we state **P3: error hypotheses should be explicitly tested.** This requires answers to counterfactual questions, such as “If the predicted distractor was not there, would the model predict correctly?”

Errudite allows manual editing of individual examples (i.e., changing the input arbitrarily), a common practice to verify if the suspected error causes are really causes. While useful for quick spot tests on single instances, manual editing does not scale. For scalable counterfactual analysis, Errudite uses **rules** to rewrite all relevant instances within a group – similar to search and replace but with the flexibility and power of the Errudite DSL.

A rewrite rule is specified using the syntax `rewrite(target,from→to)`, where **target** indicates the part of the instance that should be rewritten by replacing **from** with **to**. Both **from** and **to** can include linguistic annotations, in ALL CAPS. A rule to replace “Who” followed by a verb with “What person” followed by the same verb is written as `rewrite(q,"who VERB"→"what person VERB")`. For convenience, Errudite also includes *default rules* suggested in formative interviews with MC experts, such as *“remove all sentences except the one that contains the ground truth”*, and *“replace pronouns (he) with raw references (John Smith) from a coreference model.”*

Returning to our distractor example, we can verify whether distractors are causing mistakes by using a rewrite rule on the `is_distracted` group, replacing the predicted distractor with a non-entity, placeholder token `"#"`: `rewrite(c, STRING(p(m))→"#")`.



A Prediction span remains fixed, 45 instances (23%)

Q: How many of Jacksonville’s city residents are younger than 18?
C: ... with **23.9%** under the age of 18, ~~10.5%~~ # from 18 to 24...

B Prediction changes to correct, 91 instances (48%)

Q: How many kilometers is Warsaw from the Carpathian Mountains?
C: Warsaw lies in east-central Poland about **300** km (~~190~~ #mi) from...

C Prediction changes but remains incorrect, 56 instances (29%)

Q: Who created the 2005 theme for Doctor Who?
C: ...~~John Debney~~ # created a new arrangement of Ron Grainer’s ... **Murray Gold** provided a new arrangement...

Figure 4: Updated prediction in response to the rewrite rule `rewrite(c,STRING(p(m))→"#")`.

The results from the rewrite rule are presented in Figure 4. The model predicts the same span (now containing the meaningless token `"#"`) 23% of the time (A), changes to the correct span 48% (B) of the time and predicts a different wrong span 29% of the time (C). While case B indicates that the distractor was indeed causing a misprediction, in case A it seems other factors are at play. In case C, further analysis indicates that the predicted span is almost always a different distractor (i.e., has the same entity type). Thus, while BiDAF is fairly accurate when the distractors are present and the entity type is matched (88%), *when it is incorrect*, it seems distractors are indeed confusing the model. This kind of analysis is rarely seen (if at all) in the literature; yet it helps users develop insights not available through data grouping.

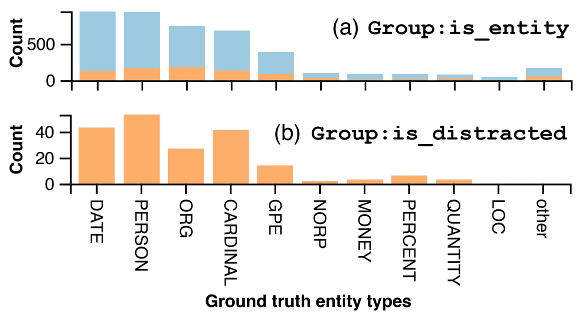


Figure 5: The distribution of `ENT(g)` in group (a) `is_entity` and (b) `is_distracted`. The histogram shows the absolute frequency and incorrect/correct ratio for each attribute value.

4 Interactive User Interface

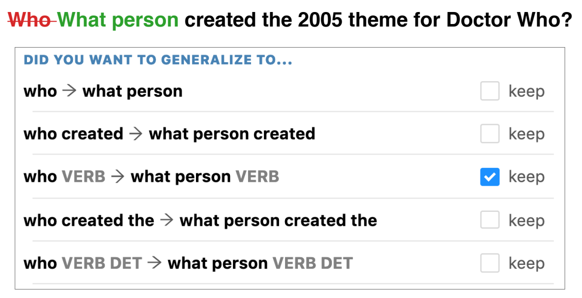
We now walk through the interactive interface of Errudite in more detail. The interface not only integrates the entire analysis process, but also provides additional exploration support such as visualizing data distributions, suggesting potential queries, and presenting the grouping and rewriting results. While not strictly necessary for the error analysis principles previously outlined, it makes their application much more straightforward by helping users formulate and inspect their hypotheses in real time, and at scale (P2).

Attribute distribution. To guide the exploration, group creation and refinement, Errudite supports defining complex attributes and inspecting their distributions. An example in Figure 5 shows the histogram of ground truth entity types. It displays the relative frequency of different entity answers, as well as the proportion of incorrect predictions. The histograms are updated to show conditional distributions when a user selects a group. Figure 5(a) shows histograms for the ground truth entity type in the group `is_entity`: when the answer is an entity, it is most often a DATE, PERSON, ORG, or CARDINAL. Figure 5(b) displays the same histogram for the group `is_distracted`. We note that the frequency of "distraction" mistakes for PERSON and CARDINAL are higher, while lower for ORG, relative to the base frequencies in Figure 5(a), an insight that may warrant further investigation.

Programming-by-Demonstration. To make it easier for users to formulate group queries and rewrite rules, interactive selections can trigger suggestions for related DSL statements. If a user selects any text span in an instance in the central browser, she is shown suggestions for related queries. For example, selecting "John" in Figure 1 (or Figure 2) triggers the following suggestions:

```
starts_with(p(m), pattern="NNP")              1
starts_with(p(m), pattern="PERSON")           2
answer_type(g) == answer_type(p(m))           3
exact_match(m) == 0                           4
is_correct_sent(m) == False                   5
overlap(q, sentence(p(m))) >                  6
   overlap(q, sentence(g))                    7
```

These suggestions cover pattern searches (lines 1-2) ranked by their occurrence frequency and error rate, and target comparisons (lines 4-7), which are particularly relevant when the prediction or ground truth is selected. Selecting a different text span yields different suggestions, heuristically ranked and filtered with the goal of surfacing queries likely to be of interest.



~~Who~~ What person created the 2005 theme for Doctor Who?

DID YOU WANT TO GENERALIZE TO...

who → what person	☐ keep
who created → what person created	☐ keep
who VERB → what person VERB	✓ keep
who created the → what person created the	☐ keep
who VERB DET → what person VERB DET	☐ keep

Figure 6: Rewrite rules inferred from an edit to an individual instance.

For rewrite rules, we use a technique inspired by Ribeiro et al. (2018) to generalize manual edits into suggested rewrite rules: including context, POS tags and named entities, attempting to maximize coverage and relevance without redundancy. Figure 6 shows an example in which various suggestions are displayed after a user rewrites an instance by changing "Who" to "What person." Appendix B provides a more detailed description of our searching and ranking criteria.

Layout. The UI (Figure 1) contains three main components. The central component contains a filter panel (C) and an instance browser (D), which help examine the results of data groupings or rewrite rules for iterative refinement. The collapsible sidebar on the left contains a list of different models being analyzed with summary statistics (A) and customizable attribute histograms (B). The one on the right contains a list of saved data groups (E) and rewrite rules (F); these can be loaded into the central component via mouse click. All groups and rewrite rules can be saved and loaded through the interface, so the analysis can be easily shared and reproduced.

5 User Study

We conducted user studies to evaluate Errudite. Though less common in NLP, this type of evaluation is widely used in fields like Human-Computer Interaction for understanding how certain methods or systems impact the intended user group (Nielsen, 1994; Olsen Jr, 2007) — precisely our objective here. We recruited ten participants with prior Machine Comprehension experience (developed 1-6 models each, $\mu = 3.1$, $\sigma = 2.02$) for a 90-minute study: four NLP graduate students and six researchers or QA engineers from industry. Participants analyzed BiDAF on SQuAD v1.1.

User studies can take various forms, ranging from experiments that quantitatively compare human performance, to interviews or observational studies that qualitatively inspect users' behaviors and perspectives. We take a more qualitative approach, as we are primarily interested in how Errudite shapes participants' error analysis experience. The study started with a background survey about users' prior experience in MC and error analysis. After a walk-through tour of Errudite (described in Appendix A.3), participants were asked to perform two tasks: **Replication** (§5.1), in which they attempted to reproduce the error analysis from Seo et al. (2016); and **Exploration** (§5.2), in which they freely explored the model and reported their findings. We collected multiple subjective measures from participants in the form of five-point Likert scale ratings (Likert, 1932), with 5 being strongly positive and 1 strongly negative. Participants were compensated at a rate of $25/hr.

5.1 Task 1: Replication

The goal of this task was two-fold: (1) to verify if Errudite is flexible enough to support the creation of groups traditionally labeled by hand, and

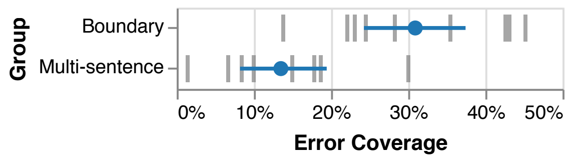
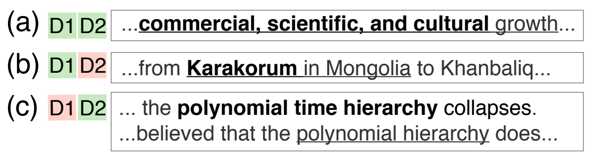


Figure 7: Percentage of errors covered by user-defined groups: *Boundary* (μ = 30.9%, σ = 10.5%) and *Multi-sentence* (μ = 13.5%, σ = 8.29%). The dispersion of grey ticks shows that users come up with different definitions for groups described by Seo et al. (2016), even when they think they replicated the group faithfully.



(a) D1 D2 ...**commercial, scientific, and cultural** growth...

(b) D1 D2 ...from **Karakorum** in Mongolia to Khanbaliq...

(c) D1 D2 ... the **polynomial time hierarchy** collapses.
...believed that the polynomial hierarchy does...

Figure 8: Example instances that fall into different user-defined *Boundary* groups.

(2) to assess the reproducibility of current ad-hoc error analysis methods. Seo et al. (2016) manually labeled 50 instances predicted incorrectly by BiDAF into different error groups. We asked participants to generalize these groups to the whole validation set after reading the relevant section in Seo et al. (2016). For learning purposes, we first asked users to inspect two data groups that we created using Errudite, and evaluate if they captured the same semantics as the original group: incorrect preprocessing (*Preprocess*) and paraphrase problems (*Paraphrase*). Users then created their own groups to replicate two others from Seo et al. (2016): imprecise answer boundaries (*Boundary*) and multi-sentence issues (*Multi-sentence*).

Results. Participants rated the accuracy of the replication of each group after seeing a variety of examples, i.e., "how close the approximation matches the paper definition." For the groups we wrote queries for, participants were confident that *Preprocess* was accurate (μ = 4.3, σ = 0.64), but ambivalent towards *Paraphrase* (μ = 3.1, σ = 0.54). Participants' comments indicated the ambivalence did not come from Errudite: 6 participants disagreed with the example given by Seo et al. (2016), and participants who gave low ratings found *Paraphrase* itself too fuzzy and confusing to formalize. Despite being used widely as an error group (Kundu and Ng, 2018; Chen et al., 2016), participants had conflicting understandings of *Paraphrase*, either as "the question and the ground truth sentence are semantically similar but with great lexical variations", or "the predicted answer is a paraphrased version of the ground truth."

When replicating groups themselves, participants were able to express the queries they wanted. Participants were not very confident in the accuracy of their produced *Multi-sentence* group (μ = 2.8, σ = 1.32), for reasons similar to *Paraphrase*: they thought the group was under-specified in the original analysis. More interestingly, users were the most confident in the fidelity of an apparently "easy" group *Boundary* (μ = 4.8, σ = 0.60), yet the groups they produced were wildly different (Figure 7). While users were able to express what they thought was meant by "imprecise error boundaries", they applied different definitions.

For example, one user defined the group as (D1) "the predicted span can be off by at most two tokens both on the left and right" (yielding 22.1% of all BiDAF errors), while another defined it as (D2) "there is no exact match but high overlap — $F1$ is higher than 0.7" (yielding 13.8% of all errors). Figure 8 shows samples that fit the two definitions or just one of them. Errudite makes the different interpretations explicit. The author of D2 observed examples like Figure 8(c) in his samples, but decided ultimately that what mattered was just the returned short text, not the span index. In contrast, D1's author carefully refined his initial query precisely to rule out cases like Figure 8(c).

In summary, users were able to express their intended groups well with Errudite, but they were unable to consistently replicate the analysis of Seo et al. (2016) — even when they thought they did — due to the ambiguity inherent in manual grouping.

5.2 Task 2: Exploration

To assess the usefulness of Errudite, we let participants freely analyze BiDAF. We asked them to "think aloud" in real time, vocalizing their hypotheses, intriguing observations, objectives, and expectations. At the end of the session, subjects rated each of their discovered insights in terms of (1) importance (very trivial to very helpful), (2) confidence in insight correctness, and (3) relative ease of discovery compared to existing methods.

Results. All participants found at least one insight by building semantically meaningful groups or rewrite rules. On average, subjects reported μ = 2.1 findings (σ = 0.94). Some insights *confirmed prior hypotheses* about BiDAF more formally, increasing users' confidence. For example, one user created a group to verify that mistakes frequently

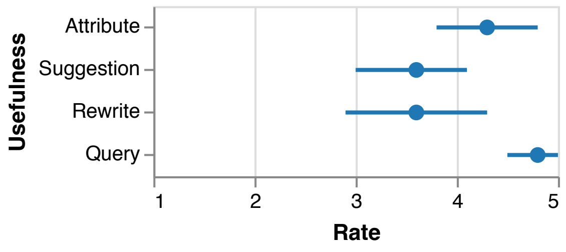


Figure 9: Usefulness of function modules in the tool.

occur when there is significant overlap between the question and a sentence that does not contain the ground truth. Indeed, that group accounts for about 18% of BiDAF errors. Other insights *extended previous knowledge*, such as explorations by two users who examined low performance on "why" questions (Appendix A.4). They also *rejected some prior hypotheses* after using Errudite, such as the distractor case in §3. Participants rated their findings to be important (μ = 3.7, σ = 1.12), were confident that their findings were valid (μ = 4.0, σ = 1.05), and consistently agreed that Errudite made finding insights easier (μ = 4.9, σ = 0.35). Participants agreed that they *learned more about the model* (μ = 3.9, σ= 0.94), and valued Errudite's support for assessing their hypotheses.

5.3 Usability and User Feedback

When rating the usefulness of different components of the tool (Figure 9), users rated the DSL (μ = 4.8, σ = 0.40) and the attribute distribution (μ = 4.3, σ = 0.78) as very useful, and rated query suggestions (μ= 3.6, σ = 0.91) and rewrite rules (μ = 3.6, σ = 1.11) as potentially useful. We hypothesize that rewrite rules pose a learning curve that makes them difficult to evaluate in a single session. This kind of counterfactual analysis is not common and a few participants were concerned about possible unintended side effects of edits.

We also asked participants to describe their impressions with free-form comments, which were very positive for all of them – all thought Errudite enhanced their error analysis experience. In particular, four users stated that they felt it *systematically scaled up* the analysis, making it more precise and thus inspiring more confidence. Five users noted *how much faster exploration became* with Errudite, and how having a good set of building blocks and visualizations let them bypass the large coding overhead needed to otherwise test a single hypothesis about a model.

6 Related Work

6.1 Data Grouping

Non-manual data grouping typically follows one of two extremes. Most of the literature relies on data groups that are very coarse and easy to program (e.g., based on question length and answer types (Kafle and Kanan, 2017; Agrawal et al., 2016; Shen et al., 2017)). While useful and accessible, they do not allow more semantically meaningful observations (like *distractors* or *paraphrases*). In contrast, some define groups that are highly specific to a particular dataset or model, such as hand-crafting factors to quantify MC instance difficulties (Rondeau and Hazen, 2018). While often insightful, these suffer from potential pitfalls similar to labeling individual instances: they are laborious, often subjective, and hard to reproduce. In other words, just as in manual error labeling mentioned in §1, typical automatic grouping also struggles with the trade-off between being reproducible/scalable, and being in-depth and meaningful. In contrast, Errudite addresses the challenge with an expressive domain-specific language, which helps users build filters that can slice the entire dataset, and thereby build scalable and semantically meaningful groups.

Chung et al. (2018) made a similar attempt to balance the trade-off in Slice Finder, a framework that uses statistical techniques to identify large and interpretable slices that models perform poorly on. However, their purely automated data slicing does not allow users to customize groups based on their own hypotheses. Furthermore, Slice Finder only uses predefined attributes. While this is reasonable in the context of structured data classifier that they tested (with features explicitly defined), it is not flexible enough for unstructured text in NLP. Other interactive error analysis tools tend to face similar customization issues. QADiver (Lee et al., 2018) enriches question attributes in SQuAD 2.0 by including factors like word frequencies and question-context word match ratios, but users cannot query or create groups based on these attributes. QSAnglyzer (Chen and Kim, 2017) aims at category-oriented analysis by pre-defining seven groups for QA models, but there is limited support for group customization. ActiVis (Kahng et al., 2018) allows for flexible data attribute and group definitions, but only supports group creation prior to the interactive process. Rarely does a user to know which group they want to inspect beforehand, and thus it is to be expected that users would revert to coarse and easy-to-program groups. Errudite emphasizes customization: it allows users to define extractors for rich instance attributes, and helps them adjust their groups in real-time with quick trial and error, visualizations, and suggestions based on programming-by-demonstration.

6.2 Counterfactual Analysis

Counterfactual attacks to models have taken various forms, e.g., by adding distracting sentences to the context in MC (Jia and Liang, 2017), or feeding partial questions or wrong images into models (Agrawal et al., 2016; Mudrakarta et al., 2018; Feng et al., 2018). Slightly closer to our work is SEARs (Ribeiro et al., 2018) (also incorporated into QADiver), which also takes the form of rewrite rules: it generates semantic-preserving rules that cause models to change predictions. However, these focus on robustness, i.e., counterfactual perturbations are mainly for the purpose of detecting over-stability or over-sensitivity. In contrast, our counterfactual analysis is for the purpose of understanding *why* models fail in certain groups. Furthermore, our DSL allows for more complex counterfactual rules and for applying rules only to certain groups, such as "delete the predicted distractor for instances in the `is_distracted` group." As far as we know, such analysis is novel, and a promising direction for more in-depth error analysis.

7 Conclusion and Future Work

In this paper, we characterize deficiencies with current error analysis methods used in NLP: they are laborious and subjective, which can lead to high variance and low reproducibility. Moreover, by focusing on error cases independent of situations where the model is correct, they can yield biased results. Finally, since it is difficult to perform counterfactual analysis, the root cause of errors can easily be overlooked.

In response, we identify three principles required for successful error analysis, and present an interactive tool called Errudite to enable their application: (1) building precise instances groups with composable building blocks in a domain-specific language; (2) scaling the analysis to cover all the relevant successes and failures by automatically building large groups with filtering queries, and providing visual summaries for them; and (3) testing error hypotheses using counterfactual analysis by rewriting the instances with rules. Data groups and rewrite rules can be easily saved and shared for replication or for analysis of different models with the same groups and rules.

We conduct a detailed user study with NLP experts, confirming that Errudite makes hypothesis definitions both concrete and apparent, reduces sampling bias, and helps researchers verify the true causes of errors. We find that Errudite significantly lowers the barrier for insightful error analysis, hopefully leading to a more in-depth understanding of current models, and to safeguard deployments and improve the state of the art.

While our primary experiments are on Machine Comprehension, the DSL primitives in Errudite are general enough to make extensions to other tasks and domains straightforward. For example, we have extended Errudite to Visual Question Answering with only minor adjustments to the performance metrics and the instance browser (to include images). We share case studies in Appendices A.1 and A.2, together with further analysis on SQuAD (Appendix A.4). Similar adjustments could be done to extend Errudite to other tasks such as Machine Translation, Natural Language Inference, and text classification, along with customization of domain-specific attributes.

In the future, we hope to design and evaluate more sophisticated query and attribute suggestions to guide exploration by less expert users, as well as social features that facilitate collaboration within an organization to promote sharing, review, reuse, and extension of error analyses.

Acknowledgements

The project was supported by the Moore Foundation Data-Driven Discovery Investigator program, with additional support from ONR grant N00014-18-1-2193, the WRF/Cable Professorship and the Allen Institute for Artifical Intelligence. We gratefully thank Gagan Bansal, Quanze Chen, Matthew Conlen, Jane Hoffswell, Eric Horvitz, Ece Kamar, Younghoon Kim, Benjamin Lee, Halden Lin, Yang Liu, Besmira Nushi, Zening Qu, and Akshat Shrivastava for their helpful comments. We also appreciate the valuable input from our user study participants, and the constructive comments from the anonymous reviewers.

References

Aishwarya Agrawal, Dhruv Batra, and Devi Parikh. 2016. Analyzing the behavior of visual question answering models. *arXiv preprint arXiv:1606.07356*.

Betty van Aken, Julian Risch, Ralf Krestel, and Alexander Löser. 2018. Challenges for toxic comment classification: An in-depth error analysis. *arXiv preprint arXiv:1809.07572*.

Stanislaw Antol, Aishwarya Agrawal, Jiasen Lu, Margaret Mitchell, Dhruv Batra, C Lawrence Zitnick, and Devi Parikh. 2015. Vqa: Visual question answering. In *Proceedings of the IEEE international conference on computer vision*, pages 2425–2433.

Joseph Chee Chang, Saleema Amershi, and Ece Kamar. 2017. Revolt: Collaborative crowdsourcing for labeling machine learning datasets. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, pages 2334–2346. ACM.

Danqi Chen, Jason Bolton, and Christopher D Manning. 2016. A thorough examination of the cnn/daily mail reading comprehension task. *arXiv preprint arXiv:1606.02858*.

Nan-Chen Chen and Been Kim. 2017. Qsanglyzer: Visual analytics for prismatic analysis of question answering system evaluations. In *Proceedings of the IEEE Conference on Visual Analytics Science and Technology*.

Yeounoh Chung, Tim Kraska, Neoklis Polyzotis, Ki Hyun Tae, and Steven Euijong Whang. 2018. Automated data slicing for model validation:a big data - ai integration approach.

Christopher Clark and Matt Gardner. 2017. Simple and effective multi-paragraph reading comprehension. *arXiv preprint arXiv:1710.10723*.

Anthony Fader, Luke Zettlemoyer, and Oren Etzioni. 2013. Paraphrase-driven learning for open question answering. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 1608–1618.

Shi Feng, Eric Wallace, Alvin Grissom II, Pedro Rodriguez, Mohit Iyyer, and Jordan Boyd-Graber. 2018. Pathologies of neural models make interpretation difficult. In *Empirical Methods in Natural Language Processing*.

Matt Gardner, Joel Grus, Mark Neumann, Oyvind Tafjord, Pradeep Dasigi, Nelson F. Liu, Matthew Peters, Michael Schmitz, and Luke S. Zettlemoyer. 2017. Allennlp: A deep semantic natural language processing platform.

Sumit Gulwani and Prateek Jain. 2017. Programming by examples: Pl meets ml. In *Asian Symposium on Programming Languages and Systems*, pages 3–20. Springer.

Luheng He, Kenton Lee, Mike Lewis, and Luke Zettlemoyer. 2017. Deep semantic role labeling: What works and what's next. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 473–483.

Minghao Hu, Yuxing Peng, Zhen Huang, Nan Yang, Ming Zhou, et al. 2018. Read+ verify: Machine reading comprehension with unanswerable questions. *arXiv preprint arXiv:1808.05759*.

Robin Jia and Percy Liang. 2017. Adversarial examples for evaluating reading comprehension systems. *arXiv preprint arXiv:1707.07328*.

Mandar Joshi, Eunsol Choi, Daniel S Weld, and Luke Zettlemoyer. 2017. Triviaqa: A large scale distantly supervised challenge dataset for reading comprehension. *arXiv preprint arXiv:1705.03551*.

Kushal Kafle and Christopher Kanan. 2017. An analysis of visual question answering algorithms. In *Computer Vision (ICCV), 2017 IEEE International Conference on*, pages 1983–1991. IEEE.

Minsuk Kahng, Pierre Y Andrews, Aditya Kalro, and Duen Horng Polo Chau. 2018. Activis: Visual exploration of industry-scale deep neural network models. *IEEE transactions on visualization and computer graphics*, 24(1):88–97.

Sean Kandel, Andreas Paepcke, Joseph Hellerstein, and Jeffrey Heer. 2011. Wrangler: Interactive visual specification of data transformation scripts. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 3363–3372. ACM.

Vahid Kazemi and Ali Elqursh. 2017. Show, ask, attend, and answer: A strong baseline for visual question answering. *arXiv preprint arXiv:1704.03162*.

Souvik Kundu and Hwee Tou Ng. 2018. A question-focused multi-factor attention network for question answering. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

Tessa A. Lau, Steven A. Wolfman, Pedro M. Domingos, and Daniel S. Weld. 2003. Programming by demonstration using version space algebra. *Machine Learning*, 53:111–156.

Gyeongbok Lee, Sungdong Kim, and Seung-won Hwang. 2018. Qadiver: Interactive framework for diagnosing qa models. *arXiv preprint arXiv:1812.00161*.

Xin Li and Dan Roth. 2002. Learning question classifiers. In *Proceedings of the 19th international conference on Computational linguistics-Volume 1*, pages 1–7. Association for Computational Linguistics.

Rensis Likert. 1932. A technique for the measurement of attitudes. *Archives of psychology*.

Xiaodong Liu, Yelong Shen, Kevin Duh, and Jianfeng Gao. 2017. Stochastic answer networks for machine reading comprehension. *arXiv preprint arXiv:1712.03556*.

Sewon Min, Minjoon Seo, and Hannaneh Hajishirzi. 2017. Question answering through transfer learning from large fine-grained supervision data. *arXiv preprint arXiv:1702.02171*.

Sewon Min, Victor Zhong, Richard Socher, and Caiming Xiong. 2018. Efficient and robust question answering from minimal context over documents. *arXiv preprint arXiv:1805.08092*.

Pramod Kaushik Mudrakarta, Ankur Taly, Mukund Sundararajan, and Kedar Dhamdhere. 2018. Did the model understand the question? *arXiv preprint arXiv:1805.05492*.

Jakob Nielsen. 1994. *Usability engineering*. Elsevier.

Dan R Olsen Jr. 2007. Evaluating user interface systems research. In *Proceedings of the 20th annual ACM symposium on User interface software and technology*, pages 251–258. ACM.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. Squad: 100,000+ questions for machine comprehension of text. *arXiv preprint arXiv:1606.05250*.

Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. 2018. Semantically equivalent adversarial rules for debugging nlp models. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 856–865.

Marc-Antoine Rondeau and Timothy J Hazen. 2018. Systematic error analysis of the stanford question answering dataset. In *Proceedings of the Workshop on Machine Reading for Question Answering*, pages 12–20.

Minjoon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hannaneh Hajishirzi. 2016. Bidirectional attention flow for machine comprehension. *arXiv preprint arXiv:1611.01603*.

Yelong Shen, Xiaodong Liu, Kevin Duh, and Jianfeng Gao. 2017. An empirical analysis of multiple-turn reasoning strategies in reading comprehension tasks. *arXiv preprint arXiv:1711.03230*.

Soumya Wadhwa, Khyathi Raghavi Chandu, and Eric Nyberg. 2018. Comparative analysis of neural qa models on squad. In *Proceedings of the Workshop on Machine Reading for Question Answering*.

Yizhong Wang, Kai Liu, Jing Liu, Wei He, Yajuan Lyu, Hua Wu, Sujian Li, and Haifeng Wang. 2018. Multi-passage machine reading comprehension with cross-passage answer verification. *arXiv preprint arXiv:1805.02220*.

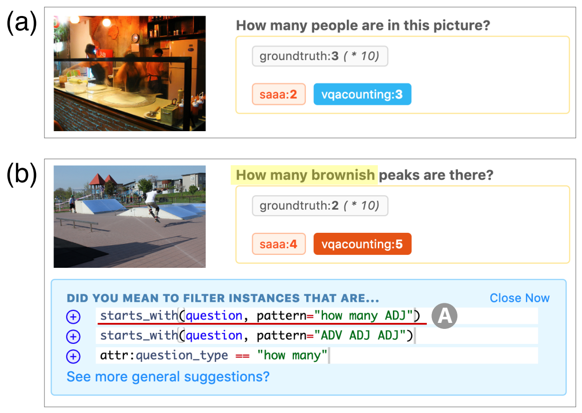
Dirk Weissenborn, Georg Wiese, and Laura Seiffe. 2017. Making neural qa as simple as possible but not simpler. *arXiv preprint arXiv:1703.04816*.

Yan Zhang, Jonathon Hare, and Adam Prügel-Bennett. 2018. Learning to count objects in natural images for visual question answering. *arXiv preprint arXiv:1802.05766*.

A Additional Use Cases

We use case studies in Visual Question Answering and Machine Comprehension to further demonstrate the usefulness of Errudite. A video demo is available at `https://youtu.be/Dil5i0AYyu8`.

A.1 VQA: Breaking down "How many"


(a)

How many people are in this picture?

groundtruth:3 (* 10)

saaa:2 vqacounting:3

(b)

How many brownish peaks are there?

groundtruth:2 (* 10)

saaa:4 vqacounting:5

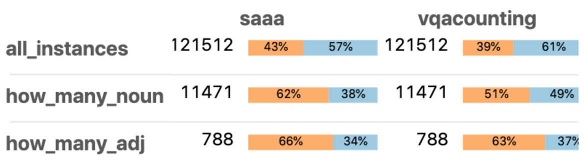
DID YOU MEAN TO FILTER INSTANCES THAT ARE... Close Now

```
starts_with(question, pattern="how many ADJ")
starts_with(question, pattern="ADV ADJ ADJ")
attr:question_type == "how many"
```

Figure 10: Two "how many" examples: VQACounting improves on SAAA for instance (a), but predicts an even higher count in (b). Highlighting "how many brownish", we create groups based on the suggested query **A**.

We demonstrate Errudite's power on comparing multiple models in the context of Visual Question Answering (VQA). We analyze SAAA (Kazemi and Elqursh, 2017) and VQACounting (Zhang et al., 2018) concurrently on the validation set of VQA v1 (Antol et al., 2015), which contains 21,512 instances. VQACounting is built on top of SAAA, with increased performance on counting questions. Querying "how many" questions, we notice two interesting cases in Figure 10: VQACounting correctly predicts the "how many people" question in (a), but is worse than SAAA (also wrong) in (b). We suspect the token following "how many" can make a difference. Highlighting "how many brownish", we follow the first returned suggestion (Figure 10A) to build a `how_many_ADJ` group (`starts_with(q, pattern="how many ADJ")`), and similarly, a `how_many_noun` group.

Per-group comparison shows VQACounting improves SAAA more on `"how many NOUN"` than `"how many ADJ"` (Figure 11) questions: the former has an increase of accuracy from 38% to 49%, whereas the latter only shows 3% improvement. However, note the group size difference: the `NOUN` group is 14 times larger than `ADJ`. In fact,



	saaa			vqacounting		
all_instances	121512	43%	57%	121512	39%	61%
how_many_noun	11471	62%	38%	11471	51%	49%
how_many_adj	788	66%	34%	788	63%	37%

Figure 11: VQACounting improves much more on `how_many_NOUN`, compared to `how_many_ADJ`, though many fewer instances follow the latter pattern.

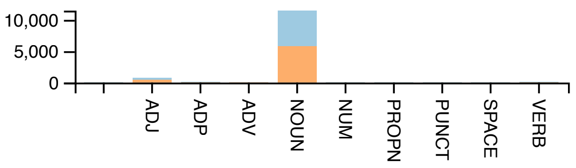


Figure 12: Extracting the POS tag for the token immediately after "how many", we notice most instances follow a `"how many NOUN"` pattern.

extracting the POS tags following "how many" into an attribute, we see `"NOUN"` drastically stands out, suggesting a very imbalanced data distribution (Figure 12).

A.2 VQA: Ambiguous Questions

With groups and attributes independent of models or predictions, Errudite can help analyze the consistencies and ambiguities of the datasets. In this case, we use Errudite to group all the "ambiguous VQA questions", or questions where the answers exhibit high human annotator disagreement. If humans cannot agree on the answer, it is to be expected that machine learning models will not be accurate. In the annotations for the VQA v1 dataset, each question collects up to 10 human answers, while in evaluation an answer is considered fully accurate if it matches the answer of at least three humans. We count the unique ground truth annotations for each instance (`count(g)`), which results in the distribution shown in Figure 13: instances with more ground truth labels are more poorly predicted. Querying for instances with `count(g) > 5`, we find many instances like the ones in Figure 14, covering 29.9% of all the errors. This means the dataset is far from "clean" and that 30% of the model's mistakes should probably not be considered mistakes.

A.3 MC: Incorrect Pre-processing

We used the following case as the tutorial demo in our user study (§5). When sorting instances by their *F1* score, instances like those in Figure 15(a) appear. Due to incorrect tokenization, BiDAF

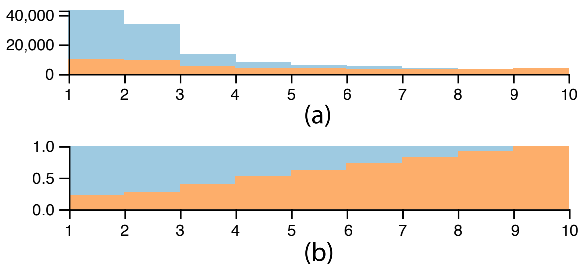
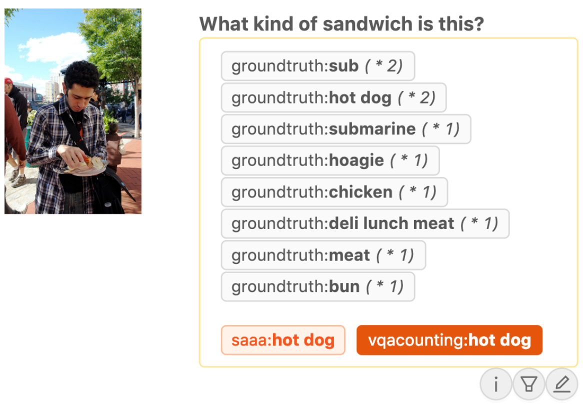


Figure 13: The VQACounting model becomes worse as the count of distinct ground truth annotations (i.e., human disagreement) grows: (a) shows instance counts in each ground truth count bin; (b) with the counts normalized, emphasizing the incorrect proportions.



What kind of sandwich is this?

groundtruth:**sub** *(* 2)*
groundtruth:**hot dog** *(* 2)*
groundtruth:**submarine** *(* 1)*
groundtruth:**hoagie** *(* 1)*
groundtruth:**chicken** *(* 1)*
groundtruth:**deli lunch meat** *(* 1)*
groundtruth:**meat** *(* 1)*
groundtruth:**bun** *(* 1)*

saaa:**hot dog** vqacounting:**hot dog**

Figure 14: An instance with 8 distinct ground truth annotations, and neglible inter-annotator agreement.

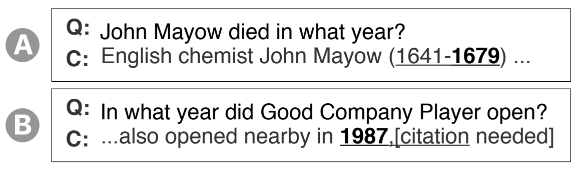
treats "1641-1679" as one token, and its mismatch with the ground truth "1679" evaluation (which is token-wise) will result in *F1* = 0. We simulate the above tokenization issue with a query that states (a) even though at the character level the ground truth answer is a substring of the prediction, (b) the two don't have token level overlap:

```
STRING(g) in STRING(p(m)) and f1(m) == 0
```

Among the 26 instances returned (0.8% of all incorrect instances), we find multiple instances like the ones in Figure 15(a), and also unexpected cases like Figure 15(b).

It is unclear from these examples if tokenization is the only issue. To further assess, we define a rewrite rule that separates dashes from nearby words: `rewrite(sentence(g), "-"` → `" - ")`.

The rewritten instances are then queryable using a wrapper function: `apply(func, rewrite="rule_name")` runs the query function `func` on the new instances generated by rule `"rule_name"`. We use the queries in Figure 16 to further divide the 13 instances rewritten (the rule cannot edit additional cases like in Figure 15(b)):

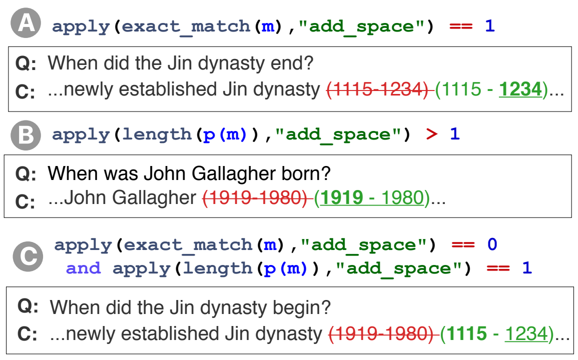


A
Q: John Mayow died in what year?
C: English chemist John Mayow (1641-**1679**) ...

B
Q: In what year did Good Company Player open?
C: ...also opened nearby in **1987**,[citation needed]

Figure 15: Two instances suspected to be wrong due to tokenization.

4 were predicted correctly after the rewrite, 5 remained the same (with spaces added), and 4 returned a different incorrect span after the rewrite. This counterfactual analysis confirms that these errors are not solely due to preprocessing errors.



A `apply(exact_match(m),"add_space") == 1`
Q: When did the Jin dynasty end?
C: ...newly established Jin dynasty ~~(1115-1234)~~ (1115 - **1234**)...

B `apply(length(p(m)),"add_space") > 1`
Q: When was John Gallagher born?
C: ...John Gallagher ~~(1919-1980)~~ (**1919** - 1980)...

C `apply(exact_match(m),"add_space") == 0`
`and apply(length(p(m)),"add_space") == 1`
Q: When did the Jin dynasty begin?
C: ...newly established Jin dynasty ~~(1919-1980)~~ (**1115** - 1234)...

Figure 16: Three types of changed predictions on instances generated with `add_space`.

A.4 MC: "Why" Questions

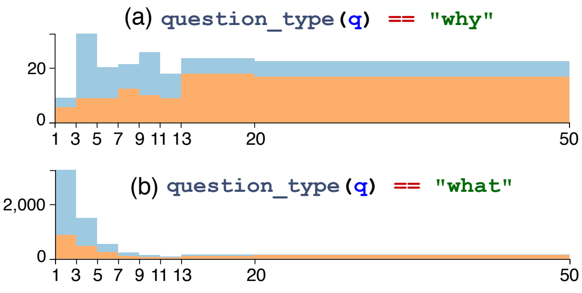
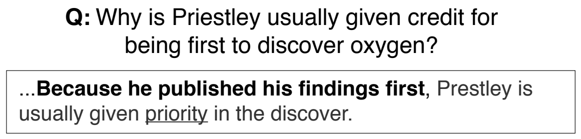


Figure 17: The prediction length `length(p(m))` is much longer for (a) only "why" questions than for (b) only "what" questions.

We merge two cases from our user study in §5 to demonstrate how participants P1 and P2 can start with similar attributes and then diverge and discover complementary insights.

Both participants started by grouping "why" questions, as they observed them to have much lower performance than other primary question types. P1 realized these questions had longer predictions, and the ground truths were usually a



Q: Why is Priestley usually given credit for being first to discover oxygen?

...**Because he published his findings first**, Prestley is usually given priority in the discover.

Figure 18: A "why" question where the model ignored the apparent hint "because."

small substring of the prediction (with multiple unnecessary tokens on both ends). Meanwhile, "what" questions have relatively shorter predictions. He hypothesized that reframing "why" to "what" questions could result in reasonable prediction lengths, and created a rule `rewrite(q, "Why VERB"→"What is the reason that")` to confirm it. Out of 151 rewritten instances, 46 had shorter predictions, and 6 had longer ones; the remaining instances had unchanged predictions. Out of the 19 instances where *F1* improved after the rewrite (`apply(f1(m), rewrite="why_to_what") > f1(m)`), 13 had the prediction shortened to approximately the correct ground truth answer.

P2 found the example in Figure 18 and chose a different angle. He was surprised to see the incorrect prediction, when the ground truth contained the word "because", which should make the prediction easier for BiDAF. Grouping all "why" questions with a `"because"` in their context:

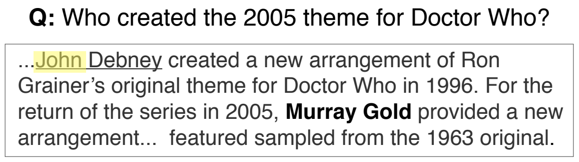
```
question_type(q) == "why"
and has_pattern(c, pattern="because")
```

he found most instances still had a prediction following `"because"`, and that removing `"because"` from the context made predictions worse. He confirmed that "because" was indeed an essential signal. The prediction in Figure 18 remained the same, and P2 therefore hypothesized that aggressive pattern matching affected this instance, as all the words surrounding the prediction "priority" were in the question. He was also surprised that there were only 40 instances in the `because` group, and suggested more labeling might easily help bump up the performance.

The two participants explored complementary angles on "why" question, suggesting the value of collaborative sharing among Errudite users.

B Programming-by-Demonstration

To help users express their intent, Errudite supports programming by demonstration (PBD) (Gulwani and Jain, 2017), a well-recognized technique for synthesizing targeted programs from specific examples. It has been widely applied to tasks like data wrangling (Kandel et al., 2011) and text editing (Lau et al., 2003). Here, we explain the heuristics used for ranking query suggestions and extracting rewrite rules.



Q: Who created the 2005 theme for Doctor Who?

...John Debney created a new arrangement of Ron Grainer's original theme for Doctor Who in 1996. For the return of the series in 2005, **Murray Gold** provided a new arrangement... featured sampled from the 1963 original.

Figure 19: The illustrating example we used in the paper; we repeat it here to explain our programming-by-demonstration heuristics. The scenario here assumes "John" is selected by a user.

B.1 Query Ranking

Pattern	R_e	C_d	S_u
`"NNP"`	27.1%	35.7%	1.90
`"PERSON"`	22.1%	10.3%	0.56
`"john"`	20.1%	0.4%	0.40
`"how many ADJ"`	62.9%	0.6%	1.27
`"ADV ADJ ADJ"`	62.5%	0.7%	1.26

Table 2: Patterns and their associated usefulness in Figure 19 (top 3 lines) and Figure 10 (bottom 2 lines)

As users interact with instances, Errudite detects and returns potential queries that can assist generalization from a single observation to a larger set. As running examples, we explain our query ranking methods assuming "John" is selected in Figure 19, and "How many brownish" is selected in Figure 10. There are three broad types of suggestions with different granularity. To ensure diversity, our suggestions cover at least one query from each type, and the inter-type suggestion ranking will always be as the following:

Span-related suggestions closely relate to the specific token(s) selected ("John" in Figure 19). The most typical span-related suggestions are pattern searches. We generate a list of possible linguistic patterns from the cross-product of raw token text with POS tags (coarse for multiple tokens, and fine-grained for single tokens), as well as the entity type (if any). The resulting possible patterns for "John" are `"John"`, `"NNP"`, `"PERSON"`. Similarly, in Figure 10, "how many brownish" results in `"how many brownish"`, `"how many ADJ"`, `"ADV ADJ ADJ"`, etc. The functional predicate used differs if the selected span lies at the beginning, middle, or end of a target (`start_with`, `has_pattern`, and `end_with`).

Target-related suggestions are based on the target under inspection. For instance, we return `question_type` when a user interacts with the question (`q`) in Figure 10. A prediction (`p`) as in Figure 19 will instead trigger different levels of comparisons with the ground truth, including accuracy checks (`exact_match` and `is_correct_sent`), answer type comparisons (`ENT(p) == ENT(q)`), answer offsets (`answer_offset_delta`) and sentence level comparisons (`overlap`):

```
answer_type(g) == answer_type(p(m))
exact_match(m) == 0
is_correct_sent(m) == False
overlap(q, sentence(p(m))) >
   overlap(q, sentence(g))
```

Instance-level suggestions are conventional attributes that domain experts often find useful. For example, performance, question type, and answer type are considered the most important "instance" suggestions if they are not triggered by the target-related suggestions. In addition, lengths of inputs also belong to this suggestion type.

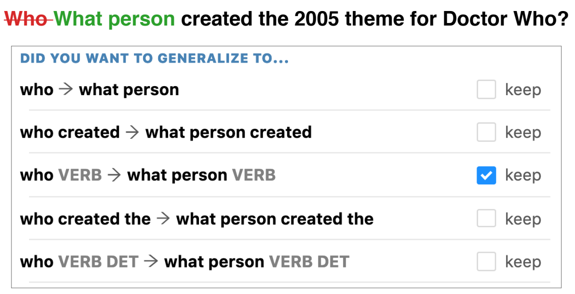
To perform intra-group ranking, we precompute the resulting groups for each candidate suggestion, and rank their in-group error rate R_e and dataset coverage C_d, maximizing a usefulness score:

$$S_u = \frac{R_e}{|C_d - 50\%|} \quad (1)$$

Intuitively, R_e measures group difficulty. We would like to prioritize patterns that will return subsets that are not well-handled on average, resulting in high in-group error rate. The $|C_d - 50\%|$ term, on the other hand, ensures reasonable coverage. We prioritize groups that lean towards 50% coverage of the entire validation set, so to penalize patterns that cover too few instances to be significant, or those covering too many instances that essentially return the entire dataset. Taking the ranking of span-related suggestions as an example, candidate patterns for Figure 19 and Figure 10, and their scores S_u, are shown in Table 2.

B.2 Rewrite Rule Extraction

When a source x is edited to x', we propose a set of rules $R = \{r_1, ..., r_m\}$ in the same manner as Ribeiro et al. (2018): we test the exact matching, and select the minimal contiguous sequence that turns x to x', with their immediate contexts and linguistic features. While Ribeiro et al. (2018) use only text and POS tags, we further extend to include entity types.



~~Who~~ What person created the 2005 theme for Doctor Who?

DID YOU WANT TO GENERALIZE TO...

who → what person	keep
who created → what person created	keep
who VERB → what person VERB	[✓ keep]
who created the → what person created the	keep
who VERB DET → what person VERB DET	keep

Figure 20: Rewrite rules inferred from an edit on an individual instance.

Then, we apply every rule in the candidate set onto a random subset of instances $S = \{s_1, ..., s_n\}, n = 100$. Similar to Ribeiro et al. (2018), we prioritize rules that have (1) high coverage and (2) low redundancy, while loosening their constraint on semantic equivalence: rules resulting in different semantics are still valid in our error cause testing context. In addition, we heuristically score the linguistic features used based on their specificity: we consider raw text the most specific, POS tag the least, and penalize rules that are too general and abstract (as they are likely to result in unexpected changes). For example, in addition to the rules reported in Figure 20, an additional rule found in the candidate set from "Who" to "What person" was `"NOUN"`→`"What person"`. By editing random `NOUN`s, this rule will have high coverage, but our specificity score weights it down enough that `"Who"`→`"What person"` is ranked more highly. We then report the five highest-ranked candidate rules to the user.

C Survey: Error Analysis Sample Sizes

Table 3 lists the 10 papers we surveyed to inspect the scale of the *status quo* error analysis practice. Papers are randomly selected from top tier conferences, and either develop novel MC models (our primary test case), or focus on error analysis.

Paper	Sample size
(Seo et al., 2016)	50
(Kundu and Ng, 2018)	50
(Hu et al., 2018)	50
(Min et al., 2018)	50
(Weissenborn et al., 2017)	55
(Chen et al., 2016)	100
(Min et al., 2017)	100
(Wadhwa et al., 2018)	100
(Fader et al., 2013)	100
(van Aken et al., 2018)	200
Average	*85.5*

Table 3: Surveyed papers and their error sample sizes.

D DSL Documentation

Here we list the functions defined in our domain-specific language for MC and VQA.

Converters and Targets

Get targets: These targets contain text spans post-processed with state-of-the-art POS taggers, lemmatizers and NER models, along with metadata such as example id, or (in the answer case) the model that generated it. When additional metadata is not used, `Target` can be treated just as `Span` in a function, or a piece of text with its linguistic features.

1. `question|context|groundtruth`→`Target`: Automatically query the target object (`Question` and `Answer` in VQA and MC, as well as `Context` in MC).
2. `prediction(model:str)`→`Target`: Get the prediction object of a given model.

Converters that extract sub-spans, short phrases, or sentences from targets.

1. `token(span:Span,idxes:int|int[],pattern:str)`→`Token|Token[]`: Get a list of tokens from the target based on `idxes` (sub-list) and `pattern` (in the form of, for example, `"(what, which) NOUN)"`. `pattern` automatically detects queries on POS tags and entity types.
2. `sentence(target:Target,shift:int|int[])`→`Span`: *[MC only]* Get the sentence that contains a given answer. Shift indicates if neighboring sentences should be included. If `shift==0`, then the actual sentence is returned; if `shift==[-2,-1,1,2]`, then the four sentences surrounding the answer sentence are returned.

General Computation

1. `apply(func:Callable,rewrite:str)`→`any`: Applies query functions to instances rewritten by the named rule `rewrite`.
2. `abs(num:float|int)`→`float|int`: Returns the absolute value.
3. `truncate(num:float|int, min_value:float|int, max_value::float|int)`→`float|int`: Clamps a given number to a given domain.
4. `is_digit(input:any)`→`bool`: Determines if an input is a number, or – in the case of a string input – if it can be parsed into a number.
5. `digitize(input:any)`→`float|int`: Parses an input into a number if `is_digit(input) == True`; Otherwise returns `None`.
6. `length(span:Span)`→`int`: The length of a given span, in tokens.
7. `[has_any|has_all](container:Span,contained:Span)`→`int`: Determines whether one list `container` contains any (or all) of the members present in another lists.
8. `count(vars:list)`→`int`: Count the number of members in the input list.
9. `freq(target:Target,target_type:str)`→`str`: Returns the frequency of a token occurring in the training data, given a target_type (`"question"` or `"answer"` in MC; However, `freq` can be on other targets given other tasks).

Linguistic Attributes

1. `[LEMMA|POS|TAG|ENT](span:Span,get_root:bool,pattern:str)`→`str|str[]`: Return the specified linguistic feature of a span with one more more tokens. If pattern is specified (the same as in `token`), gets the sub-list of spans in the span list. If `get_root==True`, gets the single linguistic feature of the “primary” token, or the one within the ground truth span that is highest in the dependency parsing tree.
2. `STRING(span:Span)`→`str`: Get the raw string from a given span.
3. `[has_pattern|starts_with|ends_with](span:Span,pattern:str)`→`bool`: To determine whether the targeted span contains a certain pattern.

Performance Metrics

1. `[f1|exact_match|precision|recall|accuracy|confidence](model:str)`→`float`: Get the specified performance metric for one instance, given the selected model. Confidence is for

both QA and VQA, which is usually the model prediction probability. Accuracy is for VQA, and the others are for QA.

2. `is_correct_sent(model:str)`→`bool`: *[MC only]* Determine if the given model locates the sentence with the ground truth, regardless of span-level correctness.

Between-target Relations

1. `overlap(span1:Span,span2:Span,pattern:str)`→`float`: A directional overlapping: returns the ratio of tokens in `span1` that also occur in `target2`. If `pattern` is provided, it is used to filter to matching tokens in `span1` and `target2`. For example, if `pattern=="NOUN"`, then the overlap will only be on tokens with a NOUN tag.

Domain-Specific Attributes

1. `question_type(question:Target)`→`str`: Returns the question type: either the WH-word or the first word in a sentence.
2. `answer_type(answer:Answer)`→`str`: Returns the answer type, computed based on TREC (Li and Roth, 2002) and the named entities of the answer. Returns one of the following: ABBR, DESC, ENTY, HUM, LOC, NUM.
3. `answer_offset_delta(prediction:Answer, direction:str)`→`int`: *[MC only]* Compute the offset between prediction and ground truth in the left or right direction. Returns the position difference.
4. `answer_offset_span(prediction:Answer, direction:str)`→`Span`: *[MC only]* Compute the offset between prediction and ground truth in the left or right direction. Returns the actual span(s).
5. `dep_distance(answer:Answer,pattern:str)`→`float`: *[MC only]* Dependency distance between a key question token and the answer token. The key is computed by finding tokens that do not occur frequently in the context and is not far from the given answer. Pattern fixes the keyword linguistic feature.